

---

# **MPC5606BK Microcontroller Reference Manual**

**Devices Supported:**

**MPC5606BK  
MPC5605BK**

**MPC5606BKRM**

Rev. 2  
05/2014

This page is intentionally left blank.

# Chapter 1

## Preface

|       |  |    |
|-------|--|----|
| 1.1   | Overview .....                           | 21 |
| 1.2   | Audience .....                           | 21 |
| 1.3   | Guide to this reference manual .....     | 21 |
| 1.4   | Register description conventions .....   | 25 |
| 1.5   | References .....                         | 26 |
| 1.6   | How to use the MPC5606BK documents ..... | 26 |
| 1.6.1 | The MPC5606BK document set .....         | 26 |
| 1.6.2 | Reference manual content .....           | 27 |
| 1.7   | Using the MPC5606BK .....                | 28 |
| 1.7.1 | Hardware design .....                    | 28 |
| 1.7.2 | Input/output pins .....                  | 29 |
| 1.7.3 | Software design .....                    | 29 |
| 1.7.4 | Other features .....                     | 30 |

# Chapter 2

## Introduction

|        |  |    |
|--------|--|----|
| 2.1    | The MPC5606BK microcontroller family .....               | 31 |
| 2.2    | MPC5606BK device comparison .....                        | 31 |
| 2.3    | Device block diagram .....                               | 32 |
| 2.4    | Feature details .....                                    | 35 |
| 2.4.1  | e200z0h core processor .....                             | 35 |
| 2.4.2  | Crossbar switch (XBAR) .....                             | 35 |
| 2.4.3  | Interrupt Controller (INTC) .....                        | 35 |
| 2.4.4  | System Integration Unit Lite (SIUL) .....                | 36 |
| 2.4.5  | Flash memory .....                                       | 36 |
| 2.4.6  | SRAM .....   | 38 |
| 2.4.7  | Memory Protection Unit (MPU) .....                       | 38 |
| 2.4.8  | Boot Assist Module (BAM) .....                           | 38 |
| 2.4.9  | Enhanced Modular Input Output System (eMIOS) .....       | 39 |
| 2.4.10 | Deserial Serial Peripheral Interface Module (DSPI) ..... | 40 |
| 2.4.11 | Controller Area Network module (FlexCAN) .....           | 40 |
| 2.4.12 | System clocks and clock generation .....                 | 41 |
| 2.4.13 | System timers .....                                      | 42 |
| 2.4.14 | System watchdog timer .....                              | 43 |
| 2.4.15 | Inter-Integrated Circuit (I2C) module .....              | 43 |
| 2.4.16 | On-chip voltage regulator (VREG) .....                   | 43 |
| 2.4.17 | Analog-to-Digital Converter (ADC) .....                  | 44 |
| 2.4.18 | Enhanced Direct Memory Access controller (eDMA) .....    | 45 |
| 2.4.19 | Cross Trigger Unit (CTU) .....                           | 45 |
| 2.4.20 | Serial communication interface module (LINFlex) .....    | 46 |
| 2.4.21 | JTAG Controller (JTAGC) .....                            | 47 |
| 2.5    | Developer support .....                                  | 47 |

## Chapter 3 Memory Map

## Chapter 4 Signal description

|     |                       |    |
|-----|-----------------------|----|
| 4.1 | Package pinouts ..... | 53 |
| 4.2 | Pin muxing .....      | 55 |

## Chapter 5 Microcontroller Boot

|       |   |    |
|-------|---|----|
| 5.1   | Boot mechanism .....                                | 75 |
| 5.1.1 | Flash memory boot .....                             | 76 |
| 5.1.2 | Serial boot mode .....                              | 78 |
| 5.1.3 | Censorship .....                                    | 78 |
| 5.2   | Boot Assist Module (BAM) .....                      | 83 |
| 5.2.1 | BAM software flow .....                             | 83 |
| 5.2.2 | LINFlex (RS232) boot .....                          | 91 |
| 5.2.3 | FlexCAN boot .....                                  | 92 |
| 5.3   | System Status and Configuration Module (SSCM) ..... | 94 |
| 5.3.1 | Introduction .....                                  | 94 |
| 5.3.2 | Features .....                                      | 94 |
| 5.3.3 | Modes of operation .....                            | 95 |
| 5.3.4 | Memory map and register description .....           | 95 |

## Chapter 6 Clock Description

|       |  |     |
|-------|--|-----|
| 6.1   | Clock architecture .....   | 105 |
| 6.2   | Clock gating .....   | 106 |
| 6.3   | Fast external crystal oscillator (FXOSC) digital interface ..... | 107 |
| 6.3.1 | Main features .....  | 107 |
| 6.3.2 | Functional description .....                                     | 107 |
| 6.3.3 | Register description .....                                       | 108 |
| 6.4   | Slow external crystal oscillator (SXOSC) digital interface ..... | 109 |
| 6.4.1 | Introduction .....   | 109 |
| 6.4.2 | Main features .....  | 109 |
| 6.4.3 | Functional description .....                                     | 109 |
| 6.4.4 | Register description .....                                       | 110 |
| 6.5   | Slow internal RC oscillator (SIRC) digital interface .....       | 111 |
| 6.5.1 | Introduction .....   | 111 |
| 6.5.2 | Functional description .....                                     | 112 |
| 6.5.3 | Register description .....                                       | 112 |
| 6.6   | Fast internal RC oscillator (FIRC) digital interface .....       | 113 |
| 6.6.1 | Introduction .....   | 113 |
| 6.6.2 | Functional description .....                                     | 113 |
| 6.6.3 | Register description .....                                       | 114 |

|       |   |     |
|-------|---|-----|
| 6.7   | Frequency-modulated phase-locked loop (FMPLL) | 115 |
| 6.7.1 | Introduction                                  | 115 |
| 6.7.2 | Overview                                      | 115 |
| 6.7.3 | Features                                      | 115 |
| 6.7.4 | Memory map                                    | 116 |
| 6.7.5 | Register description                          | 116 |
| 6.7.6 | Functional description                        | 120 |
| 6.7.7 | Recommendations                               | 122 |
| 6.8   | Clock monitor unit (CMU)                      | 123 |
| 6.8.1 | Introduction                                  | 123 |
| 6.8.2 | Main features                                 | 123 |
| 6.8.3 | Block diagram                                 | 124 |
| 6.8.4 | Functional description                        | 124 |
| 6.8.5 | Memory map and register description           | 126 |

## Chapter 7 Clock Generation Module (MC\_CGM)

|       |   |     |
|-------|---|-----|
| 7.1   | Overview  | 131 |
| 7.2   | Features  | 132 |
| 7.3   | Modes of operation                                  | 133 |
| 7.3.1 | Normal and reset modes of operation                 | 133 |
| 7.4   | External signal description                         | 133 |
| 7.5   | Memory map and register definition                  | 133 |
| 7.5.1 | Register descriptions                               | 137 |
| 7.5.2 | Output Clock Division Select Register (CGM_OCDS_SC) | 138 |
| 7.5.3 | System Clock Select Status Register (CGM_SC_SS)     | 139 |
| 7.6   | Functional Description                              | 142 |
| 7.6.1 | System Clock Generation                             | 142 |
| 7.6.2 | Output Clock Multiplexing                           | 143 |
| 7.6.3 | Output Clock Division Selection                     | 144 |

## Chapter 8 Mode Entry Module (MC\_ME)

|       |  |     |
|-------|--|-----|
| 8.1   | Overview                                   | 145 |
| 8.1.1 | Features                                   | 145 |
| 8.1.2 | Modes of operation                         | 146 |
| 8.2   | External signal description                | 147 |
| 8.3   | Memory map and register definition         | 147 |
| 8.3.1 | Register descriptions                      | 150 |
| 8.4   | Functional description                     | 163 |
| 8.4.1 | Mode transition request                    | 163 |
| 8.4.2 | Modes details                              | 164 |
| 8.4.3 | Mode transition process                    | 167 |
| 8.4.4 | Protection of mode configuration registers | 175 |
| 8.4.5 | Mode transition interrupts                 | 175 |

|       |                           |     |
|-------|---------------------------|-----|
| 8.4.6 | Application example ..... | 177 |
|-------|---------------------------|-----|

## Chapter 9 Reset Generation Module (MC\_RGM)

|       |  |     |
|-------|--|-----|
| 9.1   | Introduction .....                       | 179 |
| 9.1.1 | Overview .....                           | 179 |
| 9.1.2 | Features .....                           | 180 |
| 9.1.3 | Modes of operation .....                 | 181 |
| 9.2   | External signal description .....        | 181 |
| 9.3   | Memory map and register definition ..... | 182 |
| 9.3.1 | Register descriptions .....              | 183 |
| 9.4   | Functional description .....             | 188 |
| 9.4.1 | Reset state machine .....                | 188 |
| 9.4.2 | Destructive resets .....                 | 191 |
| 9.4.3 | External reset .....                     | 192 |
| 9.4.4 | Functional resets .....                  | 192 |
| 9.4.5 | Alternate event generation .....         | 192 |
| 9.4.6 | Boot mode capturing .....                | 193 |

## Chapter 10 Power Control Unit (MC\_PCU)

|        |  |     |
|--------|--|-----|
| 10.1   | Introduction .....                       | 195 |
| 10.1.1 | Overview .....                           | 195 |
| 10.1.2 | Features .....                           | 196 |
| 10.1.3 | Modes of operation .....                 | 196 |
| 10.2   | External signal description .....        | 197 |
| 10.3   | Memory map and register definition ..... | 197 |
| 10.3.1 | Register descriptions .....              | 199 |
| 10.4   | Functional description .....             | 202 |
| 10.4.1 | General .....                            | 202 |
| 10.4.2 | Reset / Power-On Reset .....             | 203 |
| 10.4.3 | MC_PCU configuration .....               | 203 |
| 10.4.4 | Mode transitions .....                   | 203 |
| 10.5   | Initialization information .....         | 205 |
| 10.6   | Application information .....            | 206 |
| 10.6.1 | STANDBY Mode Considerations .....        | 206 |

## Chapter 11 Voltage Regulators and Power Supplies

|        |  |     |
|--------|--|-----|
| 11.1   | Voltage regulators .....                 | 207 |
| 11.1.1 | High power regulator (HPREG) .....       | 207 |
| 11.1.2 | Low power regulator (LPREG) .....        | 207 |
| 11.1.3 | Ultra low power regulator (ULPREG) ..... | 208 |
| 11.1.4 | LVDs and POR .....                       | 208 |
| 11.1.5 | VREG digital interface .....             | 208 |

|        |                                 |     |
|--------|---------------------------------|-----|
| 11.1.6 | Register description .....      | 209 |
| 11.2   | Power supply strategy .....     | 209 |
| 11.3   | Power domain organization ..... | 210 |

## Chapter 12 Wakeup Unit (WKPU)

|         |  |     |
|---------|--|-----|
| 12.1    | Overview .....   | 213 |
| 12.2    | Features .....   | 215 |
| 12.3    | External signal description .....                                  | 216 |
| 12.4    | Memory map and register description .....                          | 216 |
| 12.4.1  | Memory map .....   | 216 |
| 12.4.2  | NMI Status Flag Register (NSR) .....                               | 217 |
| 12.4.3  | NMI Configuration Register (NCR) .....                             | 218 |
| 12.4.4  | Wakeup/Interrupt Status Flag Register (WISR) .....                 | 219 |
| 12.4.5  | Interrupt Request Enable Register (IRER) .....                     | 219 |
| 12.4.6  | Wakeup Request Enable Register (WRER) .....                        | 220 |
| 12.4.7  | Wakeup/Interrupt Rising-Edge Event Enable Register (WIREER) .....  | 220 |
| 12.4.8  | Wakeup/Interrupt Falling-Edge Event Enable Register (WIFEER) ..... | 221 |
| 12.4.9  | Wakeup/Interrupt Filter Enable Register (WIFER) .....              | 221 |
| 12.4.10 | Wakeup/Interrupt Pullup Enable Register (WIPUER) .....             | 222 |
| 12.5    | Functional description .....                                       | 222 |
| 12.5.1  | General .....  | 222 |
| 12.5.2  | Non-maskable interrupts .....                                      | 223 |
| 12.5.3  | External wakeups/interrupts .....                                  | 224 |
| 12.5.4  | On-chip wakeups .....  | 226 |

## Chapter 13 Real Time Clock / Autonomous Periodic Interrupt (RTC/API)

|        |   |     |
|--------|---|-----|
| 13.1   | Overview .....                                  | 227 |
| 13.2   | Features .....                                  | 227 |
| 13.3   | Device-specific information .....               | 229 |
| 13.4   | Modes of operation .....                        | 229 |
| 13.4.1 | Functional mode .....                           | 229 |
| 13.4.2 | Debug mode .....                                | 230 |
| 13.5   | Register descriptions .....                     | 230 |
| 13.5.1 | RTC Supervisor Control Register (RTCSUPV) ..... | 230 |
| 13.5.2 | RTC Control Register (RTCC) .....               | 231 |
| 13.5.3 | RTC Status Register (RTCS) .....                | 233 |
| 13.5.4 | RTC Counter Register (RTCCNT) .....             | 234 |
| 13.6   | RTC functional description .....                | 234 |
| 13.7   | API functional description .....                | 235 |

## Chapter 14 CAN Sampler

|      |                    |     |
|------|--------------------|-----|
| 14.1 | Introduction ..... | 237 |
|------|--------------------|-----|

|        |   |     |
|--------|---|-----|
| 14.2   | Main features .....                       | 237 |
| 14.3   | Memory map and register description ..... | 238 |
| 14.3.1 | Control Register (CR) .....               | 238 |
| 14.3.2 | CAN Sampler Sample Registers 0–11 .....   | 239 |
| 14.4   | Functional description .....              | 239 |
| 14.4.1 | Enabling/disabling the CAN sampler .....  | 240 |
| 14.4.2 | Selecting the Rx port .....               | 240 |
| 14.4.3 | Baud rate generation .....                | 241 |

## Chapter 15 e200z0h Core

|        |   |     |
|--------|---|-----|
| 15.1   | Overview .....                              | 245 |
| 15.2   | Microarchitecture summary .....             | 245 |
| 15.3   | Block diagram .....                         | 247 |
| 15.4   | Features .....                              | 247 |
| 15.4.1 | Instruction unit features .....             | 248 |
| 15.4.2 | Integer unit features .....                 | 248 |
| 15.4.3 | Load/Store unit features .....              | 249 |
| 15.4.4 | e200z0h system bus features .....           | 249 |
| 15.5   | Core registers and programmer's model ..... | 249 |

## Chapter 16 Enhanced Direct Memory Access (eDMA)

|        |  |     |
|--------|--|-----|
| 16.1   | Device-specific features .....                 | 253 |
| 16.2   | Introduction .....                             | 253 |
| 16.2.1 | Features .....                                 | 254 |
| 16.3   | Memory map and register definition .....       | 255 |
| 16.3.1 | Memory map .....                               | 255 |
| 16.3.2 | Register descriptions .....                    | 257 |
| 16.4   | Functional description .....                   | 278 |
| 16.4.1 | eDMA basic data flow .....                     | 280 |
| 16.5   | Initialization / application information ..... | 283 |
| 16.5.1 | eDMA initialization .....                      | 283 |
| 16.5.2 | DMA programming errors .....                   | 285 |
| 16.5.3 | DMA request assignments .....                  | 286 |
| 16.5.4 | DMA arbitration mode considerations .....      | 286 |
| 16.5.5 | DMA transfer .....                             | 287 |
| 16.5.6 | TCD status .....                               | 290 |
| 16.5.7 | Channel linking .....                          | 291 |
| 16.5.8 | Dynamic programming .....                      | 292 |

## Chapter 17 eDMA Channel Multiplexer (DMA\_MUX)

|      |                    |     |
|------|--------------------|-----|
| 17.1 | Introduction ..... | 295 |
| 17.2 | Features .....     | 295 |



|        |   |     |
|--------|---|-----|
| 17.3   | Modes of operation .....                                | 296 |
| 17.4   | External signal description .....                       | 296 |
| 17.5   | Memory map and register definition .....                | 296 |
| 17.5.1 | Channel configuration registers (CHCONFIGn) .....       | 297 |
| 17.6   | DMA_MUX inputs .....                                    | 298 |
| 17.6.1 | DMA_MUX peripheral sources .....                        | 298 |
| 17.6.2 | DMA_MUX periodic trigger inputs .....                   | 300 |
| 17.7   | Functional description .....                            | 300 |
| 17.7.1 | eDMA channels with periodic triggering capability ..... | 300 |
| 17.7.2 | eDMA channels with no triggering capability .....       | 302 |
| 17.8   | Initialization/Application information .....            | 303 |
| 17.8.1 | Reset .....   | 303 |
| 17.8.2 | Enabling and configuring sources .....                  | 303 |

## Chapter 18 Interrupt Controller (INTC)

|         |   |     |
|---------|---|-----|
| 18.1    | Introduction .....  | 307 |
| 18.2    | Features .....  | 307 |
| 18.3    | Block diagram .....   | 309 |
| 18.4    | Modes of operation .....  | 309 |
| 18.4.1  | Normal mode .....   | 309 |
| 18.5    | Memory map and register description .....                           | 311 |
| 18.5.1  | Module memory map .....   | 311 |
| 18.5.2  | Register description .....  | 311 |
| 18.6    | Functional description .....  | 319 |
| 18.6.1  | Interrupt request sources .....                                     | 327 |
| 18.6.2  | Priority management .....   | 328 |
| 18.6.3  | Handshaking with processor .....                                    | 330 |
| 18.7    | Initialization/application information .....                        | 332 |
| 18.7.1  | Initialization flow .....   | 332 |
| 18.7.2  | Interrupt exception handler .....                                   | 332 |
| 18.7.3  | ISR, RTOS, and task hierarchy .....                                 | 334 |
| 18.7.4  | Order of execution .....  | 335 |
| 18.7.5  | Priority ceiling protocol .....                                     | 336 |
| 18.7.6  | Selecting priorities according to request rates and deadlines ..... | 336 |
| 18.7.7  | Software configurable interrupt requests .....                      | 337 |
| 18.7.8  | Lowering priority within an ISR .....                               | 338 |
| 18.7.9  | Negating an interrupt request outside of its ISR .....              | 338 |
| 18.7.10 | Examining LIFO contents .....                                       | 339 |

## Chapter 19 Crossbar Switch (XBAR)

|      |                     |     |
|------|---------------------|-----|
| 19.1 | Introduction .....  | 341 |
| 19.2 | Block diagram ..... | 341 |
| 19.3 | Overview .....      | 342 |

|        |                              |     |
|--------|------------------------------|-----|
| 19.4   | Features .....               | 342 |
| 19.5   | Modes of operation .....     | 342 |
| 19.5.1 | Normal mode .....            | 342 |
| 19.5.2 | Debug mode .....             | 342 |
| 19.6   | Functional description ..... | 342 |
| 19.6.1 | Overview .....               | 342 |
| 19.6.2 | General operation .....      | 343 |
| 19.6.3 | Master ports .....           | 343 |
| 19.6.4 | Slave ports .....            | 344 |
| 19.6.5 | Priority assignment .....    | 344 |
| 19.6.6 | Arbitration .....            | 344 |

## Chapter 20 System Integration Unit Lite (SIUL)

|        |  |     |
|--------|--|-----|
| 20.1   | Introduction .....                                 | 347 |
| 20.2   | Overview .....                                     | 347 |
| 20.3   | Features .....                                     | 349 |
| 20.4   | External signal description .....                  | 349 |
| 20.4.1 | Detailed signal descriptions .....                 | 350 |
| 20.5   | Memory map and register description .....          | 351 |
| 20.5.1 | SIUL memory map .....                              | 351 |
| 20.5.2 | Register protection .....                          | 352 |
| 20.5.3 | Register descriptions .....                        | 353 |
| 20.6   | Functional description .....                       | 372 |
| 20.6.1 | Pad control .....                                  | 372 |
| 20.6.2 | General purpose input and output pads (GPIO) ..... | 372 |
| 20.6.3 | External interrupts .....                          | 373 |
| 20.7   | Pin muxing .....                                   | 374 |

## Chapter 21 Memory Protection Unit (MPU)

|        |  |     |
|--------|--|-----|
| 21.1   | Introduction .....                                       | 375 |
| 21.2   | Features .....   | 376 |
| 21.3   | Modes of operation .....                                 | 377 |
| 21.4   | External signal description .....                        | 377 |
| 21.5   | Memory map and register description .....                | 377 |
| 21.5.1 | Memory map .....   | 378 |
| 21.5.2 | Register description .....                               | 379 |
| 21.6   | Functional description .....                             | 390 |
| 21.6.1 | Access evaluation macro .....                            | 390 |
| 21.6.2 | Putting it all together and AHB error terminations ..... | 391 |
| 21.7   | Initialization information .....                         | 392 |
| 21.8   | Application information .....                            | 392 |

## Chapter 22

### Inter-Integrated Circuit Bus Controller Module (I<sup>2</sup>C)

|        |  |     |
|--------|--|-----|
| 22.1   | Introduction .....   | 397 |
| 22.1.1 | Overview .....   | 397 |
| 22.1.2 | Features .....   | 397 |
| 22.1.3 | Block diagram .....  | 398 |
| 22.2   | External signal description .....                                  | 398 |
| 22.2.1 | SCL .....  | 398 |
| 22.2.2 | SDA .....  | 398 |
| 22.3   | Memory map and register description .....                          | 398 |
| 22.3.1 | Module memory map .....  | 398 |
| 22.3.2 | I <sup>2</sup> C Bus Address Register (IBAD) .....                 | 399 |
| 22.3.3 | I <sup>2</sup> C Bus Frequency Divider Register (IBFD) .....       | 400 |
| 22.3.4 | I <sup>2</sup> C Bus Control Register (IBCR) .....                 | 406 |
| 22.3.5 | I <sup>2</sup> C Bus Status Register (IBSR) .....                  | 407 |
| 22.3.6 | I <sup>2</sup> C Bus Data I/O Register (IBDR) .....                | 408 |
| 22.3.7 | I <sup>2</sup> C Bus Interrupt Configuration Register (IBIC) ..... | 409 |
| 22.4   | DMA Interface .....  | 409 |
| 22.5   | Functional description .....                                       | 411 |
| 22.5.1 | I-Bus protocol .....   | 411 |
| 22.5.2 | Interrupts .....   | 414 |
| 22.6   | Initialization/application information .....                       | 415 |
| 22.6.1 | I <sup>2</sup> C programming examples .....                        | 415 |

## Chapter 23

### LIN Controller (LINFlex)

|        |  |     |
|--------|--|-----|
| 23.1   | Introduction .....                         | 421 |
| 23.2   | Main features .....                        | 421 |
| 23.2.1 | LIN mode features .....                    | 421 |
| 23.2.2 | UART mode features .....                   | 421 |
| 23.2.3 | Features common to LIN and UART .....      | 421 |
| 23.3   | General description .....                  | 422 |
| 23.4   | Fractional baud rate generation .....      | 423 |
| 23.5   | Operating modes .....                      | 425 |
| 23.5.1 | Initialization mode .....                  | 426 |
| 23.5.2 | Normal mode .....                          | 426 |
| 23.5.3 | Low power mode (Sleep) .....               | 426 |
| 23.6   | Test modes .....                           | 426 |
| 23.6.1 | Loop Back mode .....                       | 426 |
| 23.6.2 | Self Test mode .....                       | 427 |
| 23.7   | Memory map and registers description ..... | 427 |
| 23.7.1 | Memory map .....                           | 427 |
| 23.8   | Functional description .....               | 453 |
| 23.8.1 | UART mode .....                            | 453 |
| 23.8.2 | LIN mode .....                             | 455 |

|        |                             |     |
|--------|-----------------------------|-----|
| 23.8.3 | 8-bit timeout counter ..... | 463 |
| 23.8.4 | Interrupts .....            | 465 |

## Chapter 24 LIN Controller (LINFlexD)

|          |   |     |
|----------|---|-----|
| 24.1     | Introduction .....                                  | 467 |
| 24.2     | Main features .....                                 | 467 |
| 24.2.1   | LIN mode features .....                             | 468 |
| 24.2.2   | UART mode features .....                            | 468 |
| 24.3     | The LIN protocol .....                              | 469 |
| 24.3.1   | Dominant and recessive logic levels .....           | 469 |
| 24.3.2   | LIN frames .....                                    | 469 |
| 24.3.3   | LIN header .....                                    | 470 |
| 24.3.4   | Response .....                                      | 471 |
| 24.4     | LINFlexD and software intervention .....            | 472 |
| 24.5     | Summary of operating modes .....                    | 472 |
| 24.6     | Controller-level operating modes .....              | 473 |
| 24.6.1   | Initialization mode .....                           | 473 |
| 24.6.2   | Normal mode .....                                   | 474 |
| 24.6.3   | Sleep (low-power) mode .....                        | 474 |
| 24.7     | LIN modes .....                                     | 474 |
| 24.7.1   | Master mode .....                                   | 474 |
| 24.7.2   | Slave mode .....                                    | 476 |
| 24.7.3   | Slave mode with identifier filtering .....          | 478 |
| 24.7.4   | Slave mode with automatic resynchronization .....   | 481 |
| 24.8     | Test modes .....                                    | 482 |
| 24.8.1   | Loop Back mode .....                                | 482 |
| 24.8.2   | Self Test mode .....                                | 483 |
| 24.9     | UART mode .....                                     | 483 |
| 24.9.1   | Data frame structure .....                          | 483 |
| 24.9.2   | Buffer .....  | 485 |
| 24.9.3   | UART transmitter .....                              | 485 |
| 24.9.4   | UART receiver .....                                 | 486 |
| 24.10    | Memory map and register description .....           | 488 |
| 24.10.1  | LIN control register 1 (LINCR1) .....               | 488 |
| 24.10.2  | LIN interrupt enable register (LINIER) .....        | 491 |
| 24.10.3  | LIN status register (LINSR) .....                   | 493 |
| 24.10.4  | LIN error status register (LINESR) .....            | 496 |
| 24.10.5  | UART mode control register (UARTCR) .....           | 497 |
| 24.10.6  | UART mode status register (UARTSR) .....            | 500 |
| 24.10.7  | LIN timeout control status register (LINTCSR) ..... | 502 |
| 24.10.8  | LIN output compare register (LINOOCR) .....         | 503 |
| 24.10.9  | LIN timeout control register (LINTOCR) .....        | 504 |
| 24.10.10 | LIN fractional baud rate register (LINFBR) .....    | 505 |
| 24.10.11 | LIN integer baud rate register (LINIBRR) .....      | 505 |

|          |  |     |
|----------|--|-----|
| 24.10.12 | LIN checksum field register (LINCFR)               | 506 |
| 24.10.13 | LIN control register 2 (LINCR2)                    | 507 |
| 24.10.14 | Buffer identifier register (BIDR)                  | 508 |
| 24.10.15 | Buffer data register least significant (BDRL)      | 509 |
| 24.10.16 | Buffer data register most significant (BDRM)       | 510 |
| 24.10.17 | Identifier filter enable register (IFER)           | 511 |
| 24.10.18 | Identifier filter match index (IFMI)               | 512 |
| 24.10.19 | Identifier filter mode register (IFMR)             | 513 |
| 24.10.20 | Identifier filter control registers (IFCR0–IFCR15) | 513 |
| 24.10.21 | Global control register (GCR)                      | 514 |
| 24.10.22 | UART preset timeout register (UARTPTO)             | 516 |
| 24.10.23 | UART current timeout register (UARTCTO)            | 516 |
| 24.10.24 | DMA Tx enable register (DMATXE)                    | 517 |
| 24.10.25 | DMA Rx enable register (DMARXE)                    | 518 |
| 24.11    | DMA interface                                      | 518 |
| 24.11.1  | Master node, TX mode                               | 519 |
| 24.11.2  | Master node, RX mode                               | 522 |
| 24.11.3  | Slave node, TX mode                                | 524 |
| 24.11.4  | Slave node, RX mode                                | 527 |
| 24.11.5  | UART node, TX mode                                 | 530 |
| 24.11.6  | UART node, RX mode                                 | 532 |
| 24.11.7  | Use cases and limitations                          | 535 |
| 24.12    | Functional description                             | 536 |
| 24.12.1  | 8-bit timeout counter                              | 536 |
| 24.12.2  | Interrupts   | 537 |
| 24.12.3  | Fractional baud rate generation                    | 539 |
| 24.13    | Programming considerations                         | 540 |
| 24.13.1  | Master node  | 540 |
| 24.13.2  | Slave node   | 541 |
| 24.13.3  | Extended frames                                    | 545 |
| 24.13.4  | Timeout  | 545 |
| 24.13.5  | UART mode  | 546 |

## Chapter 25 FlexCAN

|        |                                     |     |
|--------|-------------------------------------|-----|
| 25.1   | Information specific to this device | 547 |
| 25.1.1 | Device-specific features            | 547 |
| 25.2   | Introduction                        | 547 |
| 25.2.1 | Overview                            | 548 |
| 25.2.2 | FlexCAN module features             | 549 |
| 25.2.3 | Modes of operation                  | 549 |
| 25.3   | External signal description         | 550 |
| 25.3.1 | Overview                            | 550 |
| 25.3.2 | Signal descriptions                 | 551 |
| 25.4   | Memory map/register definition      | 551 |

|         |  |     |
|---------|--|-----|
| 25.4.1  | FlexCAN memory mapping .....                         | 551 |
| 25.4.2  | Message Buffer Structure .....                       | 553 |
| 25.4.3  | Rx FIFO structure .....                              | 556 |
| 25.4.4  | Register descriptions .....                          | 557 |
| 25.5    | Functional description .....                         | 576 |
| 25.5.1  | Overview .....                                       | 576 |
| 25.5.2  | Local priority transmission .....                    | 577 |
| 25.5.3  | Transmit process .....                               | 577 |
| 25.5.4  | Arbitration process .....                            | 578 |
| 25.5.5  | Receive process .....                                | 579 |
| 25.5.6  | Matching process .....                               | 580 |
| 25.5.7  | Data coherence .....                                 | 581 |
| 25.5.8  | Rx FIFO .....  | 584 |
| 25.5.9  | CAN protocol related features .....                  | 584 |
| 25.5.10 | Modes of operation details .....                     | 588 |
| 25.5.11 | Interrupts .....                                     | 589 |
| 25.5.12 | Bus interface .....                                  | 590 |
| 25.6    | Initialization/application information .....         | 591 |
| 25.6.1  | FlexCAN initialization sequence .....                | 591 |
| 25.6.2  | FlexCAN addressing and RAM size configurations ..... | 592 |

## Chapter 26

### Deserial Serial Peripheral Interface (DSPI)

|        |  |     |
|--------|--|-----|
| 26.1   | Introduction .....   | 593 |
| 26.2   | Features .....   | 594 |
| 26.3   | Modes of operation .....   | 595 |
| 26.3.1 | Master mode .....  | 595 |
| 26.3.2 | Slave mode .....   | 595 |
| 26.3.3 | Module Disable mode .....  | 595 |
| 26.3.4 | Debug mode .....   | 596 |
| 26.4   | External signal description .....  | 596 |
| 26.4.1 | Signal overview .....  | 596 |
| 26.4.2 | Signal names and descriptions .....  | 596 |
| 26.5   | Memory map and register description .....                                  | 597 |
| 26.5.1 | Memory map .....   | 597 |
| 26.5.2 | DSPI Module Configuration Register (DSPIx_MCR) .....                       | 598 |
| 26.5.3 | DSPI Transfer Count Register (DSPIx_TCR) .....                             | 601 |
| 26.5.4 | DSPI Clock and Transfer Attributes Registers 0–5 (DSPIx_CTARn) .....       | 602 |
| 26.5.5 | DSPI Status Register (DSPIx_SR) .....                                      | 610 |
| 26.5.6 | DSPI DMA / Interrupt Request Select and Enable Register (DSPIx_RSER) ..... | 612 |
| 26.5.7 | DSPI PUSH TX FIFO Register (DSPIx_PUSHR) .....                             | 614 |
| 26.5.8 | DSPI POP RX FIFO Register (DSPIx_POPR) .....                               | 616 |
| 26.5.9 | DSPI Transmit FIFO Registers 0–3 (DSPIx_TXFRn) .....                       | 617 |
| 26.6   | Functional description .....   | 618 |
| 26.6.1 | Modes of operation .....   | 619 |

|        |   |     |
|--------|---|-----|
| 26.6.2 | Start and stop of DSPI transfers .....                | 620 |
| 26.6.3 | Serial peripheral interface (SPI) configuration ..... | 621 |
| 26.6.4 | DSPI baud rate and clock delay generation .....       | 624 |
| 26.6.5 | Transfer formats .....                                | 627 |
| 26.6.6 | Continuous serial communications clock .....          | 634 |
| 26.6.7 | Interrupt/DMA requests .....                          | 635 |
| 26.6.8 | Power saving features .....                           | 637 |
| 26.7   | Initialization and application information .....      | 638 |
| 26.7.1 | How to change queues .....                            | 638 |
| 26.7.2 | Baud rate settings .....                              | 638 |
| 26.7.3 | Delay settings .....                                  | 640 |
| 26.7.4 | Calculation of FIFO pointer addresses .....           | 640 |

## Chapter 27 Timers

|        |  |     |
|--------|--|-----|
| 27.1   | Introduction .....                               | 645 |
| 27.2   | Technical overview .....                         | 645 |
| 27.2.1 | Overview of the STM .....                        | 647 |
| 27.2.2 | Overview of the eMIOS .....                      | 647 |
| 27.2.3 | Overview of the PIT .....                        | 649 |
| 27.3   | System Timer Module (STM) .....                  | 649 |
| 27.3.1 | Introduction .....                               | 649 |
| 27.3.2 | External signal description .....                | 650 |
| 27.3.3 | Memory map and register definition .....         | 650 |
| 27.3.4 | Functional description .....                     | 654 |
| 27.4   | Enhanced Modular IO Subsystem (eMIOS) .....      | 655 |
| 27.4.1 | Introduction .....                               | 655 |
| 27.4.2 | External signal description .....                | 658 |
| 27.4.3 | Memory map and register description .....        | 658 |
| 27.4.4 | Functional description .....                     | 670 |
| 27.4.5 | Initialization/Application information .....     | 700 |
| 27.5   | Periodic Interrupt Timer (PIT) .....             | 704 |
| 27.5.1 | Introduction .....                               | 704 |
| 27.5.2 | Features .....                                   | 704 |
| 27.5.3 | Signal description .....                         | 705 |
| 27.5.4 | Memory map and register description .....        | 705 |
| 27.5.5 | Functional description .....                     | 709 |
| 27.5.6 | Initialization and application information ..... | 710 |

## Chapter 28 Analog-to-Digital Converter (ADC)

|        |                                      |     |
|--------|--------------------------------------|-----|
| 28.1   | Overview .....                       | 715 |
| 28.1.1 | Device-specific features .....       | 715 |
| 28.1.2 | Device-specific implementation ..... | 716 |
| 28.2   | Introduction .....                   | 717 |

|         |   |     |
|---------|---|-----|
| 28.3    | Functional description .....                        | 717 |
| 28.3.1  | Analog channel conversion .....                     | 717 |
| 28.3.2  | Analog clock generator and conversion timings ..... | 720 |
| 28.3.3  | ADC sampling and conversion timing .....            | 720 |
| 28.3.4  | ADC CTU (Cross Triggering Unit) .....               | 725 |
| 28.3.5  | Presampling .....                                   | 726 |
| 28.3.6  | Programmable analog watchdog .....                  | 727 |
| 28.3.7  | DMA functionality .....                             | 728 |
| 28.3.8  | Interrupts .....                                    | 728 |
| 28.3.9  | External decode signals delay .....                 | 729 |
| 28.3.10 | Power-down mode .....                               | 729 |
| 28.3.11 | Auto-clock-off mode .....                           | 729 |
| 28.4    | Register descriptions .....                         | 730 |
| 28.4.1  | Introduction .....                                  | 730 |
| 28.4.2  | Control logic registers .....                       | 737 |
| 28.4.3  | Interrupt registers .....                           | 740 |
| 28.4.4  | DMA registers .....                                 | 748 |
| 28.4.5  | Threshold registers .....                           | 752 |
| 28.4.6  | Presampling registers .....                         | 753 |
| 28.4.7  | Conversion timing registers CTR[0..2] .....         | 756 |
| 28.4.8  | Mask registers .....                                | 757 |
| 28.4.9  | Delay registers .....                               | 761 |
| 28.4.10 | Data registers .....                                | 763 |
| 28.4.11 | Watchdog register .....                             | 765 |

## Chapter 29 Cross Triggering Unit (CTU)

|        |  |     |
|--------|--|-----|
| 29.1   | Introduction .....   | 779 |
| 29.2   | Main features .....  | 779 |
| 29.3   | Block diagram .....  | 779 |
| 29.4   | Memory map and register descriptions .....                     | 779 |
| 29.4.1 | Event Configuration Registers (CTU_EVTCFGRx) (x = 0..63) ..... | 780 |
| 29.5   | Functional description .....                                   | 781 |
| 29.5.1 | Channel value .....  | 783 |

## Chapter 30 Flash Memory

|        |   |     |
|--------|---|-----|
| 30.1   | Introduction .....                      | 789 |
| 30.2   | Main features .....                     | 790 |
| 30.3   | Block diagram .....                     | 790 |
| 30.4   | Functional description .....            | 791 |
| 30.4.1 | Module structure .....                  | 791 |
| 30.4.2 | Flash memory module sectorization ..... | 792 |
| 30.4.3 | TestFlash block .....                   | 793 |
| 30.4.4 | Shadow sector .....                     | 795 |



|         |  |     |
|---------|--|-----|
| 30.4.5  | User mode operation .....                            | 795 |
| 30.4.6  | Reset .....  | 796 |
| 30.4.7  | Power-down mode .....                                | 797 |
| 30.4.8  | Low power mode .....                                 | 797 |
| 30.5    | Register description .....                           | 798 |
| 30.5.1  | CFlash register description .....                    | 799 |
| 30.5.2  | DFlash register description .....                    | 834 |
| 30.6    | Programming considerations .....                     | 857 |
| 30.6.1  | Modify operation .....                               | 857 |
| 30.6.2  | Double word program .....                            | 858 |
| 30.6.3  | Sector erase .....                                   | 860 |
| 30.7    | Platform flash memory controller .....               | 868 |
| 30.7.1  | Introduction .....                                   | 868 |
| 30.7.2  | Memory map and register description .....            | 871 |
| 30.8    | Functional description .....                         | 880 |
| 30.8.1  | Access protections .....                             | 880 |
| 30.8.2  | Read cycles – Buffer miss .....                      | 880 |
| 30.8.3  | Read cycles – Buffer hit .....                       | 881 |
| 30.8.4  | Write cycles .....                                   | 881 |
| 30.8.5  | Error termination .....                              | 881 |
| 30.8.6  | Access pipelining .....                              | 881 |
| 30.8.7  | Flash error response operation .....                 | 882 |
| 30.8.8  | Bank0 page read buffers and prefetch operation ..... | 882 |
| 30.8.9  | Bank1 Temporary Holding Register .....               | 884 |
| 30.8.10 | Read-while-write functionality .....                 | 885 |
| 30.8.11 | Wait-state emulation .....                           | 886 |

## Chapter 31 Static RAM (SRAM)

|        |  |     |
|--------|--|-----|
| 31.1   | Introduction .....                               | 889 |
| 31.2   | Low power configuration .....                    | 889 |
| 31.3   | Register memory map .....                        | 889 |
| 31.4   | SRAM ECC mechanism .....                         | 890 |
| 31.4.1 | Access timing .....                              | 890 |
| 31.4.2 | Reset effects on SRAM accesses .....             | 891 |
| 31.5   | Functional description .....                     | 891 |
| 31.6   | Initialization and application information ..... | 891 |

## Chapter 32 Register Protection

|      |   |     |
|------|---|-----|
| 32.1 | Introduction .....                        | 895 |
| 32.2 | Features .....                            | 895 |
| 32.3 | Modes of operation .....                  | 896 |
| 32.4 | External signal description .....         | 896 |
| 32.5 | Memory map and register description ..... | 896 |

|        |                              |     |
|--------|------------------------------|-----|
| 32.5.1 | Memory map .....             | 897 |
| 32.5.2 | Register description .....   | 898 |
| 32.6   | Functional description ..... | 900 |
| 32.6.1 | General .....                | 900 |
| 32.6.2 | Change lock settings .....   | 900 |
| 32.6.3 | Access errors .....          | 904 |
| 32.7   | Reset .....                  | 904 |
| 32.8   | Protected registers .....    | 904 |

## Chapter 33 Software Watchdog Timer (SWT)

|        |   |     |
|--------|---|-----|
| 33.1   | Overview .....                            | 913 |
| 33.2   | Features .....                            | 913 |
| 33.3   | Modes of operation .....                  | 913 |
| 33.4   | External signal description .....         | 914 |
| 33.5   | Memory map and register description ..... | 914 |
| 33.5.1 | Memory map .....                          | 914 |
| 33.5.2 | Register description .....                | 915 |
| 33.6   | Functional description .....              | 919 |

## Chapter 34 Error Correction Status Module (ECSM)

|        |   |     |
|--------|---|-----|
| 34.1   | Introduction .....                        | 923 |
| 34.2   | Overview .....                            | 923 |
| 34.3   | Features .....                            | 923 |
| 34.4   | Memory map and register description ..... | 923 |
| 34.4.1 | Memory map .....                          | 923 |
| 34.4.2 | Register description .....                | 924 |
| 34.4.3 | Register protection .....                 | 942 |

## Chapter 35 IEEE 1149.1 Test Access Port Controller (JTAGC)

|        |   |     |
|--------|---|-----|
| 35.1   | Introduction .....                        | 945 |
| 35.2   | Block diagram .....                       | 945 |
| 35.3   | Overview .....                            | 945 |
| 35.4   | Features .....                            | 946 |
| 35.5   | Modes of operation .....                  | 946 |
| 35.5.1 | Reset .....                               | 946 |
| 35.5.2 | IEEE 1149.1-2001 defined test modes ..... | 946 |
| 35.6   | External signal description .....         | 947 |
| 35.7   | Memory map and register description ..... | 947 |
| 35.7.1 | Instruction register .....                | 947 |
| 35.7.2 | Bypass register .....                     | 948 |
| 35.7.3 | Device identification register .....      | 948 |
| 35.7.4 | Boundary scan register .....              | 949 |

|        |   |     |
|--------|---|-----|
| 35.8   | Functional description .....                        | 949 |
| 35.8.1 | JTAGC reset configuration .....                     | 949 |
| 35.8.2 | IEEE 1149.1-2001 (JTAG) Test Access Port .....      | 949 |
| 35.8.3 | TAP controller state machine .....                  | 949 |
| 35.8.4 | JTAGC instructions .....                            | 951 |
| 35.8.5 | Boundary scan .....                                 | 953 |
| 35.9   | e200z0 OnCE controller .....                        | 953 |
| 35.9.1 | e200z0 OnCE controller block diagram .....          | 953 |
| 35.9.2 | e200z0 OnCE controller functional description ..... | 954 |
| 35.9.3 | e200z0 OnCE controller register description .....   | 954 |
| 35.10  | Initialization/application information .....        | 956 |

## Appendix A Revision History

|     |   |     |
|-----|---|-----|
| A.1 | Changes between revisions 1 and 2 ..... | 957 |
|-----|---|-----|



# Chapter 1

## Preface

### 1.1 Overview

The primary objective of this document is to define the functionality of the MPC5606BK microcontroller for use by software and hardware developers. The MPC5606BK is built on Power Architecture<sup>®</sup> technology and integrates technologies that are important for today's automotive vehicle body applications.

The information in this book is subject to change without notice, as described in the disclaimers on the title page. As with any technical documentation, it is the reader's responsibility to be sure he or she is using the most recent version of the documentation.

To locate any published errata or updates for this document, visit the Freescale Web site at [freescale.com](http://freescale.com).

### 1.2 Audience

This manual is intended for system software and hardware developers and applications programmers who want to develop products with the MPC5606BK device. It is assumed that the reader understands operating systems, microprocessor system design, basic principles of software and hardware, and basic details of the Power Architecture.

### 1.3 Guide to this reference manual

Table 1-1. Guide to this reference manual

| Chapter |   | Description  | Functional group      |
|---------|---|--|-----------------------|
| #       | Title   |  |                       |
| 2       | <a href="#">Introduction</a>  | General overview, family description, feature list, and information on how to use the reference manual in conjunction with other available documents.  | Introductory material |
| 3       | <a href="#">Memory Map</a>  | Memory map of all peripherals and memory.  | Memory map            |
| 4       | <a href="#">Signal description</a>  | Pinout diagrams and descriptions of all pads.  | Signals               |
| 5       | <a href="#">Microcontroller Boot</a>  |  | Boot                  |
|         | <ul style="list-style-type: none"> <li><a href="#">Boot mechanism</a></li> </ul>                                | <ul style="list-style-type: none"> <li>Describes what configuration is required by the user and what processes are involved when the microcontroller boots from flash memory or serial boot modes.</li> <li>Describes censorship.</li> </ul> |                       |
|         | <ul style="list-style-type: none"> <li><a href="#">Boot Assist Module (BAM)</a></li> </ul>                      | Features of BAM code and when it's used.   |                       |
|         | <ul style="list-style-type: none"> <li><a href="#">System Status and Configuration Module (SSCM)</a></li> </ul> | Reports information about current state and configuration of the microcontroller.  |                       |

Table 1-1. Guide to this reference manual (continued)

| Chapter |   | Description  | Functional group   |
|---------|---|--|--|
| #       | Title   |  |  |
| 6       | <a href="#">Clock Description</a>   | <ul style="list-style-type: none"> <li>Covers configuration of all of the clock sources in the system.</li> <li>Describes the Clock Monitor Unit (CMU).</li> </ul> | Clocks and power<br><br>(includes operating mode configuration and how to wake up from low power mode) |
| 7       | <a href="#">Clock Generation Module (MC_CGM)</a>                          | Determines how the clock sources are used (including clock dividers) to generate the reference clocks for all of the modules and peripherals.                      |  |
| 8       | <a href="#">Mode Entry Module (MC_ME)</a>                                 | Determines the clock source, memory, power, and peripherals that are available in each operating mode.   |  |
| 9       | <a href="#">Reset Generation Module (MC_RGM)</a>                          | Manages the process of entering and exiting reset, allows reset sources to be configured (including LVDs), and provides status reporting.                          |  |
| 10      | <a href="#">Power Control Unit (MC_PCU)</a>                               | Controls the power to different power domains within the microcontroller (allowing SRAM to be selectively powered in STANDBY mode).                                |  |
| 11      | <a href="#">Voltage Regulators and Power Supplies</a>                     | Information on voltage regulator implementation. Includes enable bit for 5 V LVD (see also MC_RGM).  |  |
| 12      | <a href="#">Wakeup Unit (WKPU)</a>  | Always-active analog block. Details configuration of two internal (API/RTC) and 27 external (pin) low power mode wakeup sources.                                   |  |
| 13      | <a href="#">Real Time Clock / Autonomous Periodic Interrupt (RTC/API)</a> | Details configuration and operation of timers that are predominately used for system wakeup.   |  |
| 14      | <a href="#">CAN Sampler</a>   | Details on how to configure the CAN sampler, which is used to capture the identifier frame of a CAN message when the microcontroller is in low power mode.         |  |

Table 1-1. Guide to this reference manual (continued)

| Chapter |  | Description  | Functional group      |
|---------|--|--|-----------------------|
| #       | Title  |  |                       |
| 15      | e200z0h Core   | Overview on cores. For more details consult the core reference manuals available on <a href="http://www.freescale.com">www.freescale.com</a> .   | Core platform modules |
| 16      | Enhanced Direct Memory Access (eDMA)                 | Operation and configuration information on the 32-channel direct memory access that can be used to transfer data between any memory mapped locations. Certain peripherals have eDMA triggers that can be used to feed configuration data to, or read results from the peripherals. |                       |
| 17      | eDMA Channel Multiplexer (DMA_MUX)                   | Operation and configuration information for the eDMA multiplexer, which takes the possible eDMA sources (triggers from the DSPI, eMIOS, I <sup>2</sup> C, ADC, and LINFlexD) and multiplexes them onto the eDMA channels.  |                       |
| 18      | Interrupt Controller (INTC)                          | Provides the configuration and control of all of the external interrupts (non-core) that are then routed to the IVOR4 core interrupt vector.   |                       |
| 19      | Crossbar Switch (XBAR)                               | Describes the connections of the XBAR masters and slaves on this microcontroller.  |                       |
| 21      | Memory Protection Unit (MPU)                         | The MPU sits on the slave side of the XBAR and allows highly configurable control over all master accesses to the memory.  |                       |
| 20      | System Integration Unit Lite (SIUL)                  | How to configure the pins or ports for input or output functions including external interrupts.  | Ports                 |
| 22      | Inter-Integrated Circuit Bus Controller Module (I2C) | These chapters describe the configuration and operation of the various communication modules. Some of these modules support DMA requests to fill / empty buffer queues to minimize CPU overhead.   | Communication modules |
| 23      | LIN Controller (LINFlex)                             |  |                       |
| 24      | LIN Controller (LINFlexD)                            |  |                       |
| 25      | FlexCAN  |  |                       |
| 26      | Deserial Serial Peripheral Interface (DSPI)          |  |                       |

Table 1-1. Guide to this reference manual (continued)

| Chapter |   | Description  | Functional group |
|---------|---|--|------------------|
| #       | Title   |  |                  |
| 27      | <a href="#">Timers</a>  |  | Timer modules    |
|         | <ul style="list-style-type: none"> <li>• <a href="#">Technical overview</a></li> </ul>                    | Gives an overview of the available system timer modules showing links to other modules as well as tables detailing the external pins associated with eMIOS timer channels.   |                  |
|         | <ul style="list-style-type: none"> <li>• <a href="#">System Timer Module (STM)</a></li> </ul>             | A simple 32-bit free running counter with 4 compare channels with interrupt on match. It can be read at any time; this is very useful for measuring execution times.   |                  |
|         | <ul style="list-style-type: none"> <li>• <a href="#">Enhanced Modular IO Subsystem (eMIOS)</a></li> </ul> | Highly configurable timer module(s) supporting PWM, output compare, and input capture features. Includes interrupt and eDMA support.   |                  |
|         | <ul style="list-style-type: none"> <li>• <a href="#">Periodic Interrupt Timer (PIT)</a></li> </ul>        | Set of 32-bit countdown timers that provide periodic events (which can trigger an interrupt) with automatic reload.  |                  |
| 28      | <a href="#">Analog-to-Digital Converter (ADC)</a>   | Details the configuration and operation of the ADC modules as well as detailing the channels that are shared between the 10-bit and 12-bit ADC. The ADC is tightly linked to the INTC, eDMA, PIT, and CTU. When used in conjunction with these other modules, the CPU overhead for an ADC conversion is significantly reduced. | ADC system       |
| 29      | <a href="#">Cross Triggering Unit (CTU)</a>   | The CTU allows an ADC conversion to be automatically triggered based on an eMIOS event (like a PWM output going high) or a PIT_RTI event with no CPU intervention.   |                  |
| 30      | <a href="#">Flash Memory</a>  | Details the code and data flash memory structure (with ECC), block sizes and the flash memory port configuration, including wait states, line buffer configuration, and pre-fetch control.   | Memory           |
| 31      | <a href="#">Static RAM (SRAM)</a>   | Details the structure of the SRAM (with ECC). There are no user configurable registers associated with the SRAM.   |                  |



Table 1-1. Guide to this reference manual (continued)

| Chapter |   | Description   | Functional group |
|---------|---|---|------------------|
| #       | Title   |   |                  |
| 32      | <a href="#">Register Protection</a>                             | Certain registers in each peripheral can be protected from further writes using the register protection mechanism detailed in this section. Registers can either be configured to be unlocked via a soft lock bit or locked until the next reset. | Integrity        |
| 33      | <a href="#">Software Watchdog Timer (SWT)</a>                   | The SWT offers a selection of configurable modes that can be used to monitor the operation of the microcontroller and /or reset the device or trigger an interrupt if the SWT is not correctly serviced. The SWT is enabled out of reset.         |                  |
| 34      | <a href="#">Error Correction Status Module (ECSM)</a>           | Provides information about the last reset, general device information, system fault information and detailed ECC error information.   |                  |
| 35      | <a href="#">IEEE 1149.1 Test Access Port Controller (JTAGC)</a> | Used for boundary scan as well as device debug.   | Debug            |

## 1.4 Register description conventions

The register information for MPC5606BK is presented in:

- Memory maps containing:
  - An offset from the module's base address
  - The name and acronym/abbreviation of each register
  - The page number on which each register is described
- Register figures
- Field-description tables
- Associated text

The register figures show the field structure using the conventions in [Figure 1-1](#).

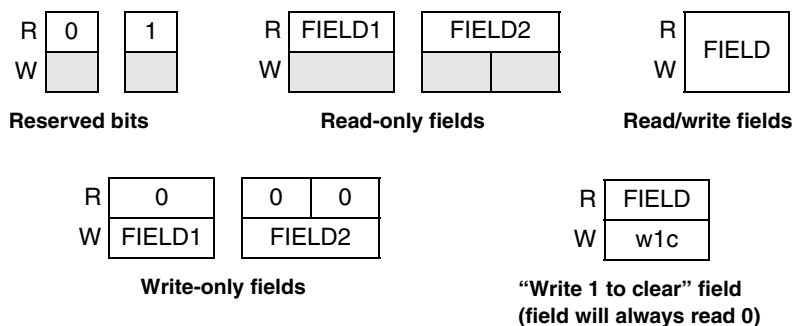


Figure 1-1. Register figure conventions

The numbering of register bits and fields on MPC5606BK is as follows:

- Register bit numbers, shown at the top of each figure, use the standard Power Architecture bit ordering (0, 1, 2, ...) where bit 0 is the most significant bit (MSB).
- Multi-bit fields within a register use conventional bit ordering (... , 2, 1, 0) where bit 0 is the least significant bit (LSB).

## 1.5 References

In addition to this reference manual, the following documents provide additional information on the operation of the MPC5606BK:

- IEEE 1149.1-2001 standard—IEEE Standard Test Access Port and Boundary-Scan Architecture
- Power Architecture Book E V1.0  
([http://www.freescale.com/files/32bit/doc/user\\_guide/BOOK\\_EUM.pdf](http://www.freescale.com/files/32bit/doc/user_guide/BOOK_EUM.pdf))

## 1.6 How to use the MPC5606BK documents

This section:

- Describes how the MPC5606BK documents provide information on the microcontroller
- Makes recommendations on how to use the documents in a system design

### 1.6.1 The MPC5606BK document set

The MPC5606BK document set comprises:

- This reference manual (provides information on the features of the logical blocks on the device and how they are integrated with each other)
- The device data sheet (specifies the electrical characteristics of the device)
- The device product brief

The following reference documents (available online at [www.freescale.com](http://www.freescale.com)) are also available to support the CPU on this device:

- *Programmer's Reference Manual for Freescale Embedded Processors*
- *e200z0 Power Architecture Core Reference Manual*
- *Variable-Length Encoding (VLE) Programming Environments Manual*

The aforementioned documents describe all of the functional and electrical characteristics of the MPC5606BK microcontroller.

Depending on your task, you may need to refer to multiple documents to make design decisions. However, in general the use of the documents can be divided up as follows:

- Use the reference manual (this document) during software development and when allocating functions during system design.
- Use the data sheet when designing hardware and optimizing power consumption.
- Use the CPU reference documents when:
  - Configuring CPU memory and branch optimizations

- Doing detailed software development in assembly language
- Debugging complex software interactions

## 1.6.2 Reference manual content

The content in this document focuses on the functionality of the microcontroller rather than its performance. Most chapters describe the functionality of a particular on-chip module, such as a CAN controller or timer. The remaining chapters describe how these modules are integrated into the memory map, how they are powered and clocked, and the pin-out of the device.

In general, when an individual module is enabled for use all of the detail required to configure and operate it is contained in the dedicated chapter. In some cases there are multiple implementations of this module, however, there is only one chapter for each type of module in use. For this reason, the address of registers in each module is normally provided as an offset from a base address that can be found in [Chapter 3, Memory Map](#). The benefit of this approach is that software developed for a particular module can be easily reused on this device and on other related devices that use the same modules.

The steps to enable a module for use varies but typically these require configuration of the integration features of the microcontroller. The module will normally have to be powered and enabled at system level, then a clock may have to be explicitly chosen, and finally (if required) the input and output connections to the external system must be configured.

The primary integration chapters of the reference manual contain most of the information required to enable the modules. There are special cases where a chapter may describe module functionality and some integration features for convenience — for example, the microcontroller input/output (SIUL) module. Integration and functional content is provided in the manual as shown in [Table 1-2](#).

**Table 1-2. Reference manual integration and functional content**

| Chapter            | Integration content  | Functional content                                  |
|--------------------|--|---|
| Introduction       | <ul style="list-style-type: none"> <li>• The main features on chip</li> <li>• A summary of the functions provided by each module</li> </ul>  | —   |
| Memory Map         | How the memory map is allocated, including: <ul style="list-style-type: none"> <li>• Internal RAM</li> <li>• Flash memory</li> <li>• External memory-mapped resources and the location of the registers used by the peripherals<sup>1</sup></li> </ul> | —   |
| Signal Description | How the signals from each of the modules are combined and brought to a particular pin on a package   | —   |
| Boot Assist Module | CPU boot sequence from reset<br>Implementation of the boot options if internal flash memory is not used  |   |
| Clock Description  | Clocking architecture of the device (which clock is available for the system and each peripheral)  | Description of operation of different clock sources |

**Table 1-2. Reference manual integration and functional content (continued)**

| Chapter                               | Integration content  | Functional content                                      |
|---------------------------------------|--|---|
| eDMA Channel Multiplexer              | Source values for module eDMA channels   | How to connect a module eDMA channel to the eDMA module |
| Interrupt Controller                  | Interrupt vector table   | Operation of the module                                 |
| Mode Entry Module                     | Module numbering for control and status  | Operation of operating modes                            |
| System Integration Unit Lite          | How input signals are mapped to individual modules including external interrupt pins | Operation of GPIO                                       |
| Voltage regulators and power supplies | Power distribution to the MCU  | —   |
| Wakeup Unit                           | Allocation of inputs to the Wakeup Unit  | Operation of the wakeup feature                         |

<sup>1</sup> To find the address of a register in a particular module take the start address of the module given in the memory map and add the offset for the register given in the module chapter.

## 1.7 Using the MPC5606BK

There are many different approaches to designing a system using the MPC5606BK so the guidance in this section is provided as an example of how the documents can be applied in this task.

Familiarity with the MPC5606BK modules can help ensure that its features are being optimally used in a system design. Therefore, the current chapter is a good starting point. Further information on the detailed features of a module are provided within the module chapters. These, combined with the current chapter, should provide a good introduction to the functions available on the MCU.

### 1.7.1 Hardware design

The MPC5606BK requires that certain pins are connected to particular power supplies, system functions, and other voltage levels for operation.

The MPC5606BK internal logic operates from 1.2 V (nominal) supplies that are normally supplied by the on-chip voltage regulator from a 5 V or 3.3 V supply. The 3.3–5 V ( $\pm 10\%$ ) supply is also used to supply the input/output pins on the MCU. [Chapter 4, Signal description](#), describes the power supply pin names, numbers, and their purpose. For more detail on the voltage supply of each pin, see [Chapter 11, Voltage Regulators and Power Supplies](#). For specifications of the voltage ranges and limits and decoupling of the power supplies see the MPC5606BK data sheet.

Certain pins have dedicated functions that affect the behavior of the MCU after reset. These include pins to force test or alternate boot conditions and debug features. These are described in [Chapter 4, Signal description](#), and a hardware designer should take care that these pins are connected to allow correct operation.

Beyond power supply and pins that have special functions there are also pins that have special system purposes such as oscillator and reset pins. These are also described in [Chapter 4, Signal description](#). The reset pin is bidirectional, and its function is closely tied to the reset generation module [[Chapter 9, Reset Generation Module \(MC\\_RGM\)](#)]. The crystal oscillator pins are dedicated to this function but the

oscillator is not started automatically after reset. The oscillator module is described in [Section 6.3, Fast external crystal oscillator \(FXOSC\) digital interface](#), along with the internal clock architecture and the other oscillator sources on chip.

## 1.7.2 Input/output pins

The majority of the pins on the MCU are input/output pins, which may either operate as general purpose pins or be connected to a particular on-chip module. The arrangement allows a function to be available on several pins. The system designer should allocate the function for the pin before connecting to external hardware. The software should then choose the correct function to match the hardware. The pad characteristics can vary depending on the functions on the pad. [Chapter 4, Signal description](#), describes each pad type (for example, S, M, or J). Two pads may be able to carry the same function but have different pad types. The electrical specification of the pads is described in the data sheet dependent on the function enabled and the pad type.

There are three modules that configure the various functions available:

- System Integration Unit Lite (SIUL)
- Wakeup Unit (WKPU)
- 32 KHz oscillator (SXOSC)

The SIUL configures the digital pin functions. Each pin has a register (PCR) in the module that allows selection of the output functions that is connected to the pin. The available settings for the PCR are described in [Section 4.2, Pin muxing](#). Inputs are selected using the PSMI registers; these are described in [Chapter 20, System Integration Unit Lite \(SIUL\)](#). (PSMI registers connect a module to one of several pins, whereas the PCR registers connect a pin to one of several modules).

The WKPU provides the ability to cause interrupts and wake the MCU from low power modes, and operates independently from the SIUL.

In addition to digital I/O functions, the SXOSC is a “special function” that provides a slow external crystal. The SXOSC is enabled independently from the digital I/O, which means that the digital function on the pin must be disabled when the SXOSC is active.

The ADC functions are enabled using the PCRs.

## 1.7.3 Software design

Certain modules provide system integration functions, and other modules (such as timers) provide specific functions.

From reset, the modules involved in configuring the system for application software are:

- Boot Assist Module (BAM) — determines the selected boot source
- Reset Generation Module (MC\_RGM) — determines the behavior of the MCU when various reset sources are triggered and reports the source of the reset
- Mode Entry Module (MC\_ME) — controls the operating mode the MCU is in and configures the peripherals, clocks, and power supplies for each of the modes
- Power Control Unit (MC\_PCU) — determines which power domains are active

- Clock Generation Module (MC\_CGM) — chooses the clock source for the system and many peripherals

After reset, the MCU will automatically select the appropriate reset source and begin to execute code. At this point the system clock is the 16 MHz FIRC (internal) oscillator, the CPU is in supervisor mode and all the memory is available. Initialization is required before most peripherals may be used and before the SRAM can be read (since the SRAM is protected by ECC, the syndrome will generally be uninitialized after reset and reads would fail the check). Accessing disabled features causes error conditions or interrupts.

A typical startup routine would involve initializing the software environment including stacks, heaps, memory, and variable initialization; and configuring the MCU for the application.

The MC\_ME module enables the modules and other features like clocks. It is therefore an essential part of the initialization and operation software. In general, the software will configure an MC\_ME mode to make certain peripherals, clocks, and memory active and then switch to that mode.

[Chapter 6, Clock Description](#), includes a graphic of the clock architecture of the MCU. This can be used to determine how to configure the MC\_CGM module. In general software will configure the module to enable the required clocks and PLLs and route these to the active modules.

After these steps are complete it is possible to configure the input/output pins and the modules for the application.

## 1.7.4 Other features

The MC\_ME module manages low power modes and so it is likely that it will be used to switch into different configurations (module sets, clocks) depending on the application requirements.

The MCU includes two other features to improve the integrity of the application:

- It is possible to enable a software watchdog (SWT) immediately at reset or afterwards to help detect code runaway.
- Individual register settings can be protected from unintended writes using the features of the Register Protection module. The protected registers are shown in [Chapter 32, Register Protection](#).

Other integration functionality is provided by the System Status and Configuration Module (SSCM).

# Chapter 2

## Introduction

### 2.1 The MPC5606BK microcontroller family

The MPC5606BK is a family of Power Architecture<sup>®</sup>-based microcontrollers that target automotive vehicle body applications such as:

- Central body electronics
- Vehicle body controllers
- Smart junction boxes
- Front modules
- Body peripherals
- Door control
- Seat control

The MPC5606BK family expands the range of the MPC560xB/C microcontroller family. It provides the scalability needed to implement platform approaches and delivers the performance required by increasingly sophisticated software architectures. The advanced and cost-efficient host processor core of the MPC5606BK automotive controller family complies with the Power Architecture embedded category, and only implements the VLE (variable-length encoding) APU, providing improved code density. It operates at speeds as high as 64 MHz and offers high performance processing optimized for low power consumption. It also capitalizes on the available development infrastructure of current Power Architecture devices and is supported with software drivers, operating systems, and configuration code to assist with users implementations.

This document describes the features of the family and options available within the family members, and highlights important electrical and physical characteristics of the device.

### 2.2 MPC5606BK device comparison

Table 2-1 summarizes the MPC5606BK family of microcontrollers.

**Table 2-1. MPC5606BK family comparison<sup>1</sup>**

| Feature                      | MPC5605BK      |          |          | MPC5606BK |          |          |
|------------------------------|----------------|----------|----------|-----------|----------|----------|
|                              | 100 LQFP       | 144 LQFP | 176 LQFP | 100 LQFP  | 144 LQFP | 176 LQFP |
| Package                      | 100 LQFP       | 144 LQFP | 176 LQFP | 100 LQFP  | 144 LQFP | 176 LQFP |
| CPU                          | e200z0h        |          |          |           |          |          |
| Execution speed <sup>2</sup> | Up to 64 MHz   |          |          |           |          |          |
| Code flash memory            | 768 KB         |          |          | 1024 KB   | 1 MB     |          |
| Data flash memory            | 64 (4 x 16) KB |          |          |           |          |          |
| SRAM                         | 64 KB          |          |          | 64 KB     | 80 KB    |          |
| MPU                          | 8-entry        |          |          |           |          |          |
| eDMA                         | 16 ch          |          |          |           |          |          |

Table 2-1. MPC5606BK family comparison<sup>1</sup> (continued)

| Feature                                       | MPC5605BK        |                  |       | MPC5606BK        |                  |       |
|---|------------------|------------------|-------|------------------|------------------|-------|
| 10-bit ADC                                    | Yes              |                  |       |                  |                  |       |
| dedicated <sup>3</sup>                        | 7 ch             | 15 ch            | 29 ch | 7 ch             | 15 ch            | 29 ch |
| shared with 12-bit ADC                        | 19 ch            |                  |       |                  |                  |       |
| 12-bit ADC                                    | Yes              |                  |       |                  |                  |       |
| dedicated <sup>4</sup>                        | 5 ch             |                  |       |                  |                  |       |
| shared with 10-bit ADC                        | 19 ch            |                  |       |                  |                  |       |
| Total timer I/O <sup>5</sup><br>eMIOS         | 37 ch,<br>16-bit | 64 ch,<br>16-bit |       | 37 ch,<br>16-bit | 64 ch,<br>16-bit |       |
| Counter / OPWM / ICOC <sup>6</sup>            | 10 ch            |                  |       |                  |                  |       |
| O(I)PWM / OPWFMB / OPWMCB / ICOC <sup>7</sup> | 7 ch             |                  |       |                  |                  |       |
| O(I)PWM / ICOC <sup>8</sup>                   | 7 ch             | 14 ch            |       | 7 ch             | 14 ch            |       |
| OPWM / ICOC <sup>9</sup>                      | 13 ch            | 33 ch            |       | 13 ch            | 33 ch            |       |
| SCI (LINFlex)                                 | 4                | 6                | 8     | 4                | 6                | 8     |
| SPI (DSPI)                                    | 3                | 5                | 6     | 3                | 5                | 6     |
| CAN (FlexCAN)                                 | 6                |                  |       |                  |                  |       |
| I <sup>2</sup> C                              | 1                |                  |       |                  |                  |       |
| 32 KHz oscillator                             | Yes              |                  |       |                  |                  |       |
| GPIO <sup>10</sup>                            | 77               | 121              | 149   | 77               | 121              | 149   |
| Debug   | JTAG             |                  |       |                  |                  |       |

<sup>1</sup> Feature set dependent on selected peripheral multiplexing; table shows example.

<sup>2</sup> Based on 125 °C ambient operating temperature.

<sup>3</sup> Not shared with 12-bit ADC, but possibly shared with other alternate functions.

<sup>4</sup> Not shared with 10-bit ADC, but possibly shared with other alternate functions.

<sup>5</sup> Refer to eMIOS section of device reference manual for information on the channel configuration and functions.

<sup>6</sup> Each channel supports a range of modes including Modulus counters, PWM generation, Input Capture, Output Compare.

<sup>7</sup> Each channel supports a range of modes including PWM generation with dead time, Input Capture, Output Compare.

<sup>8</sup> Each channel supports a range of modes including PWM generation, Input Capture, Output Compare, Period and Pulse width measurement.

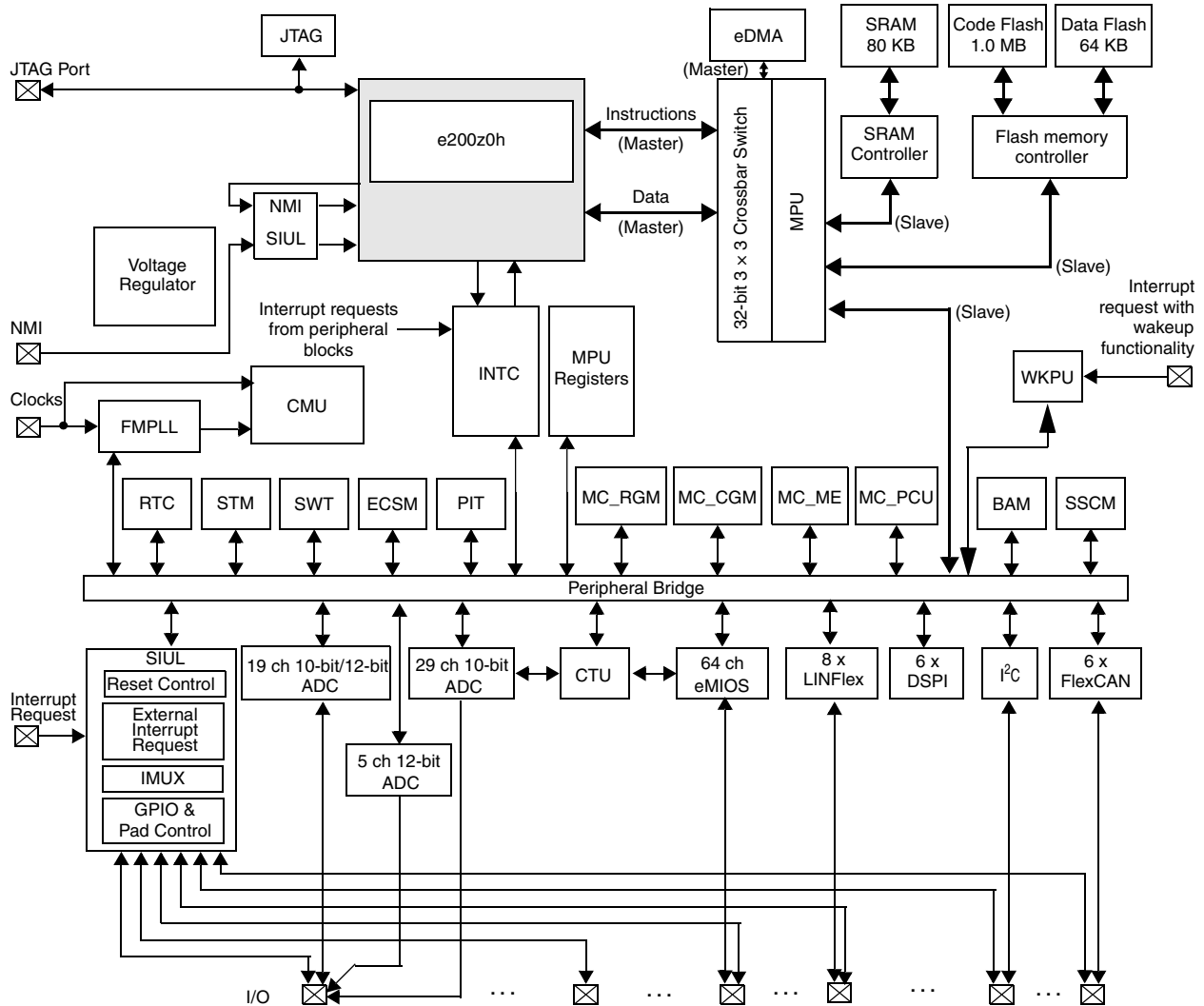
<sup>9</sup> Each channel supports a range of modes including PWM generation, Input Capture, and Output Compare.

<sup>10</sup> Maximum I/O count based on multiplexing with peripherals.

## 2.3 Device block diagram

Figure 2-1 shows a top-level block diagram of the MPC5606BK.





Legend:

|                  |                                       |         |  |
|------------------|---------------------------------------|---------|--|
| ADC              | Analog-to-Digital Converter           | LINFlex | Serial Communication Interface (LIN support) |
| BAM              | Boot Assist Module                    | MC_CGM  | Clock Generation Module                      |
| FlexCAN          | Controller Area Network               | MC_ME   | Mode Entry Module                            |
| CFlash           | Code flash memory                     | MPU     | Memory Protection Unit                       |
| CMU              | Clock Monitor Unit                    | NMI     | Non-Maskable Interrupt                       |
| CTU              | Cross Triggering Unit                 | MC_PCU  | Power Control Unit                           |
| DFlash           | Data flash memory                     | MC_RGM  | Reset Generation Module                      |
| DSPI             | Deserial Serial Peripheral Interface  | PIT     | Periodic Interrupt Timer                     |
| eDMA             | Enhanced Direct Memory Access         | RTC     | Real-Time Clock                              |
| eMIOS            | Enhanced Modular Input Output System  | SIUL    | System Integration Unit Lite                 |
| FMPPLL           | Frequency-Modulated Phase-Locked Loop | SRAM    | Static Random-Access Memory                  |
| I <sup>2</sup> C | Inter-integrated Circuit Bus          | SSCM    | System Status Configuration Module           |
| IMUX             | Internal Multiplexer                  | STM     | System Timer Module                          |
| INTC             | Interrupt Controller                  | SWT     | Software Watchdog Timer                      |
| JTAG             | JTAG controller                       | WKPU    | Wakeup Unit                                  |

Figure 2-1. MPC5606BK block diagram

Table 2-2 summarizes the functions of the blocks present on the MPC5606BK.

Table 2-2. MPC5606BK series block summary

| Block  | Function   |
|--|--|
| Analog-to-digital converter (ADC)                | Converts analog voltages to digital values   |
| Boot assist module (BAM)                         | A block of read-only memory containing VLE code which is executed according to the boot mode of the device   |
| Clock generation module (MC_CGM)                 | Provides logic and control required for the generation of system and peripheral clocks   |
| Crossbar switch (XBAR)                           | Supports simultaneous connections between three master ports and three slave ports. The crossbar supports a 32-bit address bus width and a 64-bit data bus width.  |
| Cross triggering unit (CTU)                      | Enables synchronization of ADC conversions with a timer event from the eMIOS or from the PIT   |
| Deserial serial peripheral interface (DSPI)      | Provides a synchronous serial interface for communication with external devices  |
| Enhanced modular input output system (eMIOS)     | Provides the functionality to generate or measure events   |
| Flash memory                                     | Provides non-volatile storage for program code, constants and variables  |
| FlexCAN (controller area network)                | Supports the standard CAN communications protocol  |
| Frequency-modulated phase-locked loop (FMPLL)    | Generates high-speed system clocks and supports programmable frequency modulation  |
| Internal multiplexer (IMUX) SIU subblock         | Allows flexible mapping of peripheral interface on the different pins of the device  |
| Inter-integrated circuit (I <sup>2</sup> C™) bus | A two wire bidirectional serial bus that provides a simple and efficient method of data exchange between devices   |
| Interrupt controller (INTC)                      | Provides priority-based preemptive scheduling of interrupt requests  |
| JTAG controller                                  | Provides the means to test chip functionality and connectivity while remaining transparent to system logic when not in test mode   |
| LINFlex controller                               | Manages a high number of LIN (Local Interconnect Network protocol) messages efficiently with a minimum of CPU load   |
| Memory protection unit (MPU)                     | Provides hardware access control for all memory references generated in a device   |
| Periodic interrupt timer (PIT)                   | Produces periodic interrupts and triggers  |
| Real-time counter (RTC)                          | A free running counter used for time keeping applications, the RTC can be configured to generate an interrupt at a predefined interval independent of the mode of operation (run mode or low-power mode)                 |
| Reset generation module (MC_RGM)                 | Centralizes reset sources and manages the device reset sequence of the device  |
| Static random-access memory (SRAM)               | Provides storage for program code, constants, and variables  |
| System integration unit lite (SIUL)              | Provides control over all the electrical pad controls and up 32 ports with 16 bits of bidirectional, general-purpose input and output signals and supports up to 32 external interrupts with trigger event configuration |
| System timer module (STM)                        | Provides a set of output compare events to support the Automotive Open System Architecture (AUTOSAR) and operating system tasks  |

## 2.4 Feature details

### 2.4.1 e200z0h core processor

The e200z0h core includes the following features:

- High performance, e200z0h core processor for managing peripherals and interrupts
- Single issue 4-stage pipelined in-order execution, 32-bit Power Architecture CPU
- Variable length encoding (VLE), allowing mixed 16-bit and 32-bit instructions
  - Results in efficient code size footprint
  - Minimizes impact on performance
- Branch processing acceleration using lookahead instruction buffer
- Load/store unit
  - 1-cycle load latency
  - Misaligned access support
  - No load-to-use pipeline bubbles
- 32-bit general purpose registers (GPRs)
- Separate instruction bus and load/store bus Harvard architecture
- Hardware vectored interrupt support
- Multi-cycle divide word (divw) and load multiple word (lmw) store multiple word (smw) multiple class instructions, can be interrupted to prevent increases in interrupt latency

### 2.4.2 Crossbar switch (XBAR)

The following summarizes the MPC5606BK's implementation of the crossbar switch:

- Three master ports:
  - CPU instruction bus
  - CPU load/store bus
  - eDMA
- Multiple bus slaves to enable access to flash memory, SRAM, and peripherals
- Crossbar supports as many as two consecutive transfers at any one time
- 32-bit internal address bus, 32-bit internal data bus
- Fixed priority arbitration based on port master

### 2.4.3 Interrupt Controller (INTC)

The MPC5606BK implements an interrupt controller that features the following:

- Unique 9-bit vector for each of the 231 separate interrupt sources
- Eight software triggerable interrupt sources

- 16 priority levels with fixed hardware arbitration within priority levels for each interrupt source
- Ability to modify the ISR or task priority
  - Modifying the priority can be used to implement Priority Ceiling Protocol for accessing shared resources
- External high priority interrupt directly accessing the main core critical interrupt mechanism

#### 2.4.4 System Integration Unit Lite (SIUL)

The SIUL features the following:

- As many as four levels of internal pin multiplexing, allowing exceptional flexibility in the allocation of device functions for each package
- Centralized general purpose input output (GPIO) control of as many as 149 input/output pins (package dependent)
- All GPIO pins independently configurable to support pull-up pull down, or no pull
- Reading and writing to GPIO supported both as individual pins and 16-bit wide ports
- All peripheral pins can be alternatively configured as both general purpose input or output pins except ADC channels which support alternative configuration as general purpose inputs, with selected pins able to also support outputs
- Direct readback of the pin value supported on all digital output pins through the SIUL
- Configurable digital input filter that can be applied to as many as 16 general purpose input pins for noise elimination on external interrupts
- Register configuration protected against change with soft lock for temporary guard or hard lock to prevent modification until next reset
- Support for two 32-bit virtual ports via the DSPI serialization

#### 2.4.5 Flash memory

The on-chip flash memory on the MPC5606BK features the following:

- As much as 1.0 MB burst flash memory
  - $4 \times$  128-bit page buffers with programmable prefetch control
  - Typical flash-memory access time: 0 wait-state for buffer hits, 2 wait-states for page buffer miss at 64 MHz
  - Page buffers can be allocated for code-only, fixed partitions of code and data, all available for any access
  - 64-bit ECC with single-bit correction, double-bit detection for data integrity
- Censorship protection scheme to prevent flash-memory content visibility
- Separate dedicated data flash memory (DFlash) for EEPROM emulation
  - Four erase sectors, each containing 16 KB of memory
  - Offers read-while-write functionality from main program space

- Small block flash-memory arrangement in main array to support features such as boot block, operating system block
- Hardware managed flash memory writes, erase and verify sequence
- Flash-memory partitioning (see [Table 2-3](#))
- Error correction status
  - Configurable error-correcting codes (ECC) reporting for SRAM and flash memory
  - Supports optional reporting of single-bit errors
  - Protected mechanism for reporting of corrected ECC values
  - Error address recorded including Access type and Master
  - Flash-memory ECC reporting registers mirrored into ECSM address space but data comes from the flash-memory module
  - Flash-memory module can be interrogated to provide ECC bit error location
  - Margin read for flash-memory array supported for initial program verification

**Table 2-3. Flash memory partitioning**

| Array   | Address                  | MPC5605B | MPC5606B |
|---------|--------------------------|----------|----------|
|         |                          | 768 MB   | 1 MB     |
| Array_A | Flash_Base + 0x0000_0000 | 32 KB    | 32 KB    |
|         | Flash_Base + 0x0000_8000 | 16 KB    | 16 KB    |
|         | Flash_Base + 0x0000_C000 | 16 KB    | 16 KB    |
|         | Flash_Base + 0x0001_0000 | 32 KB    | 32 KB    |
|         | Flash_Base + 0x0001_8000 | 32 KB    | 32 KB    |
|         | Flash_Base + 0x0002_0000 | 128 KB   | 128 KB   |
|         | Flash_Base + 0x0004_0000 | 128 KB   | 128 KB   |
|         | Flash_Base + 0x0006_0000 | 128 KB   | 128 KB   |
| Array_B | Flash_Base + 0x0008_0000 | 128 KB   | 128 KB   |
|         | Flash_Base + 0x000A_0000 | 128 KB   | 128 KB   |
|         | Flash_Base + 0x000C_0000 | —        | 128 KB   |
|         | Flash_Base + 0x000E_0000 | —        | 128 KB   |
| Array_C | Flash_Base + 0x0010_0000 | —        | —        |
|         | Flash_Base + 0x0012_0000 | —        | —        |
|         | Flash_Base + 0x0014_0000 | —        | —        |
|         | Flash_Base + 0x0016_0000 | —        | —        |

**Table 2-3. Flash memory partitioning (continued)**

| Array   | Address                        | MPC5605B | MPC5606B |
|---------|--------------------------------|----------|----------|
|         |                                | 768 MB   | 1 MB     |
| Array_D | Data Flash Block + 0x0000_0000 | 16 KB    | 16 KB    |
|         | Data Flash Block + 0x0000_4000 | 16 KB    | 16 KB    |
|         | Data Flash Block + 0x0000_8000 | 16 KB    | 16 KB    |
|         | Data Flash Block + 0x0000_C000 | 16 KB    | 16 KB    |

## 2.4.6 SRAM

The on-chip SRAM on the MPC5606BK features the following:

- As much as 80 KB general purpose SRAM
- Typical SRAM access time: 0 wait-state for reads and 32-bit writes; 1 wait-state for 8- and 16-bit writes if back to back with a read to same memory block
- 32-bit ECC with single-bit correction, double-bit detection for data integrity
- Supports byte (8-bit), half word (16-bit), and word (32-bit) writes for optimal use of memory
- User transparent ECC encoding and decoding for byte, half word, and word accesses
- Separate internal power domain applied to 32 KB SRAM block or 8 KB SRAM block during STANDBY modes to retain contents during low power mode

## 2.4.7 Memory Protection Unit (MPU)

The MPU provides the following features

- Eight region descriptors for per-master protection
- Start and end address defined with 32-byte granularity
- Overlapping regions supported
- Protection attributes can optionally include process ID
- Protection offered for Threeconcurrent read ports
- Read and write attributes for all masters
- Execute and supervisor/user mode attributes for processor masters

## 2.4.8 Boot Assist Module (BAM)

The device implements a Boot Assist Module (BAM):

- Block of read-only memory containing VLE code which is executed according to boot mode of the device
- Download of code into internal SRAM possible via FlexCAN or LINFlex, after which code can be executed

## 2.4.9 Enhanced Modular Input Output System (eMIOS)

The MPC5606BK implements a scaled-down version of the eMIOS module:

- As many as 64 timed I/O channels with 16-bit counter resolution
- Buffered updates
- Support for shifted PWM outputs to minimize occurrence of concurrent edges
- Supports configurable trigger outputs for ADC conversion for synchronization to channel output waveforms
- Edge-aligned output pulse width modulation
  - Programmable pulse period and duty cycle
  - Supports 0% and 100% duty cycle
  - Shared or independent time bases
- DMA transfer support available

Table 2-4 shows the supported eMIOS modes.

**Table 2-4. Supported eMIOS channel modes**

| Mode   |        | Channel type                |   |                   |                |
|--|--------|-----------------------------|---|-------------------|----------------|
| Description  | Name   | Counter /<br>OPWM /<br>ICOC | O(I)PWM /<br>OPWFMB /<br>OPWMCB /<br>ICOC | O(I)PWM /<br>ICOC | OPWM /<br>ICOC |
| Double action output compare                         | DAOC   | x                           | x   | x                 | —              |
| General purpose input / output                       | GPIO   | x                           | x   | x                 | x              |
| Input filter   | IPF    | x                           | x   | x                 | x              |
| Input period measurement                             | IPM    | x                           | x   | x                 | —              |
| Input pulse width measurement                        | IPWM   | x                           | x   | x                 | —              |
| Modulus counter                                      | MC     | x                           | —   | —                 | —              |
| Modulus counter buffered (up / down)                 | MCB    | x                           | x   | —                 | —              |
| Output pulse width and frequency modulation buffered | OPWFMB | x                           | x   | —                 | —              |
| Output pulse width modulation buffered               | OPWMB  | —                           | x   | x                 | x              |
| Center aligned output PWM buffered with dead time    | OPWMCB | —                           | x   | —                 | —              |
| Output pulse width modulation trigger                | OPWMT  | x                           | x   | x                 | x              |
| Pulse edge accumulation                              | PEA    | x                           | —   | —                 | —              |
| Pulse edge counting                                  | PEC    | x                           | —   | —                 | —              |
| Quadrature decode                                    | QDEC   | x                           | —   | —                 | —              |
| Single action input capture                          | SAIC   | x                           | x   | x                 | x              |
| Single action output compare                         | SAOC   | x                           | x   | x                 | x              |

Table 2-5 shows the maximum eMIOS channel allocation.

Table 2-5. eMIOS configuration

| Channel type                                  | Maximum number of channels |         | Total |
|---|----------------------------|---------|-------|
|   | eMIOS_0                    | eMIOS_1 |       |
| Counter / OPWM / ICOC <sup>1</sup>            | 5                          | 5       | 10    |
| O(I)PWM / OPWFMB / OPWMCB / ICOC <sup>2</sup> | 7                          | 0       | 7     |
| O(I)PWM / ICOC <sup>3</sup>                   | 7                          | 7       | 14    |
| OPWM / ICOC <sup>4</sup>                      | 13                         | 20      | 33    |

<sup>1</sup> Each channel supports a range of modes including Modulus counters, PWM generation, Input Capture, Output Compare.

<sup>2</sup> Each channel supports a range of modes including PWM generation with dead time, Input Capture, Output Compare.

<sup>3</sup> Each channel supports a range of modes including PWM generation, Input Capture, Output Compare, Period and Pulse width measurement.

<sup>4</sup> Each channel supports a range of modes including PWM generation, Input Capture, Output Compare.

## 2.4.10 Deserial Serial Peripheral Interface Module (DSPI)

The DSPI features the following:

- As many as six DSPI modules supported
- Full duplex, synchronous transfers
- Master or slave operation
- Programmable master bit rates
- Programmable clock polarity and phase
- End-of-transmission interrupt flag
- Programmable transfer baud rate
- Programmable data frames from 4 to 16 bits
- As many as six chip select lines available, depending on package and pin multiplexing, to enable 64 external devices to be selected using external muxing from a single DSPI
- As many as eight transfer types, independently configurable for each DSPI using the clock and transfer attributes registers
- Chip select strobe available as alternate function on one of the chip select pins for deglitching
- FIFOs for buffering as many as four transfers on the transmit and receive side
- General purpose I/O functionality on pins when not used for SPI
- Queueing operation possible through use of eDMA
- 32-bit serialization of data enabling virtual GPIO ports on 2 DSPI modules

## 2.4.11 Controller Area Network module (FlexCAN)

The enhanced FlexCAN module features the following:

- As many as six FlexCAN modules supported



- Compliant with CAN protocol specification, version 2.0B active
- 64 mailboxes per FlexCAN module
  - Mailboxes configurable while module remains synchronized to CAN bus
  - Each mailbox configurable as transmit or receive
- Transmit features
  - Supports configuration of multiple mailboxes to form message queues of scalable depth
  - Arbitration scheme according to message ID or message buffer number
  - Internal arbitration to guarantee no inner or outer priority inversion
  - Transmit abort procedure and notification
- Receive features
  - Individual programmable filters for each mailbox
  - Eight mailboxes configurable as a 6-entry receive FIFO
  - Eight programmable acceptance filters for receive FIFO
- Programmable clock source
  - System clock
  - Direct oscillator clock to avoid PLL jitter
- Listen only mode capabilities
- CAN sampler available for connection to one of available CAN module pads
  - Supports capturing of first message identifier while in STOP or STANDBY modes

## 2.4.12 System clocks and clock generation

The following list summarizes the system clock and clock generation on the MPC5606BK:

- System clock can be derived from the following sources
  - External crystal oscillator
  - FMPLL
  - 16 MHz fast internal RC oscillator
- Programmable output clock divider of system clock ( $\div 1$ ,  $\div 2$ ,  $\div 4$ )
- Separate programmable peripheral bus clock divider ratio ( $\div 1$ ,  $\div 2$ ,  $\div 4$ ) applied to system clock
- Frequency modulated phase-locked loop (FMPLL)
  - Input clock frequency from 4 MHz to 16 MHz
  - Clock sources: external oscillator or internal FIRC oscillator
  - Lock detect circuitry continuously monitors lock status
  - Loss of clock (LOC) detection for reference and feedback clocks
  - On-chip loop filter
    - Improves electromagnetic interference performance
    - Reduces number of external components required
- On-chip fast external crystal oscillator supporting 4 MHz to 16 MHz

- Dedicated 16 MHz fast internal RC oscillator
  - Used as default clock source out of reset
  - Provides clock for rapid startup from low power modes
  - Provides back-up clock in the event of FMPLL or external oscillator clock failure
  - Offers independent clock source for the watchdog timer
  - 5% accuracy over the operating temperature range
  - Trimming registers to support frequency adjustment with in-application calibration
- Dedicated 128 kHz slow internal RC oscillator for low power mode operation and self wakeup
  - 5% accuracy
  - Trimming registers to support improve accuracy with in-application calibration
- 32-KHz low power external oscillator for low power real time clock

### 2.4.13 System timers

The system timers include:

- Peripheral Interrupt Timer (PIT) timers (including ADC trigger)
- One Real-Time Counter (RTC) timer

The PIT is an array of timers that can be used to raise interrupts, trigger CTU channels and ADC conversions. The RTC supports wakeup from low power modes or real-time clock generation.

#### 2.4.13.1 Periodic Interrupt Timer module (PIT)

The PIT features the following:

- Eight general purpose interrupt timers
- As many as two interrupt timers for triggering ADC injected conversions (one for 10-bit ADC, one for 12-bit ADC)
- As many as four interrupt timers for triggering DMA transfers
- As many as two interrupt timers for triggering CTU
- 32-bit counter resolution
- Clocked by system clock frequency

#### 2.4.13.2 Real-Time Counter (RTC)

The RTC features the following:

- Configurable resolution for different timeout periods
  - 1 sec resolution for > 1 hour period
  - 1 ms resolution for 2 second period
- Selectable clock sources
  - 128 kHz slow internal RC oscillator
  - Divided 16 MHz fast internal RC oscillator

- External 32 KHz crystal
- Supports continued operation through all resets except POR (power-on reset)

### 2.4.14 System watchdog timer

The watchdog on the MPC5606BK features the following:

- Activation by software or out of reset
- 32-bit modulus counter
- Clock source: robust 128 kHz slow internal RC oscillator (divisible by 1 to 32)
- Supports normal or windowed mode
- Configurable response on timeout: reset, interrupt, or interrupt followed by reset
- Reset by writing a software key to memory mapped register
- Support for protected access to watchdog control registers with optional soft and hard locks
  - Soft lock allows temporary locking of configuration
  - Once enabled, hard lock prevents any changes until after a reset
- Supports halting during low power modes

### 2.4.15 Inter-Integrated Circuit (I<sup>2</sup>C) module

The I<sup>2</sup>C module features the following:

- One I<sup>2</sup>C module supported
- 2-wire bidirectional serial bus for on-board communications
- Compatibility with I<sup>2</sup>C bus standard
- Multimaster operation
- Software-programmable for one of 256 different serial clock frequencies
- Software-selectable acknowledge bit
- Interrupt-driven, byte-by-byte data transfer
- Arbitration-lost interrupt with automatic mode switching from master to slave
- Calling address identification interrupt
- Start and stop signal generation/detection
- Repeated START signal generation
- Acknowledge bit generation/detection
- Bus-busy detection

### 2.4.16 On-chip voltage regulator (VREG)

The on-chip voltage regulator includes the following features:

- Regulates 3.3 or 5 V  $\pm 10\%$  input to generate all internal supplies for internal control
- Manages power gating

- Low power regulators support operation when in STOP and STANDBY modes to minimize power consumption
- Fast startup on-chip regulators for rapid exit from low power modes
- Low voltage reset supported on all internal supplies

### 2.4.17 Analog-to-Digital Converter (ADC)

The ADC features the following:

- Two ADC modules, one 10-bit resolution and one 12-bit resolution supporting synchronous conversions on channels
- $0-V_{DD}$  common mode conversion range
- Independent reference supplies for each ADC
- Conversion times of  $< 1 \mu\text{s}$  available
- As many as 53 single ended inputs channels, expandable to 81 channels with external multiplexers
- As many as 19 shared channels, among which, 16 called ANP are mapped on dedicated pins, not multiplexed with any other functionality, in order to improve the accuracy. All other channels, called ANS or ANX are multiplexed with other functionalities.
  - As many as 19 channels shared between 10-bit and 12-bit ADCs
  - As many as five dedicated 12-bit ADC channels
  - As many as 29 dedicated 10-bit ADC channels
- Externally multiplexed channels
  - Internal control to support generation of external analog multiplexor selection
  - Four internal channels optionally used to support externally multiplex inputs, providing transparent control for additional ADC channels
  - Each of the three channels supports as many as eight externally muxed inputs
  - Individual dedicated result register also available for externally muxed conversion channels
  - Three independently configurable sample and conversion times for high occurrence channels, internally muxed channels and externally muxed channels
- Configurable right-aligned or left-aligned result formats
- Support for one-shot, scan and injection conversion modes
- Independently configurable parameters for channels:
  - Offset refresh
  - Sampling
- Conversion triggering support
  - Internal conversion triggering from periodic interrupt timer (PIT) or timed I/O module (eMIOS) through cross triggering unit (CTU)
  - Internal conversion triggering from periodic interrupt timer (PIT)
  - One input pin configurable as external conversion trigger source

- As many as six configurable analog comparator channels offering range comparison with triggered alarm
  - Greater than
  - Less than
  - Out of range
- All unused analog pins available as general purpose input pins
- Unused 10-bit ADC analog pins, with the exception of the 19 dedicated high accuracy channels, available as general purpose output pins
- Power-down mode
- Supports DMA transfer of results based on end of conversion chain or each conversion
- Separate dedicated DMA request for injection mode

### 2.4.18 Enhanced Direct Memory Access controller (eDMA)

The following summarizes the MPC5606BK's implementation of the eDMA controller:

- 16 channels to support independent 8-, 16-, or 32-bit single value or block transfers
- Support of variable sized queues and circular queues
- Source and destination address registers independently configured to post-increment or remain constant
- Each transfer initiated by peripheral, CPU, periodic timer interrupt, or eDMA channel request
- Peripheral DMA request sources possible from SPIs, I<sup>2</sup>C, 10-bit ADC, 12-bit ADC, eMIOS, and GPIOs
- Each eDMA channel able to optionally send interrupt request to CPU on completion of single value or block transfer
- DMA transfers possible between system memories and all accessible memory mapped locations including peripheral and registers
- Programmable DMA channel mux allows assignment of any DMA source to any available DMA channel with as many as 64 potential request sources

### 2.4.19 Cross Trigger Unit (CTU)

The CTU enables the synchronization of ADC conversions with a timer event. Its key features are:

- Single cycle delayed trigger output; trigger output is a combination of 64 (generic value) input flags/events connected to different timers in the system
- Triggers ADC conversions from any eMIOS channel
- Triggers ADC conversions from as many as two dedicated PITs
- Maskable interrupt generation whenever a trigger output is generated
- One event configuration register dedicated to each timer event allows to define the corresponding ADC channel
- Acknowledgment signal to eMIOS/PIT for clearing the flag

- Synchronization with ADC to avoid collision

### 2.4.20 Serial communication interface module (LINFlex)

The LINFlex on the MPC5606BK features the following:

- As many as eight LINFlex modules supported
- Supports LIN master mode, LIN slave mode and UART mode
- DMA connected on LINFlex\_0 and LINFlex\_1
- LINFlex\_0 supporting LIN master and slave mode; LINFlex\_1 to LINFlex\_7 modules supporting LIN master mode
- LIN state machine compliant to LIN 1.3, 2.0 and 2.1 specifications
- Handles LIN frame transmission and reception without CPU intervention
- LIN features
  - Autonomous LIN frame handling
  - Message buffer to store identified and up to 8 data bytes
  - Supports message length of up to 64 bytes
  - Detection and flagging of LIN errors
    - Sync field; delimiter; ID parity; bit, framing; checksum and timeout errors
  - Classic or extended checksum calculation
  - Configurable break duration of up to 36-bit times
  - Programmable baud rate prescalers (13-bit mantissa, 4-bit fractional)
  - Diagnostic features
    - Loop back; Self Test; LIN bus stuck dominant detection
  - Interrupt driven operation with 16 interrupt sources
- LIN slave mode features
  - Autonomous LIN header handling
  - Autonomous LIN response handling
  - 16 identifiers filters for discarding irrelevant LIN frames
- UART mode
  - Full-duplex operation
  - Standard non return-to-zero (NRZ) mark/space format
  - Data buffers with 4-bytes receive, 4-bytes transmit
  - Configurable word length (8-bit or 9-bit words)
  - Error detection and flagging
    - Parity, noise and framing errors
  - Interrupt driven operation with four interrupt sources
  - Separate transmitter and receiver CPU interrupt sources
  - 16-bit programmable baud rate modulus counter and 16-bit fractional

— Two receiver wakeup methods

MPC5606BK devices include two functionally-different LINFlex controller types. These are distinguished in the documentation by the abbreviations “LINFlex” and “LINFlexD”. The latter name represents the DMA support available on this controller type. The MPC5606BK devices combine these two types to provide as many as eight modules supporting the LINFlex protocol. Table 2-6 shows the module (instance) numbers and the corresponding functional controller type.

**Table 2-6. LINFlex numbering and naming**

| Module numbers | Module version |
|----------------|----------------|
| 0 and 1        | LINFlexD       |
| 2–7            | LINFlex        |

### 2.4.21 JTAG Controller (JTAGC)

JTAG features the following:

- JTAG low pin count interface (IEEE 1149.1) test access port (TAP) interface
- Backward compatible to standard JTAG IEEE 1149.1-2001 test access port (TAP) interface
- Supports boundary scan testing
- All JTAG pins reusable in application as standard IOs

## 2.5 Developer support

The MPC5606BK MCU tools and third-party developers are similar to those used for the Freescale MPC5500 product family, offering a widespread, established network of tool and software vendors.

The following development support will be available:

- Automotive evaluation boards (EVB) featuring CAN, LIN interfaces, and more
- Compilers
- Debuggers
- JTAG interface

The following software support will be available:

- OSEK solutions will be available from multiple third parties
- CAN and LIN drivers
- AutoSAR package

This page is intentionally left blank.



## Chapter 3

# Memory Map

Table 3-1 shows the memory map for the MPC5606BK. All addresses on the device, including those that are reserved, are identified in the table. The addresses represent the physical addresses assigned to each IP block.

**Table 3-1. MPC5606BK memory map**

| Start address | End address | Size (KB) | Region name                           |
|---------------|-------------|-----------|---------------------------------------|
| 0x0000_0000   | 0x0000_7FFF | 32        | Code flash memory array 0             |
| 0x0000_8000   | 0x0000_BFFF | 16        | Code flash memory array 0             |
| 0x0000_C000   | 0x0000_FFFF | 16        | Code flash memory array 0             |
| 0x0001_0000   | 0x0001_7FFF | 32        | Code flash memory array 0             |
| 0x0001_8000   | 0x0001_FFFF | 32        | Code flash memory array 0             |
| 0x0002_0000   | 0x0003_FFFF | 128       | Code flash memory array 0             |
| 0x0004_0000   | 0x0005_FFFF | 128       | Code flash memory array 0             |
| 0x0006_0000   | 0x0007_FFFF | 128       | Code flash memory array 0             |
| 0x0008_0000   | 0x0009_FFFF | 128       | Code flash memory array 0             |
| 0x000A_0000   | 0x000B_FFFF | 128       | Code flash memory array 0             |
| 0x000C_0000   | 0x000D_FFFF | 128       | Code flash memory array 0             |
| 0x000E_0000   | 0x000F_FFFF | 128       | Code flash memory array 0             |
| 0x0010_0000   | 0x001F_FFFF | 1024      | Reserved                              |
| 0x0020_0000   | 0x0020_3FFF | 16        | Flash memory shadow array             |
| 0x0020_4000   | 0x003F_FFFF | 2032      | Reserved                              |
| 0x0040_0000   | 0x0040_3FFF | 16        | Code flash memory array 0 test sector |
| 0x0040_4000   | 0x007F_FFFF | 4080      | Reserved                              |
| 0x0080_0000   | 0x0080_3FFF | 16        | Data flash memory array 0             |
| 0x0080_4000   | 0x0080_7FFF | 16        | Data flash memory array 0             |
| 0x0080_8000   | 0x0080_BFFF | 16        | Data flash memory array 0             |
| 0x0080_C000   | 0x0080_FFFF | 16        | Data flash memory array 0             |
| 0x0081_0000   | 0x00BF_FFFF | 4032      | Reserved                              |
| 0x00C0_0000   | 0x00C0_3FFF | 16        | Data flash memory array 0 test sector |
| 0x00C0_4000   | 0x00FF_FFFF | 4080      | Reserved                              |
| 0x0100_0000   | 0x1FFF_FFFF | 507904    | Flash memory emulation mapping        |
| 0x2000_0000   | 0x3FFF_FFFF | 524288    | Reserved                              |
| 0x4000_0000   | 0x4001_3FFF | 80        | SRAM                                  |

Table 3-1. MPC5606BK memory map (continued)

| Start address                   | End address | Size (KB) | Region name                       |
|---------------------------------|-------------|-----------|-----------------------------------|
| 0x4001_4000                     | 0xBFFF_FFFF | 2097072   | Reserved                          |
| <b>Off-platform peripherals</b> |             |           |                                   |
| 0xC000_0000                     | 0xC3F8_7FFF | 65056     | Reserved                          |
| 0xC3F8_8000                     | 0xC3F8_BFFF | 16        | Code flash memory 0 configuration |
| 0xC3F8_C000                     | 0xC3F8_FFFF | 16        | Data flash memory 0 configuration |
| 0xC3F9_0000                     | 0xC3F9_3FFF | 16        | SIUL                              |
| 0xC3F9_4000                     | 0xC3F9_7FFF | 16        | WKPU                              |
| 0xC3F9_8000                     | 0xC3F9_FFFF | 32        | Reserved                          |
| 0xC3FA_0000                     | 0xC3FA_3FFF | 16        | eMIOS_0                           |
| 0xC3FA_4000                     | 0xC3FA_7FFF | 16        | eMIOS_1                           |
| 0xC3FA_8000                     | 0xC3FD_7FFF | 192       | Reserved                          |
| 0xC3FD_8000                     | 0xC3FD_BFFF | 16        | SSCM                              |
| 0xC3FD_C000                     | 0xC3FD_FFFF | 16        | MC_ME                             |
| 0xC3FE_0000                     | 0xC3FE_3FFF | 16        | MC_CGM                            |
| 0xC3FE_4000                     | 0xC3FE_7FFF | 16        | MC_RGM                            |
| 0xC3FE_8000                     | 0xC3FE_BFFF | 16        | MC_PCU                            |
| 0xC3FE_C000                     | 0xC3FE_FFFF | 16        | RTC/API                           |
| 0xC3FF_0000                     | 0xC3FF_3FFF | 16        | PIT                               |
| 0xC3FF_4000                     | 0xFFDF_FFFF | 981040    | Reserved                          |
| 0xFFE0_0000                     | 0xFFE0_3FFF | 16        | ADC_0                             |
| 0xFFE0_4000                     | 0xFFE0_7FFF | 16        | ADC_1                             |
| 0xFFE0_8000                     | 0xFFE2_FFFF | 160       | Reserved                          |
| 0xFFE3_0000                     | 0xFFE3_3FFF | 16        | I2C_0                             |
| 0xFFE3_4000                     | 0xFFE3_FFFF | 48        | Reserved                          |
| 0xFFE4_0000                     | 0xFFE4_3FFF | 16        | LINFlex_0                         |
| 0xFFE4_4000                     | 0xFFE4_7FFF | 16        | LINFlex_1                         |
| 0xFFE4_8000                     | 0xFFE4_BFFF | 16        | LINFlex_2                         |
| 0xFFE4_C000                     | 0xFFE4_FFFF | 16        | LINFlex_3                         |
| 0xFFE5_0000                     | 0xFFE5_3FFF | 16        | LINFlex_4                         |
| 0xFFE5_4000                     | 0xFFE5_7FFF | 16        | LINFlex_5                         |
| 0xFFE5_8000                     | 0xFFE5_BFFF | 16        | LINFlex_6                         |
| 0xFFE5_C000                     | 0xFFE5_FFFF | 16        | LINFlex_7                         |
| 0xFFE6_0000                     | 0xFFE6_3FFF | 16        | Reserved                          |

Table 3-1. MPC5606BK memory map (continued)

| Start address | End address  | Size (KB) | Region name                            |
|---------------|--------------|-----------|--|
| 0xFFE6_4000   | 0xFFE6_7FFF  | 16        | CTU                                    |
| 0xFFE6_8000   | 0xFFE6_FFFF  | 32        | Reserved                               |
| 0xFFE7_0000   | 0xFFE7_3FFF  | 16        | CAN sampler                            |
| 0xFFE7_4000   | 0xFFE7_FFFF  | 48        | Reserved                               |
| 0xFFE8_0000   | 0xFFEF_FFFF  | 512       | Mirrored range<br>0x3F80000—0xC3FFFFFF |
| 0xFFFF0_0000  | 0xFFFF0_FFFF | 64        | Reserved                               |
| 0xFFFF1_0000  | 0xFFFF1_3FFF | 16        | MPU                                    |
| 0xFFFF1_4000  | 0xFFFF3_7FFF | 144       | Reserved                               |
| 0xFFFF3_8000  | 0xFFFF3_BFFF | 16        | SWT                                    |
| 0xFFFF3_C000  | 0xFFFF3_FFFF | 16        | STM                                    |
| 0xFFFF4_0000  | 0xFFFF4_3FFF | 16        | ECSM                                   |
| 0xFFFF4_4000  | 0xFFFF4_7FFF | 16        | eDMA                                   |
| 0xFFFF4_8000  | 0xFFFF4_BFFF | 16        | INTC                                   |
| 0xFFFF4_C000  | 0xFFFF8_FFFF | 272       | Reserved                               |
| 0xFFFF9_0000  | 0xFFFF9_3FFF | 16        | DSPI_0                                 |
| 0xFFFF9_4000  | 0xFFFF9_7FFF | 16        | DSPI_1                                 |
| 0xFFFF9_8000  | 0xFFFF9_BFFF | 16        | DSPI_2                                 |
| 0xFFFF9_C000  | 0xFFFF9_FFFF | 16        | DSPI_3                                 |
| 0xFFFFA_0000  | 0xFFFFA_3FFF | 16        | DSPI_4                                 |
| 0xFFFFA_4000  | 0xFFFFA_7FFF | 16        | DSPI_5                                 |
| 0xFFFFA_8000  | 0xFFFFB_FFFF | 96        | Reserved                               |
| 0xFFFFC_0000  | 0xFFFFC_3FFF | 16        | FlexCAN_0                              |
| 0xFFFFC_4000  | 0xFFFFC_7FFF | 16        | FlexCAN_1                              |
| 0xFFFFC_8000  | 0xFFFFC_BFFF | 16        | FlexCAN_2                              |
| 0xFFFFC_C000  | 0xFFFFC_FFFF | 16        | FlexCAN_3                              |
| 0xFFFFD_0000  | 0xFFFFD_3FFF | 16        | FlexCAN_4                              |
| 0xFFFFD_4000  | 0xFFFFD_7FFF | 16        | FlexCAN_5                              |
| 0xFFFFD_8000  | 0xFFFFD_BFFF | 16        | Reserved                               |
| 0xFFFFD_C000  | 0xFFFFD_FFFF | 16        | DMA_MUX                                |
| 0xFFFFE_0000  | 0xFFFFF_BFFF | 144       | Reserved                               |
| 0xFFFFF_C000  | 0xFFFFF_FFFF | 16        | BAM                                    |

This page is intentionally left blank.

# Chapter 4

## Signal Description

### 4.1 Package pinouts

Figure 4-1, Figure 4-2, and Figure 4-3 show the location of the signals on the available packages for this device.

For more information on pin multiplexing on this chip, see Table 4-1.

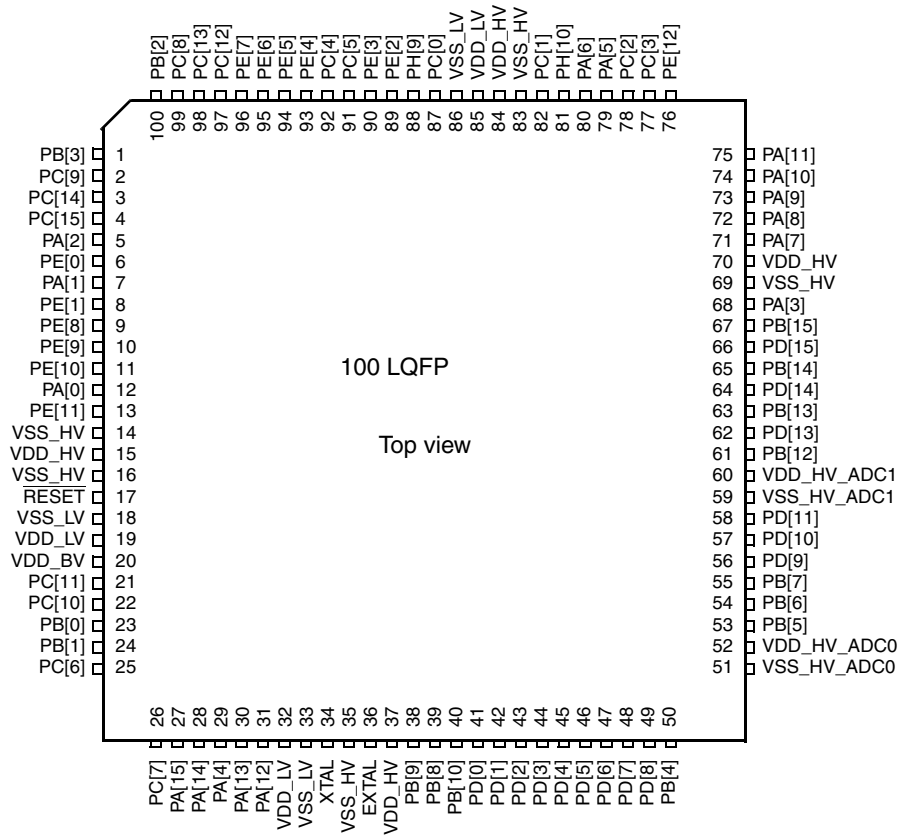


Figure 4-1. 100-pin LQFP pinout

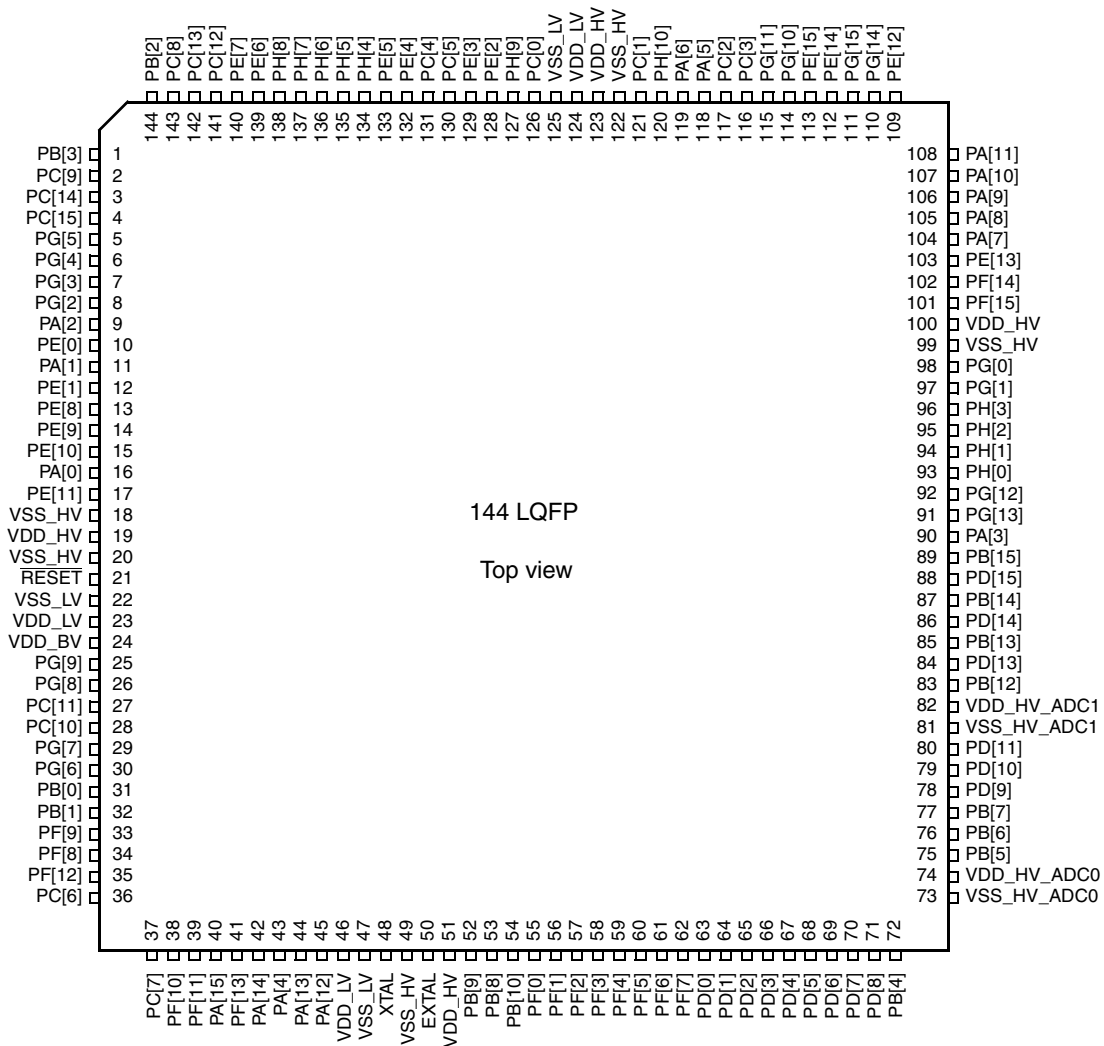


Figure 4-2. 144-pin LQFP pinout

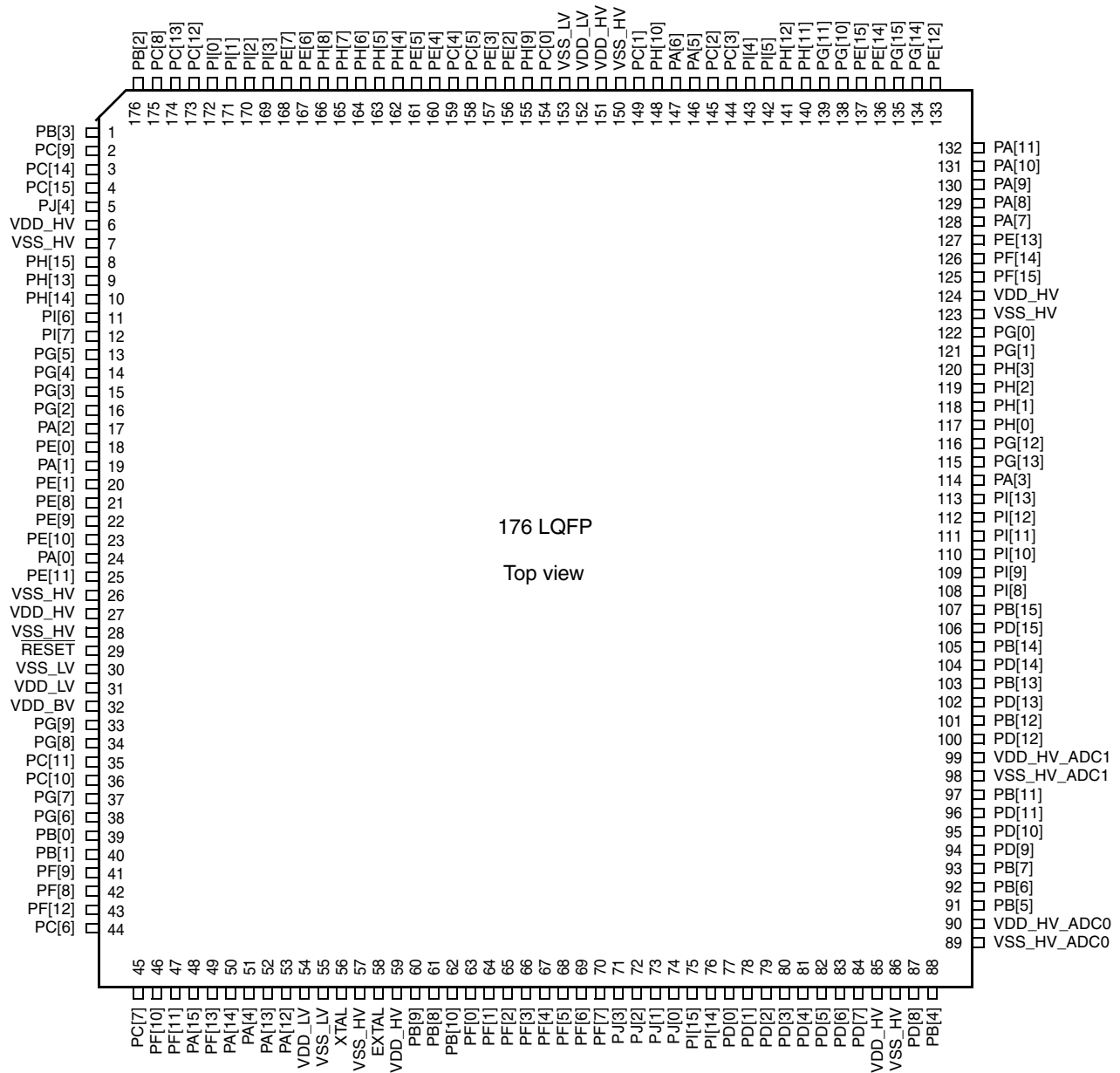


Figure 4-3. 176-pin LQFP pinout

## 4.2 Pin muxing

Table 4-1 defines the pin list and muxing for this device.

Each entry of Table 4-1 shows all the possible configurations for each pin, via the alternate functions. The default function assigned to each pin after reset is indicated by AF0.

Table 4-1. Functional port pins

| Port pin      | PCR register | Alternate function <sup>1</sup>    | Function  | Peripheral  | I/O direction                    | Pad type <sup>2</sup> | RESET config. <sup>3</sup> | Pin number |          |          |
|---------------|--------------|------------------------------------|---|---|----------------------------------|-----------------------|----------------------------|------------|----------|----------|
|               |              |                                    |   |   |                                  |                       |                            | 100 LQFP   | 144 LQFP | 176 LQFP |
| <b>Port A</b> |              |                                    |   |   |                                  |                       |                            |            |          |          |
| PA[0]         | PCR[0]       | AF0<br>AF1<br>AF2<br>AF3<br>—      | GPIO[0]<br>E0UC[0]<br>CLKOUT<br>E0UC[13]<br>WKUP[19] <sup>4</sup>     | SIUL<br>eMIOS_0<br>MC_CGM<br>eMIOS_0<br>WKUP            | I/O<br>I/O<br>O<br>I/O<br>I      | M                     | Tristate                   | 12         | 16       | 24       |
| PA[1]         | PCR[1]       | AF0<br>AF1<br>AF2<br>AF3<br>—      | GPIO[1]<br>E0UC[1]<br>NMI <sup>5</sup><br>—<br>WKUP[2] <sup>(4)</sup> | SIUL<br>eMIOS_0<br>WKUP<br>—<br>WKUP                    | I/O<br>I/O<br>I<br>—<br>I        | S                     | Tristate                   | 7          | 11       | 19       |
| PA[2]         | PCR[2]       | AF0<br>AF1<br>AF2<br>AF3<br>—      | GPIO[2]<br>E0UC[2]<br>—<br>MA[2]<br>WKUP[3] <sup>(4)</sup>            | SIUL<br>eMIOS_0<br>—<br>ADC_0<br>WKUP                   | I/O<br>I/O<br>—<br>O<br>I        | S                     | Tristate                   | 5          | 9        | 17       |
| PA[3]         | PCR[3]       | AF0<br>AF1<br>AF2<br>AF3<br>—<br>— | GPIO[3]<br>E0UC[3]<br>LIN5TX<br>CS4_1<br>EIRQ[0]<br>ADC1_S[0]         | SIUL<br>eMIOS_0<br>LINFlex_5<br>DSPI_1<br>SIUL<br>ADC_1 | I/O<br>I/O<br>O<br>O<br>I<br>I   | J                     | Tristate                   | 68         | 90       | 114      |
| PA[4]         | PCR[4]       | AF0<br>AF1<br>AF2<br>AF3<br>—<br>— | GPIO[4]<br>E0UC[4]<br>—<br>CS0_1<br>LIN5RX<br>WKUP[9] <sup>(4)</sup>  | SIUL<br>eMIOS_0<br>—<br>DSPI_1<br>LINFlex_5<br>WKUP     | I/O<br>I/O<br>—<br>I/O<br>I<br>I | S                     | Tristate                   | 29         | 43       | 51       |
| PA[5]         | PCR[5]       | AF0<br>AF1<br>AF2<br>AF3           | GPIO[5]<br>E0UC[5]<br>LIN4TX<br>—                                     | SIUL<br>eMIOS_0<br>LINFlex_4<br>—                       | I/O<br>I/O<br>O<br>—             | M                     | Tristate                   | 79         | 118      | 146      |
| PA[6]         | PCR[6]       | AF0<br>AF1<br>AF2<br>AF3<br>—<br>— | GPIO[6]<br>E0UC[6]<br>—<br>CS1_1<br>EIRQ[1]<br>LIN4RX                 | SIUL<br>eMIOS_0<br>—<br>DSPI_1<br>SIUL<br>LINFlex_4     | I/O<br>I/O<br>—<br>O<br>I<br>I   | S                     | Tristate                   | 80         | 119      | 147      |
| PA[7]         | PCR[7]       | AF0<br>AF1<br>AF2<br>AF3<br>—<br>— | GPIO[7]<br>E0UC[7]<br>LIN3TX<br>—<br>EIRQ[2]<br>ADC1_S[1]             | SIUL<br>eMIOS_0<br>LINFlex_3<br>—<br>SIUL<br>ADC_1      | I/O<br>I/O<br>O<br>—<br>I<br>I   | J                     | Tristate                   | 71         | 104      | 128      |



Table 4-1. Functional port pins (continued)

| Port pin      | PCR register | Alternate function <sup>1</sup>                        | Function  | Peripheral   | I/O direction                         | Pad type <sup>2</sup> | RESET config. <sup>3</sup> | Pin number |          |          |
|---------------|--------------|--|---|--|---------------------------------------|-----------------------|----------------------------|------------|----------|----------|
|               |              |  |   |  |                                       |                       |                            | 100 LQFP   | 144 LQFP | 176 LQFP |
| PA[8]         | PCR[8]       | AF0<br>AF1<br>AF2<br>AF3<br>—<br>N/A <sup>6</sup><br>— | GPIO[8]<br>E0UC[8]<br>E0UC[14]<br>—<br>EIRQ[3]<br>ABS[0]<br>LIN3RX  | SIUL<br>eMIOS_0<br>eMIOS_0<br>—<br>SIUL<br>BAM<br>LINFlex_3              | I/O<br>I/O<br>I/O<br>—<br>I<br>I<br>I | S                     | Input,<br>weak<br>pull-up  | 72         | 105      | 129      |
| PA[9]         | PCR[9]       | AF0<br>AF1<br>AF2<br>AF3<br>N/A <sup>6</sup>           | GPIO[9]<br>E0UC[9]<br>—<br>CS2_1<br>FAB                             | SIUL<br>eMIOS_0<br>—<br>DSPI_1<br>BAM                                    | I/O<br>I/O<br>—<br>O<br>I             | S                     | Pull-<br>down              | 73         | 106      | 130      |
| PA[10]        | PCR[10]      | AF0<br>AF1<br>AF2<br>AF3<br>—                          | GPIO[10]<br>E0UC[10]<br>SDA<br>LIN2TX<br>ADC1_S[2]                  | SIUL<br>eMIOS_0<br>I <sup>2</sup> C_0<br>LINFlex_2<br>ADC_1              | I/O<br>I/O<br>I/O<br>O<br>I           | J                     | Tristate                   | 74         | 107      | 131      |
| PA[11]        | PCR[11]      | AF0<br>AF1<br>AF2<br>AF3<br>—<br>—<br>—                | GPIO[11]<br>E0UC[11]<br>SCL<br>—<br>EIRQ[16]<br>LIN2RX<br>ADC1_S[3] | SIUL<br>eMIOS_0<br>I <sup>2</sup> C_0<br>—<br>SIUL<br>LINFlex_2<br>ADC_1 | I/O<br>I/O<br>I/O<br>—<br>I<br>I<br>I | J                     | Tristate                   | 75         | 108      | 132      |
| PA[12]        | PCR[12]      | AF0<br>AF1<br>AF2<br>AF3<br>—<br>—                     | GPIO[12]<br>—<br>E0UC[28]<br>CS3_1<br>EIRQ[17]<br>SIN_0             | SIUL<br>—<br>eMIOS_0<br>DSPI_1<br>SIUL<br>DSPI_0                         | I/O<br>—<br>I/O<br>O<br>I<br>I        | S                     | Tristate                   | 31         | 45       | 53       |
| PA[13]        | PCR[13]      | AF0<br>AF1<br>AF2<br>AF3                               | GPIO[13]<br>SOUT_0<br>E0UC[29]<br>—                                 | SIUL<br>DSPI_0<br>eMIOS_0<br>—   | I/O<br>O<br>I/O<br>—                  | M                     | Tristate                   | 30         | 44       | 52       |
| PA[14]        | PCR[14]      | AF0<br>AF1<br>AF2<br>AF3<br>—                          | GPIO[14]<br>SCK_0<br>CS0_0<br>E0UC[0]<br>EIRQ[4]                    | SIUL<br>DSPI_0<br>DSPI_0<br>eMIOS_0<br>SIUL                              | I/O<br>I/O<br>I/O<br>I/O<br>I         | M                     | Tristate                   | 28         | 42       | 50       |
| PA[15]        | PCR[15]      | AF0<br>AF1<br>AF2<br>AF3<br>—                          | GPIO[15]<br>CS0_0<br>SCK_0<br>E0UC[1]<br>WKUP[10] <sup>(4)</sup>    | SIUL<br>DSPI_0<br>DSPI_0<br>eMIOS_0<br>WKUP                              | I/O<br>I/O<br>I/O<br>I/O<br>I         | M                     | Tristate                   | 27         | 40       | 48       |
| <b>Port B</b> |              |  |   |  |                                       |                       |                            |            |          |          |

Table 4-1. Functional port pins (continued)

| Port pin | PCR register | Alternate function <sup>1</sup>         | Function   | Peripheral  | I/O direction                       | Pad type <sup>2</sup> | RESET config. <sup>3</sup> | Pin number |          |          |
|----------|--------------|---|--|---|-------------------------------------|-----------------------|----------------------------|------------|----------|----------|
|          |              |   |  |   |                                     |                       |                            | 100 LQFP   | 144 LQFP | 176 LQFP |
| PB[0]    | PCR[16]      | AF0<br>AF1<br>AF2<br>AF3                | GPIO[16]<br>CAN0TX<br>E0UC[30]<br>LIN0TX                                     | SIUL<br>FlexCAN_0<br>eMIOS_0<br>LINFlex_0                       | I/O<br>O<br>I/O<br>O                | M                     | Tristate                   | 23         | 31       | 39       |
| PB[1]    | PCR[17]      | AF0<br>AF1<br>AF2<br>AF3<br>—<br>—<br>— | GPIO[17]<br>—<br>E0UC[31]<br>—<br>WKUP[4] <sup>(4)</sup><br>CAN0RX<br>LIN0RX | SIUL<br>—<br>eMIOS_0<br>—<br>WKUP<br>FlexCAN_0<br>LINFlex_0     | I/O<br>—<br>I/O<br>—<br>I<br>I<br>I | S                     | Tristate                   | 24         | 32       | 40       |
| PB[2]    | PCR[18]      | AF0<br>AF1<br>AF2<br>AF3                | GPIO[18]<br>LIN0TX<br>SDA<br>E0UC[30]  | SIUL<br>LINFlex_0<br>I <sup>2</sup> C_0<br>eMIOS_0              | I/O<br>O<br>I/O<br>I/O              | M                     | Tristate                   | 100        | 144      | 176      |
| PB[3]    | PCR[19]      | AF0<br>AF1<br>AF2<br>AF3<br>—<br>—      | GPIO[19]<br>E0UC[31]<br>SCL<br>—<br>WKUP[11] <sup>(4)</sup><br>LIN0RX        | SIUL<br>eMIOS_0<br>I <sup>2</sup> C_0<br>—<br>WKUP<br>LINFlex_0 | I/O<br>I/O<br>I/O<br>—<br>I<br>I    | S                     | Tristate                   | 1          | 1        | 1        |
| PB[4]    | PCR[20]      | AF0<br>AF1<br>AF2<br>AF3<br>—<br>—<br>— | —<br>—<br>—<br>—<br>ADC0_P[0]<br>ADC1_P[0]<br>GPIO[20]                       | —<br>—<br>—<br>—<br>ADC_0<br>ADC_1<br>SIUL                      | —<br>—<br>—<br>—<br>I<br>I<br>I     | I                     | Tristate                   | 50         | 72       | 88       |
| PB[5]    | PCR[21]      | AF0<br>AF1<br>AF2<br>AF3<br>—<br>—<br>— | —<br>—<br>—<br>—<br>ADC0_P[1]<br>ADC1_P[1]<br>GPIO[21]                       | —<br>—<br>—<br>—<br>ADC_0<br>ADC_1<br>SIUL                      | —<br>—<br>—<br>—<br>I<br>I<br>I     | I                     | Tristate                   | 53         | 75       | 91       |
| PB[6]    | PCR[22]      | AF0<br>AF1<br>AF2<br>AF3<br>—<br>—<br>— | —<br>—<br>—<br>—<br>ADC0_P[2]<br>ADC1_P[2]<br>GPIO[22]                       | —<br>—<br>—<br>—<br>ADC_0<br>ADC_1<br>SIUL                      | —<br>—<br>—<br>—<br>I<br>I<br>I     | I                     | Tristate                   | 54         | 76       | 92       |

Table 4-1. Functional port pins (continued)

| Port pin | PCR register | Alternate function <sup>1</sup> | Function                  | Peripheral | I/O direction | Pad type <sup>2</sup> | RESET config. <sup>3</sup> | Pin number |          |          |
|----------|--------------|---------------------------------|---------------------------|------------|---------------|-----------------------|----------------------------|------------|----------|----------|
|          |              |                                 |                           |            |               |                       |                            | 100 LQFP   | 144 LQFP | 176 LQFP |
| PB[7]    | PCR[23]      | AF0                             | —                         | —          | —             | I                     | Tristate                   | 55         | 77       | 93       |
|          |              | AF1                             | —                         | —          | —             |                       |                            |            |          |          |
|          |              | AF2                             | —                         | —          | —             |                       |                            |            |          |          |
|          |              | AF3                             | —                         | —          | —             |                       |                            |            |          |          |
|          |              | —                               | ADC0_P[3]                 | ADC_0      | I             |                       |                            |            |          |          |
|          |              | —                               | ADC1_P[3]                 | ADC_1      | I             |                       |                            |            |          |          |
|          |              | —                               | GPIO[23]                  | SIUL       | I             |                       |                            |            |          |          |
| PB[8]    | PCR[24]      | AF0                             | GPIO[24]                  | SIUL       | I             | I                     | —                          | 39         | 53       | 61       |
|          |              | AF1                             | —                         | —          | —             |                       |                            |            |          |          |
|          |              | AF2                             | —                         | —          | —             |                       |                            |            |          |          |
|          |              | AF3                             | —                         | —          | —             |                       |                            |            |          |          |
|          |              | —                               | OSC32K_XTAL <sup>7</sup>  | OSC32K     | —             |                       |                            |            |          |          |
|          |              | —                               | WKUP[25]                  | WKUP       | I             |                       |                            |            |          |          |
|          |              | —                               | ADC0_S[0]                 | ADC_0      | I             |                       |                            |            |          |          |
| —        | ADC1_S[4]    | ADC_1                           | I                         |            |               |                       |                            |            |          |          |
| PB[9]    | PCR[25]      | AF0                             | GPIO[25]                  | SIUL       | I             | I                     | —                          | 38         | 52       | 60       |
|          |              | AF1                             | —                         | —          | —             |                       |                            |            |          |          |
|          |              | AF2                             | —                         | —          | —             |                       |                            |            |          |          |
|          |              | AF3                             | —                         | —          | —             |                       |                            |            |          |          |
|          |              | —                               | OSC32K_EXTAL <sup>7</sup> | OSC32K     | —             |                       |                            |            |          |          |
|          |              | —                               | WKUP[26]                  | WKUP       | I             |                       |                            |            |          |          |
|          |              | —                               | ADC0_S[1]                 | ADC_0      | I             |                       |                            |            |          |          |
| —        | ADC1_S[5]    | ADC_1                           | I                         |            |               |                       |                            |            |          |          |
| PB[10]   | PCR[26]      | AF0                             | GPIO[26]                  | SIUL       | I/O           | J                     | Tristate                   | 40         | 54       | 62       |
|          |              | AF1                             | —                         | —          | —             |                       |                            |            |          |          |
|          |              | AF2                             | —                         | —          | —             |                       |                            |            |          |          |
|          |              | AF3                             | —                         | —          | —             |                       |                            |            |          |          |
|          |              | —                               | WKUP[8] <sup>4</sup>      | WKUP       | I             |                       |                            |            |          |          |
|          |              | —                               | ADC0_S[2]                 | ADC_0      | I             |                       |                            |            |          |          |
| —        | ADC1_S[6]    | ADC_1                           | I                         |            |               |                       |                            |            |          |          |
| PB[11]   | PCR[27]      | AF0                             | GPIO[27]                  | SIUL       | I/O           | J                     | Tristate                   | —          | —        | 97       |
|          |              | AF1                             | E0UC[3]                   | eMIOS_0    | I/O           |                       |                            |            |          |          |
|          |              | AF2                             | —                         | —          | —             |                       |                            |            |          |          |
|          |              | AF3                             | CS0_0                     | DSPI_0     | I/O           |                       |                            |            |          |          |
|          |              | —                               | ADC0_S[3]                 | ADC_0      | I             |                       |                            |            |          |          |
| PB[12]   | PCR[28]      | AF0                             | GPIO[28]                  | SIUL       | I/O           | J                     | Tristate                   | 61         | 83       | 101      |
|          |              | AF1                             | E0UC[4]                   | eMIOS_0    | I/O           |                       |                            |            |          |          |
|          |              | AF2                             | —                         | —          | —             |                       |                            |            |          |          |
|          |              | AF3                             | CS1_0                     | DSPI_0     | O             |                       |                            |            |          |          |
|          |              | —                               | ADC0_X[0]                 | ADC_0      | I             |                       |                            |            |          |          |
| PB[13]   | PCR[29]      | AF0                             | GPIO[29]                  | SIUL       | I/O           | J                     | Tristate                   | 63         | 85       | 103      |
|          |              | AF1                             | E0UC[5]                   | eMIOS_0    | I/O           |                       |                            |            |          |          |
|          |              | AF2                             | —                         | —          | —             |                       |                            |            |          |          |
|          |              | AF3                             | CS2_0                     | DSPI_0     | O             |                       |                            |            |          |          |
|          |              | —                               | ADC0_X[1]                 | ADC_0      | I             |                       |                            |            |          |          |

Table 4-1. Functional port pins (continued)

| Port pin           | PCR register | Alternate function <sup>1</sup> | Function  | Peripheral | I/O direction | Pad type <sup>2</sup> | RESET config. <sup>3</sup> | Pin number |          |          |
|--------------------|--------------|---------------------------------|-----------|------------|---------------|-----------------------|----------------------------|------------|----------|----------|
|                    |              |                                 |           |            |               |                       |                            | 100 LQFP   | 144 LQFP | 176 LQFP |
| PB[14]             | PCR[30]      | AF0                             | GPIO[30]  | SIUL       | I/O           | J                     | Tristate                   | 65         | 87       | 105      |
|                    |              | AF1                             | E0UC[6]   | eMIOS_0    | I/O           |                       |                            |            |          |          |
|                    |              | AF2                             | —         | —          | —             |                       |                            |            |          |          |
|                    |              | AF3                             | CS3_0     | DSPI_0     | O             |                       |                            |            |          |          |
|                    |              | —                               | ADC0_X[2] | ADC_0      | I             |                       |                            |            |          |          |
| PB[15]             | PCR[31]      | AF0                             | GPIO[31]  | SIUL       | I/O           | J                     | Tristate                   | 67         | 89       | 107      |
|                    |              | AF1                             | E0UC[7]   | eMIOS_0    | I/O           |                       |                            |            |          |          |
|                    |              | AF2                             | —         | —          | —             |                       |                            |            |          |          |
|                    |              | AF3                             | CS4_0     | DSPI_0     | O             |                       |                            |            |          |          |
|                    |              | —                               | ADC0_X[3] | ADC_0      | I             |                       |                            |            |          |          |
| <b>Port C</b>      |              |                                 |           |            |               |                       |                            |            |          |          |
| PC[0] <sup>8</sup> | PCR[32]      | AF0                             | GPIO[32]  | SIUL       | I/O           | M                     | Input, weak pull-up        | 87         | 126      | 154      |
|                    |              | AF1                             | —         | —          | —             |                       |                            |            |          |          |
|                    |              | AF2                             | TDI       | JTAGC      | I             |                       |                            |            |          |          |
|                    |              | AF3                             | —         | —          | —             |                       |                            |            |          |          |
| PC[1] <sup>8</sup> | PCR[33]      | AF0                             | GPIO[33]  | SIUL       | I/O           | F <sup>9</sup>        | Tristate                   | 82         | 121      | 149      |
|                    |              | AF1                             | —         | —          | —             |                       |                            |            |          |          |
|                    |              | AF2                             | TDO       | JTAGC      | O             |                       |                            |            |          |          |
|                    |              | AF3                             | —         | —          | —             |                       |                            |            |          |          |
| PC[2]              | PCR[34]      | AF0                             | GPIO[34]  | SIUL       | I/O           | M                     | Tristate                   | 78         | 117      | 145      |
|                    |              | AF1                             | SCK_1     | DSPI_1     | I/O           |                       |                            |            |          |          |
|                    |              | AF2                             | CAN4TX    | FlexCAN_4  | O             |                       |                            |            |          |          |
|                    |              | AF3                             | DEBUG[0]  | SSCM       | O             |                       |                            |            |          |          |
|                    |              | —                               | EIRQ[5]   | SIUL       | I             |                       |                            |            |          |          |
| PC[3]              | PCR[35]      | AF0                             | GPIO[35]  | SIUL       | I/O           | S                     | Tristate                   | 77         | 116      | 144      |
|                    |              | AF1                             | CS0_1     | DSPI_1     | I/O           |                       |                            |            |          |          |
|                    |              | AF2                             | MA[0]     | ADC_0      | O             |                       |                            |            |          |          |
|                    |              | AF3                             | DEBUG[1]  | SSCM       | O             |                       |                            |            |          |          |
|                    |              | —                               | EIRQ[6]   | SIUL       | I             |                       |                            |            |          |          |
|                    |              | —                               | CAN1RX    | FlexCAN_1  | I             |                       |                            |            |          |          |
| —                  | CAN4RX       | FlexCAN_4                       | I         |            |               |                       |                            |            |          |          |
| PC[4]              | PCR[36]      | AF0                             | GPIO[36]  | SIUL       | I/O           | M                     | Tristate                   | 92         | 131      | 159      |
|                    |              | AF1                             | E1UC[31]  | eMIOS_1    | I/O           |                       |                            |            |          |          |
|                    |              | AF2                             | —         | —          | —             |                       |                            |            |          |          |
|                    |              | AF3                             | DEBUG[2]  | SSCM       | O             |                       |                            |            |          |          |
|                    |              | —                               | EIRQ[18]  | SIUL       | I             |                       |                            |            |          |          |
|                    |              | —                               | SIN_1     | DSPI_1     | I             |                       |                            |            |          |          |
| —                  | CAN3RX       | FlexCAN_3                       | I         |            |               |                       |                            |            |          |          |
| PC[5]              | PCR[37]      | AF0                             | GPIO[37]  | SIUL       | I/O           | M                     | Tristate                   | 91         | 130      | 158      |
|                    |              | AF1                             | SOUT_1    | DSPI_1     | O             |                       |                            |            |          |          |
|                    |              | AF2                             | CAN3TX    | FlexCAN_3  | O             |                       |                            |            |          |          |
|                    |              | AF3                             | DEBUG[3]  | SSCM       | O             |                       |                            |            |          |          |
|                    |              | —                               | EIRQ[7]   | SIUL       | I             |                       |                            |            |          |          |

Table 4-1. Functional port pins (continued)

| Port pin | PCR register | Alternate function <sup>1</sup>         | Function   | Peripheral  | I/O direction                     | Pad type <sup>2</sup> | RESET config. <sup>3</sup> | Pin number |          |          |
|----------|--------------|---|--|---|-----------------------------------|-----------------------|----------------------------|------------|----------|----------|
|          |              |   |  |   |                                   |                       |                            | 100 LQFP   | 144 LQFP | 176 LQFP |
| PC[6]    | PCR[38]      | AF0<br>AF1<br>AF2<br>AF3                | GPIO[38]<br>LIN1TX<br>E1UC[28]<br>DEBUG[4]                                 | SIUL<br>LINFlex_1<br>eMIOS_1<br>SSCM                      | I/O<br>O<br>I/O<br>O              | S                     | Tristate                   | 25         | 36       | 44       |
| PC[7]    | PCR[39]      | AF0<br>AF1<br>AF2<br>AF3<br>—<br>—      | GPIO[39]<br>—<br>E1UC[29]<br>DEBUG[5]<br>LIN1RX<br>WKUP[12] <sup>(4)</sup> | SIUL<br>—<br>eMIOS_1<br>SSCM<br>LINFlex_1<br>WKUP         | I/O<br>—<br>I/O<br>O<br>I<br>I    | S                     | Tristate                   | 26         | 37       | 45       |
| PC[8]    | PCR[40]      | AF0<br>AF1<br>AF2<br>AF3                | GPIO[40]<br>LIN2TX<br>E0UC[3]<br>DEBUG[6]                                  | SIUL<br>LINFlex_2<br>eMIOS_0<br>SSCM                      | I/O<br>O<br>I/O<br>O              | S                     | Tristate                   | 99         | 143      | 175      |
| PC[9]    | PCR[41]      | AF0<br>AF1<br>AF2<br>AF3<br>—<br>—      | GPIO[41]<br>—<br>E0UC[7]<br>DEBUG[7]<br>WKUP[13] <sup>(4)</sup><br>LIN2RX  | SIUL<br>—<br>eMIOS_0<br>SSCM<br>WKUP<br>LINFlex_2         | I/O<br>—<br>I/O<br>O<br>I<br>I    | S                     | Tristate                   | 2          | 2        | 2        |
| PC[10]   | PCR[42]      | AF0<br>AF1<br>AF2<br>AF3                | GPIO[42]<br>CAN1TX<br>CAN4TX<br>MA[1]                                      | SIUL<br>FlexCAN_1<br>FlexCAN_4<br>ADC_0                   | I/O<br>O<br>O<br>O                | M                     | Tristate                   | 22         | 28       | 36       |
| PC[11]   | PCR[43]      | AF0<br>AF1<br>AF2<br>AF3<br>—<br>—<br>— | GPIO[43]<br>—<br>—<br>MA[2]<br>WKUP[5] <sup>(4)</sup><br>CAN1RX<br>CAN4RX  | SIUL<br>—<br>—<br>ADC_0<br>WKUP<br>FlexCAN_1<br>FlexCAN_4 | I/O<br>—<br>—<br>O<br>I<br>I<br>I | S                     | Tristate                   | 21         | 27       | 35       |
| PC[12]   | PCR[44]      | AF0<br>AF1<br>AF2<br>AF3<br>—<br>—      | GPIO[44]<br>E0UC[12]<br>—<br>—<br>EIRQ[19]<br>SIN_2                        | SIUL<br>eMIOS_0<br>—<br>—<br>SIUL<br>DSPI_2               | I/O<br>I/O<br>—<br>—<br>I<br>I    | M                     | Tristate                   | 97         | 141      | 173      |
| PC[13]   | PCR[45]      | AF0<br>AF1<br>AF2<br>AF3                | GPIO[45]<br>E0UC[13]<br>SOUT_2<br>—  | SIUL<br>eMIOS_0<br>DSPI_2<br>—                            | I/O<br>I/O<br>O<br>—              | S                     | Tristate                   | 98         | 142      | 174      |
| PC[14]   | PCR[46]      | AF0<br>AF1<br>AF2<br>AF3<br>—           | GPIO[46]<br>E0UC[14]<br>SCK_2<br>—<br>EIRQ[8]                              | SIUL<br>eMIOS_0<br>DSPI_2<br>—<br>SIUL                    | I/O<br>I/O<br>I/O<br>—<br>I       | S                     | Tristate                   | 3          | 3        | 3        |

Table 4-1. Functional port pins (continued)

| Port pin      | PCR register | Alternate function <sup>1</sup> | Function  | Peripheral | I/O direction | Pad type <sup>2</sup> | RESET config. <sup>3</sup> | Pin number |          |          |
|---------------|--------------|---------------------------------|-----------|------------|---------------|-----------------------|----------------------------|------------|----------|----------|
|               |              |                                 |           |            |               |                       |                            | 100 LQFP   | 144 LQFP | 176 LQFP |
| PC[15]        | PCR[47]      | AF0                             | GPIO[47]  | SIUL       | I/O           | M                     | Tristate                   | 4          | 4        | 4        |
|               |              | AF1                             | E0UC[15]  | eMIOS_0    | I/O           |                       |                            |            |          |          |
|               |              | AF2                             | CS0_2     | DSPI_2     | I/O           |                       |                            |            |          |          |
|               |              | AF3                             | —         | —          | —             |                       |                            |            |          |          |
|               |              | —                               | EIRQ[20]  | SIUL       | I             |                       |                            |            |          |          |
| <b>Port D</b> |              |                                 |           |            |               |                       |                            |            |          |          |
| PD[0]         | PCR[48]      | AF0                             | GPIO[48]  | SIUL       | I             | I                     | Tristate                   | 41         | 63       | 77       |
|               |              | AF1                             | —         | —          | —             |                       |                            |            |          |          |
|               |              | AF2                             | —         | —          | —             |                       |                            |            |          |          |
|               |              | AF3                             | —         | —          | —             |                       |                            |            |          |          |
|               |              | —                               | WKUP[27]  | WKUP       | I             |                       |                            |            |          |          |
|               |              | —                               | ADC0_P[4] | ADC_0      | I             |                       |                            |            |          |          |
| —             | ADC1_P[4]    | ADC_1                           | I         |            |               |                       |                            |            |          |          |
| PD[1]         | PCR[49]      | AF0                             | GPIO[49]  | SIUL       | I             | I                     | Tristate                   | 42         | 64       | 78       |
|               |              | AF1                             | —         | —          | —             |                       |                            |            |          |          |
|               |              | AF2                             | —         | —          | —             |                       |                            |            |          |          |
|               |              | AF3                             | —         | —          | —             |                       |                            |            |          |          |
|               |              | —                               | WKUP[28]  | WKUP       | I             |                       |                            |            |          |          |
|               |              | —                               | ADC0_P[5] | ADC_0      | I             |                       |                            |            |          |          |
| —             | ADC1_P[5]    | ADC_1                           | I         |            |               |                       |                            |            |          |          |
| PD[2]         | PCR[50]      | AF0                             | GPIO[50]  | SIUL       | I             | I                     | Tristate                   | 43         | 65       | 79       |
|               |              | AF1                             | —         | —          | —             |                       |                            |            |          |          |
|               |              | AF2                             | —         | —          | —             |                       |                            |            |          |          |
|               |              | AF3                             | —         | —          | —             |                       |                            |            |          |          |
|               |              | —                               | ADC0_P[6] | ADC_0      | I             |                       |                            |            |          |          |
|               |              | —                               | ADC1_P[6] | ADC_1      | I             |                       |                            |            |          |          |
| PD[3]         | PCR[51]      | AF0                             | GPIO[51]  | SIUL       | I             | I                     | Tristate                   | 44         | 66       | 80       |
|               |              | AF1                             | —         | —          | —             |                       |                            |            |          |          |
|               |              | AF2                             | —         | —          | —             |                       |                            |            |          |          |
|               |              | AF3                             | —         | —          | —             |                       |                            |            |          |          |
|               |              | —                               | ADC0_P[7] | ADC_0      | I             |                       |                            |            |          |          |
|               |              | —                               | ADC1_P[7] | ADC_1      | I             |                       |                            |            |          |          |
| PD[4]         | PCR[52]      | AF0                             | GPIO[52]  | SIUL       | I             | I                     | Tristate                   | 45         | 67       | 81       |
|               |              | AF1                             | —         | —          | —             |                       |                            |            |          |          |
|               |              | AF2                             | —         | —          | —             |                       |                            |            |          |          |
|               |              | AF3                             | —         | —          | —             |                       |                            |            |          |          |
|               |              | —                               | ADC0_P[8] | ADC_0      | I             |                       |                            |            |          |          |
|               |              | —                               | ADC1_P[8] | ADC_1      | I             |                       |                            |            |          |          |
| PD[5]         | PCR[53]      | AF0                             | GPIO[53]  | SIUL       | I             | I                     | Tristate                   | 46         | 68       | 82       |
|               |              | AF1                             | —         | —          | —             |                       |                            |            |          |          |
|               |              | AF2                             | —         | —          | —             |                       |                            |            |          |          |
|               |              | AF3                             | —         | —          | —             |                       |                            |            |          |          |
|               |              | —                               | ADC0_P[9] | ADC_0      | I             |                       |                            |            |          |          |
|               |              | —                               | ADC1_P[9] | ADC_1      | I             |                       |                            |            |          |          |

Table 4-1. Functional port pins (continued)

| Port pin | PCR register | Alternate function <sup>1</sup> | Function   | Peripheral | I/O direction | Pad type <sup>2</sup> | RESET config. <sup>3</sup> | Pin number |          |          |
|----------|--------------|---------------------------------|------------|------------|---------------|-----------------------|----------------------------|------------|----------|----------|
|          |              |                                 |            |            |               |                       |                            | 100 LQFP   | 144 LQFP | 176 LQFP |
| PD[6]    | PCR[54]      | AF0                             | GPIO[54]   | SIUL       | I             | I                     | Tristate                   | 47         | 69       | 83       |
|          |              | AF1                             | —          | —          | —             |                       |                            |            |          |          |
|          |              | AF2                             | —          | —          | —             |                       |                            |            |          |          |
|          |              | AF3                             | —          | —          | —             |                       |                            |            |          |          |
|          |              | —                               | ADC0_P[10] | ADC_0      | I             |                       |                            |            |          |          |
|          |              | —                               | ADC1_P[10] | ADC_1      | I             |                       |                            |            |          |          |
| PD[7]    | PCR[55]      | AF0                             | GPIO[55]   | SIUL       | I             | I                     | Tristate                   | 48         | 70       | 84       |
|          |              | AF1                             | —          | —          | —             |                       |                            |            |          |          |
|          |              | AF2                             | —          | —          | —             |                       |                            |            |          |          |
|          |              | AF3                             | —          | —          | —             |                       |                            |            |          |          |
|          |              | —                               | ADC0_P[11] | ADC_0      | I             |                       |                            |            |          |          |
|          |              | —                               | ADC1_P[11] | ADC_1      | I             |                       |                            |            |          |          |
| PD[8]    | PCR[56]      | AF0                             | GPIO[56]   | SIUL       | I             | I                     | Tristate                   | 49         | 71       | 87       |
|          |              | AF1                             | —          | —          | —             |                       |                            |            |          |          |
|          |              | AF2                             | —          | —          | —             |                       |                            |            |          |          |
|          |              | AF3                             | —          | —          | —             |                       |                            |            |          |          |
|          |              | —                               | ADC0_P[12] | ADC_0      | I             |                       |                            |            |          |          |
|          |              | —                               | ADC1_P[12] | ADC_1      | I             |                       |                            |            |          |          |
| PD[9]    | PCR[57]      | AF0                             | GPIO[57]   | SIUL       | I             | I                     | Tristate                   | 56         | 78       | 94       |
|          |              | AF1                             | —          | —          | —             |                       |                            |            |          |          |
|          |              | AF2                             | —          | —          | —             |                       |                            |            |          |          |
|          |              | AF3                             | —          | —          | —             |                       |                            |            |          |          |
|          |              | —                               | ADC0_P[13] | ADC_0      | I             |                       |                            |            |          |          |
|          |              | —                               | ADC1_P[13] | ADC_1      | I             |                       |                            |            |          |          |
| PD[10]   | PCR[58]      | AF0                             | GPIO[58]   | SIUL       | I             | I                     | Tristate                   | 57         | 79       | 95       |
|          |              | AF1                             | —          | —          | —             |                       |                            |            |          |          |
|          |              | AF2                             | —          | —          | —             |                       |                            |            |          |          |
|          |              | AF3                             | —          | —          | —             |                       |                            |            |          |          |
|          |              | —                               | ADC0_P[14] | ADC_0      | I             |                       |                            |            |          |          |
|          |              | —                               | ADC1_P[14] | ADC_1      | I             |                       |                            |            |          |          |
| PD[11]   | PCR[59]      | AF0                             | GPIO[59]   | SIUL       | I             | I                     | Tristate                   | 58         | 80       | 96       |
|          |              | AF1                             | —          | —          | —             |                       |                            |            |          |          |
|          |              | AF2                             | —          | —          | —             |                       |                            |            |          |          |
|          |              | AF3                             | —          | —          | —             |                       |                            |            |          |          |
|          |              | —                               | ADC0_P[15] | ADC_0      | I             |                       |                            |            |          |          |
|          |              | —                               | ADC1_P[15] | ADC_1      | I             |                       |                            |            |          |          |
| PD[12]   | PCR[60]      | AF0                             | GPIO[60]   | SIUL       | I/O           | J                     | Tristate                   | —          | —        | 100      |
|          |              | AF1                             | CS5_0      | DSPI_0     | O             |                       |                            |            |          |          |
|          |              | AF2                             | E0UC[24]   | eMIOS_0    | I/O           |                       |                            |            |          |          |
|          |              | AF3                             | —          | —          | —             |                       |                            |            |          |          |
|          |              | —                               | ADC0_S[4]  | ADC_0      | I             |                       |                            |            |          |          |
| PD[13]   | PCR[61]      | AF0                             | GPIO[61]   | SIUL       | I/O           | J                     | Tristate                   | 62         | 84       | 102      |
|          |              | AF1                             | CS0_1      | DSPI_1     | I/O           |                       |                            |            |          |          |
|          |              | AF2                             | E0UC[25]   | eMIOS_0    | I/O           |                       |                            |            |          |          |
|          |              | AF3                             | —          | —          | —             |                       |                            |            |          |          |
|          |              | —                               | ADC0_S[5]  | ADC_0      | I             |                       |                            |            |          |          |

Table 4-1. Functional port pins (continued)

| Port pin      | PCR register | Alternate function <sup>1</sup> | Function               | Peripheral | I/O direction | Pad type <sup>2</sup> | RESET config. <sup>3</sup> | Pin number |          |          |
|---------------|--------------|---------------------------------|------------------------|------------|---------------|-----------------------|----------------------------|------------|----------|----------|
|               |              |                                 |                        |            |               |                       |                            | 100 LQFP   | 144 LQFP | 176 LQFP |
| PD[14]        | PCR[62]      | AF0                             | GPIO[62]               | SIUL       | I/O           | J                     | Tristate                   | 64         | 86       | 104      |
|               |              | AF1                             | CS1_1                  | DSPI_1     | O             |                       |                            |            |          |          |
|               |              | AF2                             | E0UC[26]               | eMIOS_0    | I/O           |                       |                            |            |          |          |
|               |              | AF3                             | —                      | —          | —             |                       |                            |            |          |          |
|               |              | —                               | ADC0_S[6]              | ADC_0      | I             |                       |                            |            |          |          |
| PD[15]        | PCR[63]      | AF0                             | GPIO[63]               | SIUL       | I/O           | J                     | Tristate                   | 66         | 88       | 106      |
|               |              | AF1                             | CS2_1                  | DSPI_1     | O             |                       |                            |            |          |          |
|               |              | AF2                             | E0UC[27]               | eMIOS_0    | I/O           |                       |                            |            |          |          |
|               |              | AF3                             | —                      | —          | —             |                       |                            |            |          |          |
|               |              | —                               | ADC0_S[7]              | ADC_0      | I             |                       |                            |            |          |          |
| <b>Port E</b> |              |                                 |                        |            |               |                       |                            |            |          |          |
| PE[0]         | PCR[64]      | AF0                             | GPIO[64]               | SIUL       | I/O           | S                     | Tristate                   | 6          | 10       | 18       |
|               |              | AF1                             | E0UC[16]               | eMIOS_0    | I/O           |                       |                            |            |          |          |
|               |              | AF2                             | —                      | —          | —             |                       |                            |            |          |          |
|               |              | AF3                             | —                      | —          | —             |                       |                            |            |          |          |
|               |              | —                               | WKUP[6] <sup>(4)</sup> | WKUP       | I             |                       |                            |            |          |          |
| —             | CAN5RX       | FlexCAN_5                       | I                      |            |               |                       |                            |            |          |          |
| PE[1]         | PCR[65]      | AF0                             | GPIO[65]               | SIUL       | I/O           | M                     | Tristate                   | 8          | 12       | 20       |
|               |              | AF1                             | E0UC[17]               | eMIOS_0    | I/O           |                       |                            |            |          |          |
|               |              | AF2                             | CAN5TX                 | FlexCAN_5  | O             |                       |                            |            |          |          |
|               |              | AF3                             | —                      | —          | —             |                       |                            |            |          |          |
| PE[2]         | PCR[66]      | AF0                             | GPIO[66]               | SIUL       | I/O           | M                     | Tristate                   | 89         | 128      | 156      |
|               |              | AF1                             | E0UC[18]               | eMIOS_0    | I/O           |                       |                            |            |          |          |
|               |              | AF2                             | —                      | —          | —             |                       |                            |            |          |          |
|               |              | AF3                             | —                      | —          | —             |                       |                            |            |          |          |
|               |              | —                               | EIRQ[21]               | SIUL       | I             |                       |                            |            |          |          |
| —             | SIN_1        | DSPI_1                          | I                      |            |               |                       |                            |            |          |          |
| PE[3]         | PCR[67]      | AF0                             | GPIO[67]               | SIUL       | I/O           | M                     | Tristate                   | 90         | 129      | 157      |
|               |              | AF1                             | E0UC[19]               | eMIOS_0    | I/O           |                       |                            |            |          |          |
|               |              | AF2                             | SOUT_1                 | DSPI_1     | O             |                       |                            |            |          |          |
|               |              | AF3                             | —                      | —          | —             |                       |                            |            |          |          |
| PE[4]         | PCR[68]      | AF0                             | GPIO[68]               | SIUL       | I/O           | M                     | Tristate                   | 93         | 132      | 160      |
|               |              | AF1                             | E0UC[20]               | eMIOS_0    | I/O           |                       |                            |            |          |          |
|               |              | AF2                             | SCK_1                  | DSPI_1     | I/O           |                       |                            |            |          |          |
|               |              | AF3                             | —                      | —          | —             |                       |                            |            |          |          |
|               |              | —                               | EIRQ[9]                | SIUL       | I             |                       |                            |            |          |          |
| PE[5]         | PCR[69]      | AF0                             | GPIO[69]               | SIUL       | I/O           | M                     | Tristate                   | 94         | 133      | 161      |
|               |              | AF1                             | E0UC[21]               | eMIOS_0    | I/O           |                       |                            |            |          |          |
|               |              | AF2                             | CS0_1                  | DSPI_1     | I/O           |                       |                            |            |          |          |
|               |              | AF3                             | MA[2]                  | ADC_0      | O             |                       |                            |            |          |          |
| PE[6]         | PCR[70]      | AF0                             | GPIO[70]               | SIUL       | I/O           | M                     | Tristate                   | 95         | 139      | 167      |
|               |              | AF1                             | E0UC[22]               | eMIOS_0    | I/O           |                       |                            |            |          |          |
|               |              | AF2                             | CS3_0                  | DSPI_0     | O             |                       |                            |            |          |          |
|               |              | AF3                             | MA[1]                  | ADC_0      | O             |                       |                            |            |          |          |
|               |              | —                               | EIRQ[22]               | SIUL       | I             |                       |                            |            |          |          |



Table 4-1. Functional port pins (continued)

| Port pin | PCR register | Alternate function <sup>1</sup> | Function                | Peripheral | I/O direction | Pad type <sup>2</sup> | RESET config. <sup>3</sup> | Pin number |          |          |
|----------|--------------|---------------------------------|-------------------------|------------|---------------|-----------------------|----------------------------|------------|----------|----------|
|          |              |                                 |                         |            |               |                       |                            | 100 LQFP   | 144 LQFP | 176 LQFP |
| PE[7]    | PCR[71]      | AF0                             | GPIO[71]                | SIUL       | I/O           | M                     | Tristate                   | 96         | 140      | 168      |
|          |              | AF1                             | E0UC[23]                | eMIOS_0    | I/O           |                       |                            |            |          |          |
|          |              | AF2                             | CS2_0                   | DSPI_0     | O             |                       |                            |            |          |          |
|          |              | AF3                             | MA[0]                   | ADC_0      | O             |                       |                            |            |          |          |
|          |              | —                               | EIRQ[23]                | SIUL       | I             |                       |                            |            |          |          |
| PE[8]    | PCR[72]      | AF0                             | GPIO[72]                | SIUL       | I/O           | M                     | Tristate                   | 9          | 13       | 21       |
|          |              | AF1                             | CAN2TX                  | FlexCAN_2  | O             |                       |                            |            |          |          |
|          |              | AF2                             | E0UC[22]                | eMIOS_0    | I/O           |                       |                            |            |          |          |
|          |              | AF3                             | CAN3TX                  | FlexCAN_3  | O             |                       |                            |            |          |          |
| PE[9]    | PCR[73]      | AF0                             | GPIO[73]                | SIUL       | I/O           | S                     | Tristate                   | 10         | 14       | 22       |
|          |              | AF1                             | —                       | —          | —             |                       |                            |            |          |          |
|          |              | AF2                             | E0UC[23]                | eMIOS_0    | I/O           |                       |                            |            |          |          |
|          |              | AF3                             | —                       | —          | —             |                       |                            |            |          |          |
|          |              | —                               | WKUP[7] <sup>(4)</sup>  | WKUP       | I             |                       |                            |            |          |          |
|          |              | —                               | CAN2RX                  | FlexCAN_2  | I             |                       |                            |            |          |          |
| —        | CAN3RX       | FlexCAN_3                       | I                       |            |               |                       |                            |            |          |          |
| PE[10]   | PCR[74]      | AF0                             | GPIO[74]                | SIUL       | I/O           | S                     | Tristate                   | 11         | 15       | 23       |
|          |              | AF1                             | LIN3TX                  | LINFlex_3  | O             |                       |                            |            |          |          |
|          |              | AF2                             | CS3_1                   | DSPI_1     | O             |                       |                            |            |          |          |
|          |              | AF3                             | E1UC[30]                | eMIOS_1    | I/O           |                       |                            |            |          |          |
|          |              | —                               | EIRQ[10]                | SIUL       | I             |                       |                            |            |          |          |
| PE[11]   | PCR[75]      | AF0                             | GPIO[75]                | SIUL       | I/O           | S                     | Tristate                   | 13         | 17       | 25       |
|          |              | AF1                             | E0UC[24]                | eMIOS_0    | I/O           |                       |                            |            |          |          |
|          |              | AF2                             | CS4_1                   | DSPI_1     | O             |                       |                            |            |          |          |
|          |              | AF3                             | —                       | —          | —             |                       |                            |            |          |          |
|          |              | —                               | LIN3RX                  | LINFlex_3  | I             |                       |                            |            |          |          |
|          |              | —                               | WKUP[14] <sup>(4)</sup> | WKUP       | I             |                       |                            |            |          |          |
| PE[12]   | PCR[76]      | AF0                             | GPIO[76]                | SIUL       | I/O           | J                     | Tristate                   | 76         | 109      | 133      |
|          |              | AF1                             | —                       | —          | —             |                       |                            |            |          |          |
|          |              | AF2                             | E1UC[19] <sup>10</sup>  | eMIOS_1    | I/O           |                       |                            |            |          |          |
|          |              | AF3                             | —                       | —          | —             |                       |                            |            |          |          |
|          |              | —                               | EIRQ[11]                | SIUL       | I             |                       |                            |            |          |          |
|          |              | —                               | SIN_2                   | DSPI_2     | I             |                       |                            |            |          |          |
| —        | ADC1_S[7]    | ADC_1                           | I                       |            |               |                       |                            |            |          |          |
| PE[13]   | PCR[77]      | AF0                             | GPIO[77]                | SIUL       | I/O           | S                     | Tristate                   | —          | 103      | 127      |
|          |              | AF1                             | SOUT_2                  | DSPI_2     | O             |                       |                            |            |          |          |
|          |              | AF2                             | E1UC[20]                | eMIOS_1    | I/O           |                       |                            |            |          |          |
|          |              | AF3                             | —                       | —          | —             |                       |                            |            |          |          |
| PE[14]   | PCR[78]      | AF0                             | GPIO[78]                | SIUL       | I/O           | S                     | Tristate                   | —          | 112      | 136      |
|          |              | AF1                             | SCK_2                   | DSPI_2     | I/O           |                       |                            |            |          |          |
|          |              | AF2                             | E1UC[21]                | eMIOS_1    | I/O           |                       |                            |            |          |          |
|          |              | AF3                             | —                       | —          | —             |                       |                            |            |          |          |
|          |              | —                               | EIRQ[12]                | SIUL       | I             |                       |                            |            |          |          |

Table 4-1. Functional port pins (continued)

| Port pin      | PCR register | Alternate function <sup>1</sup> | Function   | Peripheral                              | I/O direction             | Pad type <sup>2</sup> | RESET config. <sup>3</sup> | Pin number |          |          |
|---------------|--------------|---------------------------------|--|---|---------------------------|-----------------------|----------------------------|------------|----------|----------|
|               |              |                                 |  |   |                           |                       |                            | 100 LQFP   | 144 LQFP | 176 LQFP |
| PE[15]        | PCR[79]      | AF0<br>AF1<br>AF2<br>AF3        | GPIO[79]<br>CS0_2<br>E1UC[22]<br>—               | SIUL<br>DSPI_2<br>eMIOS_1<br>—          | I/O<br>I/O<br>I/O<br>—    | M                     | Tristate                   | —          | 113      | 137      |
| <b>Port F</b> |              |                                 |  |   |                           |                       |                            |            |          |          |
| PF[0]         | PCR[80]      | AF0<br>AF1<br>AF2<br>AF3<br>—   | GPIO[80]<br>E0UC[10]<br>CS3_1<br>—<br>ADC0_S[8]  | SIUL<br>eMIOS_0<br>DSPI_1<br>—<br>ADC_0 | I/O<br>I/O<br>O<br>—<br>I | J                     | Tristate                   | —          | 55       | 63       |
| PF[1]         | PCR[81]      | AF0<br>AF1<br>AF2<br>AF3<br>—   | GPIO[81]<br>E0UC[11]<br>CS4_1<br>—<br>ADC0_S[9]  | SIUL<br>eMIOS_0<br>DSPI_1<br>—<br>ADC_0 | I/O<br>I/O<br>O<br>—<br>I | J                     | Tristate                   | —          | 56       | 64       |
| PF[2]         | PCR[82]      | AF0<br>AF1<br>AF2<br>AF3<br>—   | GPIO[82]<br>E0UC[12]<br>CS0_2<br>—<br>ADC0_S[10] | SIUL<br>eMIOS_0<br>DSPI_2<br>—<br>ADC_0 | I/O<br>I/O<br>O<br>—<br>I | J                     | Tristate                   | —          | 57       | 65       |
| PF[3]         | PCR[83]      | AF0<br>AF1<br>AF2<br>AF3<br>—   | GPIO[83]<br>E0UC[13]<br>CS1_2<br>—<br>ADC0_S[11] | SIUL<br>eMIOS_0<br>DSPI_2<br>—<br>ADC_0 | I/O<br>I/O<br>O<br>—<br>I | J                     | Tristate                   | —          | 58       | 66       |
| PF[4]         | PCR[84]      | AF0<br>AF1<br>AF2<br>AF3<br>—   | GPIO[84]<br>E0UC[14]<br>CS2_2<br>—<br>ADC0_S[12] | SIUL<br>eMIOS_0<br>DSPI_2<br>—<br>ADC_0 | I/O<br>I/O<br>O<br>—<br>I | J                     | Tristate                   | —          | 59       | 67       |
| PF[5]         | PCR[85]      | AF0<br>AF1<br>AF2<br>AF3<br>—   | GPIO[85]<br>E0UC[22]<br>CS3_2<br>—<br>ADC0_S[13] | SIUL<br>eMIOS_0<br>DSPI_2<br>—<br>ADC_0 | I/O<br>I/O<br>O<br>—<br>I | J                     | Tristate                   | —          | 60       | 68       |
| PF[6]         | PCR[86]      | AF0<br>AF1<br>AF2<br>AF3<br>—   | GPIO[86]<br>E0UC[23]<br>CS1_1<br>—<br>ADC0_S[14] | SIUL<br>eMIOS_0<br>DSPI_1<br>—<br>ADC_0 | I/O<br>I/O<br>O<br>—<br>I | J                     | Tristate                   | —          | 61       | 69       |
| PF[7]         | PCR[87]      | AF0<br>AF1<br>AF2<br>AF3<br>—   | GPIO[87]<br>—<br>CS2_1<br>—<br>ADC0_S[15]        | SIUL<br>—<br>DSPI_1<br>—<br>ADC_0       | I/O<br>—<br>O<br>—<br>I   | J                     | Tristate                   | —          | 62       | 70       |

Table 4-1. Functional port pins (continued)

| Port pin      | PCR register | Alternate function <sup>1</sup> | Function                | Peripheral | I/O direction | Pad type <sup>2</sup> | RESET config. <sup>3</sup> | Pin number |          |          |
|---------------|--------------|---------------------------------|-------------------------|------------|---------------|-----------------------|----------------------------|------------|----------|----------|
|               |              |                                 |                         |            |               |                       |                            | 100 LQFP   | 144 LQFP | 176 LQFP |
| PF[8]         | PCR[88]      | AF0                             | GPIO[88]                | SIUL       | I/O           | M                     | Tristate                   | —          | 34       | 42       |
|               |              | AF1                             | CAN3TX                  | FlexCAN_3  | O             |                       |                            |            |          |          |
|               |              | AF2                             | CS4_0                   | DSPI_0     | O             |                       |                            |            |          |          |
|               |              | AF3                             | CAN2TX                  | FlexCAN_2  | O             |                       |                            |            |          |          |
| PF[9]         | PCR[89]      | AF0                             | GPIO[89]                | SIUL       | I/O           | S                     | Tristate                   | —          | 33       | 41       |
|               |              | AF1                             | E1UC[1]                 | eMIOS_1    | I/O           |                       |                            |            |          |          |
|               |              | AF2                             | CS5_0                   | DSPI_0     | O             |                       |                            |            |          |          |
|               |              | AF3                             | —                       | —          | —             |                       |                            |            |          |          |
|               |              | —                               | WKUP[22] <sup>(4)</sup> | WKUP       | I             |                       |                            |            |          |          |
|               |              | —                               | CAN2RX                  | FlexCAN_2  | I             |                       |                            |            |          |          |
| —             | CAN3RX       | FlexCAN_3                       | I                       |            |               |                       |                            |            |          |          |
| PF[10]        | PCR[90]      | AF0                             | GPIO[90]                | SIUL       | I/O           | M                     | Tristate                   | —          | 38       | 46       |
|               |              | AF1                             | CS1_0                   | DSPI_0     | O             |                       |                            |            |          |          |
|               |              | AF2                             | LIN4TX                  | LINFlex_4  | O             |                       |                            |            |          |          |
|               |              | AF3                             | E1UC[2]                 | eMIOS_1    | I/O           |                       |                            |            |          |          |
| PF[11]        | PCR[91]      | AF0                             | GPIO[91]                | SIUL       | I/O           | S                     | Tristate                   | —          | 39       | 47       |
|               |              | AF1                             | CS2_0                   | DSPI_0     | O             |                       |                            |            |          |          |
|               |              | AF2                             | E1UC[3]                 | eMIOS_1    | I/O           |                       |                            |            |          |          |
|               |              | AF3                             | —                       | —          | —             |                       |                            |            |          |          |
|               |              | —                               | WKUP[15] <sup>(4)</sup> | WKUP       | I             |                       |                            |            |          |          |
|               |              | —                               | LIN4RX                  | LINFlex_4  | I             |                       |                            |            |          |          |
| PF[12]        | PCR[92]      | AF0                             | GPIO[92]                | SIUL       | I/O           | M                     | Tristate                   | —          | 35       | 43       |
|               |              | AF1                             | E1UC[25]                | eMIOS_1    | I/O           |                       |                            |            |          |          |
|               |              | AF2                             | LIN5TX                  | LINFlex_5  | O             |                       |                            |            |          |          |
|               |              | AF3                             | —                       | —          | —             |                       |                            |            |          |          |
| PF[13]        | PCR[93]      | AF0                             | GPIO[93]                | SIUL       | I/O           | S                     | Tristate                   | —          | 41       | 49       |
|               |              | AF1                             | E1UC[26]                | eMIOS_1    | I/O           |                       |                            |            |          |          |
|               |              | AF2                             | —                       | —          | —             |                       |                            |            |          |          |
|               |              | AF3                             | —                       | —          | —             |                       |                            |            |          |          |
|               |              | —                               | WKUP[16] <sup>(4)</sup> | WKUP       | I             |                       |                            |            |          |          |
|               |              | —                               | LIN5RX                  | LINFlex_5  | I             |                       |                            |            |          |          |
| PF[14]        | PCR[94]      | AF0                             | GPIO[94]                | SIUL       | I/O           | M                     | Tristate                   | —          | 102      | 126      |
|               |              | AF1                             | CAN4TX                  | FlexCAN_4  | O             |                       |                            |            |          |          |
|               |              | AF2                             | E1UC[27]                | eMIOS_1    | I/O           |                       |                            |            |          |          |
|               |              | AF3                             | CAN1TX                  | FlexCAN_1  | O             |                       |                            |            |          |          |
| PF[15]        | PCR[95]      | AF0                             | GPIO[95]                | SIUL       | I/O           | S                     | Tristate                   | —          | 101      | 125      |
|               |              | AF1                             | E1UC[4]                 | eMIOS_1    | I/O           |                       |                            |            |          |          |
|               |              | AF2                             | —                       | —          | —             |                       |                            |            |          |          |
|               |              | AF3                             | —                       | —          | —             |                       |                            |            |          |          |
|               |              | —                               | EIRQ[13]                | SIUL       | I             |                       |                            |            |          |          |
|               |              | —                               | CAN1RX                  | FlexCAN_1  | I             |                       |                            |            |          |          |
| —             | CAN4RX       | FlexCAN_4                       | I                       |            |               |                       |                            |            |          |          |
| <b>Port G</b> |              |                                 |                         |            |               |                       |                            |            |          |          |

Table 4-1. Functional port pins (continued)

| Port pin | PCR register | Alternate function <sup>1</sup>    | Function  | Peripheral   | I/O direction                    | Pad type <sup>2</sup> | RESET config. <sup>3</sup> | Pin number |          |          |
|----------|--------------|------------------------------------|---|--|----------------------------------|-----------------------|----------------------------|------------|----------|----------|
|          |              |                                    |   |  |                                  |                       |                            | 100 LQFP   | 144 LQFP | 176 LQFP |
| PG[0]    | PCR[96]      | AF0<br>AF1<br>AF2<br>AF3           | GPIO[96]<br>CAN5TX<br>E1UC[23]<br>—   | SIUL<br>FlexCAN_5<br>eMIOS_1<br>—                    | I/O<br>O<br>I/O<br>—             | M                     | Tristate                   | —          | 98       | 122      |
| PG[1]    | PCR[97]      | AF0<br>AF1<br>AF2<br>AF3<br>—<br>— | GPIO[97]<br>—<br>E1UC[24]<br>—<br>EIRQ[14]<br>CAN5RX                        | SIUL<br>—<br>eMIOS_1<br>—<br>SIUL<br>FlexCAN_5       | I/O<br>—<br>I/O<br>—<br>I<br>I   | S                     | Tristate                   | —          | 97       | 121      |
| PG[2]    | PCR[98]      | AF0<br>AF1<br>AF2<br>AF3           | GPIO[98]<br>E1UC[11]<br>SOUT_3<br>—   | SIUL<br>eMIOS_1<br>DSPI_3<br>—                       | I/O<br>I/O<br>O<br>—             | M                     | Tristate                   | —          | 8        | 16       |
| PG[3]    | PCR[99]      | AF0<br>AF1<br>AF2<br>AF3<br>—      | GPIO[99]<br>E1UC[12]<br>CS0_3<br>—<br>WKUP[17] <sup>(4)</sup>               | SIUL<br>eMIOS_1<br>DSPI_3<br>—<br>WKUP               | I/O<br>I/O<br>O<br>—<br>I        | S                     | Tristate                   | —          | 7        | 15       |
| PG[4]    | PCR[100]     | AF0<br>AF1<br>AF2<br>AF3           | GPIO[100]<br>E1UC[13]<br>SCK_3<br>—   | SIUL<br>eMIOS_1<br>DSPI_3<br>—                       | I/O<br>I/O<br>I/O<br>—           | M                     | Tristate                   | —          | 6        | 14       |
| PG[5]    | PCR[101]     | AF0<br>AF1<br>AF2<br>AF3<br>—<br>— | GPIO[101]<br>E1UC[14]<br>—<br>—<br>WKUP[18] <sup>(4)</sup><br>SIN_3         | SIUL<br>eMIOS_1<br>—<br>—<br>WKUP<br>DSPI_3          | I/O<br>I/O<br>—<br>—<br>I<br>I   | S                     | Tristate                   | —          | 5        | 13       |
| PG[6]    | PCR[102]     | AF0<br>AF1<br>AF2<br>AF3           | GPIO[102]<br>E1UC[15]<br>LIN6TX<br>—  | SIUL<br>eMIOS_1<br>LINFlex_6<br>—                    | I/O<br>I/O<br>O<br>—             | M                     | Tristate                   | —          | 30       | 38       |
| PG[7]    | PCR[103]     | AF0<br>AF1<br>AF2<br>AF3<br>—<br>— | GPIO[103]<br>E1UC[16]<br>E1UC[30]<br>—<br>WKUP[20] <sup>(4)</sup><br>LIN6RX | SIUL<br>eMIOS_1<br>eMIOS_1<br>—<br>WKUP<br>LINFlex_6 | I/O<br>I/O<br>I/O<br>—<br>I<br>I | S                     | Tristate                   | —          | 29       | 37       |
| PG[8]    | PCR[104]     | AF0<br>AF1<br>AF2<br>AF3<br>—      | GPIO[104]<br>E1UC[17]<br>LIN7TX<br>CS0_2<br>EIRQ[15]                        | SIUL<br>eMIOS_1<br>LINFlex_7<br>DSPI_2<br>SIUL       | I/O<br>I/O<br>O<br>I/O<br>I      | S                     | Tristate                   | —          | 26       | 34       |

Table 4-1. Functional port pins (continued)

| Port pin      | PCR register | Alternate function <sup>1</sup>    | Function   | Peripheral  | I/O direction                    | Pad type <sup>2</sup> | RESET config. <sup>3</sup> | Pin number |          |          |
|---------------|--------------|------------------------------------|--|---|----------------------------------|-----------------------|----------------------------|------------|----------|----------|
|               |              |                                    |  |   |                                  |                       |                            | 100 LQFP   | 144 LQFP | 176 LQFP |
| PG[9]         | PCR[105]     | AF0<br>AF1<br>AF2<br>AF3<br>—<br>— | GPIO[105]<br>E1UC[18]<br>—<br>SCK_2<br>WKUP[21] <sup>4</sup><br>LIN7RX | SIUL<br>eMIOS_1<br>—<br>DSPI_2<br>WKUP<br>LINFlex_7 | I/O<br>I/O<br>—<br>I/O<br>I<br>I | S                     | Tristate                   | —          | 25       | 33       |
| PG[10]        | PCR[106]     | AF0<br>AF1<br>AF2<br>AF3<br>—      | GPIO[106]<br>E0UC[24]<br>E1UC[31]<br>—<br>SIN_4                        | SIUL<br>eMIOS_0<br>eMIOS_1<br>—<br>DSPI_4           | I/O<br>I/O<br>I/O<br>—<br>I      | S                     | Tristate                   | —          | 114      | 138      |
| PG[11]        | PCR[107]     | AF0<br>AF1<br>AF2<br>AF3           | GPIO[107]<br>E0UC[25]<br>CS0_4<br>—                                    | SIUL<br>eMIOS_0<br>DSPI_4<br>—                      | I/O<br>I/O<br>O<br>—             | M                     | Tristate                   | —          | 115      | 139      |
| PG[12]        | PCR[108]     | AF0<br>AF1<br>AF2<br>AF3           | GPIO[108]<br>E0UC[26]<br>SOUT_4<br>—                                   | SIUL<br>eMIOS_0<br>DSPI_4<br>—                      | I/O<br>I/O<br>O<br>—             | M                     | Tristate                   | —          | 92       | 116      |
| PG[13]        | PCR[109]     | AF0<br>AF1<br>AF2<br>AF3           | GPIO[109]<br>E0UC[27]<br>SCK_4<br>—                                    | SIUL<br>eMIOS_0<br>DSPI_4<br>—                      | I/O<br>I/O<br>I/O<br>—           | M                     | Tristate                   | —          | 91       | 115      |
| PG[14]        | PCR[110]     | AF0<br>AF1<br>AF2<br>AF3           | GPIO[110]<br>E1UC[0]<br>—<br>—   | SIUL<br>eMIOS_1<br>—<br>—                           | I/O<br>I/O<br>—<br>—             | S                     | Tristate                   | —          | 110      | 134      |
| PG[15]        | PCR[111]     | AF0<br>AF1<br>AF2<br>AF3<br>—      | GPIO[111]<br>E1UC[1]<br>—<br>—<br>—                                    | SIUL<br>eMIOS_1<br>—<br>—<br>—                      | I/O<br>I/O<br>—<br>—<br>—        | M                     | Tristate                   | —          | 111      | 135      |
| <b>Port H</b> |              |                                    |  |   |                                  |                       |                            |            |          |          |
| PH[0]         | PCR[112]     | AF0<br>AF1<br>AF2<br>AF3<br>—      | GPIO[112]<br>E1UC[2]<br>—<br>—<br>SIN_1                                | SIUL<br>eMIOS_1<br>—<br>—<br>DSPI_1                 | I/O<br>I/O<br>—<br>—<br>I        | M                     | Tristate                   | —          | 93       | 117      |
| PH[1]         | PCR[113]     | AF0<br>AF1<br>AF2<br>AF3           | GPIO[113]<br>E1UC[3]<br>SOUT_1<br>—                                    | SIUL<br>eMIOS_1<br>DSPI_1<br>—                      | I/O<br>I/O<br>O<br>—             | M                     | Tristate                   | —          | 94       | 118      |

Table 4-1. Functional port pins (continued)

| Port pin            | PCR register | Alternate function <sup>1</sup> | Function                                | Peripheral                          | I/O direction          | Pad type <sup>2</sup> | RESET config. <sup>3</sup> | Pin number |          |          |
|---------------------|--------------|---------------------------------|---|-------------------------------------|------------------------|-----------------------|----------------------------|------------|----------|----------|
|                     |              |                                 |   |                                     |                        |                       |                            | 100 LQFP   | 144 LQFP | 176 LQFP |
| PH[2]               | PCR[114]     | AF0<br>AF1<br>AF2<br>AF3        | GPIO[114]<br>E1UC[4]<br>SCK_1<br>—      | SIUL<br>eMIOS_1<br>DSPI_1<br>—      | I/O<br>I/O<br>I/O<br>— | M                     | Tristate                   | —          | 95       | 119      |
| PH[3]               | PCR[115]     | AF0<br>AF1<br>AF2<br>AF3        | GPIO[115]<br>E1UC[5]<br>CS0_1<br>—      | SIUL<br>eMIOS_1<br>DSPI_1<br>—      | I/O<br>I/O<br>I/O<br>— | M                     | Tristate                   | —          | 96       | 120      |
| PH[4]               | PCR[116]     | AF0<br>AF1<br>AF2<br>AF3        | GPIO[116]<br>E1UC[6]<br>—<br>—          | SIUL<br>eMIOS_1<br>—<br>—           | I/O<br>I/O<br>—<br>—   | M                     | Tristate                   | —          | 134      | 162      |
| PH[5]               | PCR[117]     | AF0<br>AF1<br>AF2<br>AF3        | GPIO[117]<br>E1UC[7]<br>—<br>—          | SIUL<br>eMIOS_1<br>—<br>—           | I/O<br>I/O<br>—<br>—   | S                     | Tristate                   | —          | 135      | 163      |
| PH[6]               | PCR[118]     | AF0<br>AF1<br>AF2<br>AF3        | GPIO[118]<br>E1UC[8]<br>—<br>MA[2]      | SIUL<br>eMIOS_1<br>—<br>ADC_0       | I/O<br>I/O<br>—<br>O   | M                     | Tristate                   | —          | 136      | 164      |
| PH[7]               | PCR[119]     | AF0<br>AF1<br>AF2<br>AF3        | GPIO[119]<br>E1UC[9]<br>CS3_2<br>MA[1]  | SIUL<br>eMIOS_1<br>DSPI_2<br>ADC_0  | I/O<br>I/O<br>O<br>O   | M                     | Tristate                   | —          | 137      | 165      |
| PH[8]               | PCR[120]     | AF0<br>AF1<br>AF2<br>AF3        | GPIO[120]<br>E1UC[10]<br>CS2_2<br>MA[0] | SIUL<br>eMIOS_1<br>DSPI_2<br>ADC_0  | I/O<br>I/O<br>O<br>O   | M                     | Tristate                   | —          | 138      | 166      |
| PH[9] <sup>8</sup>  | PCR[121]     | AF0<br>AF1<br>AF2<br>AF3        | GPIO[121]<br>—<br>TCK<br>—              | SIUL<br>—<br>JTAGC<br>—             | I/O<br>—<br>I<br>—     | S                     | Input,<br>weak<br>pull-up  | 88         | 127      | 155      |
| PH[10] <sup>8</sup> | PCR[122]     | AF0<br>AF1<br>AF2<br>AF3        | GPIO[122]<br>—<br>TMS<br>—              | SIUL<br>—<br>JTAGC<br>—             | I/O<br>—<br>I<br>—     | M                     | Input,<br>weak<br>pull-up  | 81         | 120      | 148      |
| PH[11]              | PCR[123]     | AF0<br>AF1<br>AF2<br>AF3        | GPIO[123]<br>SOUT_3<br>CS0_4<br>E1UC[5] | SIUL<br>DSPI_3<br>DSPI_4<br>eMIOS_1 | I/O<br>O<br>I/O<br>I/O | M                     | Tristate                   | —          | —        | 140      |
| PH[12]              | PCR[124]     | AF0<br>AF1<br>AF2<br>AF3        | GPIO[124]<br>SCK_3<br>CS1_4<br>E1UC[25] | SIUL<br>DSPI_3<br>DSPI_4<br>eMIOS_1 | I/O<br>I/O<br>I/O<br>— | M                     | Tristate                   | —          | —        | 141      |

Table 4-1. Functional port pins (continued)

| Port pin      | PCR register | Alternate function <sup>1</sup>    | Function  | Peripheral                             | I/O direction                  | Pad type <sup>2</sup> | RESET config. <sup>3</sup> | Pin number |          |          |
|---------------|--------------|------------------------------------|---|--|--------------------------------|-----------------------|----------------------------|------------|----------|----------|
|               |              |                                    |   |  |                                |                       |                            | 100 LQFP   | 144 LQFP | 176 LQFP |
| PH[13]        | PCR[125]     | AF0<br>AF1<br>AF2<br>AF3           | GPIO[125]<br>SOUT_4<br>CS0_3<br>E1UC[26]                        | SIUL<br>DSPI_4<br>DSPI_3<br>eMIOS_1    | I/O<br>O<br>I/O<br>—           | M                     | Tristate                   | —          | —        | 9        |
| PH[14]        | PCR[126]     | AF0<br>AF1<br>AF2<br>AF3           | GPIO[126]<br>SCK_4<br>CS1_3<br>E1UC[27]                         | SIUL<br>DSPI_4<br>DSPI_3<br>eMIOS_1    | I/O<br>I/O<br>I/O<br>—         | M                     | Tristate                   | —          | —        | 10       |
| PH[15]        | PCR[127]     | AF0<br>AF1<br>AF2<br>AF3           | GPIO[127]<br>SOUT_5<br>—<br>E1UC[17]                            | SIUL<br>DSPI_5<br>—<br>eMIOS_1         | I/O<br>O<br>—<br>—             | M                     | Tristate                   | —          | —        | 8        |
| <b>Port I</b> |              |                                    |   |  |                                |                       |                            |            |          |          |
| PI[0]         | PCR[128]     | AF0<br>AF1<br>AF2<br>AF3           | GPIO[128]<br>E0UC[28]<br>—<br>—                                 | SIUL<br>eMIOS_0<br>—<br>—              | I/O<br>I/O<br>—<br>—           | S                     | Tristate                   | —          | —        | 172      |
| PI[1]         | PCR[129]     | AF0<br>AF1<br>AF2<br>AF3<br>—<br>— | GPIO[129]<br>E0UC[29]<br>—<br>—<br>WKUP[24] <sup>(4)</sup><br>— | SIUL<br>eMIOS_0<br>—<br>—<br>WKUP<br>— | I/O<br>I/O<br>—<br>—<br>I<br>— | S                     | Tristate                   | —          | —        | 171      |
| PI[2]         | PCR[130]     | AF0<br>AF1<br>AF2<br>AF3           | GPIO[130]<br>E0UC[30]<br>—<br>—                                 | SIUL<br>eMIOS_0<br>—<br>—              | I/O<br>I/O<br>—<br>—           | S                     | Tristate                   | —          | —        | 170      |
| PI[3]         | PCR[131]     | AF0<br>AF1<br>AF2<br>AF3<br>—<br>— | GPIO[131]<br>E0UC[31]<br>—<br>—<br>WKUP[23] <sup>(4)</sup><br>— | SIUL<br>eMIOS_0<br>—<br>—<br>WKUP<br>— | I/O<br>I/O<br>—<br>—<br>I<br>— | S                     | Tristate                   | —          | —        | 169      |
| PI[4]         | PCR[132]     | AF0<br>AF1<br>AF2<br>AF3           | GPIO[132]<br>E1UC[28]<br>SOUT_4<br>—                            | SIUL<br>eMIOS_1<br>DSPI_4<br>—         | I/O<br>I/O<br>O<br>—           | S                     | Tristate                   | —          | —        | 143      |
| PI[5]         | PCR[133]     | AF0<br>AF1<br>AF2<br>AF3           | GPIO[133]<br>E1UC[29]<br>SCK_4<br>—                             | SIUL<br>eMIOS_1<br>DSPI_4<br>—         | I/O<br>I/O<br>I/O<br>—         | S                     | Tristate                   | —          | —        | 142      |
| PI[6]         | PCR[134]     | AF0<br>AF1<br>AF2<br>AF3           | GPIO[134]<br>E1UC[30]<br>CS0_4<br>—                             | SIUL<br>eMIOS_1<br>DSPI_4<br>—         | I/O<br>I/O<br>I/O<br>—         | S                     | Tristate                   | —          | —        | 11       |

Table 4-1. Functional port pins (continued)

| Port pin | PCR register | Alternate function <sup>1</sup>    | Function  | Peripheral                             | I/O direction                | Pad type <sup>2</sup> | RESET config. <sup>3</sup> | Pin number |          |          |
|----------|--------------|------------------------------------|---|--|------------------------------|-----------------------|----------------------------|------------|----------|----------|
|          |              |                                    |   |  |                              |                       |                            | 100 LQFP   | 144 LQFP | 176 LQFP |
| PI[7]    | PCR[135]     | AF0<br>AF1<br>AF2<br>AF3           | GPIO[135]<br>E1UC[31]<br>CS1_4<br>—             | SIUL<br>eMIOS_1<br>DSPI_4<br>—         | I/O<br>I/O<br>I/O<br>—       | S                     | Tristate                   | —          | —        | 12       |
| PI[8]    | PCR[136]     | AF0<br>AF1<br>AF2<br>AF3<br>—      | GPIO[136]<br>—<br>—<br>—<br>ADC0_S[16]          | SIUL<br>—<br>—<br>—<br>ADC_0           | I/O<br>—<br>—<br>—<br>I      | J                     | Tristate                   | —          | —        | 108      |
| PI[9]    | PCR[137]     | AF0<br>AF1<br>AF2<br>AF3<br>—      | GPIO[137]<br>—<br>—<br>—<br>ADC0_S[17]          | SIUL<br>—<br>—<br>—<br>ADC_0           | I/O<br>—<br>—<br>—<br>I      | J                     | Tristate                   | —          | —        | 109      |
| PI[10]   | PCR[138]     | AF0<br>AF1<br>AF2<br>AF3<br>—      | GPIO[138]<br>—<br>—<br>—<br>ADC0_S[18]          | SIUL<br>—<br>—<br>—<br>ADC_0           | I/O<br>—<br>—<br>—<br>I      | J                     | Tristate                   | —          | —        | 110      |
| PI[11]   | PCR[139]     | AF0<br>AF1<br>AF2<br>AF3<br>—<br>— | GPIO[139]<br>—<br>—<br>—<br>ADC0_S[19]<br>SIN_3 | SIUL<br>—<br>—<br>—<br>ADC_0<br>DSPI_3 | I/O<br>—<br>—<br>—<br>I<br>I | J                     | Tristate                   | —          | —        | 111      |
| PI[12]   | PCR[140]     | AF0<br>AF1<br>AF2<br>AF3<br>—      | GPIO[140]<br>CS0_3<br>—<br>—<br>ADC0_S[20]      | SIUL<br>DSPI_3<br>—<br>—<br>ADC_0      | I/O<br>I/O<br>—<br>—<br>I    | J                     | Tristate                   | —          | —        | 112      |
| PI[13]   | PCR[141]     | AF0<br>AF1<br>AF2<br>AF3<br>—      | GPIO[141]<br>CS1_3<br>—<br>—<br>ADC0_S[21]      | SIUL<br>DSPI_3<br>—<br>—<br>ADC_0      | I/O<br>I/O<br>—<br>—<br>I    | J                     | Tristate                   | —          | —        | 113      |
| PI[14]   | PCR[142]     | AF0<br>AF1<br>AF2<br>AF3<br>—<br>— | GPIO[142]<br>—<br>—<br>—<br>ADC0_S[22]<br>SIN_4 | SIUL<br>—<br>—<br>—<br>ADC_0<br>DSPI_4 | I/O<br>—<br>—<br>—<br>I<br>I | J                     | Tristate                   | —          | —        | 76       |
| PI[15]   | PCR[143]     | AF0<br>AF1<br>AF2<br>AF3<br>—      | GPIO[143]<br>CS0_4<br>—<br>—<br>ADC0_S[23]      | SIUL<br>DSPI_4<br>—<br>—<br>ADC_0      | I/O<br>I/O<br>—<br>—<br>I    | J                     | Tristate                   | —          | —        | 75       |



Table 4-1. Functional port pins (continued)

| Port pin      | PCR register | Alternate function <sup>1</sup>    | Function  | Peripheral                             | I/O direction                | Pad type <sup>2</sup> | RESET config. <sup>3</sup> | Pin number |          |          |
|---------------|--------------|------------------------------------|---|--|------------------------------|-----------------------|----------------------------|------------|----------|----------|
|               |              |                                    |   |  |                              |                       |                            | 100 LQFP   | 144 LQFP | 176 LQFP |
| <b>Port J</b> |              |                                    |   |  |                              |                       |                            |            |          |          |
| PJ[0]         | PCR[144]     | AF0<br>AF1<br>AF2<br>AF3<br>—      | GPIO[144]<br>CS1_4<br>—<br>—<br>ADC0_S[24]      | SIUL<br>DSPI_4<br>—<br>—<br>ADC_0      | I/O<br>I/O<br>—<br>—<br>I    | J                     | Tristate                   | —          | —        | 74       |
| PJ[1]         | PCR[145]     | AF0<br>AF1<br>AF2<br>AF3<br>—<br>— | GPIO[145]<br>—<br>—<br>—<br>ADC0_S[25]<br>SIN_5 | SIUL<br>—<br>—<br>—<br>ADC_0<br>DSPI_5 | I/O<br>—<br>—<br>—<br>I<br>I | J                     | Tristate                   | —          | —        | 73       |
| PJ[2]         | PCR[146]     | AF0<br>AF1<br>AF2<br>AF3<br>—      | GPIO[146]<br>CS0_5<br>—<br>—<br>ADC0_S[26]      | SIUL<br>DSPI_5<br>—<br>—<br>ADC_0      | I/O<br>I/O<br>—<br>—<br>I    | J                     | Tristate                   | —          | —        | 72       |
| PJ[3]         | PCR[147]     | AF0<br>AF1<br>AF2<br>AF3<br>—      | GPIO[147]<br>CS1_5<br>—<br>—<br>ADC0_S[27]      | SIUL<br>DSPI_5<br>—<br>—<br>ADC_0      | I/O<br>I/O<br>—<br>—<br>I    | J                     | Tristate                   | —          | —        | 71       |
| PJ[4]         | PCR[148]     | AF0<br>AF1<br>AF2<br>AF3           | GPIO[148]<br>SCK_5<br>E1UC[18]<br>—             | SIUL<br>DSPI_5<br>eMIOS_1<br>—         | I/O<br>I/O<br>—<br>—         | M                     | Tristate                   | —          | —        | 5        |

<sup>1</sup> Alternate functions are chosen by setting the values of the PCR.PA bitfields inside the SIUL module. PCR.PA = 00 → AF0; PCR.PA = 01 → AF1; PCR.PA = 10 → AF2; PCR.PA = 11 → AF3. This is intended to select the output functions; to use one of the input functions, the PCR.IBE bit must be written to '1', regardless of the values selected in the PCR.PA bitfields. For this reason, the value corresponding to an input only function is reported as "—".

<sup>2</sup> See Table 4-2.

<sup>3</sup> The RESET configuration applies during and after reset.

<sup>4</sup> All WKUP pins also support external interrupt capability. See the WKPU chapter of the *MPC5606BK Microcontroller Reference Manual* for further details.

<sup>5</sup> NMI has higher priority than alternate function. When NMI is selected, the PCR.AF field is ignored.

<sup>6</sup> "Not applicable" because these functions are available only while the device is booting. See the BAM chapter of the *MPC5606BK Microcontroller Reference Manual* for details.

<sup>7</sup> Value of PCR.IBE bit must be 0.

<sup>8</sup> Out of reset all the functional pins except PC[0:1] and PH[9:10] are available to the user as GPIO. PC[0:1] are available as JTAG pins (TDI and TDO respectively). PH[9:10] are available as JTAG pins (TCK and TMS respectively). It is up to the user to configure these pins as GPIO when needed.

- <sup>9</sup> PC[1] is a fast/medium pad but is in medium configuration by default. This pad is in Alternate Function 2 mode after reset which has TDO functionality. The reset value of PCR.OBE is 1, but this setting has no impact as long as this pad stays in AF2 mode. After configuring this pad as GPIO (PCR.PA = 0), output buffer is enabled as reset value of PCR.OBE = 1.
- <sup>10</sup> Not available in 100LQFP package.

**Table 4-2. Pad types**

| Type | Description                      |
|------|----------------------------------|
| F    | Fast                             |
| I    | Input only with analog feature   |
| J    | Input/output with analog feature |
| M    | Medium                           |
| S    | Slow                             |

# Chapter 5

## Microcontroller Boot

This chapter explains the process of booting the microcontroller. The following entities are involved in the boot process:

- [Boot Assist Module \(BAM\)](#)
- [System Status and Configuration Module \(SSCM\)](#)
- Flash memory boot sectors (see [Chapter 30, Flash Memory](#))
- Memory Management Unit (MMU)

### 5.1 Boot mechanism

This section describes the configuration required by the user, and the steps performed by the microcontroller, in order to achieve a successful boot from flash memory or serial download modes.

Two external pins on the microcontroller are latched during reset, and determine whether the microcontroller boots from flash memory or attempt a serial download via FlexCAN or LINFlex (RS232). These are:

- FAB (Force Alternate Boot mode) on pin PA[9]
- ABS (Alternate Boot Select) on pin PA[8]

[Table 5-1](#) describes the configuration options.

**Table 5-1. Boot mode selection**

| Mode                             | FAB pin (PA[9]) | ABS pin (PA[8]) |
|----------------------------------|-----------------|-----------------|
| Flash memory boot (default mode) | 0               | X               |
| Serial boot (LINFlex)            | 1               | 0               |
| Serial boot (FlexCAN)            | 1               | 1               |

The microcontroller has a weak pulldown on PA[9] and a weak pullup on PA[8]. This means that if nothing external is connected to these pins, the microcontroller will enter flash memory boot mode by default. In order to change the boot behavior, you should use external pullup or pulldown resistors on PA[9] and PA[8]. If there is any external circuitry connected to either pin, you must ensure that this does not interfere with the expected value applied to the pin at reset. Otherwise, the microcontroller may boot into an unexpected mode after reset.

The SSCM preforms a lot of the automated boot activity including reading the latched value of the FAB (PA[9]) pin to determine whether to boot from flash memory or serial boot mode. This is illustrated in [Figure 5-1](#).

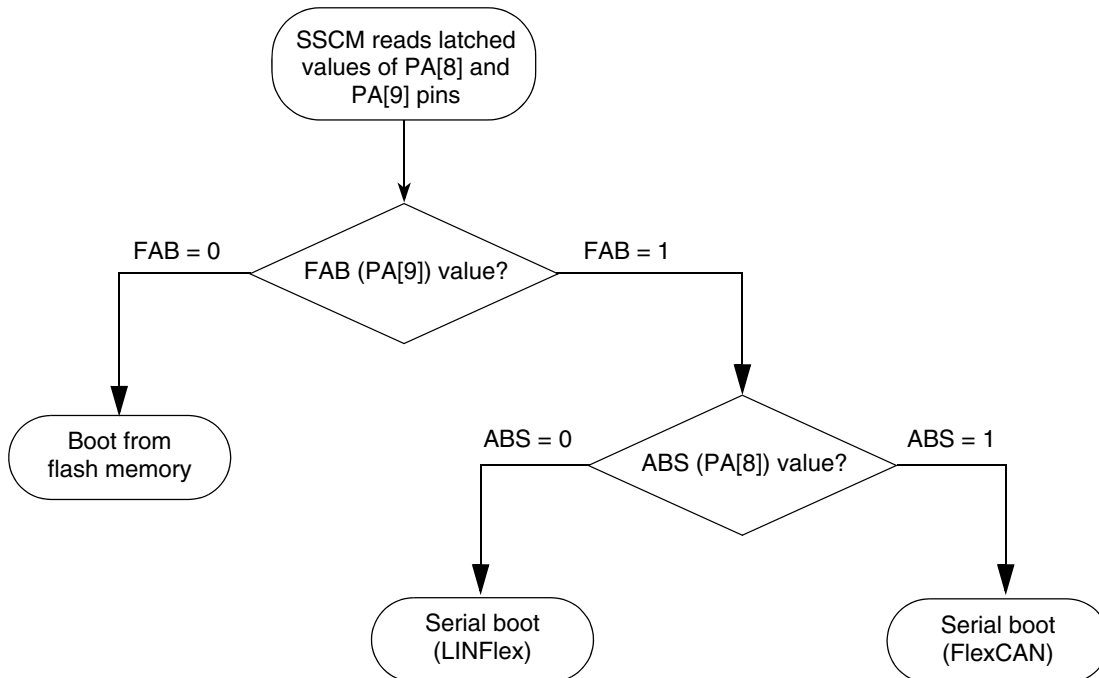


Figure 5-1. Boot mode selection

### 5.1.1 Flash memory boot

In order to successfully boot from flash memory, you must program two 32-bit fields into one of 5 possible boot blocks as detailed below. The entities to program are:

- 16-bit Reset Configuration Half Word (RCHW), which contains:
  - A `BOOT_ID` field that must be correctly set to `0x5A` in order to “validate” the boot sector
- 32-bit reset vector (this is the start address of the user code)

The location and structure of the boot sectors in flash memory are shown in [Figure 5-2](#).

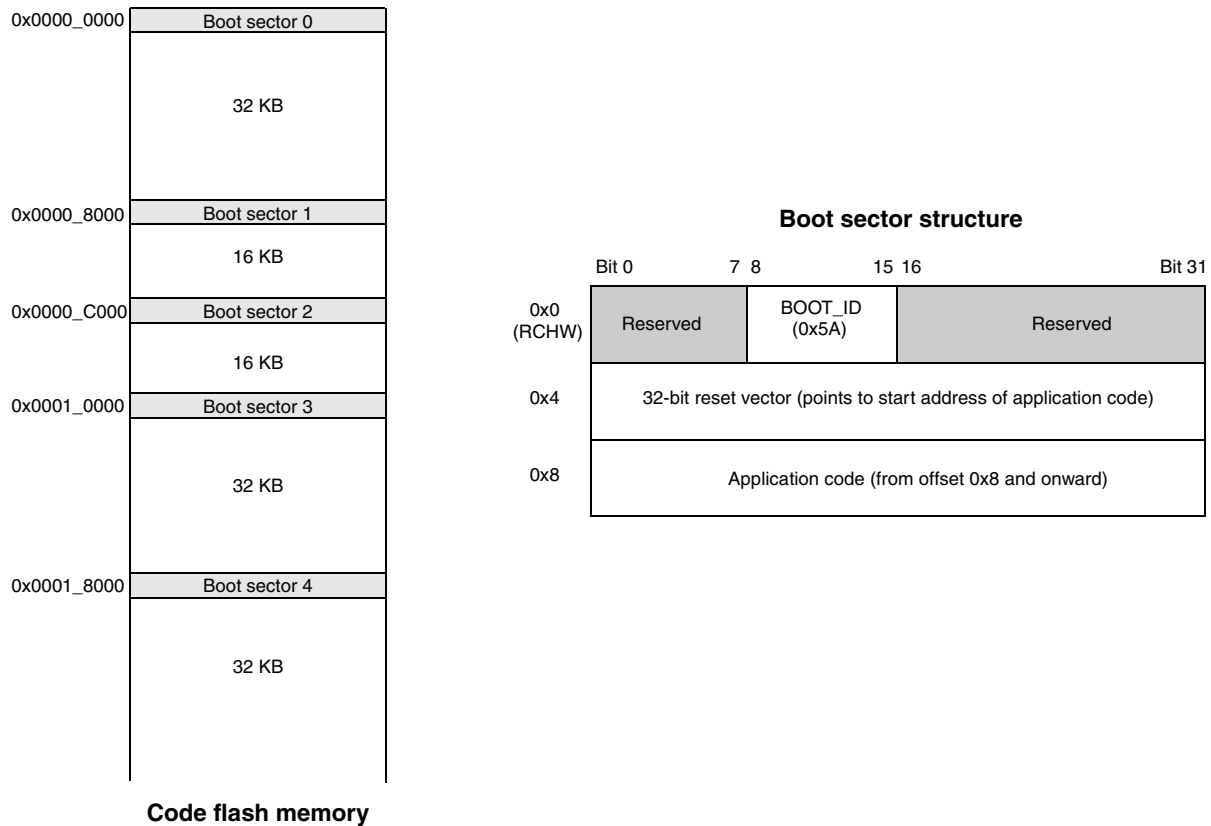


Figure 5-2. Boot sector structure

The RCHW fields are described in [Table 5-2](#).

Table 5-2. RCHW field descriptions

| Field   | Description  |
|---------|--|
| BOOT_ID | Boot identifier.<br>If BOOT_ID = 0x5A, the boot sector is considered valid and bootable. |

The SSCM performs a sequential search of each boot sector (starting at sector 0) for a valid BOOT\_ID within the RCHW. If a valid BOOT\_ID is found, the SSCM reads the boot vector address. If a valid BOOT\_ID is not found, the SSCM starts the process of putting the microcontroller into static mode.

Finally, the SSCM sets the e200z0h core instruction pointer to the reset vector address and starts the core running.

### 5.1.1.1 Static mode

If no valid BOOT\_ID within the RCHW was found, the SSCM sets the CPU core instruction pointer to the BAM address and the core starts to execute the code to enter static mode as follows:

- The core executes the “wait” instruction, which halts the core.

The sequence is illustrated in [Figure 5-3](#).

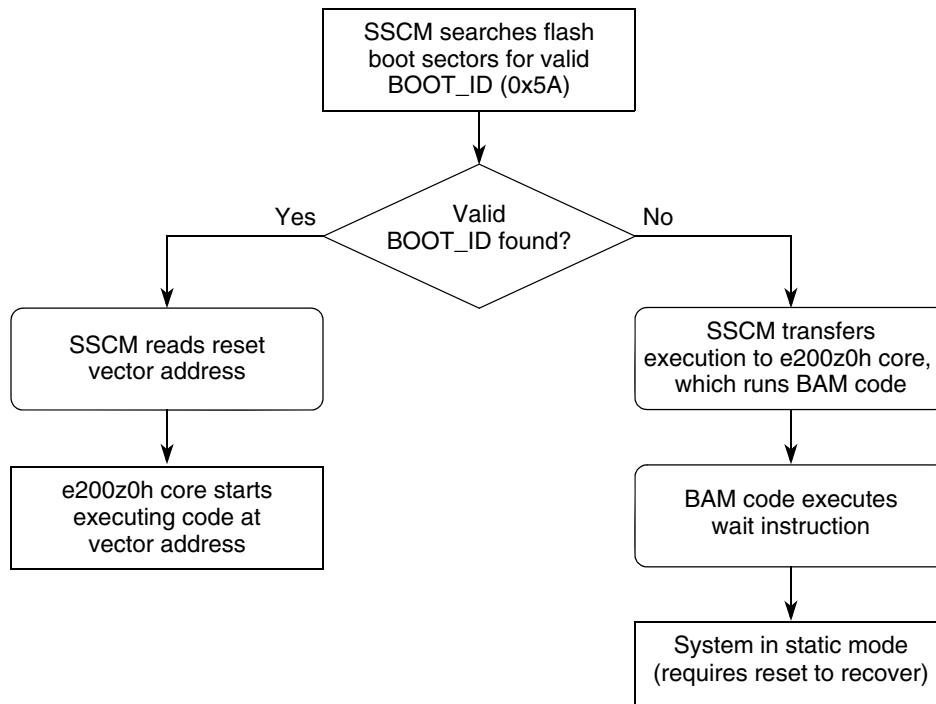


Figure 5-3. Flash memory boot mode sequence

### 5.1.1.2 Alternate boot sectors

Some applications require an alternate boot sector so that the main boot code can be erased and reprogrammed in the field. When an alternate boot is needed, you can create two bootable sectors:

- The valid boot sector located at the lowest address is the main boot sector.
- The valid boot sector located at the next available address is the alternate boot sector.

This scheme ensures that there is always one active boot sector even if the main boot sector is erased.

### 5.1.2 Serial boot mode

Serial boot provides a mechanism to download and then execute code into the microcontroller SRAM. Code may be downloaded using either FlexCAN or LINFlex (RS232). After the SSCM has detected that serial boot mode has been requested, execution is transferred to the BAM, which handles all of the serial boot mode tasks. See [Section 5.2, Boot Assist Module \(BAM\)](#), for more details.

### 5.1.3 Censorship

Censorship can be enabled to protect the contents of the flash memory from being read or modified. In order to achieve this, the censorship mechanism controls access to the:

- JTAG debug interface
- Serial boot mode (which could otherwise be used to download and execute code to query or modify the flash memory)

To regain access to the flash memory via JTAG or serial boot, a 64-bit password must be correctly entered.

### CAUTION

When censorship has been enabled, the only way to regain access is with the password. If this is forgotten or not correctly configured, then there is no way back into the device.

Two 64-bit values stored in the shadow flash control the censorship (see [Table 30-6](#) for a full description):

- Nonvolatile Private Censorship Password registers, NVPWD0 and NVPWD1
- Nonvolatile System Censorship Control registers, NVSCC0 and NVSCC1

#### 5.1.3.1 Censorship password registers (NVPWD0 and NVPWD1)

The two private password registers combine to form a 64-bit password that should be programmed to a value known only by you. After factory test these registers are programmed as shown below:

- NVPWD0 = 0xFEED\_FACE
- NVPWD1 = 0xCAFE\_BEEF

This means that even if censorship was inadvertently enabled by writing to the censorship control registers, there is an opportunity to get back into the microcontroller using the default private password of 0xFEED\_FACE\_CAFE\_BEEF.

When configuring the private password, each half word (16-bit) must contain at least one 1 and one 0. Some examples of legal and illegal passwords are shown in [Table 5-3](#):

**Table 5-3. Examples of legal and illegal passwords**

| Legal (valid) passwords | Illegal (invalid) passwords |
|-------------------------|-----------------------------|
| 0x0001_0001_0001_0001   | 0x0000_XXXX_XXXX_XXXX       |
| 0xFFFFE_FFFE_FFFE_FFFE  | 0xFFFFF_XXXX_XXXX_XXXX      |
| 0x1XXX_X2XX_XX4X_XXX8   |                             |

In uncensored devices it is possible to download code via LINFlex or FlexCAN (Serial Boot Mode) into internal SRAM even if the 64-bit private password stored in the flash and provided during the boot sequence is a password that does not conform to the password rules.

#### 5.1.3.2 Nonvolatile System Censorship Control registers (NVSCC0 and NVSCC1)

These registers are used together to define the censorship configuration. After factory test these registers are programmed as shown below, which disables censorship:

- NVSCC0 = 0x55AA\_55AA
- NVSCC1 = 0x55AA\_55AA

Each 32-bit register is split into an upper and lower 16-bit field. The upper 16 bits (the SC field) are used to control serial boot mode censorship. The lower 16 bits (the CW field) are used to control flash memory boot censorship.

**CAUTION**

If the contents of the shadow flash memory are erased and the NVSCC0,1 registers are not reprogrammed to a valid value, the microcontroller will be permanently censored with no way for you to regain access. A microcontroller in this state cannot be debugged or reflashed.

**5.1.3.3 Censorship configuration**

The steps to configuring censorship are:

1. Define a valid 64-bit password that conforms to the password rules.
2. Using the table and flow charts below, decide what level of censorship you require and configure the NVSCC0,1 values.
3. Reprogram the shadow flash memory and NVPWD0,1 and NVSCC0,1 registers with your new values. A POR is required before these will take effect.

**CAUTION**

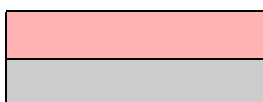
If  
 (NVSCC0 and NVSCC1 do not match)  
 or  
 (Either NVSCC0 or NVSCC1 is not set to 0x55AA)  
 then the microcontroller will be permanently censored with no way to get back in.

[Table 5-4](#) shows all the possible modes of censorship. The red shaded areas are to be avoided as these show the configuration for a device that is permanently locked out. If you wish to enable censorship with a private password there is only one valid configuration — to modify the CW field in both NVSCC0,1 registers so they match but do not equal 0x55AA. This will allow you to enter the private password in both serial and flash boot modes.



Table 5-4. Censorship configuration and truth table

| Boot configuration     |  | Serial censorship control word (NVSCC $n$ [SC]) | Censorship control word (NVSCC $n$ [CW]) | Internal flash memory state | Serial password                                  | JTAG password                                    |
|------------------------|--|---|--|-----------------------------|--|--|
| FAB pin state          | Control options                              |   |  |                             |  |  |
| 0 (flash memory boot)  | Uncensored                                   | 0XXXX AND NVSCC0 == NVSCC1                      | 0x55AA AND NVSCC0 == NVSCC1              | Enabled                     |  | N/A  |
|                        | Private flash memory password and censored   | 0x55AA AND NVSCC0 == NVSCC1                     | !0x55AA AND NVSCC0 == NVSCC1             | Enabled                     |  | NVPWD1,0 (SSCM reads flash memory <sup>1</sup> ) |
|                        | Censored with no password access (lockout)   | !0x55AA   | !0x55AA                                  | Enabled                     |  | N/A  |
| OR<br>NVSCC0 != NVSCC1 |  |   |  |                             |  |  |
| 1 (serial boot)        | Private flash memory password and uncensored | 0x55AA AND NVSCC0 == NVSCC1                     |  | Enabled                     | NVPWD0,1 (BAM reads flash memory <sup>1</sup> )  |  |
|                        | Private flash memory password and censored   | 0x55AA AND NVSCC0 == NVSCC1                     | !0x55AA AND NVSCC0 == NVSCC1             | Enabled                     | NVPWD1,0 (SSCM reads flash memory <sup>1</sup> ) |  |
|                        | Public password and uncensored               | !0x55AA AND NVSCC0 != NVSCC1                    | 0x55AA AND NVSCC0 != NVSCC1              | Enabled                     | Public (0xFEED_FACE_CAFE_BEE F)                  |  |
|                        | Public password and censored (lockout)       | !0x55AA   |  | Disabled                    | Public (0xFEED_FACE_CAFE_BEE F)                  |  |
| OR NVSCC0 != NVSCC1    |  |   |  |                             |  |  |

 = Microcontroller permanently locked out

 = Not applicable

<sup>1</sup> When the SSCM reads the passwords from flash memory, the NVPWD0 and NVPWD1 password order is swapped, so you have to submit the 64-bit password as {NVPWD1, NVPWD0}.

The flow charts in [Figure 5-4](#) and [Figure 5-5](#) provide a way to quickly check what will happen with different configurations of the NVSCC0,1 registers as well as detailing the correct way to enter the serial password. In the password examples, assume the 64-bit password has been programmed into the shadow flash memory in the order {NVPWD0, NVPWD1} and has a value of 0x01234567\_89ABCDEF.

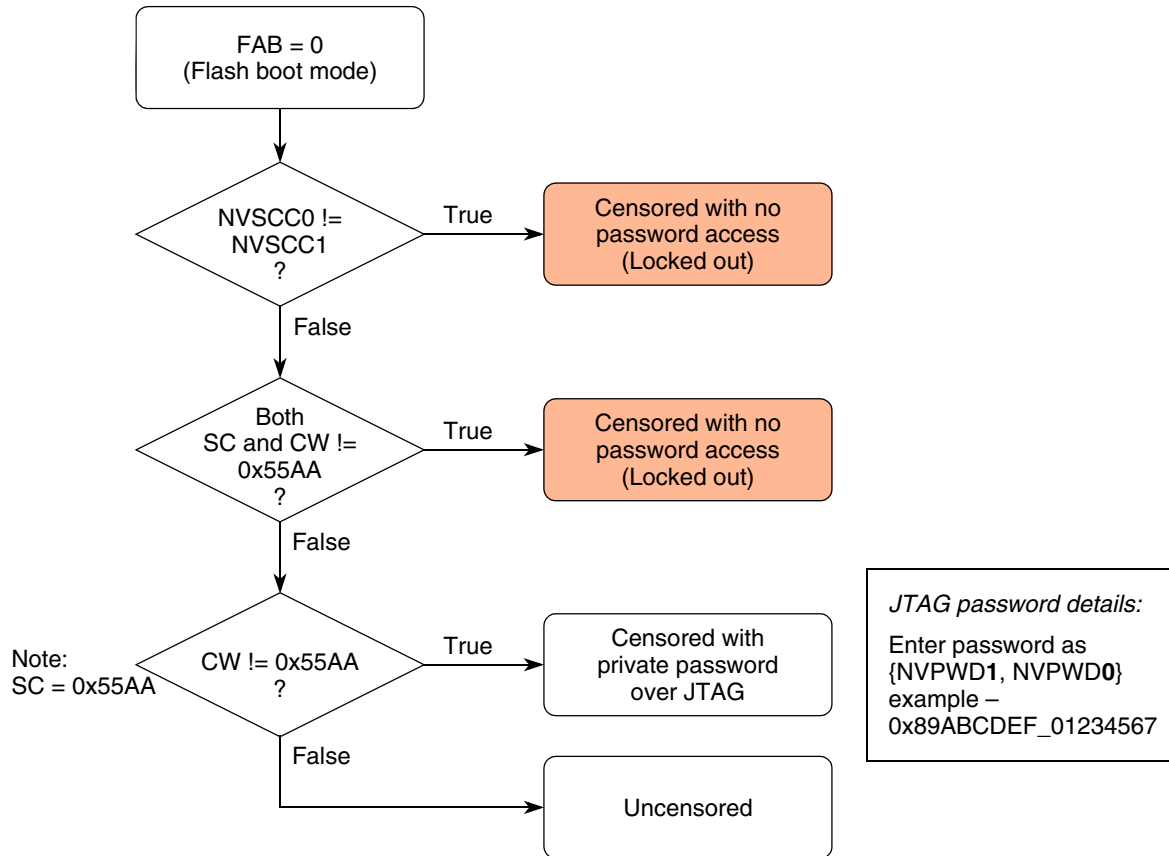


Figure 5-4. Censorship control in flash memory boot mode

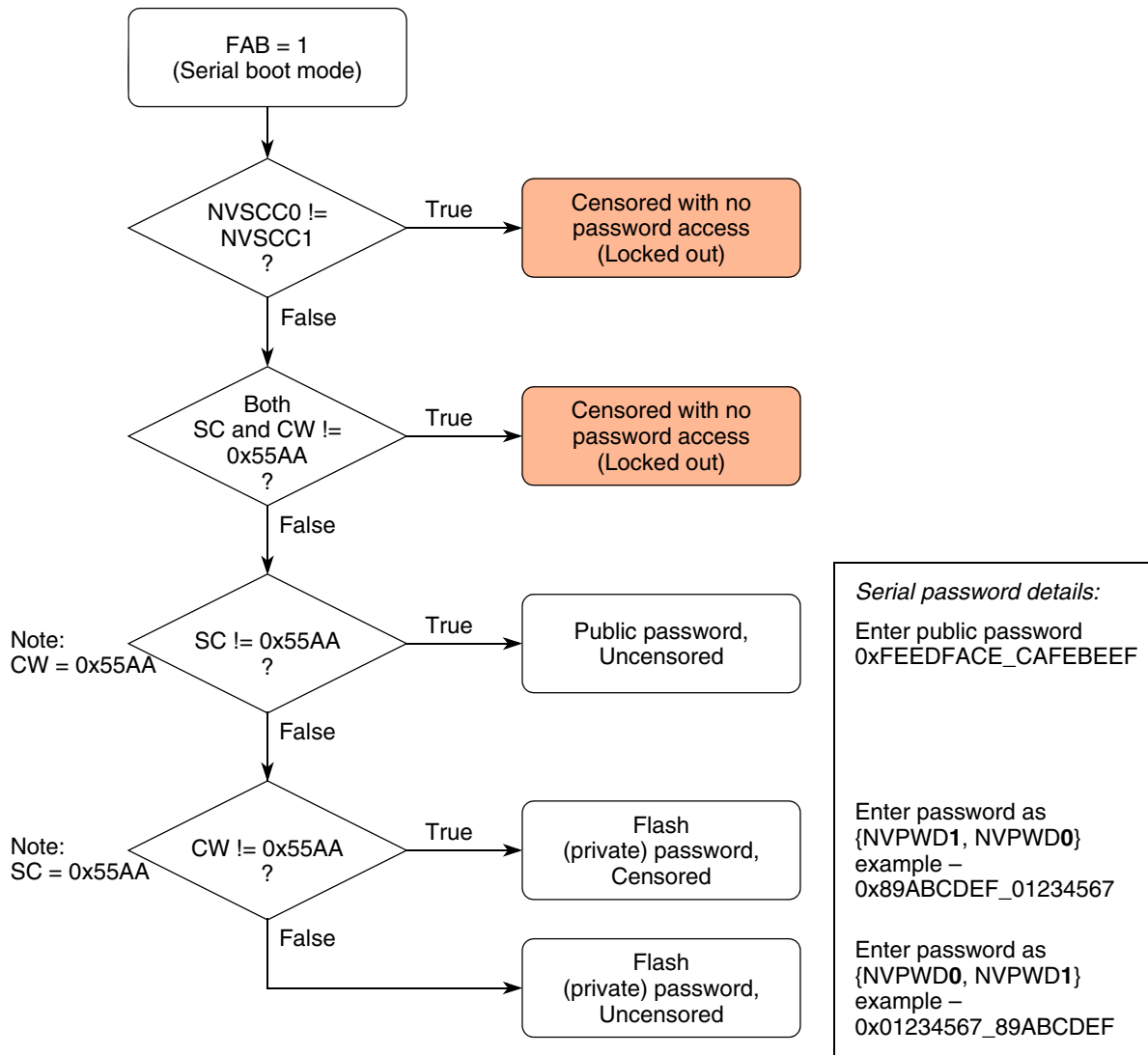


Figure 5-5. Censorship control in serial boot mode

## 5.2 Boot Assist Module (BAM)

The BAM consists of a block of ROM at address 0xFFFF\_C000 containing VLE firmware. The BAM provides 2 main functions:

- Manages the serial download (FlexCAN or LINFlex protocols supported) including support for a serial password if censorship is enabled
- Places the microcontroller into static mode if flash memory boot mode is selected and a valid BOOT\_ID is not located in one of the boot sectors by the SSCM

### 5.2.1 BAM software flow

Figure 5-6 illustrates the BAM logic flow.

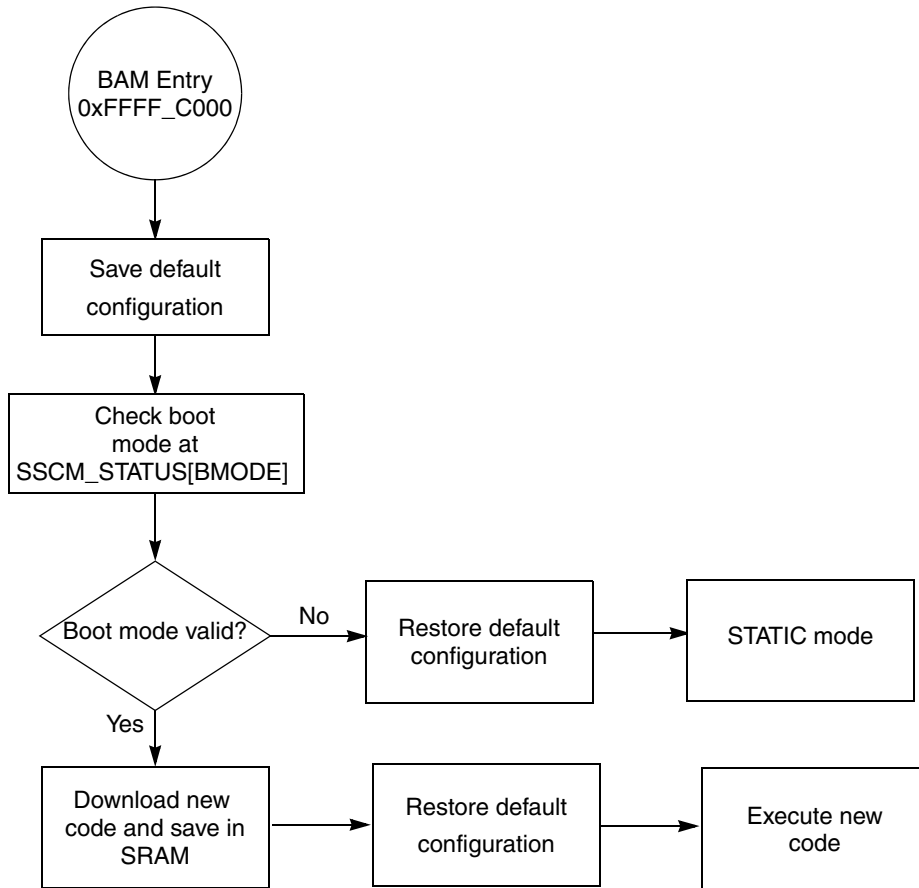


Figure 5-6. BAM logic flow

The initial (reset) device configuration is saved including the mode and clock configuration. This means that the serial download software running in the BAM can make changes to the modes and clocking, and then restore these to the default values before running the newly downloaded application code from the SRAM.

The `SSCM_STATUS[BMODE]` field indicates which boot mode is to be executed (see [Table 5-5](#)). This field is only updated during reset.

There are two conditions where the boot mode is not considered valid and the BAM pushes the microcontroller into static mode after restoring the default configuration:

- `BMODE = 011` (flash memory boot mode). This means that the SSCM has been unable to find a valid `BOOT_ID` in the boot sectors so has called the BAM
- `BMODE = reserved`

In static mode a wait instruction is executed to halt the core.

For the FlexCAN and LINFlex serial boot modes, the respective area of BAM code is executed to download the code to SRAM.

**Table 5-5. SSCM\_STATUS[BMODE] values as used by BAM**

| BMODE value | Corresponding boot mode                    |
|-------------|--|
| 000         | Reserved                                   |
| 001         | FlexCAN_0 serial boot loader               |
| 010         | LINFlex_0 (RS232 /UART) serial boot loader |
| 011         | Flash memory boot mode                     |
| 100–111     | Reserved                                   |

After the code has been downloaded to SRAM, the BAM code restores the initial device configuration, and then transfers execution to the start address of the downloaded code.

### 5.2.1.1 BAM resources

The BAM uses/initializes the following MCU resources:

- MC\_ME and MC\_CGM to initialize mode and clock sources
- FlexCAN\_0, LINFlex\_0 and the respective I/O pins when performing serial boot mode
- SSCM during password check
- SSCM to check the boot mode (see [Table 5-5](#))
- 4–16 MHz fast external crystal oscillator

The system clock is selected directly from the 4–16 MHz fast external crystal oscillator. Thus, the external oscillator frequency defines the baud rates used for serial download (see [Table 5-6](#)).

**Table 5-6. Serial boot mode – baud rates**

| FXOSC frequency (MHz) | LINFlex baud rate (baud) | CAN bit rate (bit/s) |
|-----------------------|--------------------------|----------------------|
| $f_{FXOSC}$           | $f_{FXOSC}/833$          | $f_{FXOSC}/40$       |
| 8                     | 9600                     | 200K                 |
| 12                    | 14400                    | 300K                 |
| 16                    | 19200                    | 400K                 |

### 5.2.1.2 Download and execute the new code

From a high level perspective, the download protocol follows these steps:

1. Send the 64-bit password.
2. Send the start address, size of code to be downloaded (in bytes), and the VLE bit<sup>1</sup>.
3. Download the code.

Each step must be completed before the next step starts. After the download is complete (the specified number of bytes is downloaded), the code executes from the start address.

<sup>1</sup> Since the device supports only VLE code and not Book E code, this flag is used only for backward compatibility.

The communication is done in half duplex manner, whereby the transmission from the host is followed by the microcontroller transmission mirroring the transmission back to the host:

- Host sends data to the microcontroller and waits for a response.
- MCU echoes to host the data received.
- Host verifies if echo is correct:
  - If data is correct, the host can continue to send data.
  - If data is not correct, the host stops transmission and the microcontroller enters static mode.

All multi-byte data structures are sent with MSB first.

A more detailed description of these steps follows.

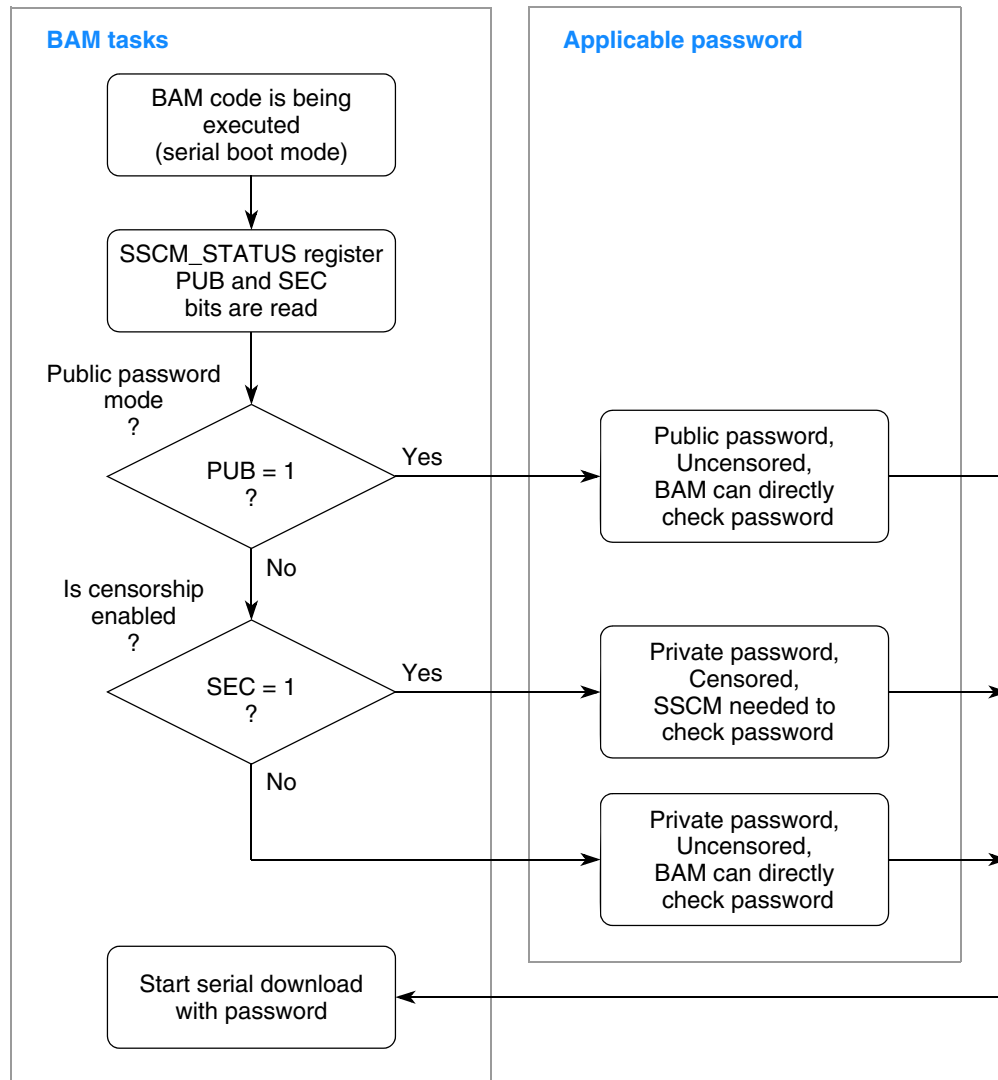
### 5.2.1.3 Censorship mode detection and serial password validation

Before the serial download can commence, the BAM code must determine which censorship mode the microcontroller is in and which password to use. It does this by reading the PUB and SEC fields in the SSCM Status Register (see [Section 5.3.4.1, System Status Register \(SSCM\\_STATUS\)](#)) as shown in [Table 5-7](#).

**Table 5-7. BAM censorship mode detection**

| SSCM_STATUS register fields |     | Mode                         | Password comparison                 |
|-----------------------------|-----|------------------------------|-------------------------------------|
| PUB                         | SEC |                              |                                     |
| 1                           | 0   | Uncensored, public password  | 0xFEED_FACE_CAFE_BEEF               |
| 0                           | 0   | Uncensored, private password | NVPWD0,1 from flash memory via BAM  |
| 0                           | 1   | Censored, private password   | NVPWD1,0 from flash memory via SSCM |

When censorship is enabled, the flash memory cannot be read by application code running in the BAM or in the SRAM. This means that the private password in the shadow flash memory cannot be read by the BAM code. In this case the SSCM is used to obtain the private password from the flash memory of the censored device. When the SSCM reads the private password it inverts the order of {NVPWD0, NVPWD1} so the password entered over the serial download needs to be {NVPWD1, NVPWD0}.



**Figure 5-7. BAM censorship mode detection**

The first thing to be downloaded is the 64-bit password. If the password does not match the stored password, then the BAM code pushes the microcontroller into static mode.

The way the password is compared with either the public or private password (depending on mode) varies depending on whether censorship is enabled as described in the following subsections.

#### 5.2.1.3.1 Censorship disabled (private or public passwords):

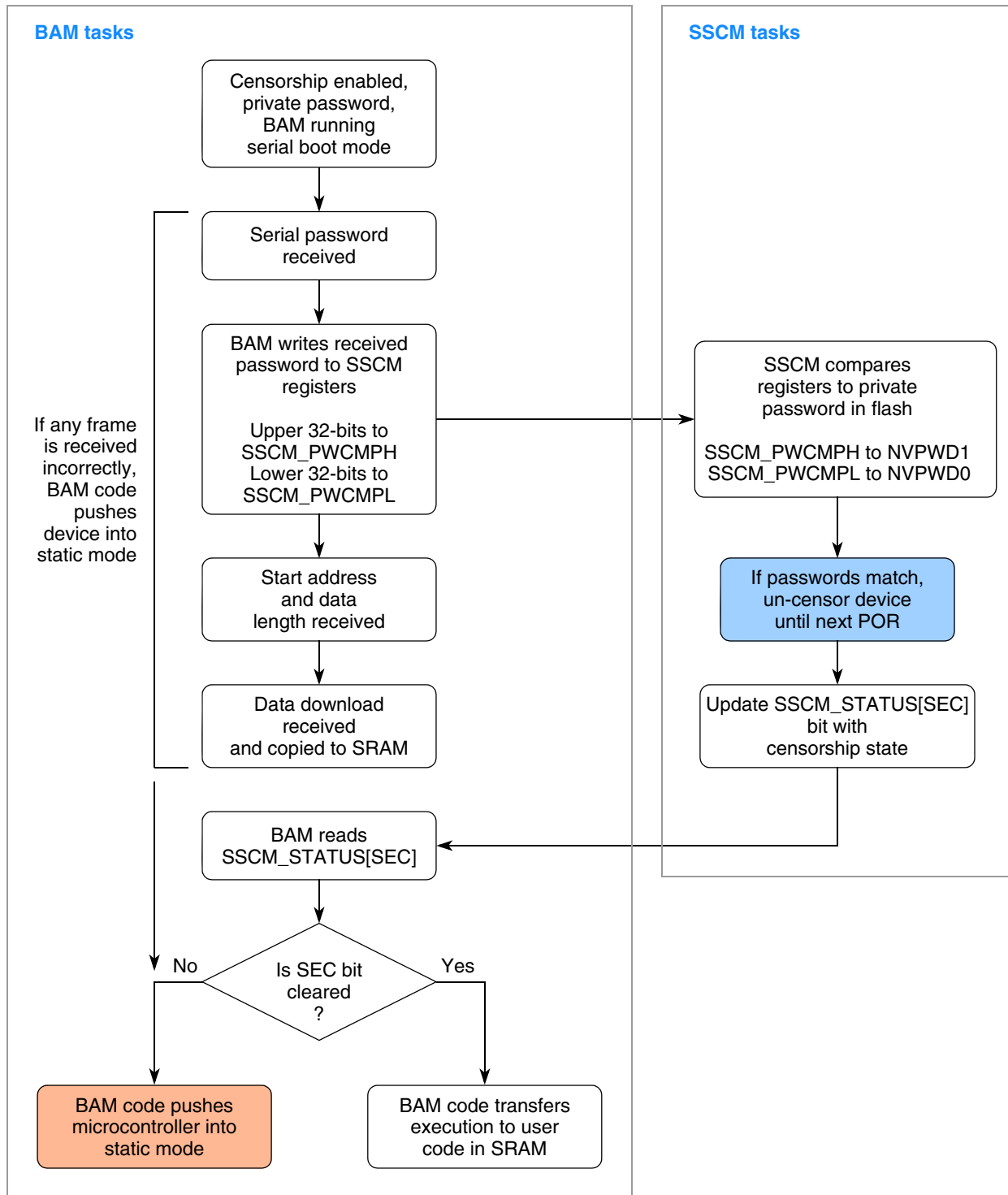
1. If the public password is used, the BAM code does a direct comparison between the serial password and 0xFEED\_FACE\_CAFE\_BEEF.
2. If the private password is used, the BAM code does a direct comparison between the serial password and the private password in flash memory, {NVPWD0, NVPWD1}.
3. If the password does not match, the BAM code immediately terminates the download and pushes the microcontroller into static mode.

### 5.2.1.3.2 Censorship enabled (private password)

1. Since the flash is secured, the SSCM is required to read the private password.
2. The BAM code writes the serial password to the SSCM\_PWCMPH and SSCM\_PWCMPH registers.
3. The BAM code then continues with the serial download (start address, data size, and data) until all the data has been copied to the SRAM.
4. In the meantime the SSCM has compared the private password in flash with the serial download password the BAM code wrote into SSCM\_PWCMPH and SSCM\_PWCMPH.
5. If the SSCM obtains a match in the passwords, the censorship is temporarily disabled (until the next reset).
6. The SSCM updates the status of the security (SEC) bit to reflect whether the passwords matched (SEC = 0) or not (SEC = 1)
7. Finally, the BAM code reads SEC. If SEC = 0, execution is transferred to the code in the SRAM. If SEC = 1, the BAM code forces the microcontroller into static mode.

Figure 5-8 shows this in more detail.





**Figure 5-8. BAM serial boot mode flow for censorship enabled and private password**

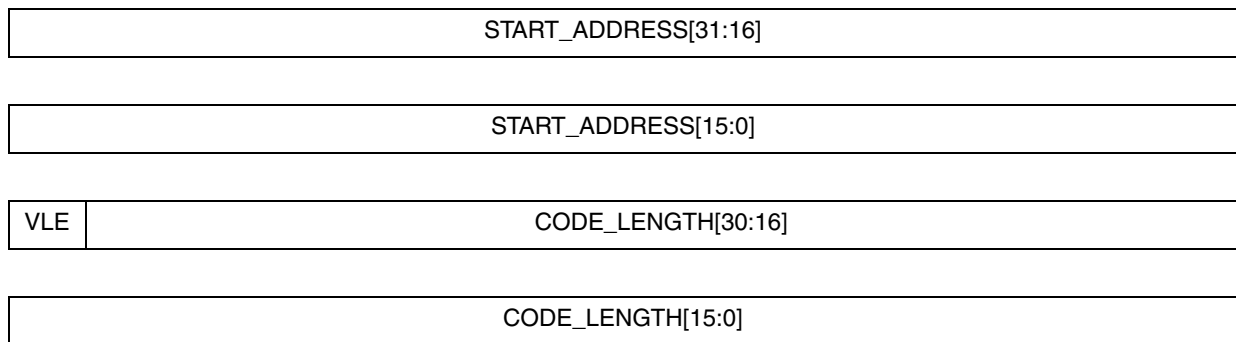
With LINFlex, any receive error will result in static mode. With FlexCAN, the host will retransmit data if there has been no acknowledgment from the microcontroller. However, there could be a situation where the receiver configuration has an error, which would result in static mode entry.

**NOTE**

In a censored device booting with serial boot mode, it is possible to read the content of the four 32-bit flash memory locations that make up the boot sector. For example, if the RCHW is stored at address 0x0000\_0000, the reads at address 0x0000\_0000, 0x0000\_0004, 0x0000\_0008, and 0x0000\_000C will return a correct value. No other flash memory locations can be read.

**5.2.1.4 Download start address, VLE bit and code size**

The next 8 bytes received by the microcontroller contain a 32-bit start address, the VLE mode bit, and a 31-bit code length, as shown in [Figure 5-9](#).



**Figure 5-9. Start address, VLE bit, and download size in bytes**

The VLE bit (Variable Length Instruction) is used to indicate whether the code to be downloaded is Book VLE or Book III-E. This device family supports only VLE = 1; the bit is used for backward compatibility.

The Start Address defines where the received data will be stored and where the MCU will branch after the download is finished. The start address is 32-bit word aligned and the 2 least significant bits are ignored by the BAM code.

**NOTE**

The start address is configurable, but must not lie within the 0x4000\_0000 to 0x4000\_00FF address range.

The Length defines how many data bytes have to be loaded.

**5.2.1.5 Download data**

Each byte of data received is stored in the microcontroller's SRAM, starting from the address specified in the previous protocol step.

The address increments until the number of bytes of data received matches the number of bytes specified by the code length.

Since the SRAM is protected by 32-bit wide Error Correction Code (ECC), the BAM code always writes bytes into SRAM grouped into 32-bit words. If the last byte received does not fall onto a 32-bit boundary, the BAM code fills any additional bytes with 0x0.

Since the ECC on the SRAM has not been initialized (except for the bytes of data that have just been downloaded), an additional dummy word of 0x0000\_0000 is written at the end of the downloaded data block to avoid any ECC errors during core prefetch.

### 5.2.1.6 Execute code

The BAM code waits for the last data byte to be received. If the operating mode is censored with a private password, then the BAM reads the SSCM status register to determine whether the serial password matched the private password. If there was a password match, then the BAM code restores the initial configuration and transfers execution to the downloaded code start address in SRAM. If the passwords did not match, the BAM code forces a static mode entry.

#### NOTE

The watchdog is disabled at the start of BAM code execution. In the case of an unexpected issue during BAM code execution, the microcontroller may be stalled and an external reset required to recover the microcontroller.

## 5.2.2 LINFlex (RS232) boot

### 5.2.2.1 Configuration

Boot according to the LINFlex boot mode download protocol (see [Section 5.2.2.2, Protocol](#)) is performed by the LINFlex\_0 module in UART (RS232) mode. Pins used are:

- LIN0TX mapped on PB[2]
- LIN0RX mapped on PB[3]

Boot from LINFlex uses the system clock driven by the 4–16 MHz external crystal oscillator (FXOSC).

The LINFlex controller is configured to operate at a baud rate = system clock frequency/833, using an 8-bit data frame without parity bit and 1 stop bit.

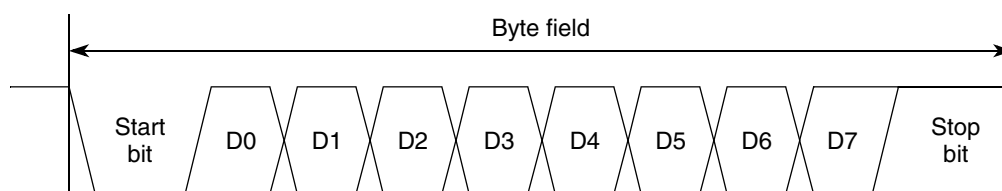


Figure 5-10. LINFlex bit timing in UART mode

### 5.2.2.2 Protocol

[Table 5-8](#) summarizes the protocol and BAM action during this boot mode.

**Table 5-8. UART boot mode download protocol**

| Protocol step | Host sent message                            | BAM response message                         | Action  |
|---------------|--|--|---|
| 1             | 64-bit password (MSB first)                  | 64-bit password                              | Password checked for validity and compared against stored password.   |
| 2             | 32-bit store address                         | 32-bit store address                         | Load address is stored for future use.  |
| 3             | VLE bit + 31-bit number of bytes (MSB first) | VLE bit + 31-bit number of bytes (MSB first) | Size of download are stored for future use. Verify if VLE bit is set to 1   |
| 4             | 8 bits of raw binary data                    | 8 bits of raw binary data                    | 8-bit data are packed into a 32-bit word. This word is saved into SRAM starting from the "Load address". "Load address" increments until the number of data received and stored matches the size as specified in the previous step. |
| 5             | None   | None   | Branch to downloaded code   |

## 5.2.3 FlexCAN boot

### 5.2.3.1 Configuration

Boot according to the FlexCAN boot mode download protocol (see [Section 5.2.3.2, Protocol](#)) is performed by the FlexCAN\_0 module. Pins used are:

- CAN0TX mapped on PB[0]
- CAN0RX mapped on PB[1]

#### NOTE

When the serial download via FlexCAN is selected and the device is part of a CAN network, the serial download may stop unexpectedly if there is any other traffic on the network. To avoid this situation, ensure that no other CAN device on the network is active during the serial download process.

Boot from FlexCAN uses the system clock driven by the 4–16 MHz fast external crystal oscillator.

The FlexCAN controller is configured to operate at a baud rate = system clock frequency/40 (see [Table 5-6](#) for examples of baud rate).

It uses the standard 11-bit identifier format detailed in FlexCAN 2.0A specification.

FlexCAN controller bit timing is programmed with 10 time quanta, and the sample point is 2 time quanta before the end, as shown in [Figure 5-11](#).

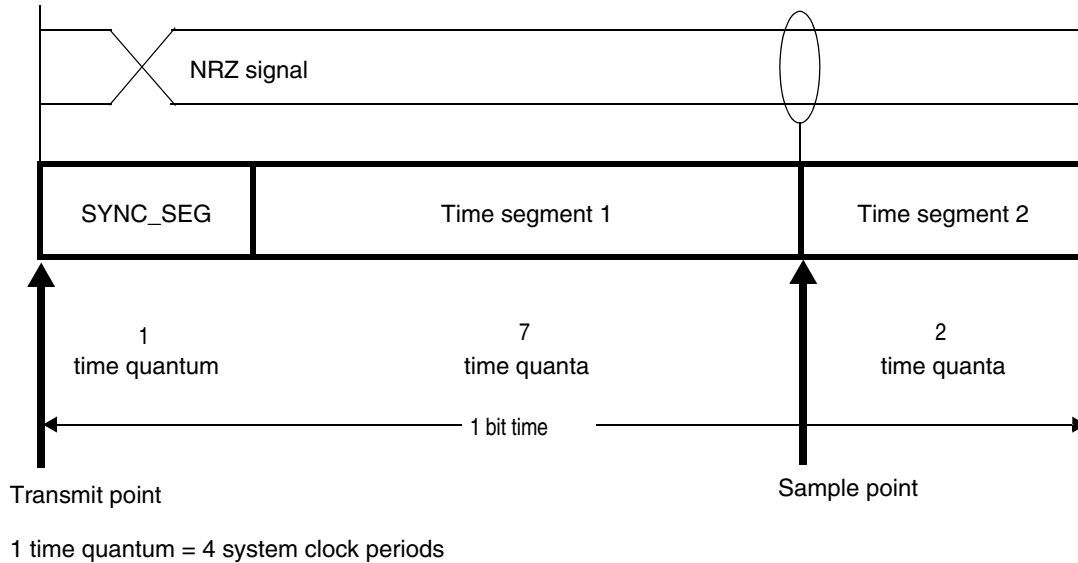


Figure 5-11. FlexCAN bit timing

### 5.2.3.2 Protocol

Table 5-9 summarizes the protocol and BAM action during this boot mode. All data are transmitted byte wise.

Table 5-9. FlexCAN boot mode download protocol

| Protocol step | Host sent message  | BAM response message   | Action   |
|---------------|--|--|--|
| 1             | CAN ID 0x011 + 64-bit password   | CAN ID 0x001 + 64-bit password   | Password checked for validity and compared against stored password   |
| 2             | CAN ID 0x012 + 32-bit store address + VLE bit + 31-bit number of bytes | CAN ID 0x002 + 32-bit store address + VLE bit + 31-bit number of bytes | Load address is stored for future use.<br>Size of download are stored for future use.<br>Verify if VLE bit is set to 1   |
| 3             | CAN ID 0x013 + 8 to 64 bits of raw binary data                         | CAN ID 0x003 + 8 to 64 bits of raw binary data                         | 8-bit data are packed into 32-bit words. These words are saved into SRAM starting from the "Load address".<br>"Load address" increments until the number of data received and stored matches the size as specified in the previous step. |
| 5             | None   | None   | Branch to downloaded code  |

## 5.3 System Status and Configuration Module (SSCM)

### 5.3.1 Introduction

The primary purpose of the SSCM is to provide information about the current state and configuration of the system that may be useful for configuring application software and for debug of the system.

On microcontrollers with a separate STANDBY power domain, the System Status block is part of that domain.

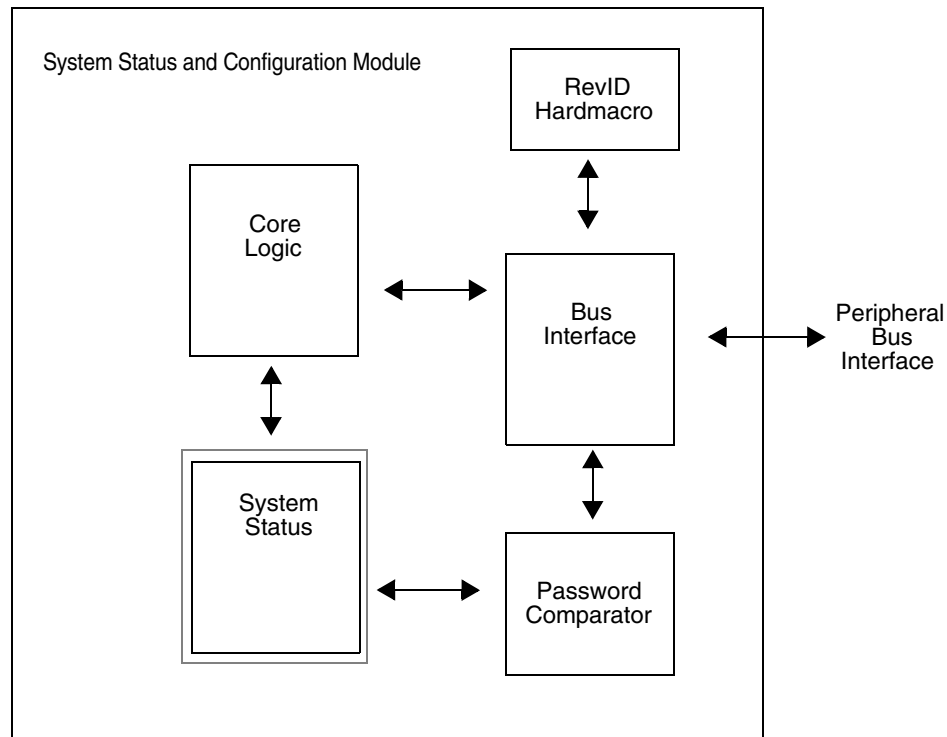


Figure 5-12. SSCM block diagram

### 5.3.2 Features

The SSCM includes these features:

- System Configuration and Status
  - Memory sizes/status
  - Microcontroller Mode and Security Status (including censorship and serial boot information)
  - Search Code Flash for bootable sector
  - Determine boot vector
- Device identification information (MCU ID registers)
- Debug Status Port enable and selection
- Bus and peripheral abort enable/disable

### 5.3.3 Modes of operation

The SSCM operates identically in all system modes.

### 5.3.4 Memory map and register description

Table 5-10 shows the memory map for the SSCM. Note that all addresses are offsets; the absolute address may be calculated by adding the specified offset to the base address of the SSCM.

Table 5-10. SSCM memory map

| Address offset | Register  | Location   |
|----------------|---|------------|
| 0x00           | System Status Register (SSCM_STATUS)                  | on page 95 |
| 0x02           | System Memory Configuration Register (SSCM_MEMCONFIG) | on page 96 |
| 0x04           | Reserved  |            |
| 0x06           | Error Configuration (SSCM_ERROR)                      | on page 97 |
| 0x08           | Debug Status Port Register (SSCM_DEBUGPORT)           | on page 98 |
| 0x0A           | Reserved  |            |
| 0x0C           | Password Comparison Register High Word (SSCM_PWCMPH)  | on page 99 |
| 0x10           | Password Comparison Register Low Word (SSCM_PWC MPL)  | on page 99 |

All registers are accessible via 8-bit, 16-bit, or 32-bit accesses. However, 16-bit accesses must be aligned to 16-bit boundaries, and 32-bit accesses must be aligned to 32-bit boundaries. As an example, the SSCM\_STATUS register is accessible by a 16-bit read/write to address Base + 0x0002, but performing a 16-bit access to Base + 0x0003 is illegal.

#### 5.3.4.1 System Status Register (SSCM\_STATUS)

The System Status register is a read-only register that reflects the current state of the system.

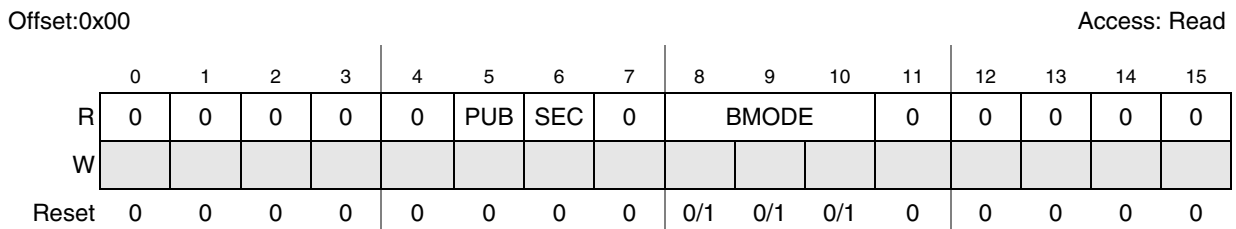


Figure 5-13. System Status Register (SSCM\_STATUS)

Table 5-11. SSCM\_STATUS allowed register accesses

| Access type | 8-bit       | 16-bit      | 32-bit <sup>1</sup> |
|-------------|-------------|-------------|---------------------|
| Read        | Allowed     | Allowed     | Allowed             |
| Write       | Not allowed | Not allowed | Not allowed         |

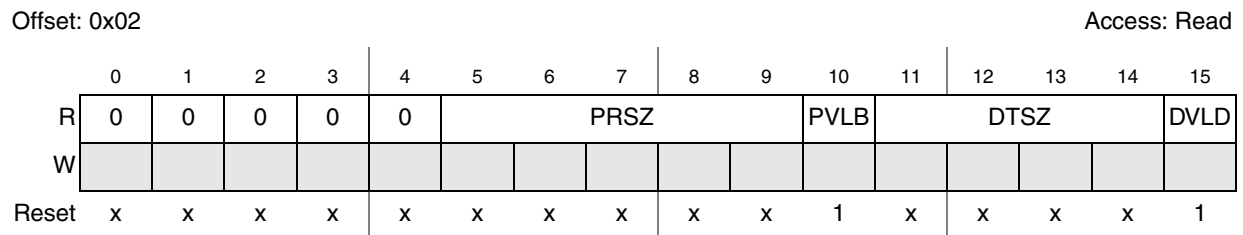
<sup>1</sup> All 32-bit accesses must be aligned to 32-bit addresses (i.e., 0x0, 0x4, 0x8 or 0xC).

**Table 5-12. SSCM\_STATUS field descriptions**

| Field | Description   |
|-------|---|
| PUB   | Public Serial Access Status. This bit indicates whether serial boot mode with public password is allowed.<br>1 Serial boot mode with public password is allowed<br>0 Serial boot mode with private flash memory password is allowed     |
| SEC   | Security Status. This bit reflects the current security state of the flash memory.<br>1 The flash memory is secured.<br>0 The flash memory is not secured.  |
| BMODE | Device Boot Mode<br>000 Reserved<br>001 FlexCAN_0 Serial Boot Loader<br>010 LINFlex_0 Serial Boot Loader<br>011 Single Chip<br>100 Reserved<br>101 Reserved<br>110 Reserved<br>111 Reserved<br>This field is only updated during reset. |

### 5.3.4.2 System Memory Configuration Register (SSCM\_MEMCONFIG)

The System Memory Configuration register is a read-only register that reflects the memory configuration of the system.



**Figure 5-14. System Memory Configuration Register (SSCM\_MEMCONFIG)**



Table 5-13. SSCM\_MEMCONFIG field descriptions

| Field | Description   |
|-------|---|
| PRSZ  | Code Flash Size<br>10000 128 KB<br>10001 256 KB<br>10010 384 KB<br>10011 512 KB<br>10101 768 KB<br>10111 1 MB<br>11011 1.5 MB   |
| PVLB  | Code Flash Available<br>This bit identifies whether or not the on-chip code Flash is available in the system memory map. The Flash may not be accessible due to security limitations, or because there is no Flash in the system.<br>1 Code Flash is available<br>0 Code Flash is not available |
| DTSZ  | Data Flash Size<br>0000 No Data Flash<br>0011 64 KB   |
| DVLD  | Data Flash Valid<br>This bit identifies whether or not the on-chip Data Flash is visible in the system memory map. The Flash may not be accessible due to security limitations, or because there is no Flash in the system.<br>1 Data Flash is visible<br>0 Data Flash is not visible           |

Table 5-14. SSCM\_MEMCONFIG allowed register accesses

| Access type | 8-bit       | 16-bit      | 32-bit                                       |
|-------------|-------------|-------------|--|
| Read        | Allowed     | Allowed     | Allowed<br>(also reads SSCM_STATUS register) |
| Write       | Not allowed | Not allowed | Not allowed                                  |

### 5.3.4.3 Error Configuration (SSCM\_ERROR)

The Error Configuration register is a read-write register that controls the error handling of the system.

Offset: 0x06

Access: Read/write

|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

Figure 5-15. Error Configuration (SSCM\_ERROR)

**Table 5-15. SSCM\_ERROR field descriptions**

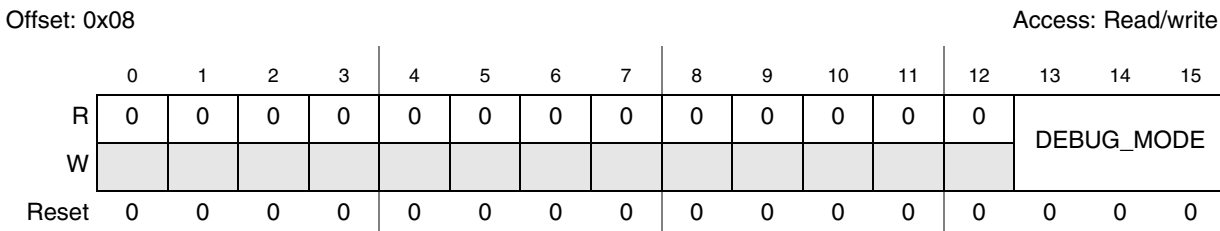
| Field | Description   |
|-------|---|
| PAE   | Peripheral Bus Abort Enable<br>This bit enables bus aborts on any access to a peripheral slot that is not used on the device. This feature is intended to aid in debugging when developing application code.<br>1 Illegal accesses to non-existing peripherals produce a Prefetch or Data Abort exception<br>0 Illegal accesses to non-existing peripherals do not produce a Prefetch or Data Abort exception   |
| RAE   | Register Bus Abort Enable<br>This bit enables bus aborts on illegal accesses to off-platform peripherals. Illegal accesses are defined as reads or writes to reserved addresses within the address space for a particular peripheral. This feature is intended to aid in debugging when developing application code.<br>1 Illegal accesses to peripherals produce a Prefetch or Data Abort exception<br>0 Illegal accesses to peripherals do not produce a Prefetch or Data Abort exception<br>Transfers to Peripheral Bus resources may be aborted even before they reach the Peripheral Bus (that is, at the PBRIDGE level). In this case, bits PAE and RAE will have no effect on the abort. |

**Table 5-16. SSCM\_ERROR allowed register accesses**

| Access type | 8-bit   | 16-bit  | 32-bit      |
|-------------|---------|---------|-------------|
| Read        | Allowed | Allowed | Allowed     |
| Write       | Allowed | Allowed | Not allowed |

### 5.3.4.4 Debug Status Port Register (SSCM\_DEBUGPORT)

The Debug Status Port register is used to (optionally) provide debug data on a set of pins.



**Figure 5-16. Debug Status Port Register (SSCM\_DEBUGPORT)**

**Table 5-17. SSCM\_DEBUGPORT field descriptions**

| Field      | Description  |
|------------|--|
| DEBUG_MODE | Debug Status Port Mode<br>This field selects the alternate debug functionality for the Debug Status Port.<br>000 No alternate functionality selected<br>001 Mode 1 selected<br>010 Mode 2 selected<br>011 Mode 3 selected<br>100 Mode 4 selected<br>101 Mode 5 selected<br>110 Mode 6 selected<br>111 Mode 7 selected<br><a href="#">Table 5-18</a> describes the functionality of the Debug Status Port in each mode. |

Table 5-18. Debug status port modes

| Pin <sub>1</sub> | Mode 1             | Mode 2              | Mode 3                | Mode 4                 | Mode 5   | Mode 6   | Mode 7   |
|------------------|--------------------|---------------------|-----------------------|------------------------|----------|----------|----------|
| 0                | SSCM_STATUS<br>[0] | SSCM_STATUS<br>[8]  | SSCM_MEMCONFI<br>G[0] | SSCM_MEMCONFI<br>G[8]  | Reserved | Reserved | Reserved |
| 1                | SSCM_STATUS<br>[1] | SSCM_STATUS<br>[9]  | SSCM_MEMCONFI<br>G[1] | SSCM_MEMCONFI<br>G[9]  | Reserved | Reserved | Reserved |
| 2                | SSCM_STATUS<br>[2] | SSCM_STATUS<br>[10] | SSCM_MEMCONFI<br>G[2] | SSCM_MEMCONFI<br>G[10] | Reserved | Reserved | Reserved |
| 3                | SSCM_STATUS<br>[3] | SSCM_STATUS<br>[11] | SSCM_MEMCONFI<br>G[3] | SSCM_MEMCONFI<br>G[11] | Reserved | Reserved | Reserved |
| 4                | SSCM_STATUS<br>[4] | SSCM_STATUS<br>[12] | SSCM_MEMCONFI<br>G[4] | SSCM_MEMCONFI<br>G[12] | Reserved | Reserved | Reserved |
| 5                | SSCM_STATUS<br>[5] | SSCM_STATUS<br>[13] | SSCM_MEMCONFI<br>G[5] | SSCM_MEMCONFI<br>G[13] | Reserved | Reserved | Reserved |
| 6                | SSCM_STATUS<br>[6] | SSCM_STATUS<br>[14] | SSCM_MEMCONFI<br>G[6] | SSCM_MEMCONFI<br>G[14] | Reserved | Reserved | Reserved |
| 7                | SSCM_STATUS<br>[7] | SSCM_STATUS<br>[15] | SSCM_MEMCONFI<br>G[7] | SSCM_MEMCONFI<br>G[15] | Reserved | Reserved | Reserved |

<sup>1</sup> All signals are active high, unless otherwise noted

PIN[0..7] referred to in [Table 5-18](#) equates to PC[2..9] (Pad 34..41).

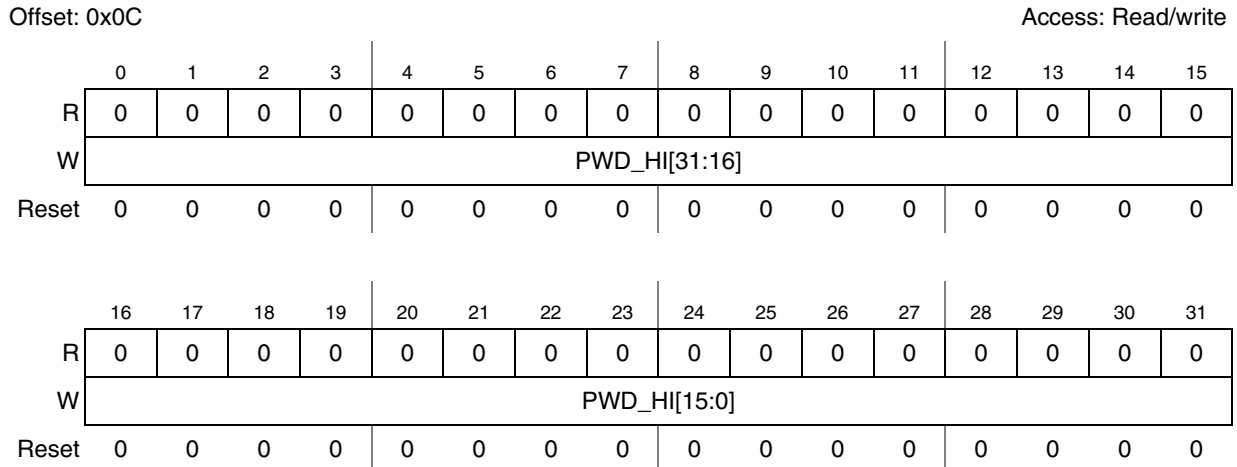
Table 5-19. SSCM\_DEBUGPORT allowed register accesses

| Access type | 8-bit   | 16-bit  | 32-bit <sup>1</sup> |
|-------------|---------|---------|---------------------|
| Read        | Allowed | Allowed | Not allowed         |
| Write       | Allowed | Allowed | Not allowed         |

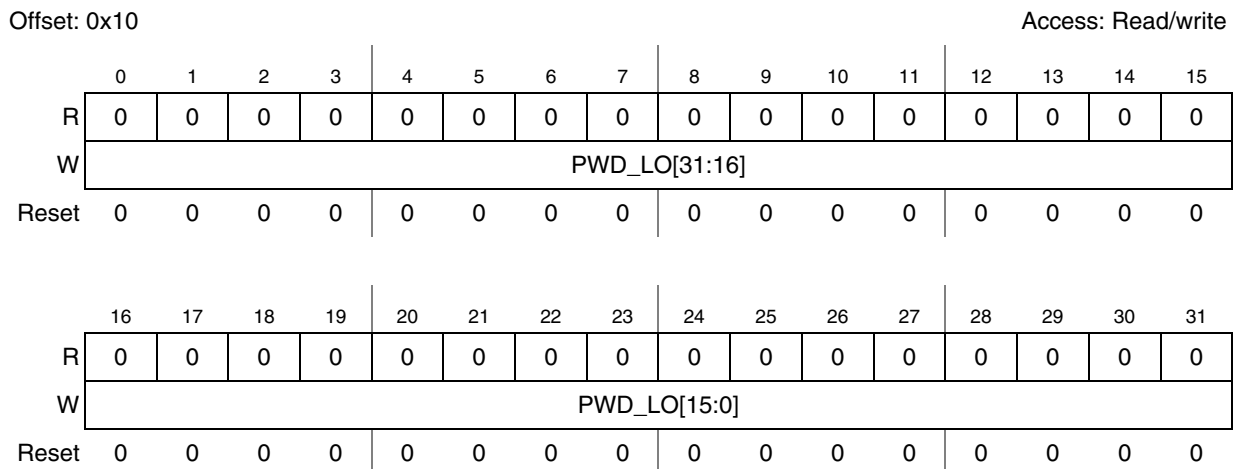
<sup>1</sup> All 32-bit accesses must be aligned to 32-bit addresses (i.e., 0x0, 0x4, 0x8 or 0xC).

### 5.3.4.5 Password comparison registers

These registers provide a means for the BAM code to unsecure the device via the SSCM if the password has been provided via serial download.



**Figure 5-17. Password Comparison Register High Word (SSCM\_PWCMPH)**



**Figure 5-18. Password Comparison Register Low Word (SSCM\_PWCMPH)**

**Table 5-20. Password Comparison Register field descriptions**

| Field  | Description                   |
|--------|-------------------------------|
| PWD_HI | Upper 32 bits of the password |
| PWD_LO | Lower 32 bits of the password |

**Table 5-21. SSCM\_PWCMPH/L allowed register accesses**

| Access type | 8-bit       | 16-bit      | 32-bit <sup>1</sup> |
|-------------|-------------|-------------|---------------------|
| Read        | Allowed     | Allowed     | Allowed             |
| Write       | Not allowed | Not allowed | Allowed             |


<sup>1</sup> All 32-bit accesses must be aligned to 32-bit addresses (i.e., 0x0, 0x4, 0x8 or 0xC).

In order to unsecure the device, the password needs to be written as follows: first the upper word to the SSCM\_PWCMPH register, then the lower word to the SSCM\_PWCMPPL register. The SSCM compares the 64-bit password entered into the SSCM\_PWCMPH / SSCM\_PWCMPPL registers with the NVPWM[1,0] private password stored in the shadow flash. If the passwords match then the SSCM temporarily unsecures the microcontroller.

This page is intentionally left blank.

---

## —— Clocks and power ——



This page is intentionally left blank.



# Chapter 6

## Clock Description

This chapter describes the clock architectural implementation for MPC5606BK.

### 6.1 Clock architecture

System clocks are generated from three sources:

- Fast external crystal oscillator 4–16 MHz (FXOSC)
- Fast internal RC oscillator 16 MHz (FIRC)
- Frequency modulated phase locked loop (FMPLL)

Additionally, there are two low power oscillators:

- Slow internal RC oscillator 128 kHz (SIRC)
- Slow external crystal oscillator 32 KHz (SXOSC)

The clock architecture is shown in [Figure 6-1](#).

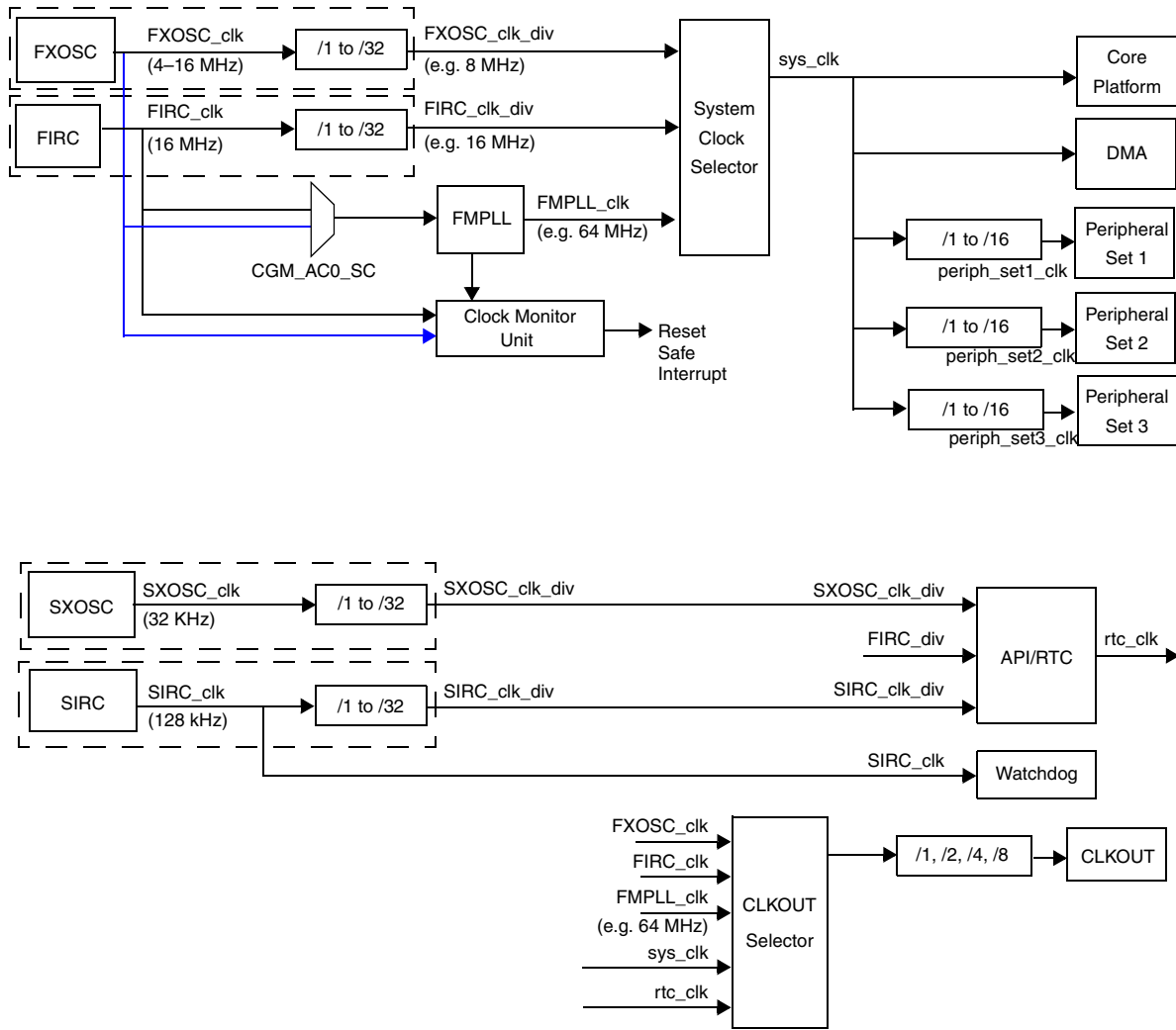


Figure 6-1. MPC5606BK system clock generation

## 6.2 Clock gating

The MPC5606BK provides the user with the possibility of gating the clock to the peripherals. Table 6-1 describes for each peripheral the associated gating register address. See the ME\_PCTLn section in this reference manual.

Additionally, peripheral set (1, 2 or 3) frequency can be configured to be an integer (1 to 16) divided version of the main system clock. See the CGM\_SC\_DC0 section in this reference manual for details.

Table 6-1. MPC5606BK — Peripheral clock sources

| Peripheral       | Register gating address offset (base = 0xC3FD_C0C0) <sup>1</sup> | Peripheral set <sup>2</sup> |
|------------------|--|-----------------------------|
| RPP_Z0H Platform | none (managed through ME mode)                                   | —                           |
| DSPI_n           | 4 + n (n = 0..5)   | 2                           |

Table 6-1. MPC5606BK — Peripheral clock sources (continued)

| Peripheral       | Register gating address offset<br>(base = 0xC3FD_C0C0) <sup>1</sup> | Peripheral set <sup>2</sup> |
|------------------|---|-----------------------------|
| FlexCAN_n        | 16 + n (n = 0..5)   | 2                           |
| ADC_0            | 32  | 3                           |
| ADC_1            | 33  | 3                           |
| I <sup>2</sup> C | 44  | 1                           |
| LINFlex_n        | 48 + n (n = 0..7)   | 1                           |
| CTU              | 57  | 3                           |
| CANS             | 60  | —                           |
| SIUL             | 68  | —                           |
| WKUP             | 69  | —                           |
| eMIOS_n          | 72 + n (n = 0..1)   | 3                           |
| RTC/API          | 91  | —                           |
| PIT              | 92  | —                           |
| CMU              | 104   | —                           |

<sup>1</sup> See the ME\_PCTL section in this reference manual for details.

<sup>2</sup> “—” means undivided system clock.

## 6.3 Fast external crystal oscillator (FXOSC) digital interface

The FXOSC digital interface controls the operation of the 4–16 MHz fast external crystal oscillator (FXOSC). It holds control and status registers accessible for application.

### 6.3.1 Main features

- Oscillator power-down control and status reporting through MC\_ME block
- Oscillator clock available interrupt
- Oscillator bypass mode
- Output clock division factors ranging from 1, 2, 3...32

### 6.3.2 Functional description

The FXOSC circuit includes an internal oscillator driver and an external crystal circuitry. It provides an output clock that can be provided to the FMPLL or used as a reference clock to specific modules depending on system needs.

The FXOSC can be controlled by the MC\_ME module. The ME\_XXX\_MC[FXOSCON] bit controls the powerdown of the oscillator based on the current device mode while ME\_GS[S\_XOSC] register provides the oscillator clock available status.

After system reset, the oscillator is put into powerdown state, and software has to switch on when required. Whenever the crystal oscillator is switched on from the off state, the OSCCNT counter starts. When it reaches the value  $EOCV[7:0] \times 512$ , the oscillator clock is made available to the system. Also, an interrupt pending FXOSC\_CTL[I\_OSC] bit is set. An interrupt is generated if the interrupt mask bit M\_OSC is set.

The oscillator circuit can be bypassed by setting FXOSC\_CTL[OSCBYP]. This bit can only be set by software. A system reset is needed to reset this bit. In this bypass mode, the output clock has the same polarity as the external clock applied on the EXTAL pin and the oscillator status is forced to 1. The bypass configuration is independent of the powerdown mode of the oscillator.

Table 6-2 shows the truth table of different oscillator configurations.

**Table 6-2. Truth table of crystal oscillator**

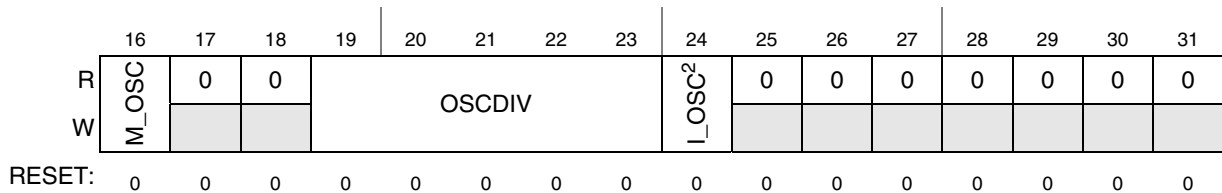
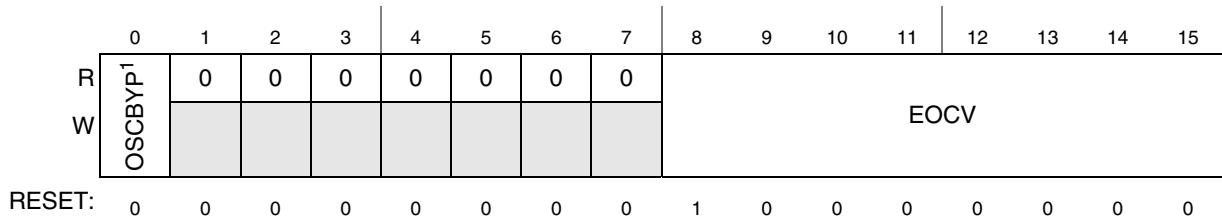
| ME_XXX_MC[FXOSCON] | FXOSC_CTL[OSCBYP] | XTAL               | EXTAL              | FXOSC | Oscillator mode      |
|--------------------|-------------------|--------------------|--------------------|-------|----------------------|
| 0                  | 0                 | No crystal, High Z | No crystal, High Z | 0     | Power down, IDDQ     |
| x                  | 1                 | x                  | Ext clock          | EXTAL | Bypass, OSC disabled |
| 1                  | 0                 | Crystal            | Crystal            | EXTAL | Normal, OSC enabled  |
|                    |                   | Gnd                | Ext clock          | EXTAL | Normal, OSC enabled  |

The FXOSC clock can be further divided by a configurable factor in the range 1 to 32 to generate the divided clock to match system requirements. This division factor is specified by FXOSC\_CTL[OSCDIV] field.

### 6.3.3 Register description

Address: 0xC3FE\_0000

Access: Special read/write



**Figure 6-2. Fast External Crystal Oscillator Control Register (FXOSC\_CTL)**

- <sup>1</sup> You can read this field, and you can write a value of 1 to it. Writing a 0 has no effect. A reset will also clear this bit.
- <sup>2</sup> You can write a value of “0” or “1” to this field. However, writing a “1” will clear this field, and writing “0” will have no effect on the field value.

Table 6-3. FXOSC\_CTL field descriptions

| Field  | Description  |
|--------|--|
| OSCBYP | Crystal Oscillator bypass.<br>This bit specifies whether the oscillator should be bypassed or not.<br>0 Oscillator output is used as root clock<br>1 EXTAL is used as root clock   |
| EOCV   | End of Count Value.<br>These bits specify the end of count value to be used for comparison by the oscillator stabilization counter OSCCNT after reset or whenever it is switched on from the off state (OSCCNT runs on the FXOSC). This counting period ensures that external oscillator clock signal is stable before it can be selected by the system. When oscillator counter reaches the value EOCV × 512, the crystal oscillator clock interrupt (I_OSC) request is generated. The OSCCNT counter will be kept under reset if oscillator bypass mode is selected. |
| M_OSC  | Crystal oscillator clock interrupt mask.<br>0 Crystal oscillator clock interrupt is masked.<br>1 Crystal oscillator clock interrupt is enabled.  |
| OSCDIV | Crystal oscillator clock division factor.<br>This field specifies the crystal oscillator output clock division factor. The output clock is divided by the factor OSCDIV+1.   |
| I_OSC  | Crystal oscillator clock interrupt.<br>This bit is set by hardware when OSCCNT counter reaches the count value EOCV × 512.<br>0 No oscillator clock interrupt occurred.<br>1 Oscillator clock interrupt pending.   |

## 6.4 Slow external crystal oscillator (SXOSC) digital interface

### 6.4.1 Introduction

The SXOSC digital interface controls the operation of the 32 KHz slow external crystal oscillator (SXOSC). It holds control and status registers accessible for application.

### 6.4.2 Main features

- Oscillator powerdown control and status
- Oscillator clock available interrupt
- Oscillator bypass mode
- Output clock division factors ranging from 1 to 32

### 6.4.3 Functional description

The SXOSC circuit includes an internal oscillator driver and an external crystal circuitry. It can be used as a reference clock to specific modules depending on system needs.

The SXOSC can be controlled via the SXOSC\_CTL register. The OSCON bit controls the powerdown while bit S\_OSC provides the oscillator clock available status.

After system reset, the oscillator is put to powerdown state, and software has to switch on when required. Whenever the SXOSC is switched on from off state, the OSCCNT counter starts. When it reaches the value  $EOCV[7:0] \times 512$ , the oscillator clock is made available to the system. Also, an interrupt pending SXOSC\_CTL[I\_OSC] bit is set. An interrupt will be generated if the interrupt mask bit M\_OSC is set.

The oscillator circuit can be bypassed by writing SXOSC\_CTL[OSCBYP] bit to 1. This bit can only be set by software. A system reset is needed to reset this bit. In this bypass mode, the output clock has the same polarity as the external clock applied on the OSC32K\_EXTAL pin and the oscillator status is forced to 1. The bypass configuration is independent of the powerdown mode of the oscillator.

Table 6-4 shows the truth table of different configurations of the oscillator.

Table 6-4. SXOSC truth table

| SXOSC_CTL fields |        | OSC32K_XTAL        | OSC32K_EXTAL       | SXOSC        | Oscillator MODE      |
|------------------|--------|--------------------|--------------------|--------------|----------------------|
| OSCON            | OSCBYP |                    |                    |              |                      |
| 0                | 0      | No crystal, High Z | No crystal, High Z | 0            | Powerdown, IDDQ      |
| x                | 1      | x                  | External clock     | OSC32K_EXTAL | Bypass, OSC disabled |
| 1                | 0      | Crystal            | Crystal            | OSC32K_EXTAL | Normal, OSC enabled  |
|                  |        | Ground             | External clock     | OSC32K_EXTAL | Normal, OSC enabled  |

The SXOSC clock can be further divided by a configurable factor in the range 1 to 32 to generate the divided clock to match system requirements. This division factor is specified by SXOSC\_CTL[OSCDIV] field.

### 6.4.4 Register description

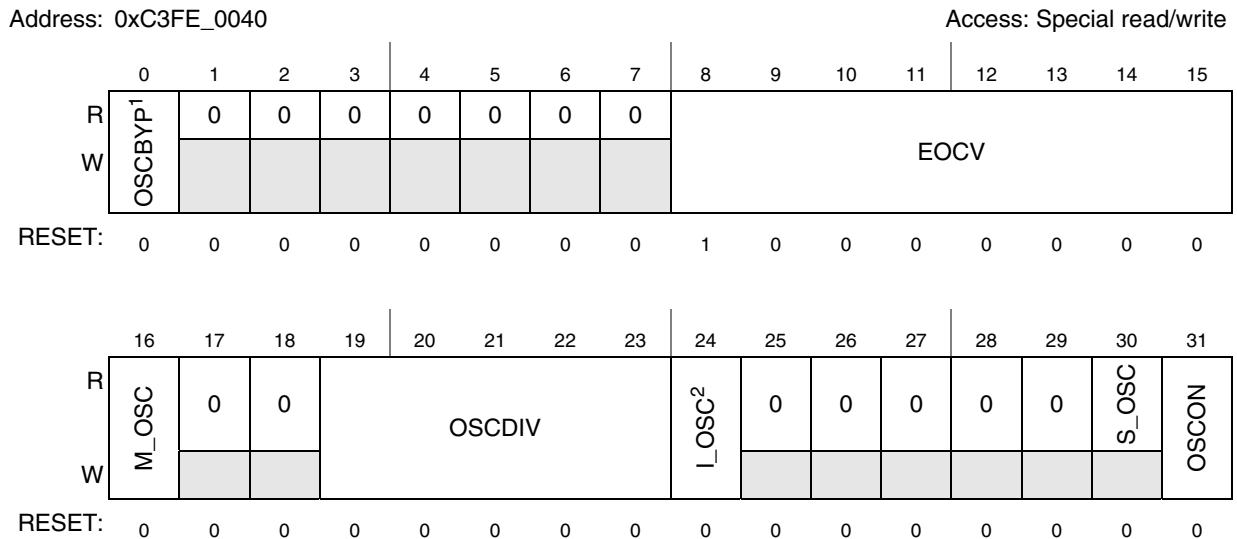


Figure 6-3. Slow External Crystal Oscillator Control Register (SXOSC\_CTL)

- <sup>1</sup> You can read this field, and you can write a value of 1 to it. Writing a 0 has no effect. A reset will also clear this bit.
- <sup>2</sup> You can write a value of “0” or “1” to this field. However, writing a “1” will clear this field, and writing “0” will have no effect on the field value.

**Table 6-5. SXOSC\_CTL field descriptions**

| Field  | Description   |
|--------|---|
| OSCBYP | Crystal Oscillator bypass.<br>This bit specifies whether the oscillator should be bypassed or not.<br>0 Oscillator output is used as root clock.<br>1 OSC32K_EXTAL is used as root clock.   |
| EOCV   | End of Count Value.<br>This field specifies the end of count value to be used for comparison by the oscillator stabilization counter OSCCNT after reset or whenever it is switched on from the off state. This counting period ensures that external oscillator clock signal is stable before it can be selected by the system. When oscillator counter reaches the value EOCV × 512, the crystal oscillator clock interrupt (I_OSC) request is generated. The OSCCNT counter will be kept under reset if oscillator bypass mode is selected. |
| M_OSC  | Crystal oscillator clock interrupt mask.<br>0 Crystal oscillator clock interrupt is masked.<br>1 Crystal oscillator clock interrupt is enabled.   |
| OSCDIV | Crystal oscillator clock division factor.<br>This field specifies the crystal oscillator output clock division factor. The output clock is divided by the factor OSCDIV + 1.  |
| I_OSC  | Crystal oscillator clock interrupt.<br>This field is set by hardware when OSCCNT counter reaches the count value EOCV × 512.<br>0 No oscillator clock interrupt occurred.<br>1 Oscillator clock interrupt pending.  |
| S_OSC  | Crystal oscillator status.<br>0 Crystal oscillator output clock is not stable.<br>1 Crystal oscillator is providing a stable clock.   |
| OSCON  | Crystal oscillator enable.<br>0 Crystal oscillator is switched off.<br>1 Crystal oscillator is switched on.   |

**NOTE**

The 32 KHz slow external crystal oscillator is by default always ON, but can be configured OFF in standby by setting the OSCON bit.

## 6.5 Slow internal RC oscillator (SIRC) digital interface

### 6.5.1 Introduction

The SIRC digital interface controls the 128 kHz slow internal RC oscillator (SIRC). It holds control and status registers accessible for application.

### 6.5.2 Functional description

The SIRC provides a low frequency ( $f_{SIRC}$ ) clock of 128 kHz requiring very low current consumption. This clock can be used as the reference clock when a fixed base time is required for specific modules.

SIRC is always on in all device modes except STANDBY mode. In STANDBY mode, it is controlled by SIRC\_CTL[SIRCON\_STDBY] bit. The clock source status is updated in SIRC\_CTL[S\_SIRC] bit.

The SIRC clock can be further divided by a configurable division factor in the range from 1 to 32 to generate the divided clock to match system requirements. This division factor is specified by SIRC\_CTL[SIRCDIV] bits.

The SIRC output frequency can be trimmed using SIRC\_CTL[SIRCTRIM]. After a power-on reset, the SIRC is trimmed using a factory test value stored in test flash memory. However, after a power-on reset the test flash memory value is not visible at SIRC\_CTL[SIRCTRIM], and this field shows a value of zero. Therefore, be aware that the SIRC\_CTL[SIRCTRIM] does not reflect the current trim value until you have written to this field. Pay particular attention to this feature when you initiate a read-modify-write operation on SIRC\_CTL, because a SIRCTRIM value of 0 may be unintentionally written back, and this may alter the SIRC frequency. In this case, you should calibrate the SIRC using the CMU or be sure that you only write to the upper 16 bits of this SIRC\_CTL.

In this oscillator, two's complement trimming method is implemented. So the trimming code increases from -16 to 15. As the trimming code increases, the internal time constant increases and frequency reduces. Please refer to the device datasheet for average frequency variation of the trimming step.

### 6.5.3 Register description

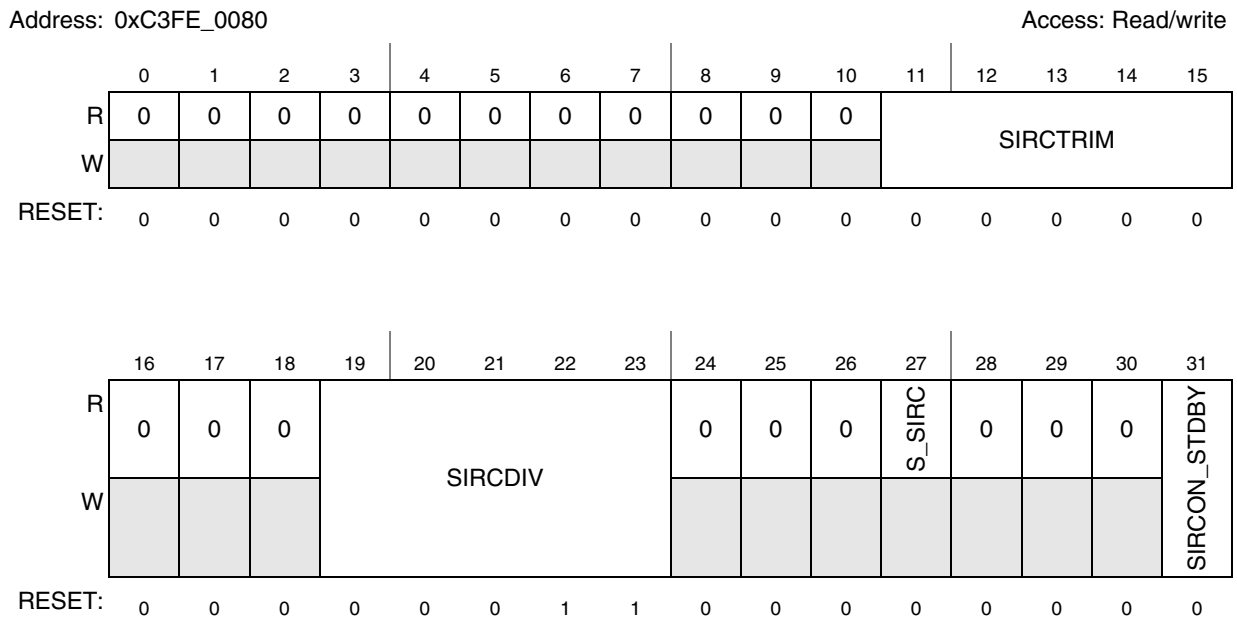


Figure 6-4. Low Power RC Control Register (SIRC\_CTL)



Table 6-6. SIRC\_CTL field descriptions

| Field        | Description  |
|--------------|--|
| SIRCTRIM     | SIRC trimming bits.<br>This field corresponds (via twos complement) to a trim factor of $-16$ to $+15$ .<br>A $+1$ change in SIRCTRIM decreases the current frequency by $\Delta_{\text{SIRCTRIM}}$ (see the device data sheet).<br>A $-1$ change in SIRCTRIM increases the current frequency by $\Delta_{\text{SIRCTRIM}}$ (see the device data sheet). |
| SIRCDIV      | SIRC clock division factor.<br>This field specifies the SIRC oscillator output clock division factor. The output clock is divided by the factor SIRCDIV+1.   |
| S_SIRC       | SIRC clock status.<br>0 SIRC is not providing a stable clock.<br>1 SIRC is providing a stable clock.   |
| SIRCON_STDBY | SIRC control in STANDBY mode.<br>0 SIRC is switched off in STANDBY mode.<br>1 SIRC is switched on in STANDBY mode.   |

## 6.6 Fast internal RC oscillator (FIRC) digital interface

### 6.6.1 Introduction

The FIRC digital interface controls the 16 MHz fast internal RC oscillator (FIRC). It holds control and status registers accessible for application.

### 6.6.2 Functional description

The FIRC provides a high frequency ( $f_{\text{FIRC}}$ ) clock of 16 MHz. This clock can be used to accelerate the exit from reset and wakeup sequence from low power modes of the system. It is controlled by the MC\_ME module based on the current device mode. The clock source status is updated in ME\_GS[S\_RC]. Please see [Chapter 8, Mode Entry Module \(MC\\_ME\)](#), for further details.

The FIRC can be further divided by a configurable division factor in the range from 1 to 32 to generate the divided clock to match system requirements. This division factor is specified by RC\_CTL[RCDIV] bits.

The FIRC output frequency can be trimmed using FIRC\_CTL[FIRCTRIM]. After a power-on reset, the FIRC is trimmed using a factory test value stored in test flash memory. However, after a power-on reset the test flash memory value is not visible at FIRC\_CTL[FIRCTRIM], and this field will show a value of 0. Therefore, be aware that the FIRC\_CTL[FIRCTRIM] field does not reflect the current trim value until you have written to it. Pay particular attention to this feature when you initiate a read-modify-write operation on FIRC\_CTL, because a FIRCTRIM value of zero may be unintentionally written back and this may alter the FIRC frequency. In this case, you should calibrate the FIRC using the CMU or ensure that you write only to the upper 16 bits of this FIRC\_CTL.

In this oscillator, two's complement trimming method is implemented. So the trimming code increases from  $-32$  to  $31$ . As the trimming code increases, the internal time constant increases and frequency reduces. Please refer to the device datasheet for average frequency variation of the trimming step.

During STANDBY mode entry process, the FIRC is controlled based on ME\_STANDBY\_MC[RCON] bit. This is the last step in the standby entry sequence. On any system wakeup event, the device exits STANDBY mode and switches on the FIRC. The actual powerdown status of the FIRC when the device is in standby is provided by RC\_CTL[FIRCON\_STDBY] bit.

### 6.6.3 Register description

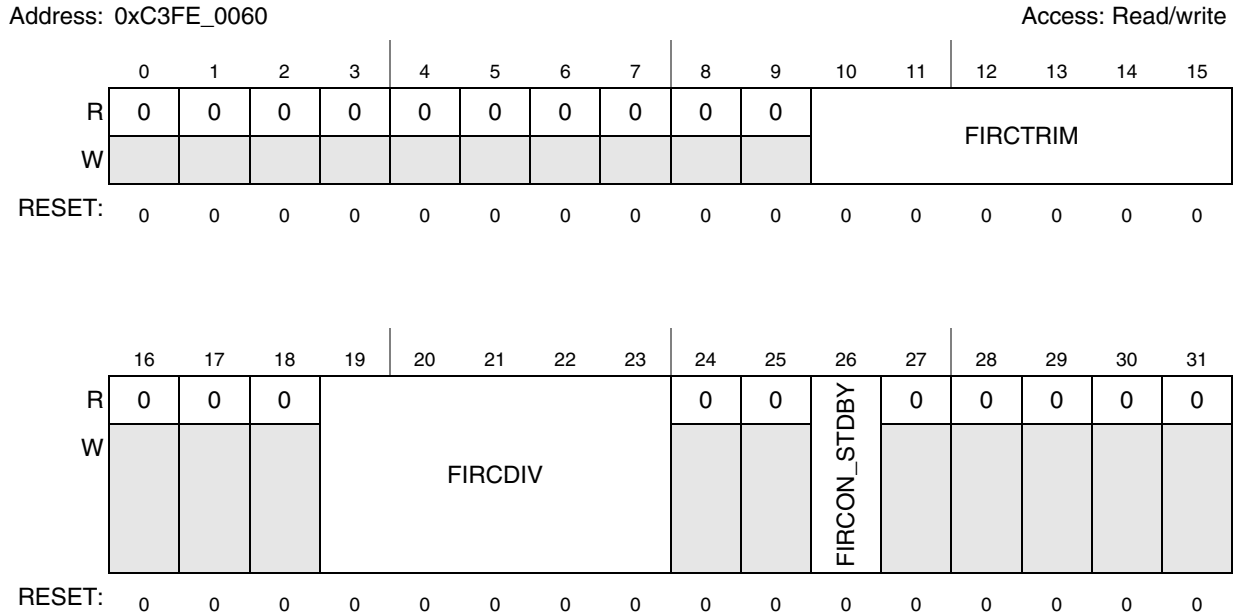


Figure 6-5. FIRC Oscillator Control Register (FIRC\_CTL)

Table 6-7. FIRC\_CTL field descriptions

| Field        | Description   |
|--------------|---|
| FIRCTRIM     | FIRC trimming bits.<br>This field corresponds (via twos complement) to a trim factor of -16 to +15.<br>A +1 change in FIRCTRIM decreases the current frequency by $\Delta_{\text{FIRCTRIM}}$ (see the device data sheet).<br>A -1 change in SIRCTRIM increases the current frequency by $\Delta_{\text{FIRCTRIM}}$ (see the device data sheet). |
| FIRCDIV      | FIRC clock division factor.<br>This field specifies the FIRC oscillator output clock division factor. The output clock is divided by the factor FIRCDIV+1.  |
| FIRCON_STDBY | FIRC control in STANDBY mode.<br>0 FIRC is switched off in STANDBY mode.<br>1 FIRC is in STANDBY mode.  |

## 6.7 Frequency-modulated phase-locked loop (FMPLL)

### 6.7.1 Introduction

This section describes the features and functions of the FMPLL module implemented in the device.

### 6.7.2 Overview

The FMPLL enables the generation of high speed system clocks from a common 4–16 MHz input clock. Further, the FMPLL supports programmable frequency modulation of the system clock. The FMPLL multiplication factor and output clock divider ratio are all software configurable.

MPC5606BK has one FMPLL that can generate the system clock and takes advantage of the FM mode.

#### NOTE

The user must take care not to program the device with a frequency higher than allowed (no hardware check).

The FMPLL block diagram is shown in [Figure 6-6](#).

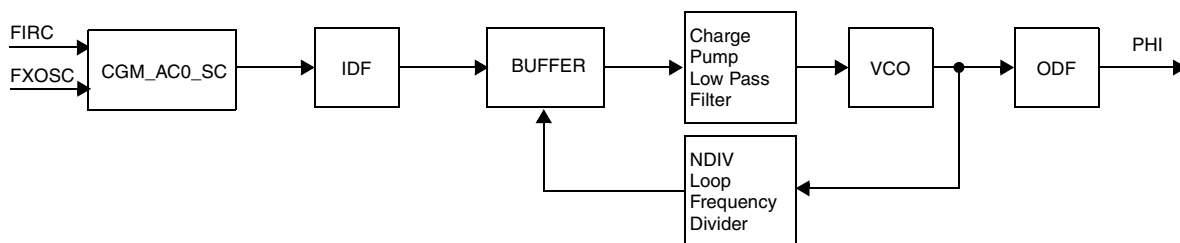


Figure 6-6. FMPLL block diagram

### 6.7.3 Features

The FMPLL has the following major features:

- Input clock frequency 4 MHz – 16 MHz
- Voltage controlled oscillator (VCO) range from 256 MHz to 512 MHz
- Frequency divider (FD) for reduced frequency operation without forcing the FMPLL to relock
- Frequency modulated FMPLL
  - Modulation enabled/disabled through software
  - Triangle wave modulation
- Programmable modulation depth
  - $\pm 0.25\%$  to  $\pm 4\%$  deviation from center spread frequency<sup>1</sup>
  - $-0.5\%$  to  $+8\%$  deviation from down spread frequency
  - Programmable modulation frequency dependent on reference frequency
- Self-locked mode (SCM) operation

1. Spread spectrum should be programmed in line with maximum datasheet frequency figures.

- 4 available modes
  - Normal mode
  - Progressive clock switching
  - Normal mode with frequency modulation
  - Powerdown mode

### 6.7.4 Memory map<sup>1</sup>

Table 6-8 shows the memory map of the FMPLL.

Table 6-8. FMPLL memory map

| Base address: 0xC3FE_00A0 |                          |                             |
|---------------------------|--------------------------|-----------------------------|
| Address offset            | Register                 | Location                    |
| 0x0                       | Control Register (CR)    | <a href="#">on page 116</a> |
| 0x4                       | Modulation Register (MR) | <a href="#">on page 119</a> |

### 6.7.5 Register description

The FMPLL operation is controlled by two registers. Those registers can be accessed and written in supervisor mode only.

#### 6.7.5.1 Control Register (CR)

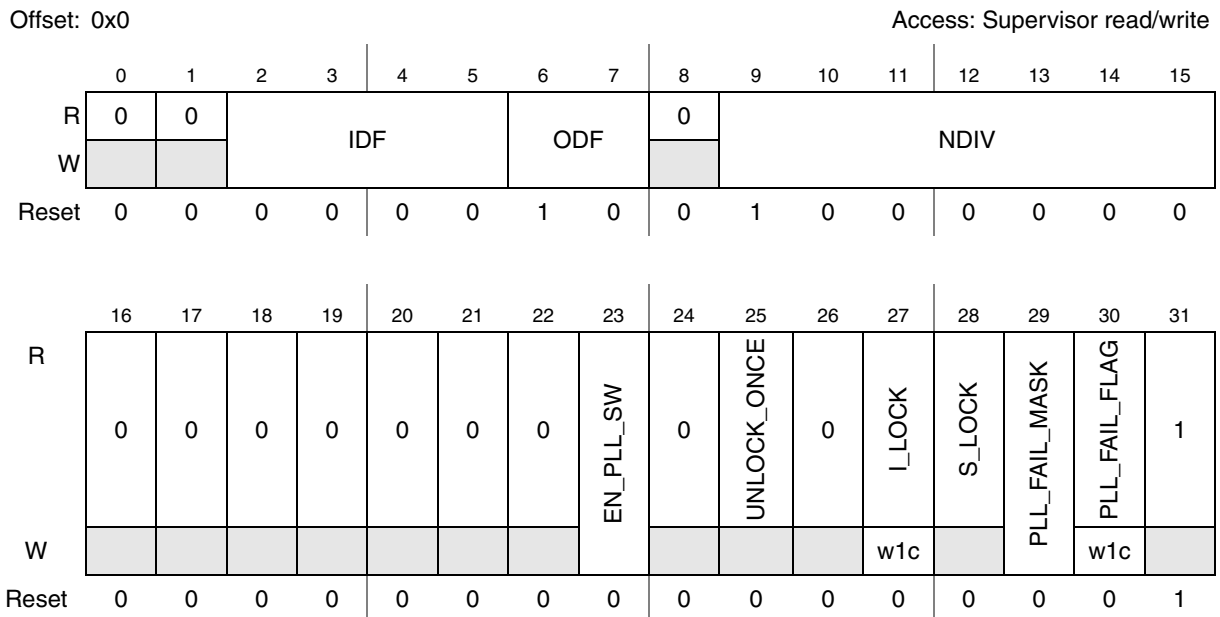


Figure 6-7. Control Register (CR)

1. FMPLL\_x are mapped through the ME\_CGM register slot

Table 6-9. CR field descriptions

| Field         | Description  |
|---------------|--|
| IDF           | The value of this field sets the FMPLL input division factor as described in Table 6-10.   |
| ODF           | The value of this field sets the FMPLL output division factor as described in Table 6-11.  |
| NDIV          | The value of this field sets the FMPLL loop division factor as described in Table 6-12.  |
| EN_PLL_SW     | This bit is used to enable progressive clock switching. After the PLL locks, the PLL output initially is divided by 8, and then progressively decreases until it reaches divide-by-1.<br>0 Progressive clock switching disabled.<br>1 Progressive clock switching enabled.<br><b>Note:</b> Progressive clock switching should not be used if a non-changing clock is needed, such as for serial communications, until the division has finished. |
| UNLOCK_ONCE   | This bit is a sticking indication of FMPLL loss of lock condition. UNLOCK_ONCE is set when the FMPLL loses lock. Whenever the FMPLL reacquires lock, UNLOCK_ONCE remains set. Only a power-on reset clears this bit.   |
| I_LOCK        | This bit is set by hardware whenever there is a lock/unlock event.   |
| S_LOCK        | This bit is an indication of whether the FMPLL has acquired lock.<br>0: FMPLL unlocked<br>1: FMPLL locked<br><b>Note:</b>  |
| PLL_FAIL_MASK | This bit is used to mask the pll_fail output.<br>0 pll_fail not masked.<br>1 pll_fail masked.  |
| PLL_FAIL_FLAG | This bit is asynchronously set by hardware whenever a loss of lock event occurs while FMPLL is switched on. It is cleared by software writing 1.   |

Table 6-10. Input divide ratios

| IDF[3:0] | Input divide ratios |
|----------|---------------------|
| 0000     | Divide by 1         |
| 0001     | Divide by 2         |
| 0010     | Divide by 3         |
| 0011     | Divide by 4         |
| 0100     | Divide by 5         |
| 0101     | Divide by 6         |
| 0110     | Divide by 7         |
| 0111     | Divide by 8         |
| 1000     | Divide by 9         |
| 1001     | Divide by 10        |
| 1010     | Divide by 11        |
| 1011     | Divide by 12        |
| 1100     | Divide by 13        |

**Table 6-10. Input divide ratios (continued)**

| IDF[3:0] | Input divide ratios |
|----------|---------------------|
| 1101     | Divide by 14        |
| 1110     | Divide by 15        |
| 1111     | Clock Inhibit       |

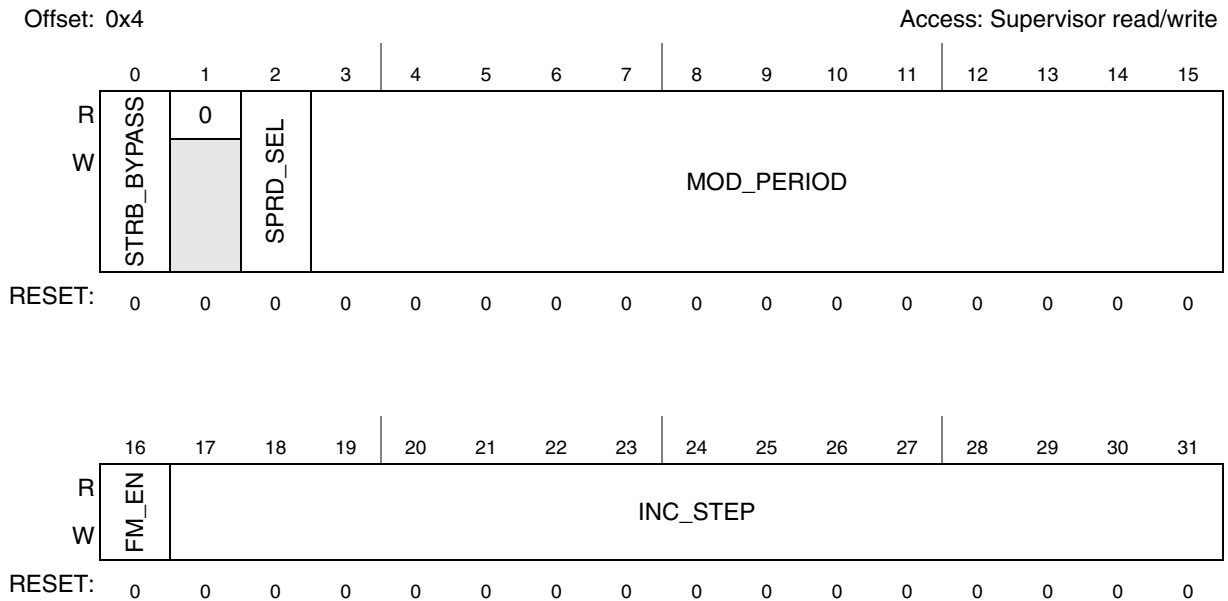
**Table 6-11. Output divide ratios**

| ODF[1:0] | Output divide ratios |
|----------|----------------------|
| 00       | Divide by 2          |
| 01       | Divide by 4          |
| 10       | Divide by 8          |
| 11       | Divide by 16         |

**Table 6-12. Loop divide ratios**

| NDIV[6:0]       | Loop divide ratios |
|-----------------|--------------------|
| 0000000–0011111 | —                  |
| 0100000         | Divide by 32       |
| 0100001         | Divide by 33       |
| 0100010         | Divide by 34       |
| ...             | ...                |
| 1011111         | Divide by 95       |
| 1100000         | Divide by 96       |
| 1100001–1111111 | —                  |

## 6.7.5.2 Modulation Register (MR)



**Figure 6-8. Modulation Register (MR)**

**Table 6-13. MR field descriptions**

| Field       | Description   |
|-------------|---|
| STRB_BYPASS | Strobe bypass.<br>The STRB_BYPASS signal is used to bypass the strobe signal used inside FMPLL to latch the correct values for control bits (INC_STEP, MOD_PERIOD, and SPRD_SEL).<br>0 Strobe is used to latch FMPLL modulation control bits<br>1 Strobe is bypassed. In this case control bits need to be static. The control bits must be changed only when FMPLL is in powerdown mode. |
| SPRD_SEL    | Spread type selection.<br>The SPRD_SEL controls the spread type in Frequency Modulation mode.<br>0 Center SPREAD<br>1 Down SPREAD   |
| MOD_PERIOD  | Modulation period.<br>The MOD_PERIOD field is the binary equivalent of the value modperiod derived from following formula: $\text{modperiod} = \frac{t_{\text{ref}}}{4 \times f_{\text{mod}}}$ where:<br>$f_{\text{ref}}$ : represents the frequency of the feedback divider<br>$f_{\text{mod}}$ : represents the modulation frequency  |

Table 6-13. MR field descriptions (continued)

| Field    | Description   |
|----------|---|
| FM_EN    | Frequency Modulation Enable. The FM_EN enables the frequency modulation.<br>0 Frequency modulation disabled<br>1 Frequency modulation enabled   |
| INC_STEP | Increment step.<br>The INC_STEP field is the binary equivalent of the value incstep derived from following formula:<br>$\text{incstep} = \text{round}\left(\frac{(2^{15} - 1) \times \text{md} \times \text{MDF}}{100 \times 5 \times \text{MODPERIOD}}\right)$<br>where:<br><i>md</i> : represents the peak modulation depth in percentage (Center spread — pk-pk = ±md, Downspread — pk-pk = -2 × md)<br><i>MDF</i> : represents the nominal value of loop divider (CR[NDIV]) |

## 6.7.6 Functional description

### 6.7.6.1 Normal mode

In Normal Mode the FMPLL inputs are driven by the CR. This means that, when the FMPLL is in lock state, the FMPLL output clock (PHI) is derived by the reference clock (CLKIN) through this relation:

$$\text{phi} = \frac{\text{clk}_{\text{in}} \cdot \text{NDIV}}{\text{IDF} \cdot \text{ODF}}$$

where the value of IDF, NDIV, and ODF are set in the CR and can be derived from [Table 6-10](#), [Table 6-11](#), and [Table 6-12](#).

Table 6-14. FMPLL lookup table

| Crystal frequency (MHz) | FMPLL output frequency (MHz) | CR field values |     |      | VCO frequency (MHz) |
|-------------------------|------------------------------|-----------------|-----|------|---------------------|
|                         |                              | IDF             | ODF | NDIV |                     |
| 8                       | 32                           | 0               | 2   | 32   | 256                 |
|                         | 64                           | 0               | 2   | 64   | 512                 |
|                         | 80                           | 0               | 1   | 40   | 320                 |
| 16                      | 32                           | 1               | 2   | 32   | 256                 |
|                         | 64                           | 1               | 2   | 64   | 512                 |
|                         | 80                           | 1               | 1   | 40   | 320                 |
| 40                      | 32                           | 4               | 2   | 32   | 256                 |
|                         | 64                           | 4               | 2   | 64   | 512                 |
|                         | 80                           | 3               | 1   | 32   | 320                 |



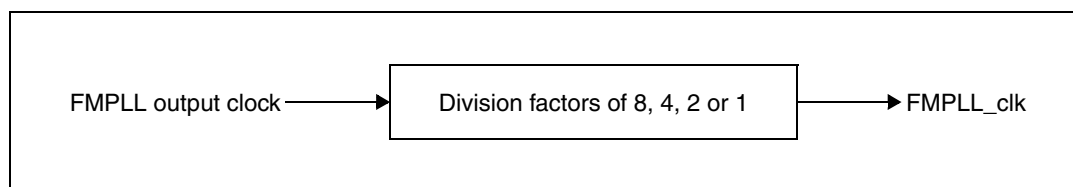
### 6.7.6.2 Progressive clock switching

Progressive clock switching allows to switch the system clock to FMPLL output clock stepping through different division factors. This means that the current consumption gradually increases and, in turn, voltage regulator response is improved.

This feature can be enabled by programming CR[EN\_PLL\_SW] bit. When enabled, the system clock is switched to divided PHI. The FMPLL\_clk divider is then progressively decreased to the target divider as shown in Table 6-15.

**Table 6-15. Progressive clock switching on pll\_select rising edge**

| Number of FMPLL output clock cycles | FMPLL_clk frequency<br>(FMPLL output clock frequency) |
|-------------------------------------|---|
| 8                                   | (FMPLL output clock frequency)/8                      |
| 16                                  | (FMPLL output clock frequency)/4                      |
| 32                                  | (FMPLL output clock frequency)/2                      |
| onward                              | FMPLL output clock frequency                          |



**Figure 6-9. FMPLL output clock division flow during progressive switching**

### 6.7.6.3 Normal mode with frequency modulation

The FMPLL default mode is without frequency modulation enabled. When frequency modulation is enabled, however, two parameters must be set to generate the desired level of modulation: the PERIOD and the STEP. The modulation waveform is always a triangle wave and its shape is not programmable.

FM mode is activated in two steps:

1. Configure the FM mode characteristics: MOD\_PERIOD, INC\_STEP.
2. Enable the FM mode by programming bit FM\_EN of the MR to 1. FM mode can only be enabled when FMPLL is in lock state.

There are two ways to latch these values inside the FMPLL, depending on the value of bit STRB\_BYPASS in the MR.

If STRB\_BYPASS is low, the modulation parameters are latched in the FMPLL only when the strobe signal goes high for at least two cycles of CLKIN clock. The strobe signal is automatically generated in the FMPLL digital interface when the modulation is enabled (FM\_EN goes high) if the FMPLL is locked (S\_LOCK = 1), or when the modulation has been enabled (FM\_EN = 1) and FMPLL enters lock state (S\_LOCK goes high).

If STRB\_BYPASS is high, the strobe signal is bypassed. In this case, control bits (MOD\_PERIOD[12:0], INC\_STEP[14:0], SPREAD\_CONTROL) need to be static or hardwired to constant values. The control bits must be changed only when the FMPLL is in powerdown mode.

The modulation depth in % is

$$\text{ModulationDepth} = \left( \frac{100 \times 5 \times \text{INCSTEP} \times \text{MODPERIOD}}{(2^{15} - 1) \times \text{MDF}} \right)$$

#### NOTE

The user must ensure that the product of INCSTEP and MODPERIOD is less than  $(2^{15} - 1)$ .



Figure 6-10. Frequency modulation

#### 6.7.6.4 Powerdown mode

To reduce consumption, the FMPLL can be switched off when not required by programming the registers ME\_x\_MC on the MC\_ME module.

#### 6.7.7 Recommendations

To avoid any unpredictable behavior of the FMPLL clock, it is recommended to follow these guidelines:

- The FMPLL VCO frequency should reside in the range 256 MHz to 512 MHz. Care is required when programming the multiplication and division factors to respect this requirement.
- Once the PLL has been locked, only the output divider can be changed.
- Use PLL progressive clock switching to ramp system clock (/8, /4, /2, /1) automatically for the case when PLL is enabled and selected as system clock.

- MOD\_PERIOD, INC\_STEP, SPREAD\_SEL bits should be modified before activating the FM mode. Then strobe has to be generated to enable the new settings. If STRB\_BYP is set to 1 then MOD\_PERIOD, INC\_STEP, and SPREAD\_SEL can be modified only when FMPLL is in powerdown mode.

## 6.8 Clock monitor unit (CMU)

### 6.8.1 Introduction

The Clock Monitor Unit (CMU), also referred to as Clock Quality Checker or Clock Fault Detector, serves two purposes. The main task is to permanently supervise the integrity of the various clock sources, for example a crystal oscillator or FMPLL. In case the FMPLL leaves an upper or lower frequency boundary or the crystal oscillator fails it can detect and forward these kind of events to the MC\_ME and the MC\_CGM. The clock management unit in turn can then switch to a SAFE mode where it uses the default safe clock source (FIRC), resets the device, or generates the interrupt according to the system needs.

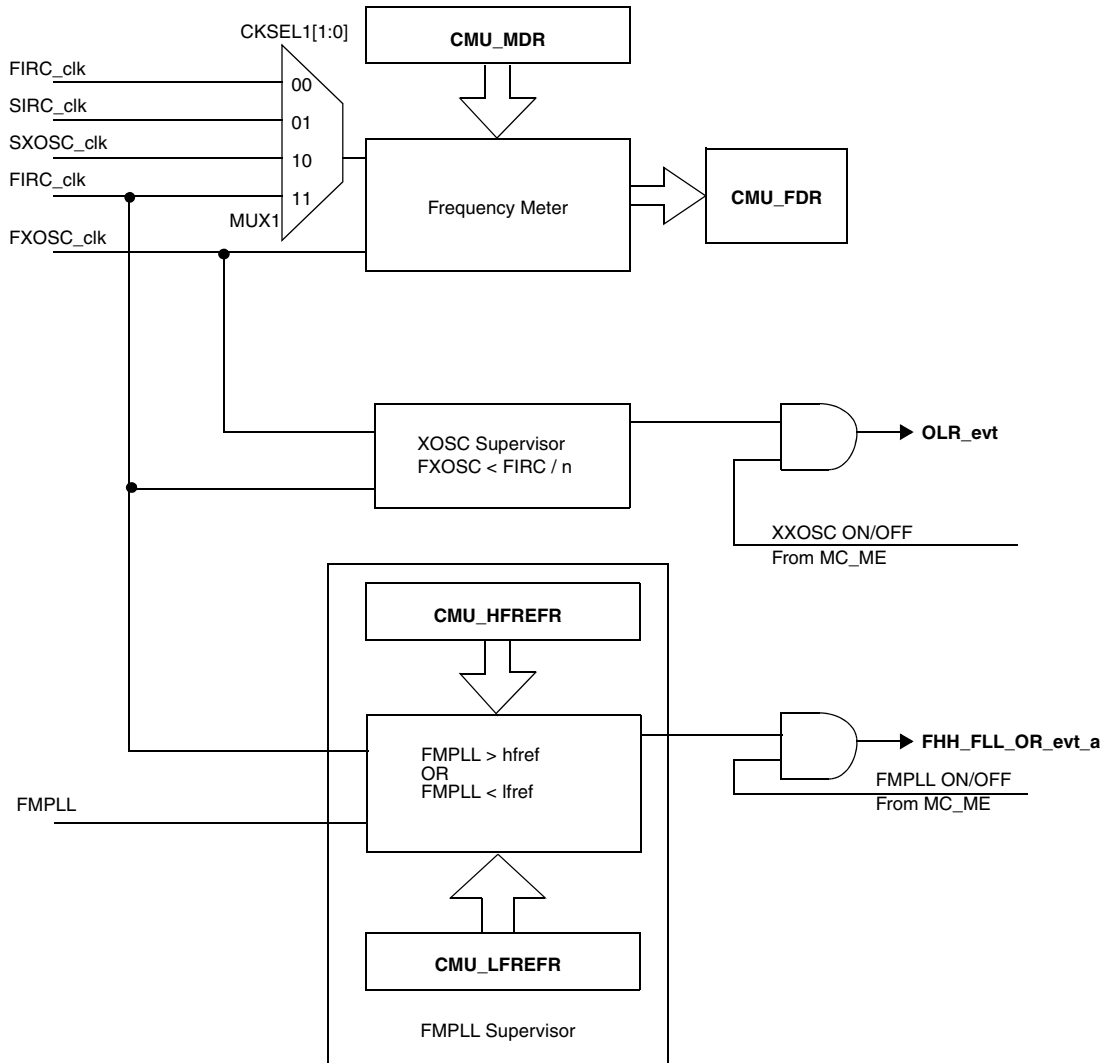
It can also monitor the external crystal oscillator clock, which must be greater than the internal RC clock divided by a division factor given by CMU\_CSR[RCDIV], and generates a system clock transition request or an interrupt when enabled.

The second task of the CMU is to provide a frequency meter, which allows to measure the frequency of one clock source vs. a reference clock. This is useful to allow the calibration of the on-chip RC oscillator(s), as well as being able to correct/calculate the time deviation of a counter that is clocked by the RC oscillator.

### 6.8.2 Main features

- FIRC, SIRC, SXOSC oscillator frequency measurement using FXOSC as reference clock
- External oscillator clock monitoring with respect to FIRC\_clk/n clock
- FMPLL clock frequency monitoring for a high and low frequency range with FIRC as reference clock
- Event generation for various failures detected inside monitoring unit

### 6.8.3 Block diagram



**OLR\_evt** : It is the event signalling XOSC failure when asserted. When this signal is asserted, RGM may generate reset, interrupt or SAFE request based on the RGM configuration.

**FHH\_FLL\_OR\_evt\_a** : It is the event signalling FMPLL failure when asserted. Based on the CMU\_HFREFR and CMU\_LFREFR configuration, if the FMPLL is greater than high frequency range or less than the low frequency range configuration, this signal is generated. When this signal is asserted, RGM may generate reset, interrupt or SAFE request based on the RGM configuration.

**Figure 6-11. Clock Monitor Unit diagram**

### 6.8.4 Functional description

The clock and frequency names referenced in this block are defined as follows:

- FXOSC\_clk: clock coming from the fast external crystal oscillator
- SXOSC\_clk: clock coming from the slow external crystal oscillator
- SIRC\_clk: clock coming from the slow (low frequency) internal RC oscillator

- FIRC\_clk: clock coming from the fast (high frequency) internal RC oscillator
- FMPLL\_clk: clock coming from the FMPLL
- $f_{\text{FXOSC\_clk}}$ : frequency of fast external crystal oscillator clock
- $f_{\text{SXOSC\_clk}}$ : frequency of slow external crystal oscillator clock
- $f_{\text{SIRC\_clk}}$ : frequency of slow (low frequency) internal RC oscillator
- $f_{\text{FIRC\_clk}}$ : frequency of fast (high frequency) internal RC oscillator
- $f_{\text{FMPLL\_clk}}$ : frequency of FMPLL clock

#### 6.8.4.1 Crystal clock monitor

If  $f_{\text{FXOSC\_clk}}$  is less than  $f_{\text{FIRC\_clk}}$  divided by  $2^{\text{RCDIV}}$  bits of the CMU\_CSR and the FXOSC\_clk is ON as signaled by the MC\_ME, then:

- An event pending bit OLRI in CMU\_ISR is set.
- A failure event OLR is signaled to the MC\_RGM, which in turn can automatically switch to a safe fallback clock and generate an interrupt or reset.

#### NOTE

Functional FXOSC monitoring can only be guaranteed when the FXOSC frequency is greater than  $(\text{FIRC} / 2^{\text{RCDIV}}) + 0.5 \text{ MHz}$ .

#### 6.8.4.2 FMPLL clock monitor

The  $f_{\text{FMPLL\_clk}}$  can be monitored by programming bit CME of the CMU\_CSR register to 1. The FMPLL\_clk monitor starts as soon as bit CME is set. This monitor can be disabled at any time by writing bit CME to 0.

If  $f_{\text{FMPLL\_clk}}$  is greater than a reference value determined by bits HFREF[11:0] of the CMU\_HFREFR and the FMPLL\_clk is ON, as signaled by the MC\_ME, then:

- An event pending bit FHHI in CMU\_ISR is set.
- A failure event is signaled to the MC\_RGM, which in turn can generate an interrupt or safe mode request or functional reset, depending on the programming model.

If  $f_{\text{FMPLL\_clk}}$  is less than a reference value determined by bits LFREF[11:0] of the CMU\_LFREFR and the FMPLL\_clk is ON, as signaled by the MC\_ME, then:

- An event pending bit FLLI in CMU\_ISR is set.
- A failure event FLL is signaled to the MC\_RGM, which in turn can generate an interrupt or safe mode request or functional reset, depending on the programming model.

#### NOTE

Functional FMPLL monitoring can only be guaranteed when the FMPLL frequency is greater than  $(\text{FIRC} / 4) + 0.5 \text{ MHz}$ .

**NOTE**

The internal RC oscillator is used as reliable reference clock for the clock supervision. In order to avoid false events, proper programming of the dividers is required. These have to take into account the accuracy and frequency deviation of the internal RC oscillator.

**NOTE**

If PLL frequency goes out of range, the CMU shall generate FMPLL flf/fhh event. It takes approximately 5  $\mu$ s to generate this event.

**6.8.4.3 Frequency meter**

The purpose of the frequency meter is twofold:

- To measure the frequency of the oscillators SIRC, FIRC or SXOSC
- To calibrate an internal RC oscillator (SIRC or FIRC) using a known frequency

**Hint:** This value can then be stored into the flash so that application software can reuse it later on.

The reference clock is always the FXOSC\_clk. The frequency meter returns a precise value of frequencies  $f_{SXOSC\_clk}$ ,  $f_{FIRC\_clk}$  or  $f_{SIRC\_clk}$  according to CKSEL1 bit value. The measure starts when bit SFM (Start Frequency Measure) in the CMU\_CSR is set to 1. The measurement duration is given by the CMU\_MDR in numbers of clock cycles of the selected clock source with a width of 20 bits. Bit SFM is reset to 0 by hardware once the frequency measurement is done, and the count is loaded in the CMU\_FDR. The frequency  $f_x^1$  can be derived from the value loaded in the CMU\_FDR as follows:

$$f_x = (f_{FXOSC} \times MD) / n \quad \text{Eqn. 6-1}$$

where n is the value in the CMU\_FDR and MD is the value in the CMU\_MDR.

The frequency meter by default evaluates  $f_{FIRC\_clk}$ , but software can swap to  $f_{SIRC\_clk}$  or  $f_{SXOSC\_clk}$  by programming the CKSEL bits in the CMU\_CSR.

**6.8.5 Memory map and register description**

The memory map of the CMU is shown in [Table 6-16](#).

**Table 6-16. CMU memory map**

| Base address: 0xC3FE_0100                            |                |             |                             |
|--|----------------|-------------|-----------------------------|
| Register name  | Address offset | Reset value | Location                    |
| Control Status Register (CMU_CSR)                    | 0x00           | 0x00000006  | <a href="#">on page 127</a> |
| Frequency Display Register (CMU_FDR)                 | 0x04           | 0x00000000  | <a href="#">on page 128</a> |
| High Frequency Reference Register FMPLL (CMU_HFREFR) | 0x08           | 0x00000FFF  | <a href="#">on page 128</a> |
| Low Frequency Reference Register FMPLL (CMU_LFREFR)  | 0x0C           | 0x00000000  | <a href="#">on page 129</a> |
| Interrupt Status Register (CMU_ISR)                  | 0x10           | 0x00000000  | <a href="#">on page 129</a> |

1. x = FIRC, SIRC or SXOSC

Table 6-16. CMU memory map (continued)

| Base address: 0xC3FE_0100               |                |             |             |
|---|----------------|-------------|-------------|
| Register name                           | Address offset | Reset value | Location    |
| Reserved                                | 0x14           | 0x00000000  | —           |
| Measurement Duration Register (CMU_MDR) | 0x18           | 0x00000000  | on page 130 |

### 6.8.5.1 Control Status Register (CMU\_CSR)

Offset: 0x00 Access: Read/write

|       |    |    |    |    |    |    |        |    |                  |    |    |    |    |       |    |       |
|-------|----|----|----|----|----|----|--------|----|------------------|----|----|----|----|-------|----|-------|
|       | 0  | 1  | 2  | 3  | 4  | 5  | 6      | 7  | 8                | 9  | 10 | 11 | 12 | 13    | 14 | 15    |
| R     | 0  | 0  | 0  | 0  | 0  | 0  | 0      | 0  | SFM <sup>1</sup> | 0  | 0  | 0  | 0  | 0     | 0  | 0     |
| W     |    |    |    |    |    |    |        |    |                  |    |    |    |    |       |    |       |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0      | 0  | 0                | 0  | 0  | 0  | 0  | 0     | 0  | 0     |
|       | 16 | 17 | 18 | 19 | 20 | 21 | 22     | 23 | 24               | 25 | 26 | 27 | 28 | 29    | 30 | 31    |
| R     | 0  | 0  | 0  | 0  | 0  | 0  | CKSEL1 |    | 0                | 0  | 0  | 0  | 0  | RCDIV |    | CME_A |
| W     |    |    |    |    |    |    |        |    |                  |    |    |    |    |       |    |       |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0      |    | 0                | 0  | 0  | 0  | 0  | 1     |    | 1     |

Figure 6-12. Control Status Register (CMU\_CSR)

<sup>1</sup> You can read this field, and you can write a value of “1” to it. Writing a “0” has no effect. A reset will also clear this bit.

Table 6-17. CMU\_CSR field descriptions

| Field  | Description  |
|--------|--|
| SFM    | Start frequency measure.<br>The software can only set this bit to start a clock frequency measure. It is reset by hardware when the measure is ready in the CMU_FDR register.<br>0 Frequency measurement completed or not yet started.<br>1 Frequency measurement not completed.   |
| CKSEL1 | Clock oscillator selection bit.<br>CKSEL1 selects the clock to be measured by the frequency meter.<br>00 FIRC_clk selected.<br>01 SIRC_clk selected.<br>10 SXOSC_clk selected.<br>11 FIRC_clk selected.  |
| RCDIV  | RC clock division factor.<br>These bits specify the RC clock division factor. The output clock is FIRC_clk divided by the factor $2^{RCDIV}$ . This output clock is used to compare with FXOSC_clk for crystal clock monitor feature. The clock division coding is as follows.<br>00 Clock divided by 1 (No division)<br>01 Clock divided by 2<br>10 Clock divided by 4<br>11 Clock divided by 8 |
| CME_A  | FMPLL_0 clock monitor enable.<br>0 FMPLL_0 monitor disabled.<br>1 FMPLL_0 monitor enabled.   |

### 6.8.5.2 Frequency Display Register (CMU\_FDR)

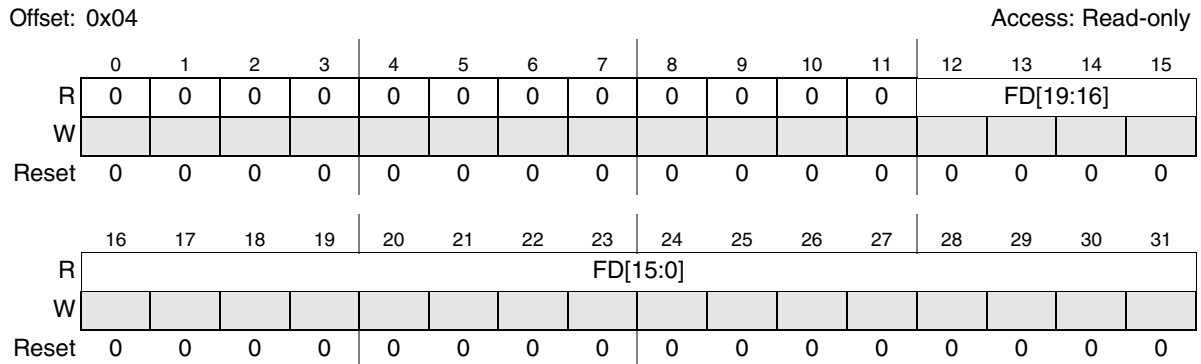


Figure 6-13. Frequency Display Register (CMU\_FDR)

Table 6-18. CMU\_FDR field descriptions

| Field | Description   |
|-------|---|
| FD    | Measured frequency bits.<br>This register shows the measured frequency $f_x$ with respect to $f_{FXOSC}$ . The measured value is given by the following formula: $f_x = (f_{FXOSC} \times MD) / n$ , where $n$ is the value in CMU_FDR register.<br><b>Note:</b> $x = \text{FIRC, SIRC or SXOSC}$ . |

### 6.8.5.3 High Frequency Reference Register FMPLL (CMU\_HFREFR)

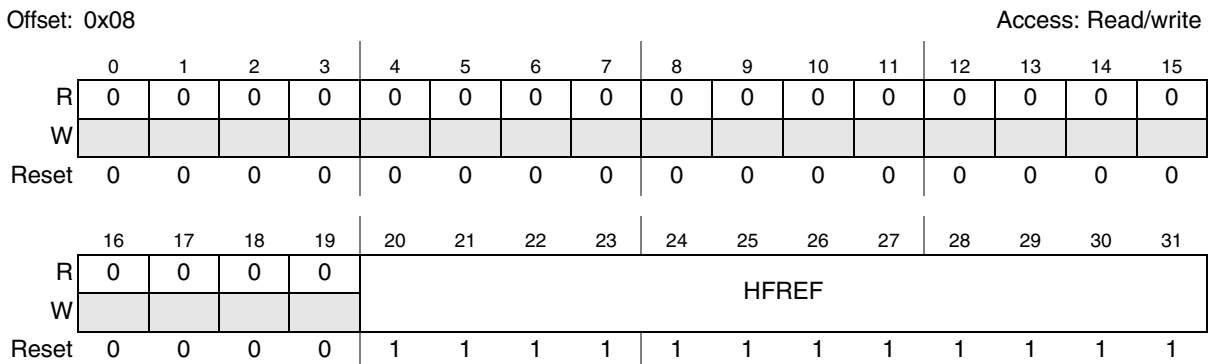


Figure 6-14. High Frequency Reference Register FMPLL (CMU\_HFREFR)

Table 6-19. CMU\_HFREFR field descriptions

| Field | Description  |
|-------|--|
| HFREF | High Frequency reference value.<br>This field determines the high reference value for the FMPLL clock. The reference value is given by: $(\text{HFREF} \div 16) \times (f_{\text{FIRC}} \div 4)$ . |



### 6.8.5.4 Low Frequency Reference Register FMPLL (CMU\_LFREFR)

Offset: 0x0C Access: Read/write

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       |    |    |    |    |       |    |    |    |    |    |    |    |    |    |    |    |
|-------|----|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|----|
|       | 16 | 17 | 18 | 19 | 20    | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| R     | 0  | 0  | 0  | 0  | LFREF |    |    |    |    |    |    |    |    |    |    |    |
| W     |    |    |    |    |       |    |    |    |    |    |    |    |    |    |    |    |
| Reset | 0  | 0  | 0  | 0  | 0     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |

Figure 6-15. Low Frequency Reference Register FMPLL (CMU\_LFREFR)

Table 6-20. CMU\_LFREFR field descriptions

| Field | Description   |
|-------|---|
| LFREF | Low Frequency reference value.<br>This field determines the low reference value for the FMPLL. The reference value is given by:<br>$(LFREF \div 16) \times (f_{FIRC} \div 4)$ . |

### 6.8.5.5 Interrupt Status Register (CMU\_ISR)

Offset: 0x10 Access: Read/write

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

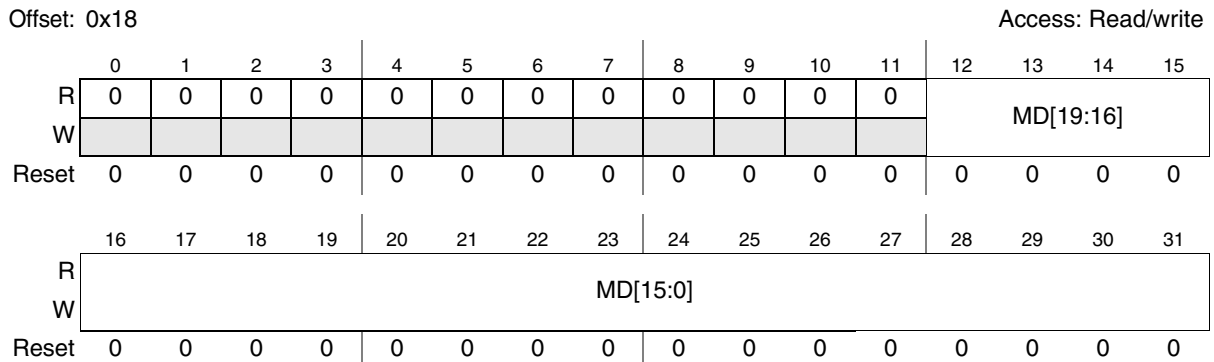
|       |    |    |    |    |    |    |    |    |    |    |    |    |     |     |      |      |
|-------|----|----|----|----|----|----|----|----|----|----|----|----|-----|-----|------|------|
|       | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28  | 29  | 30   | 31   |
| R     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | FIH | FLLI | OLRI |
| W     |    |    |    |    |    |    |    |    |    |    |    |    | w1c | w1c | w1c  |      |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 0   | 0    | 0    |

Figure 6-16. Interrupt status register (CMU\_ISR)

**Table 6-21. CMU\_ISR field descriptions**

| Field | Description   |
|-------|---|
| FHHI  | <p>FMPLL clock frequency higher than high reference interrupt.<br/>                     This bit is set by hardware when <math>f_{FMPLL\_clk}</math> becomes higher than HFREF value and FMPLL_clk is ON as signaled by the MC_ME. It can be cleared by software by writing 1.<br/>                     0 No FHH event.<br/>                     1 FHH event is pending.</p>                  |
| FLLI  | <p>FMPLL clock frequency lower than low reference event.<br/>                     This bit is set by hardware when <math>f_{FMPLL\_clk}</math> becomes lower than LFREF value and FMPLL_clk is ON as signaled by the MC_ME. It can be cleared by software by writing 1.<br/>                     0 No FLL event.<br/>                     1 FLL event is pending.</p>                         |
| OLRI  | <p>Oscillator frequency lower than RC frequency event.<br/>                     This bit is set by hardware when <math>f_{FXOSC\_clk}</math> is lower than <math>FIRC\_clk/2^{RCDIV}</math> frequency and FXOSC_clk is ON as signaled by the MC_ME. It can be cleared by software by writing 1.<br/>                     0 No OLR event.<br/>                     1 OLR event is pending.</p> |

**6.8.5.6 Measurement Duration Register (CMU\_MDR)**



**Figure 6-17. Measurement Duration Register (CMU\_MDR)**

**Table 6-22. CMU\_MDR field descriptions**

| Field | Description  |
|-------|--|
| MD    | <p>Measurement duration bits.<br/>                     This field shows the measurement duration in numbers of clock cycles of the selected clock source. This value is loaded in the frequency meter downcounter. When CMU_CSR[SFM] = 1, the downcounter starts counting.</p> |

# Chapter 7

## Clock Generation Module (MC\_CGM)

### 7.1 Overview

The clock generation module (MC\_CGM) generates reference clocks for all SoC blocks. The MC\_CGM selects one of the system clock sources to supply the system clock. The MC\_ME controls the system clock selection (see [Chapter 8, Mode Entry Module \(MC\\_ME\)](#), for more details). A set of MC\_CGM registers controls the clock dividers, which are utilized for divided system and peripheral clock generation. The memory spaces of system and peripheral clock sources that have addressable memory spaces are accessed through the MC\_CGM memory space. The MC\_CGM also selects and generates an output clock.

[Figure 7-1](#) shows the MC\_CGM block diagram.

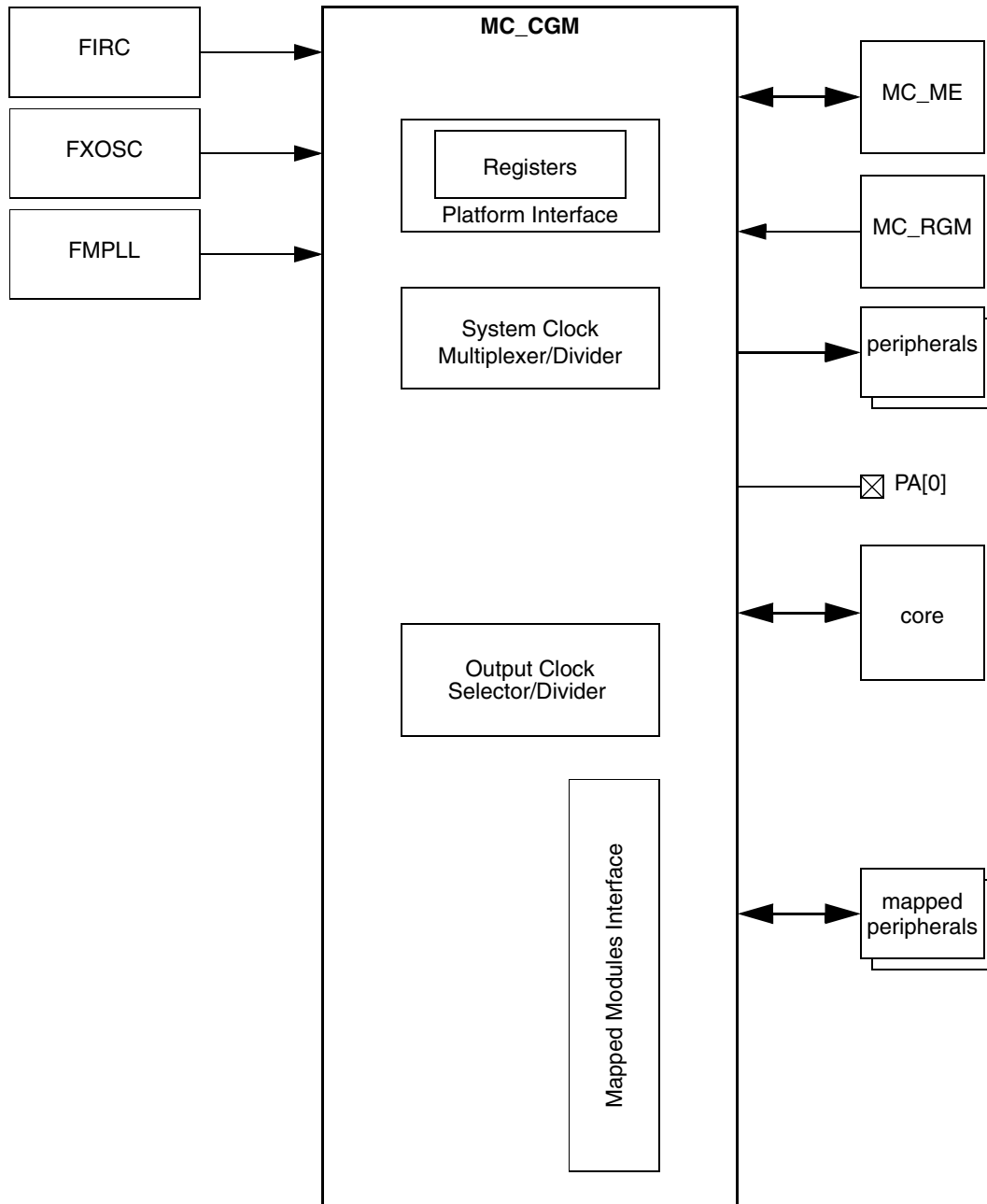


Figure 7-1. MC\_CGM block diagram

## 7.2 Features

The MC\_CGM includes the following features:

- Generates system and peripheral clocks
- Selects and enables/disables the system clock supply from system clock sources according to MC\_ME control

- Contains a set of registers to control clock dividers for divided clock generation
- Supports multiple clock sources and maps their address spaces to its memory map
- Generates an output clock
- Guarantees glitchless clock transitions when changing the system clock selection
- Supports 8-, 16-, and 32-bit wide read/write accesses

## 7.3 Modes of operation

This section describes the basic functional modes of the MC\_CGM.

### 7.3.1 Normal and reset modes of operation

During normal and reset modes of operation, the clock selection for the system clock is controlled by the MC\_ME.

## 7.4 External signal description

The MC\_CGM delivers an output clock to the PA[0] pin for off-chip use and/or observation.

## 7.5 Memory map and register definition

Table 7-1. MC\_CGM register description

| Address     | Name        | Description                          | Size | Access |            |            | Location                    |
|-------------|-------------|--------------------------------------|------|--------|------------|------------|-----------------------------|
|             |             |                                      |      | Normal | Supervisor | Test       |                             |
| 0xC3FE_0370 | CGM_OC_EN   | Output Clock Enable                  | word | read   | read/write | read/write | <a href="#">on page 138</a> |
| 0xC3FE_0374 | CGM_OCDS_SC | Output Clock Division Select         | byte | read   | read/write | read/write | <a href="#">on page 138</a> |
| 0xC3FE_0378 | CGM_SC_SS   | System Clock Select Status           | byte | read   | read       | read       | <a href="#">on page 139</a> |
| 0xC3FE_037C | CGM_SC_DC0  | System Clock Divider Configuration 0 | byte | read   | read/write | read/write | <a href="#">on page 140</a> |
| 0xC3FE_037D | CGM_SC_DC1  | System Clock Divider Configuration 1 | byte | read   | read/write | read/write | <a href="#">on page 140</a> |
| 0xC3FE_037E | CGM_SC_DC2  | System Clock Divider Configuration 2 | byte | read   | read/write | read/write | <a href="#">on page 140</a> |
| 0xC3FE_0380 | CGM_AC0_SC  | Aux Clock 0 Select Control           | word | read   | read/write | read/write | <a href="#">on page 141</a> |

### NOTE

Any access to unused registers as well as write accesses to read-only registers will:

- Not change register content

- Cause a transfer error

**Table 7-2. MC\_CGM memory map**

| Address                           | Name            | 0  | 1  | 2  | 3  | 27 | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 | 14 | 15 |
|-----------------------------------|-----------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|                                   |                 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| 0xC3FE_0000<br>...<br>0xC3FE_001C | FXOSC registers |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0xC3FE_0020<br>...<br>0xC3FE_003C | reserved        |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0xC3FE_0040<br>...<br>0xC3FE_005C | SXOSC registers |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0xC3FE_0060<br>...<br>0xC3FE_007C | FIRC registers  |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0xC3FE_0080<br>...<br>0xC3FE_009C | SIRC registers  |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0xC3FE_00A0<br>...<br>0xC3FE_00BC | FMPLL registers |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0xC3FE_00C0<br>...<br>0xC3FE_00DC | reserved        |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0xC3FE_00E0<br>...<br>0xC3FE_00FC | reserved        |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0xC3FE_0100<br>...<br>0xC3FE_011C | CMU registers   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

Table 7-2. MC\_CGM memory map (continued)

| Address                                   | Name | 0        | 1  | 2  | 3  | 27 | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 | 14 | 15 |
|---|------|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|   |      | 16       | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| 0xC3FE<br>_0120<br>...<br>0xC3FE<br>_013C |      | reserved |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0xC3FE<br>_0140<br>...<br>0xC3FE<br>_015C |      | reserved |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0xC3FE<br>_0160<br>...<br>0xC3FE<br>_017C |      | reserved |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0xC3FE<br>_0180<br>...<br>0xC3FE<br>_019C |      | reserved |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0xC3FE<br>_01A0<br>...<br>0xC3FE<br>_01BC |      | reserved |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0xC3FE<br>_01C0<br>...<br>0xC3FE<br>_01DC |      | reserved |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0xC3FE<br>_01E0<br>...<br>0xC3FE<br>_01FC |      | reserved |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0xC3FE<br>_0200<br>...<br>0xC3FE<br>_021C |      | reserved |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0xC3FE<br>_0220<br>...<br>0xC3FE<br>_023C |      | reserved |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

Table 7-2. MC\_CGM memory map (continued)

| Address                           | Name | 0        | 1  | 2  | 3  | 27 | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 | 14 | 15 |
|-----------------------------------|------|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|                                   |      | 16       | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| 0xC3FE_0240<br>...<br>0xC3FE_025C |      | reserved |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0xC3FE_0260<br>...<br>0xC3FD_C27C |      | reserved |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0xC3FE_0280<br>...<br>0xC3FE_029C |      | reserved |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0xC3FE_02A0<br>...<br>0xC3FE_02BC |      | reserved |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0xC3FE_02C0<br>...<br>0xC3FE_02DC |      | reserved |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0xC3FE_02E0<br>...<br>0xC3FE_02FC |      | reserved |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0xC3FE_0300<br>...<br>0xC3FE_031C |      | reserved |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0xC3FE_0320<br>...<br>0xC3FE_033C |      | reserved |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0xC3FE_0340<br>...<br>0xC3FE_035C |      | reserved |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |



Table 7-2. MC\_CGM memory map (continued)

| Address                           | Name               | 0  |     | 1  |        | 2  |         | 3  |        | 27 |     | 5  |    | 6  |      | 7  |    | 8 |   | 9 |   | 10 |   | 11 |   | 12 |   | 13 |   | 14 |   | 15 |   |    |
|-----------------------------------|--------------------|----|-----|----|--------|----|---------|----|--------|----|-----|----|----|----|------|----|----|---|---|---|---|----|---|----|---|----|---|----|---|----|---|----|---|----|
|                                   |                    | 16 | 17  | 18 | 19     | 20 | 21      | 22 | 23     | 24 | 25  | 26 | 27 | 28 | 29   | 30 | 31 |   |   |   |   |    |   |    |   |    |   |    |   |    |   |    |   |    |
| 0xC3FE_0360<br>...<br>0xC3FE_036C | reserved           |    |     |    |        |    |         |    |        |    |     |    |    |    |      |    |    |   |   |   |   |    |   |    |   |    |   |    |   |    |   |    |   |    |
| 0xC3FE_0370                       | CGM_OC_EN          | R  | 0   | 0  | 0      | 0  | 0       | 0  | 0      | 0  | 0   | 0  | 0  | 0  | 0    | 0  | 0  | 0 | 0 | 0 | 0 | 0  | 0 | 0  | 0 | 0  | 0 | 0  | 0 | 0  | 0 | 0  | 0 |    |
|                                   |                    | W  |     |    |        |    |         |    |        |    |     |    |    |    |      |    |    |   |   |   |   |    |   |    |   |    |   |    |   |    |   |    |   |    |
|                                   |                    | R  | 0   | 0  | 0      | 0  | 0       | 0  | 0      | 0  | 0   | 0  | 0  | 0  | 0    | 0  | 0  | 0 | 0 | 0 | 0 | 0  | 0 | 0  | 0 | 0  | 0 | 0  | 0 | 0  | 0 | 0  | 0 | EN |
|                                   |                    | W  |     |    |        |    |         |    |        |    |     |    |    |    |      |    |    |   |   |   |   |    |   |    |   |    |   |    |   |    |   |    |   |    |
| 0xC3FE_0374                       | CGM_OCDS_SC        | R  | 0   | 0  | SELDIV |    |         |    | SELCTL |    |     |    | 0  | 0  | 0    | 0  | 0  | 0 | 0 | 0 | 0 | 0  | 0 | 0  | 0 | 0  | 0 | 0  | 0 | 0  | 0 | 0  |   |    |
|                                   |                    | W  |     |    |        |    |         |    |        |    |     |    |    |    |      |    |    |   |   |   |   |    |   |    |   |    |   |    |   |    |   |    |   |    |
|                                   |                    | R  | 0   | 0  | 0      | 0  | 0       | 0  | 0      | 0  | 0   | 0  | 0  | 0  | 0    | 0  | 0  | 0 | 0 | 0 | 0 | 0  | 0 | 0  | 0 | 0  | 0 | 0  | 0 | 0  | 0 | 0  | 0 |    |
|                                   |                    | W  |     |    |        |    |         |    |        |    |     |    |    |    |      |    |    |   |   |   |   |    |   |    |   |    |   |    |   |    |   |    |   |    |
| 0xC3FE_0378                       | CGM_SC_SS          | R  | 0   | 0  | 0      | 0  | SELSTAT |    |        |    | 0   | 0  | 0  | 0  | 0    | 0  | 0  | 0 | 0 | 0 | 0 | 0  | 0 | 0  | 0 | 0  | 0 | 0  | 0 | 0  | 0 | 0  |   |    |
|                                   |                    | W  |     |    |        |    |         |    |        |    |     |    |    |    |      |    |    |   |   |   |   |    |   |    |   |    |   |    |   |    |   |    |   |    |
|                                   |                    | R  | 0   | 0  | 0      | 0  | 0       | 0  | 0      | 0  | 0   | 0  | 0  | 0  | 0    | 0  | 0  | 0 | 0 | 0 | 0 | 0  | 0 | 0  | 0 | 0  | 0 | 0  | 0 | 0  | 0 | 0  | 0 |    |
|                                   |                    | W  |     |    |        |    |         |    |        |    |     |    |    |    |      |    |    |   |   |   |   |    |   |    |   |    |   |    |   |    |   |    |   |    |
| 0xC3FE_037C                       | CGM_SC_DC<br>0...2 | R  | DE0 | 0  | 0      | 0  | DIV0    |    |        |    | DE1 | 0  | 0  | 0  | DIV1 |    |    |   |   |   |   |    |   |    |   |    |   |    |   |    |   |    |   |    |
|                                   |                    | W  |     |    |        |    |         |    |        |    |     |    |    |    |      |    |    |   |   |   |   |    |   |    |   |    |   |    |   |    |   |    |   |    |
|                                   |                    | R  | DE2 | 0  | 0      | 0  | DIV2    |    |        |    | 0   | 0  | 0  | 0  | 0    | 0  | 0  | 0 |   |   |   |    |   |    |   |    |   |    |   |    |   |    |   |    |
|                                   |                    | W  |     |    |        |    |         |    |        |    |     |    |    |    |      |    |    |   |   |   |   |    |   |    |   |    |   |    |   |    |   |    |   |    |
| 0xC3FE_0380                       | CGM_AC0_S<br>C     | R  | 0   | 0  | 0      | 0  | SELCTL  |    |        |    | 0   | 0  | 0  | 0  | 0    | 0  | 0  | 0 | 0 | 0 | 0 | 0  | 0 | 0  | 0 | 0  | 0 | 0  | 0 | 0  | 0 | 0  |   |    |
|                                   |                    | W  |     |    |        |    |         |    |        |    |     |    |    |    |      |    |    |   |   |   |   |    |   |    |   |    |   |    |   |    |   |    |   |    |
|                                   |                    | R  | 0   | 0  | 0      | 0  | 0       | 0  | 0      | 0  | 0   | 0  | 0  | 0  | 0    | 0  | 0  | 0 | 0 | 0 | 0 | 0  | 0 | 0  | 0 | 0  | 0 | 0  | 0 | 0  | 0 | 0  | 0 |    |
|                                   |                    | W  |     |    |        |    |         |    |        |    |     |    |    |    |      |    |    |   |   |   |   |    |   |    |   |    |   |    |   |    |   |    |   |    |
| 0xC3FE_0400<br>...<br>0xC3FE_3FFC | reserved           |    |     |    |        |    |         |    |        |    |     |    |    |    |      |    |    |   |   |   |   |    |   |    |   |    |   |    |   |    |   |    |   |    |

## 7.5.1 Register descriptions

All registers may be accessed as 32-bit words, 16-bit half-words, or 8-bit bytes. The bytes are ordered according to big endian. For example, the CGM\_OC\_EN register may be accessed as a word at address 0xC3FE\_0370, as a half-word at address 0xC3FE\_0372, or as a byte at address 0xC3FE\_0373.

### 7.5.1.1 Output Clock Enable Register (CGM\_OC\_EN)

Address 0xC3FE\_0370 Access: User read, Supervisor read/write, Test read/write

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|       | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| R     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | EN |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |

Figure 7-2. Output Clock Enable Register (CGM\_OC\_EN)

This register is used to enable and disable the output clock.

Table 7-3. Output Clock Enable Register (CGM\_OC\_EN) field descriptions

| Field | Description   |
|-------|---|
| EN    | <b>Output Clock Enable control</b><br>0 Output Clock is disabled<br>1 Output Clock is enabled |

### 7.5.2 Output Clock Division Select Register (CGM\_OCDS\_SC)

Address 0xC3FE\_0374 Access: User read, Supervisor read/write, Test read/write

|       |   |   |        |   |        |   |   |   |   |   |    |    |    |    |    |    |
|-------|---|---|--------|---|--------|---|---|---|---|---|----|----|----|----|----|----|
|       | 0 | 1 | 2      | 3 | 4      | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| R     | 0 | 0 | SELDIV |   | SELCTL |   |   |   | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |        |   |        |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0      | 0 | 0      | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|       | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| R     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |

Figure 7-3. Output Clock Division Select Register (CGM\_OCDS\_SC)

This register is used to select the current output clock source and its division factor before being delivered at the output clock.

**Table 7-4. Output Clock Division Select Register (CGM\_OCDS\_SC) field descriptions**

| Field  | Description   |
|--------|---|
| SELDIV | <b>Output Clock Division Select</b><br>00 output selected Output Clock without division<br>01 output selected Output Clock divided by 2<br>10 output selected Output Clock divided by 4<br>11 output selected Output Clock divided by 8   |
| SELCTL | <b>Output Clock Source Selection Control</b> — This value selects the current source for the output clock.<br>0000 4-16 MHz ext. xtal osc.<br>0001 16 MHz int. RC osc.<br>0010 freq. mod. PLL<br>0011 system clock<br>0100 RTC clock<br>0101 reserved<br>0110 reserved<br>0111 reserved<br>1000 reserved<br>1001 reserved<br>1010 reserved<br>1011 reserved<br>1100 reserved<br>1101 reserved<br>1110 reserved<br>1111 reserved |

### 7.5.3 System Clock Select Status Register (CGM\_SC\_SS)

This register provides the current clock source selection for the following clocks:

- Undivided: system clock
- Divided by system clock divider 0: peripheral set 1 clock
- Divided by system clock divider 1: peripheral set 2 clock
- Divided by system clock divider 2: peripheral set 3 clock

See [Figure 7-7](#) for details.

Address 0xC3FE\_0378 Access: User read, Supervisor read, Test read

|       |    |    |    |    |         |    |    |    |    |    |    |    |    |    |    |    |
|-------|----|----|----|----|---------|----|----|----|----|----|----|----|----|----|----|----|
|       | 0  | 1  | 2  | 3  | 4       | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 | 14 | 15 |
| R     | 0  | 0  | 0  | 0  | SELSTAT |    |    |    | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |    |    |    |    |         |    |    |    |    |    |    |    |    |    |    |    |
| Reset | 0  | 0  | 0  | 0  | 0       | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
|       | 16 | 17 | 18 | 19 | 20      | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| R     | 0  | 0  | 0  | 0  | 0       | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |    |    |    |    |         |    |    |    |    |    |    |    |    |    |    |    |
| Reset | 0  | 0  | 0  | 0  | 0       | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |

**Figure 7-4. System Clock Select Status Register (CGM\_SC\_SS)**

**Table 7-5. System Clock Select Status Register (CGM\_SC\_SS) field descriptions**

| Field   | Description  |
|---------|--|
| SELSTAT | <b>System Clock Source Selection Status</b> — This value indicates the current source for the system clock.<br>0000 16 MHz int. RC osc.<br>0001 div. 16 MHz int. RC osc.<br>0010 4-16 MHz ext. xtal osc.<br>0011 div. ext. xtal osc.<br>0100 freq. mod. PLL<br>0101 reserved<br>0110 reserved<br>0111 reserved<br>1000 reserved<br>1001 reserved<br>1010 reserved<br>1011 reserved<br>1100 reserved<br>1101 reserved<br>1110 reserved<br>1111 system clock is disabled |

### 7.5.3.1 System Clock Divider Configuration Registers (CGM\_SC\_DC0...2)

Address 0xC3FE\_037C Access: User read, Supervisor read/write, Test read/write

|       |     |    |    |    |      |    |    |    |     |    |    |    |      |    |    |    |
|-------|-----|----|----|----|------|----|----|----|-----|----|----|----|------|----|----|----|
|       | 0   | 1  | 2  | 3  | 4    | 5  | 6  | 7  | 8   | 9  | 10 | 11 | 12   | 13 | 14 | 15 |
| R     | DE0 | 0  | 0  | 0  | DIV0 |    |    |    | DE1 | 0  | 0  | 0  | DIV1 |    |    |    |
| W     |     |    |    |    |      |    |    |    |     |    |    |    |      |    |    |    |
| Reset | 1   | 0  | 0  | 0  | 0    | 0  | 0  | 0  | 1   | 0  | 0  | 0  | 0    | 0  | 0  | 0  |
|       | 16  | 17 | 18 | 19 | 20   | 21 | 22 | 23 | 24  | 25 | 26 | 27 | 28   | 29 | 30 | 31 |
| R     | DE2 | 0  | 0  | 0  | DIV2 |    |    |    | 0   | 0  | 0  | 0  | 0    | 0  | 0  | 0  |
| W     |     |    |    |    |      |    |    |    |     |    |    |    |      |    |    |    |
| Reset | 1   | 0  | 0  | 0  | 0    | 0  | 0  | 0  | 0   | 0  | 0  | 0  | 0    | 0  | 0  | 0  |

**Figure 7-5. System Clock Divider Configuration Registers (CGM\_SC\_DC0...2)**

These registers control the system clock dividers.

**Table 7-6. System Clock Divider Configuration Registers (CGM\_SC\_DC0...2) Field Descriptions**

| Field | Description   |
|-------|---|
| DE0   | <b>Divider 0 Enable</b><br>0 Disable system clock divider 0<br>1 Enable system clock divider 0  |
| DIV0  | <b>Divider 0 Division Value</b> — The resultant peripheral set 1 clock will have a period DIV0 + 1 times that of the system clock. If the DE0 is set to 0 (Divider 0 is disabled), any write access to the DIV0 field is ignored and the peripheral set 1 clock remains disabled. |
| DE1   | <b>Divider 1 Enable</b><br>0 Disable system clock divider 1<br>1 Enable system clock divider 1  |

Table 7-6. System Clock Divider Configuration Registers (CGM\_SC\_DC0...2) Field Descriptions (continued)

| Field | Description   |
|-------|---|
| DIV1  | <b>Divider 1 Division Value</b> — The resultant peripheral set 2 clock will have a period DIV1 + 1 times that of the system clock. If the DE1 is set to 0 (Divider 1 is disabled), any write access to the DIV1 field is ignored and the peripheral set 2 clock remains disabled. |
| DE2   | <b>Divider 2 Enable</b><br>0 Disable system clock divider 2<br>1 Enable system clock divider 2  |
| DIV2  | <b>Divider 2 Division Value</b> — The resultant peripheral set 3 clock will have a period DIV2 + 1 times that of the system clock. If the DE2 is set to 0 (Divider 2 is disabled), any write access to the DIV2 field is ignored and the peripheral set 3 clock remains disabled. |

### 7.5.3.2 Auxiliary Clock 0 Select Control Register (CGM\_AC0\_SC)

Address 0xC3FE\_0380

Access: User read, Supervisor read/write, Test read/write

|       |    |    |    |    |        |    |    |    |    |    |    |    |    |    |    |    |
|-------|----|----|----|----|--------|----|----|----|----|----|----|----|----|----|----|----|
|       | 0  | 1  | 2  | 3  | 4      | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 | 14 | 15 |
| R     | 0  | 0  | 0  | 0  | SELCTL |    |    |    | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |    |    |    |    |        |    |    |    |    |    |    |    |    |    |    |    |
| Reset | 0  | 0  | 0  | 0  | 0      | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
|       | 16 | 17 | 18 | 19 | 20     | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| R     | 0  | 0  | 0  | 0  | 0      | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |    |    |    |    |        |    |    |    |    |    |    |    |    |    |    |    |
| Reset | 0  | 0  | 0  | 0  | 0      | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |

Figure 7-6. Auxiliary Clock 0 Select Control Register (CGM\_AC0\_SC)

This register is used to select the current clock source for the FMPLL reference clock.

Table 7-7. Auxiliary Clock 0 Select Control Register (CGM\_AC0\_SC) Field Descriptions

| Field  | Description   |
|--------|---|
| SELCTL | <b>Auxiliary Clock 0 Source Selection Control</b> — This value selects the current source for auxiliary clock 0.<br>0000 FXOSC<br>0001 FIRC<br>0010 reserved<br>0011 reserved<br>0100 reserved<br>0101 reserved<br>0110 reserved<br>0111 reserved<br>1000 reserved<br>1001 reserved<br>1010 reserved<br>1011 reserved<br>1100 reserved<br>1101 reserved<br>1110 reserved<br>1111 reserved |

## 7.6 Functional Description

### 7.6.1 System Clock Generation

Figure 7-7 shows the block diagram of the system clock generation logic. The MC\_ME provides the system clock select and switch mask (see Chapter 8, Mode Entry Module (MC\_ME), for more details), and the MC\_RGM provides the safe clock request (see Chapter 9, Reset Generation Module (MC\_RGM), for more details). The safe clock request forces the selector to select the 16 MHz int. RC osc. as the system clock and to ignore the system clock select.

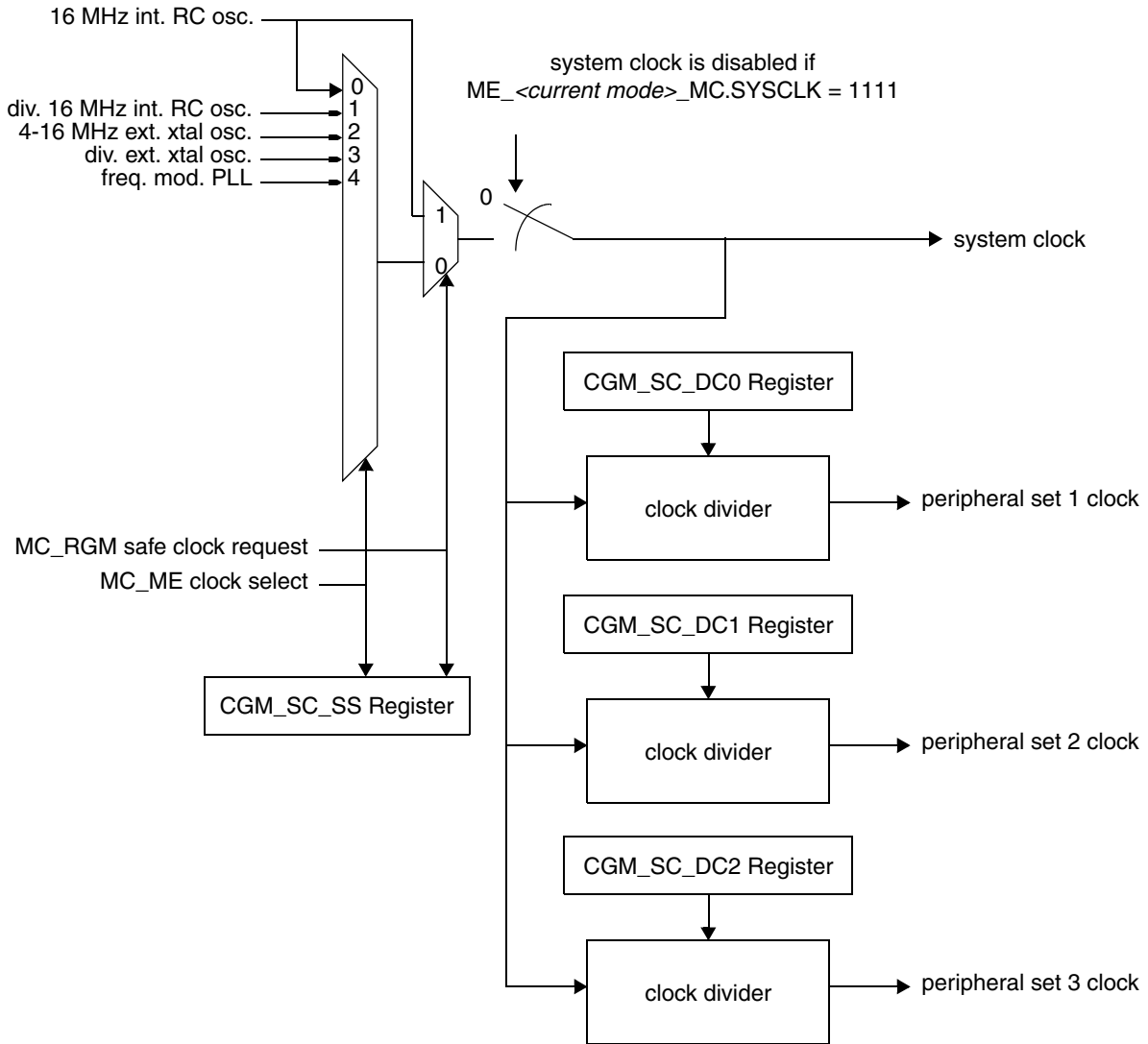


Figure 7-7. MC\_CGM System Clock Generation Overview

### 7.6.1.1 System Clock Source Selection

During normal operation, the system clock selection is controlled:

- On a SAFE mode or reset event, by the MC\_RGM
- Otherwise, by the MC\_ME

### 7.6.1.2 System Clock Disable

During normal operation, the system clock can be disabled by the MC\_ME.

### 7.6.1.3 System Clock Dividers

The MC\_CGM generates three derived clocks from the system clock.

### 7.6.1.4 Dividers Functional Description

Dividers are utilized for the generation of divided system and peripheral clocks. The MC\_CGM has the following control registers for built-in dividers:

- [Section 7.5.3.1, System Clock Divider Configuration Registers \(CGM\\_SC\\_DC0...2\)](#)

The reset value of all counters is 1. If a divider has its DE bit in the respective configuration register set to 0 (the divider is disabled), any value in its DIVn field is ignored.

## 7.6.2 Output Clock Multiplexing

The MC\_CGM contains a multiplexing function for a number of clock sources that can then be utilized as output clock sources. The selection is done via the CGM\_OCDS\_SC register.

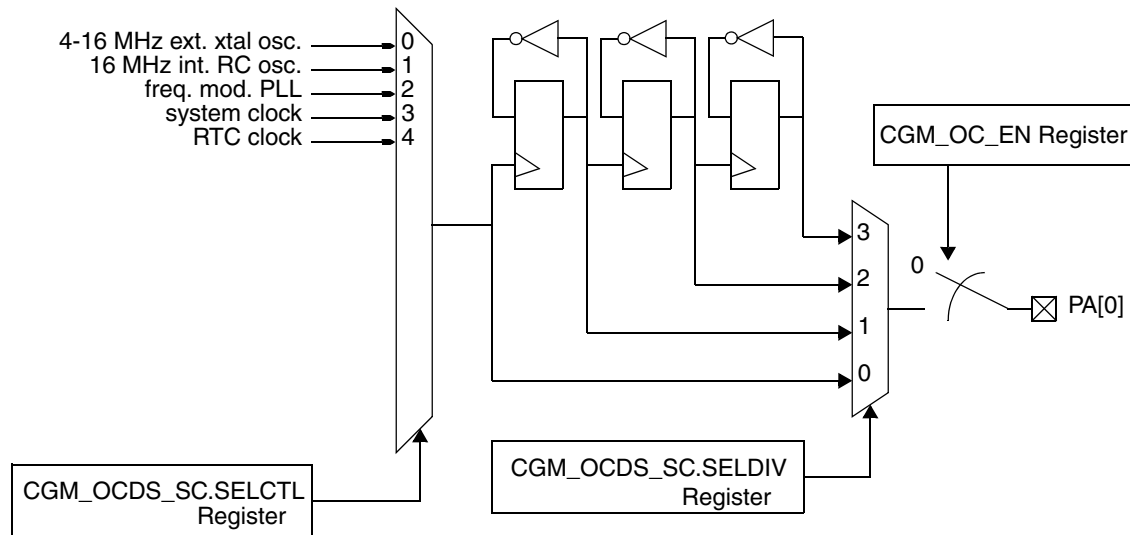


Figure 7-8. MC\_CGM Output Clock Multiplexer and PA[0] Generation

### 7.6.3 Output Clock Division Selection

The MC\_CGM provides the following output signals for the output clock generation:

- PA[0] (see [Figure 7-8](#)). This signal is generated by utilizing one of the 3-stage ripple counter outputs or the selected signal without division. The non-divided signal is not guaranteed to be 50% duty cycle by the MC\_CGM.
- the MC\_CGM also has an output clock enable register (see [Section 7.5.1.1, Output Clock Enable Register \(CGM\\_OC\\_EN\)](#)), which contains the output clock enable/disable control bit.



# Chapter 8

## Mode Entry Module (MC\_ME)

### 8.1 Overview

The MC\_ME controls the SoC modex and mode transition sequences in all functional states. It also contains configuration, control, and status registers accessible for the application.

Figure 8-1 shows the MC\_ME block diagram.

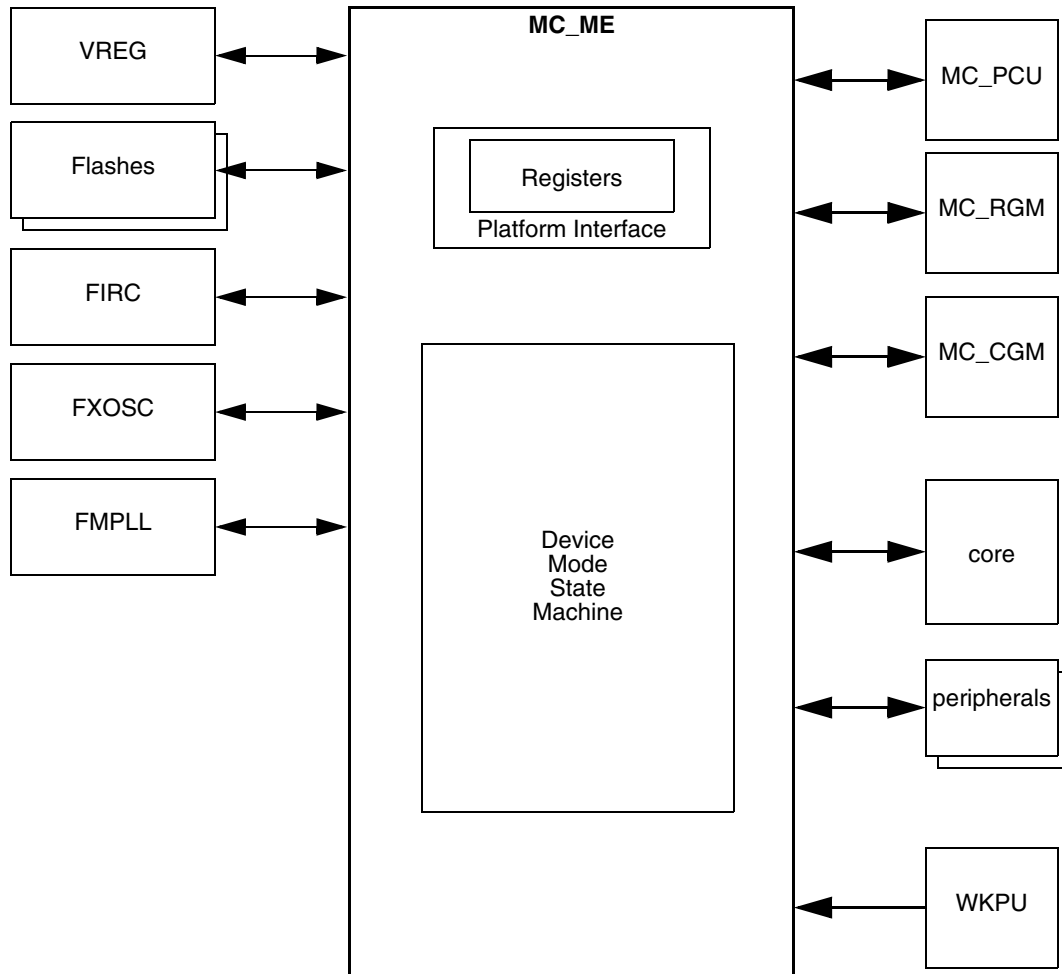


Figure 8-1. MC\_ME block diagram

#### 8.1.1 Features

The MC\_ME includes the following features:

- Control of the available modes by the ME\_ME register
- Definition of various device mode configurations by the ME\_<mode>\_MC registers
- Control of the actual device mode by the ME\_MCTL register

- Capture of the current mode and various resource status within the contents of the ME\_GS register
- Optional generation of various mode transition interrupts
- Status bits for each cause of invalid mode transitions

## 8.1.2 Modes of operation

The MC\_ME is based on several device modes corresponding to different usage models of the device. Each mode is configurable and can define a policy for energy and processing power management to fit particular system requirements. An application can easily switch from one mode to another depending on the current needs of the system. The operating modes controlled by the MC\_ME are divided into system and user modes. The system modes are modes such as RESET, DRUN, SAFE, and TEST. These modes aim to ease the configuration and monitoring of the system. The user modes are modes such as RUN0...3, HALT, STOP which can be configured to meet the application requirements in terms of energy management and available processing power. The modes DRUN, SAFE, TEST, and RUN0...3 are the device software running modes.

Table 8-1 describes the MC\_ME modes.

**Table 8-1. MC\_ME mode descriptions**

| Name     | Description   | Entry  | Exit   |
|----------|---|--|--|
| RESET    | This is a chip-wide virtual mode during which the application is not active. The system remains in this mode until all resources are available for the embedded software to take control of the device. It manages hardware initialization of chip configuration, voltage regulators, oscillators, PLLs, and flash modules. | System reset assertion from MC_RGM   | System reset deassertion from MC_RGM   |
| DRUN     | This is the entry mode for the embedded software. It provides full accessibility to the system and enables the configuration of the system at startup. It provides the unique gate to enter USER modes. BAM when present is executed in DRUN mode.  | System reset deassertion from MC_RGM, software request from SAFE, TEST, and RUN0...3       | System reset assertion, RUN0...3, TEST via software, SAFE via software or hardware failure.                  |
| SAFE     | This chip-wide service mode may be entered on the detection of a recoverable error. It forces the system into a predefined safe configuration from which the system may try to recover.   | Hardware failure, software request from DRUN, TEST, and RUN0...3                           | System reset assertion, DRUN via software  |
| TEST     | This chip-wide service mode provides a control environment for device self-test. It may enable the application to run its own self-test like flash checksum, memory BIST, etc.  | Software request from DRUN   | System reset assertion, DRUN via software  |
| RUN0...3 | These are software running modes where most processing activity is done. These various run modes allow to enable different clock & power configurations of the system with respect to each other.   | Software request from DRUN, interrupt event from HALT, interrupt or wakeup event from STOP | System reset assertion, SAFE via software or hardware failure, other RUN0...3 modes, HALT, STOP via software |

Table 8-1. MC\_ME mode descriptions (continued)

| Name | Description  | Entry                          | Exit  |
|------|--|--------------------------------|---|
| HALT | This is a reduced-activity low-power mode during which the clock to the core is disabled. It can be configured to switch off analog peripherals like PLL, flash, main regulator, etc. for efficient power management at the cost of higher wakeup latency. | Software request from RUN0...3 | System reset assertion, SAFE on hardware failure, RUN0...3 on interrupt event                 |
| STOP | This is an advanced low-power mode during which the clock to the core is disabled. It may be configured to switch off most of the peripherals including oscillator for efficient power management at the cost of higher wakeup latency.                    | Software request from RUN0...3 | System reset assertion, SAFE on hardware failure, RUN0...3 on interrupt event or wakeup event |

## 8.2 External signal description

The MC\_ME has no connections to any external pins.

## 8.3 Memory map and register definition

The MC\_ME contains registers for:

- Mode selection and status reporting
- Mode configuration
- Mode transition interrupts status and mask control

Table 8-2. MC\_ME register description

| Address     | Name    | Description                    | Size | Access |            |            | Location                    |
|-------------|---------|--------------------------------|------|--------|------------|------------|-----------------------------|
|             |         |                                |      | Normal | Supervisor | Test       |                             |
| 0xC3FD_C000 | ME_GS   | Global Status                  | word | read   | read       | read       | <a href="#">on page 151</a> |
| 0xC3FD_C004 | ME_MCTL | Mode Control                   | word | read   | read/write | read/write | <a href="#">on page 152</a> |
| 0xC3FD_C008 | ME_ME   | Mode Enable                    | word | read   | read/write | read/write | <a href="#">on page 153</a> |
| 0xC3FD_C00C | ME_IS   | Interrupt Status               | word | read   | read/write | read/write | <a href="#">on page 154</a> |
| 0xC3FD_C010 | ME_IM   | Interrupt Mask                 | word | read   | read/write | read/write | <a href="#">on page 155</a> |
| 0xC3FD_C014 | ME_IMTS | Invalid Mode Transition Status | word | read   | read/write | read/write | <a href="#">on page 156</a> |
| 0xC3FD_C018 | ME_DMTS | Debug Mode Transition Status   | word | read   | read       | read       | <a href="#">on page 157</a> |

**Table 8-2. MC\_ME register description (continued)**

| Address     | Name        | Description              | Size | Access |            |            | Location    |
|-------------|-------------|--------------------------|------|--------|------------|------------|-------------|
|             |             |                          |      | Normal | Supervisor | Test       |             |
| 0xC3FD_C020 | ME_RESET_MC | RESET Mode Configuration | word | read   | read       | read       | on page 159 |
| 0xC3FD_C024 | ME_TEST_MC  | TEST Mode Configuration  | word | read   | read/write | read/write | on page 159 |
| 0xC3FD_C028 | ME_SAFE_MC  | SAFE Mode Configuration  | word | read   | read/write | read/write | on page 160 |
| 0xC3FD_C02C | ME_DRUN_MC  | DRUN Mode Configuration  | word | read   | read/write | read/write | on page 160 |
| 0xC3FD_C030 | ME_RUN0_MC  | RUN0 Mode Configuration  | word | read   | read/write | read/write | on page 161 |
| 0xC3FD_C034 | ME_RUN1_MC  | RUN1 Mode Configuration  | word | read   | read/write | read/write | on page 161 |
| 0xC3FD_C038 | ME_RUN2_MC  | RUN2 Mode Configuration  | word | read   | read/write | read/write | on page 161 |
| 0xC3FD_C03C | ME_RUN3_MC  | RUN3 Mode Configuration  | word | read   | read/write | read/write | on page 161 |
| 0xC3FD_C040 | ME_HALT_MC  | HALT Mode Configuration  | word | read   | read/write | read/write | on page 161 |
| 0xC3FD_C048 | ME_STOP_MC  | STOP Mode Configuration  | word | read   | read/write | read/write | on page 161 |

**NOTE**

Any access to unused registers as well as write accesses to read-only registers will:

- Not change register content
- Cause a transfer error

**Table 8-3. MC\_ME Memory Map**

|             |         |   |             |   |   |   |   |   |   |   |   |   |   |   |   |        |          |   |
|-------------|---------|---|-------------|---|---|---|---|---|---|---|---|---|---|---|---|--------|----------|---|
|             |         | W |             |   |   |   |   |   |   |   |   |   |   |   |   |        |          |   |
|             |         | R | 0           | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | S_FIRC | S_SYSCLK |   |
|             |         | W |             |   |   |   |   |   |   |   |   |   |   |   |   |        |          |   |
| 0xC3FD_C004 | ME_MCTL | R | TARGET_MODE |   |   |   | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0      | 0        |   |
|             |         | W |             |   |   |   |   |   |   |   |   |   |   |   |   |        |          |   |
|             |         | R | 1           | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1      | 1        | 1 |
|             |         | W | KEY         |   |   |   |   |   |   |   |   |   |   |   |   |        |          |   |

Table 8-3. MC\_ME Memory Map (continued)

|             |             |   |   |   |   |   |   |           |           |           |   |   |          |          |         |         |        |       |
|-------------|-------------|---|---|---|---|---|---|-----------|-----------|-----------|---|---|----------|----------|---------|---------|--------|-------|
| 0xC3FD_C008 | ME_ME       | R | 0 | 0 | 0 | 0 | 0 | 0         | 0         | 0         | 0 | 0 | 0        | 0        | 0       | 0       | 0      |       |
|             |             | W |   |   |   |   |   |           |           |           |   |   |          |          |         |         |        |       |
| 0xC3FD_C00C | ME_IS       | R | 0 | 0 | 0 | 0 | 0 | 0         | 0         | 0         | 0 | 0 | 0        | 0        | 0       | 0       | 0      |       |
|             |             | W |   |   |   |   |   |           |           |           |   |   |          |          |         |         |        |       |
|             |             | R | 0 | 0 | 0 | 0 | 0 | 0         | 0         | 0         | 0 | 0 | 0        | 0        | I_ICONF | I_IMODE | I_SAFE | I_MTC |
|             | W           |   |   |   |   |   |   |           |           |           |   |   | w1c      | w1c      | w1c     | w1c     |        |       |
| 0xC3FD_C010 | ME_IM       | R | 0 | 0 | 0 | 0 | 0 | 0         | 0         | 0         | 0 | 0 | 0        | 0        | 0       | 0       | 0      |       |
|             |             | W |   |   |   |   |   |           |           |           |   |   |          |          |         |         |        |       |
|             |             | R | 0 | 0 | 0 | 0 | 0 | 0         | 0         | 0         | 0 | 0 | 0        | 0        | M_ICONF | M_IMODE | M_SAFE | M_MTC |
|             |             | W |   |   |   |   |   |           |           |           |   |   |          |          |         |         |        |       |
| 0xC3FD_C014 | ME_IMTS     | R | 0 | 0 | 0 | 0 | 0 | 0         | 0         | 0         | 0 | 0 | 0        | 0        | 0       | 0       | 0      |       |
|             |             | W |   |   |   |   |   |           |           |           |   |   |          |          |         |         |        |       |
|             |             | R | 0 | 0 | 0 | 0 | 0 | 0         | 0         | 0         | 0 | 0 | 0        | S_MTI    | S_MRI   | S_DMA   | S_NMA  | S_SEA |
|             |             | W |   |   |   |   |   |           |           |           |   |   |          | w1c      | w1c     | w1c     | w1c    | w1c   |
| 0xC3FD_C018 | ME_DMTS     | R | 0 | 0 | 0 | 0 | 0 | 0         | 0         | MPH_BUSY  | 0 | 0 | PMC_PROG | CORE_DBG | 0       | 0       | SMR    |       |
|             |             | W |   |   |   |   |   |           |           |           |   |   |          |          |         |         |        |       |
|             |             | R | 0 |   |   |   | 0 | SYSCLK_SW | DFLASH_SC | CFLASH_SC |   | 0 | 0        | 0        |         |         |        |       |
|             |             | W |   |   |   |   |   |           |           |           |   |   |          |          |         |         |        |       |
| 0xC3FD_C01C | reserved    |   |   |   |   |   |   |           |           |           |   |   |          |          |         |         |        |       |
| 0xC3FD_C020 | ME_RESET_MC | R | 0 | 0 | 0 | 0 | 0 | 0         | 0         | PDO       | 0 | 0 | MVRON    | DFLAON   | CFLAON  |         |        |       |
|             |             | W |   |   |   |   |   |           |           |           |   |   |          |          |         |         |        |       |
|             |             | R | 0 | 0 | 0 | 0 | 0 | 0         | 0         | 0         | 0 |   | FIRCON   | SYSCLK   |         |         |        |       |
|             |             | W |   |   |   |   |   |           |           |           |   |   |          |          |         |         |        |       |

**Table 8-3. MC\_ME Memory Map (continued)**

|                                   |                |   |   |   |   |   |   |   |   |   |     |      |      |        |        |        |      |      |       |
|-----------------------------------|----------------|---|---|---|---|---|---|---|---|---|-----|------|------|--------|--------|--------|------|------|-------|
| 0xC3FD_C024                       | ME_TEST_MC     | R | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | PDO | 0    | 0    | MVRON  | DFLAON | CFLAON |      |      |       |
|                                   |                | W |   |   |   |   |   |   |   |   |     |      |      |        |        |        |      |      |       |
| 0xC3FD_C028                       | ME_SAFE_MC     | R | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | PDO | 0    | 0    | MVRON  | DFLAON | CFLAON |      |      |       |
|                                   |                | W |   |   |   |   |   |   |   |   |     |      |      |        |        |        |      |      |       |
|                                   |                | R | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0   |      |      | FIRCON | SYSCLK |        |      |      |       |
|                                   |                | W |   |   |   |   |   |   |   |   |     |      |      |        |        |        |      |      |       |
| 0xC3FD_C02C                       | ME_DRUN_MC     | R | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | PDO | 0    | 0    | MVRON  | DFLAON | CFLAON |      |      |       |
|                                   |                | W |   |   |   |   |   |   |   |   |     |      |      |        |        |        |      |      |       |
| 0xC3FD_C030<br>...<br>0xC3FD_C03C | ME_RUN0...3_MC | R | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | PDO | 0    | 0    | MVRON  | DFLAON | CFLAON |      |      |       |
|                                   |                | W |   |   |   |   |   |   |   |   |     |      |      |        |        |        |      |      |       |
| 0xC3FD_C044                       | reserved       |   |   |   |   |   |   |   |   |   |     |      |      |        |        |        |      |      |       |
| 0xC3FD_C0<br>...<br>0xC3FD_C      | reserved       |   |   |   |   |   |   |   |   |   |     |      |      |        |        |        |      |      |       |
| 0xC3FD_C080<br>...<br>0xC3FD_C09C | ME_RUN_PC0...7 | R | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0   | 0    | 0    | 0      | 0      | 0      | 0    |      |       |
|                                   |                | W |   |   |   |   |   |   |   |   |     |      |      |        |        |        |      |      |       |
|                                   |                | R | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |     | RUN3 | RUN2 | RUN1   | RUN0   | DRUN   | SAFE | TEST | RESET |
|                                   |                | W |   |   |   |   |   |   |   |   |     |      |      |        |        |        |      |      |       |
| 0xC3FD_C150<br>...<br>0xC3FD_FFFC | reserved       |   |   |   |   |   |   |   |   |   |     |      |      |        |        |        |      |      |       |

### 8.3.1 Register descriptions

Unless otherwise noted, all registers may be accessed as 32-bit words, 16-bit half-words, or 8-bit bytes. The bytes are ordered according to big endian. For example, the ME\_RUN\_PC0 register may be accessed

as a word at address 0xC3FD\_C080, as a half-word at address 0xC3FD\_C082, or as a byte at address 0xC3FD\_C083.

### 8.3.1.1 Global Status Register (ME\_GS)

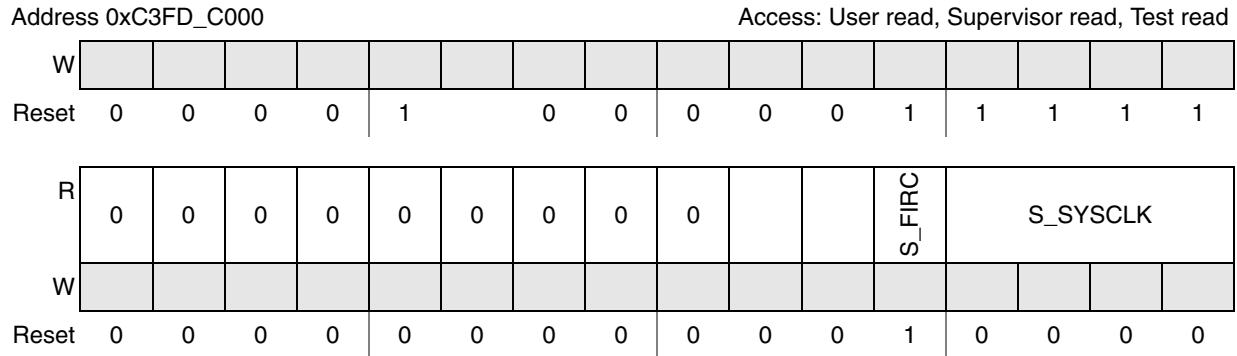


Figure 8-2. Global Status Register (ME\_GS)

This register contains global mode status.

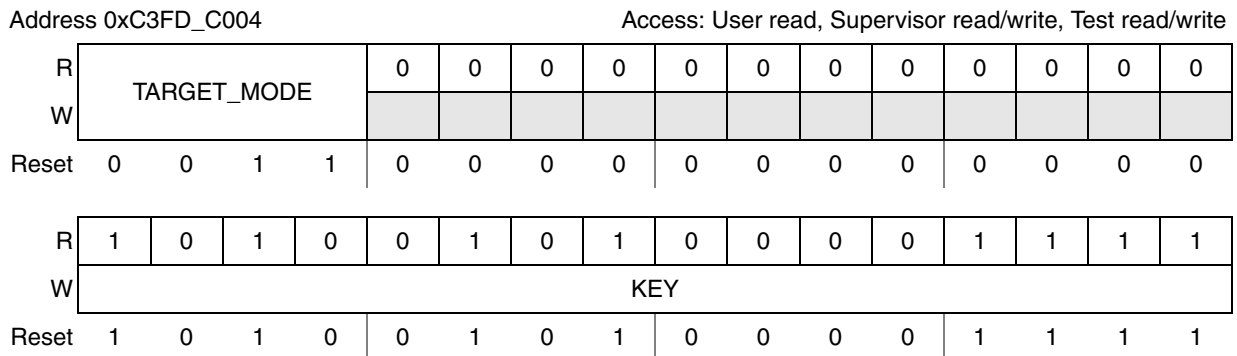
Table 8-4. Global Status Register (ME\_GS) Field Descriptions

| Field          | Description   |
|----------------|---|
| S_CURRENT_MODE | <b>Current device mode status</b><br>0000 RESET<br>0001 TEST<br>0010 SAFE<br>0011 DRUN<br>0100 RUN0<br>0101 RUN1<br>0110 RUN2<br>0111 RUN3<br>1000 HALT<br>1001 reserved<br>1010 STOP<br>1011 reserved<br>1100 reserved<br>1101<br>1110 reserved<br>1111 reserved   |
| S_MTRANS       | <b>Mode transition status</b><br>0 Mode transition process is not active<br>1 Mode transition is ongoing  |
| S_PDO          | <b>Output power-down status</b> — This bit specifies output power-down status of I/Os. This bit is asserted whenever outputs of pads are forced to high impedance state or the pads power sequence driver is switched off.<br>0 No automatic safe gating of I/Os used and pads power sequence driver is enabled<br>1 In SAFE/TEST modes, outputs of pads are forced to high impedance state and pads power sequence driver is disabled. The inputs are level unchanged. In STOP mode, only pad power sequence driver is disabled but the state of the output is kept. |
| S_MVR          | <b>Main voltage regulator status</b><br>0 Main voltage regulator is not ready<br>1 Main voltage regulator is ready for use  |

**Table 8-4. Global Status Register (ME\_GS) Field Descriptions (continued)**

| Field    | Description  |
|----------|--|
| S_DFLA   | <b>Data flash availability status</b><br>00 Data flash is not available<br>01 Data flash is in power-down mode<br>10 Data flash is in low-power mode<br>11 Data flash is in normal mode and available for use  |
| S_CFLA   | <b>Code flash availability status</b><br>00 Code flash is not available<br>01 Code flash is in power-down mode<br>10 Code flash is in low-power mode<br>11 Code flash is in normal mode and available for use  |
| S_FIRC   | <b>fast internal RC oscillator (16 MHz) status</b><br>0 fast internal RC oscillator (16 MHz) is not stable<br>1 fast internal RC oscillator (16 MHz) is providing a stable clock   |
| S_SYSClk | <b>System clock switch status</b> — These bits specify the system clock currently used by the system.<br>0000 16 MHz int. RC osc.<br>0001 div. 16 MHz int. RC osc.<br>0010 4–16 MHz ext. xtal osc.<br>0011 div. ext. xtal osc.<br>0100 freq. mod. PLL<br>0101 reserved<br>0110 reserved<br>0111 reserved<br>1000 reserved<br>1001 reserved<br>1010 reserved<br>1011 reserved<br>1100 reserved<br>1101 reserved<br>1110 reserved<br>1111 system clock is disabled |

### 8.3.1.2 Mode Control Register (ME\_MCTL)



**Figure 8-3. Mode Control Register (ME\_MCTL)**

This register is used to trigger software-controlled mode changes. Depending on the modes as enabled by ME\_ME register bits, configurations corresponding to unavailable modes are reserved and access to ME\_<mode>\_MC registers must respect this for successful mode requests.



**NOTE**

Byte and half-word write accesses are not allowed for this register as a predefined key is required to change its value.

**Table 8-5. Mode Control Register (ME\_MCTL) Field Descriptions**

| Field       | Description  |
|-------------|--|
| TARGET_MODE | <p><b>Target device mode</b> — These bits provide the target device mode to be entered by software programming. The mechanism to enter into any mode by software requires the write operation twice: first time with key, and second time with inverted key. These bits are automatically updated by hardware while entering SAFE on hardware request. Also, while exiting from the HALT and STOP modes on hardware exit events, these are updated with the appropriate RUN0...3 mode value.</p> <p>0000 RESET<br/>           0001 TEST<br/>           0010 SAFE<br/>           0011 DRUN<br/>           0100 RUN0<br/>           0101 RUN1<br/>           0110 RUN2<br/>           0111 RUN3<br/>           1000 HALT<br/>           1001 reserved<br/>           1010 STOP<br/>           1011 reserved<br/>           1100 reserved<br/>           1101<br/>           1110 reserved<br/>           1111 reserved</p> |
| KEY         | <p><b>Control key</b> — These bits enable write access to this register. Any write access to the register with a value different from the keys is ignored. Read access will always return inverted key.</p> <p>KEY: 0101101011110000 (0x5AF0)<br/>           INVERTED KEY: 1010010100001111 (0xA50F)</p>   |

**8.3.1.3 Mode Enable Register (ME\_ME)**

Address 0xC3FD\_C008

Access: User read, Supervisor read/write, Test read/write

|       |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|-------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |   |
| W     |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |   |
| Reset | 0 | 0 |   | 0 |   | 0 |   |   |   |   |   | 1 |   | 1 | 1 | 1 |

**Figure 8-4. Mode Enable Register (ME\_ME)**

This register allows a way to disable the device modes that are not required for a given device. RESET, SAFE, DRUN, and RUN0 modes are always enabled.

**Table 8-6. Mode Enable Register (ME\_ME) Field Descriptions**

| Field | Description   |
|-------|---|
| STOP  | <b>STOP mode enable</b><br>0 STOP mode is disabled<br>1 STOP mode is enabled    |
| HALT  | <b>HALT mode enable</b><br>0 HALT mode is disabled<br>1 HALT mode is enabled    |
| RUN3  | <b>RUN3 mode enable</b><br>0 RUN3 mode is disabled<br>1 RUN3 mode is enabled    |
| RUN2  | <b>RUN2 mode enable</b><br>0 RUN2 mode is disabled<br>1 RUN2 mode is enabled    |
| RUN1  | <b>RUN1 mode enable</b><br>0 RUN1 mode is disabled<br>1 RUN1 mode is enabled    |
| RUN0  | <b>RUN0 mode enable</b><br>0 RUN0 mode is disabled<br>1 RUN0 mode is enabled    |
| DRUN  | <b>DRUN mode enable</b><br>0 DRUN mode is disabled<br>1 DRUN mode is enabled    |
| SAFE  | <b>SAFE mode enable</b><br>0 SAFE mode is disabled<br>1 SAFE mode is enabled    |
| TEST  | <b>TEST mode enable</b><br>0 TEST mode is disabled<br>1 TEST mode is enabled    |
| RESET | <b>RESET mode enable</b><br>0 RESET mode is disabled<br>1 RESET mode is enabled |

### 8.3.1.4 Interrupt Status Register (ME\_IS)

Address 0xC3FD\_C00C

Access: User read, Supervisor read/write, Test read/write

|       |   |   |   |   |   |   |   |   |   |   |   |         |         |        |       |
|-------|---|---|---|---|---|---|---|---|---|---|---|---------|---------|--------|-------|
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0       | 0       | 0      | 0     |
| W     |   |   |   |   |   |   |   |   |   |   |   |         |         |        |       |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0       | 0       | 0      | 0     |
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | I_ICONF | I_IMODE | I_SAFE | I_MTC |
| W     |   |   |   |   |   |   |   |   |   |   |   | w1c     | w1c     | w1c    | w1c   |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0       | 0       | 0      | 0     |

**Figure 8-5. Interrupt Status Register (ME\_IS)**

This register provides the current interrupt status.

**Table 8-7. Interrupt Status Register (ME\_IS) Field Descriptions**

| Field   | Description  |
|---------|--|
| I_ICONF | <b>Invalid mode configuration interrupt</b> — This bit is set whenever a write operation to ME_<mode>_MC registers with invalid mode configuration is attempted. It is cleared by writing a 1 to this bit.<br>0 No invalid mode configuration interrupt occurred<br>1 Invalid mode configuration interrupt is pending  |
| I_IMODE | <b>Invalid mode interrupt</b> — This bit is set whenever an invalid mode transition is requested. It is cleared by writing a 1 to this bit.<br>0 No invalid mode interrupt occurred<br>1 Invalid mode interrupt is pending   |
| I_SAFE  | <b>SAFE mode interrupt</b> — This bit is set whenever the device enters SAFE mode on hardware requests generated in the system. It is cleared by writing a 1 to this bit.<br>0 No SAFE mode interrupt occurred<br>1 SAFE mode interrupt is pending   |
| I_MTC   | <b>Mode transition complete interrupt</b> — This bit is set whenever the mode transition process completes (S_MTRANS transits from 1 to 0). It is cleared by writing a 1 to this bit. This mode transition interrupt bit will not be set while entering low-power modes HALT, STOP.<br>0 No mode transition complete interrupt occurred<br>1 Mode transition complete interrupt is pending |

### 8.3.1.5 Interrupt Mask Register (ME\_IM)

Address 0xC3FD\_C010

Access: User read, Supervisor read/write, Test read/write

|       |   |   |   |   |   |   |   |   |   |   |   |         |         |        |       |
|-------|---|---|---|---|---|---|---|---|---|---|---|---------|---------|--------|-------|
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0       | 0       | 0      | 0     |
| W     |   |   |   |   |   |   |   |   |   |   |   |         |         |        |       |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0       | 0       | 0      | 0     |
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | M_ICONF | M_IMODE | M_SAFE | M_MTC |
| W     |   |   |   |   |   |   |   |   |   |   |   |         |         |        |       |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0       | 0       | 0      | 0     |

**Figure 8-6. Interrupt Mask Register (ME\_IM)**

This register controls whether an event generates an interrupt or not.

**Table 8-8. Interrupt Mask Register (ME\_IM) Field Descriptions**

| Field   | Description   |
|---------|---|
| M_ICONF | <b>Invalid mode configuration interrupt mask</b><br>0 Invalid mode interrupt is masked<br>1 Invalid mode interrupt is enabled |
| M_IMODE | <b>Invalid mode interrupt mask</b><br>0 Invalid mode interrupt is masked<br>1 Invalid mode interrupt is enabled               |

Table 8-8. Interrupt Mask Register (ME\_IM) Field Descriptions (continued)

| Field  | Description   |
|--------|---|
| M_SAFE | <b>SAFE mode interrupt mask</b><br>0 SAFE mode interrupt is masked<br>1 SAFE mode interrupt is enabled  |
| M_MTC  | <b>Mode transition complete interrupt mask</b><br>0 Mode transition complete interrupt is masked<br>1 Mode transition complete interrupt is enabled |

### 8.3.1.6 Invalid Mode Transition Status Register (ME\_IMTS)

Address 0xC3FD\_C014

Access: User read, Supervisor read/write, Test read/write

|       |   |   |   |   |   |   |   |   |   |   |       |       |       |       |       |
|-------|---|---|---|---|---|---|---|---|---|---|-------|-------|-------|-------|-------|
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0     | 0     | 0     | 0     | 0     |
| W     |   |   |   |   |   |   |   |   |   |   |       |       |       |       |       |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0     | 0     | 0     | 0     | 0     |
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | S_MTI | S_MRI | S_DMA | S_NMA | S_SEA |
| W     |   |   |   |   |   |   |   |   |   |   | w1c   | w1c   | w1c   | w1c   | w1c   |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0     | 0     | 0     | 0     | 0     |

Figure 8-7. Invalid Mode Transition Status Register (ME\_IMTS)

This register provides the status bits for each cause of invalid mode interrupt.

Table 8-9. Invalid Mode Transition Status Register (ME\_IMTS) Field Descriptions

| Field | Description  |
|-------|--|
| S_MTI | <b>Mode Transition Illegal status</b> — This bit is set whenever a new mode is requested while some other mode transition process is active (S_MTRANS is 1). Please see <a href="#">Section 8.4.5, Mode transition interrupts</a> , for the exceptions to this behavior. It is cleared by writing a 1 to this bit.<br>0 Mode transition requested is not illegal<br>1 Mode transition requested is illegal |
| S_MRI | <b>Mode Request Illegal status</b> — This bit is set whenever the target mode requested is not a valid mode with respect to current mode. It is cleared by writing a 1 to this bit.<br>0 Target mode requested is not illegal with respect to current mode<br>1 Target mode requested is illegal with respect to current mode  |
| S_DMA | <b>Disabled Mode Access status</b> — This bit is set whenever the target mode requested is one of those disabled modes determined by ME_ME register. It is cleared by writing a 1 to this bit.<br>0 Target mode requested is not a disabled mode<br>1 Target mode requested is a disabled mode   |

**Table 8-9. Invalid Mode Transition Status Register (ME\_IMTS) Field Descriptions (continued)**

| Field | Description   |
|-------|---|
| S_NMA | <b>Non-existing Mode Access status</b> — This bit is set whenever the target mode requested is one of those non existing modes determined by ME_ME register. It is cleared by writing a 1 to this bit.<br>0 Target mode requested is an existing mode<br>1 Target mode requested is a non-existing mode   |
| S_SEA | <b>SAFE Event Active status</b> — This bit is set whenever the device is in SAFE mode, SAFE event bit is pending and a new mode requested other than RESET/SAFE modes. It is cleared by writing a 1 to this bit.<br>0 No new mode requested other than RESET/SAFE while SAFE event is pending<br>1 New mode requested other than RESET/SAFE while SAFE event is pending |

### 8.3.1.7 Debug Mode Transition Status Register (ME\_DMTS)

Address 0xC3FD\_C018

Access: User read, Supervisor read/write, Test read/write

|       |   |   |   |   |   |          |           |           |          |   |   |          |          |   |   |     |
|-------|---|---|---|---|---|----------|-----------|-----------|----------|---|---|----------|----------|---|---|-----|
| R     | 0 | 0 | 0 | 0 | 0 | 0        | 0         | 0         | MPH_BUSY | 0 | 0 | PMC_PROG | CORE_DBG | 0 | 0 | SMR |
| W     |   |   |   |   |   |          |           |           |          |   |   |          |          |   |   |     |
| Reset | 0 | 0 | 0 | 0 | 0 | 0        | 0         | 0         | 0        | 0 | 0 | 0        | 0        | 0 | 0 | 0   |
| R     | 0 |   |   |   | 0 | SYCLK_SW | DFLASH_SC | CFLASH_SC |          | 0 | 0 | 0        |          |   |   |     |
| W     |   |   |   |   |   |          |           |           |          |   |   |          |          |   |   |     |
| Reset | 0 | 0 | 0 | 0 | 0 | 0        | 0         | 0         | 0        | 0 | 0 | 0        | 0        | 0 | 0 | 0   |

**Figure 8-8. Debug Mode Transition Status Register (ME\_DMTS)**

This register provides the status of different factors that influence mode transitions. It is used to give an indication of why a mode transition indicated by ME\_GS.S\_MTRANS may be taking longer than expected.

#### NOTE

The ME\_DMTS register does not indicate whether a mode transition is ongoing. Therefore, some ME\_DMTS bits may still be asserted after the mode transition has completed.

**Table 8-10. Debug Mode Transition Status Register (ME\_DMTS) Field Descriptions**

| Field     | Description   |
|-----------|---|
| MPH_BUSY  | MC_ME/MC_PCU Handshake Busy indicator — This bit is set if the MC_ME has requested a mode change from the MC_PCU and the MC_PCU has not yet responded. It is cleared when the MC_PCU has responded.<br>0 Handshake is not busy<br>1 Handshake is busy   |
| PMC_PROG  | MC_PCU Mode Change in Progress indicator — This bit is set if the MC_PCU is in the process of powering up or down power domains. It is cleared when all power-up/down processes have completed.<br>0 Power-up/down transition is not in progress<br>1 Power-up/down transition is in progress   |
| CORE_DBG  | Processor is in Debug mode indicator — This bit is set while the processor is in debug mode.<br>0 The processor is not in debug mode<br>1 The processor is in debug mode  |
| SMR       | SAFE mode request from MC_RGM is active indicator — This bit is set if a hardware SAFE mode request has been triggered. It is cleared when the hardware SAFE mode request has been cleared.<br>0 A SAFE mode request is not active<br>1 A SAFE mode request is active   |
| FIRC_SC   | FIRC State Change during mode transition indicator — This bit is set when the fast internal RC oscillator (16 MHz) is requested to change its power up/down state. It is cleared when the fast internal RC oscillator (16 MHz) has completed its state change.<br>0 No state change is taking place<br>1 A state change is taking place |
| SYSCLK_SW | System Clock Switching pending status —<br>0 No system clock source switching is pending<br>1 A system clock source switching is pending  |
| DFLASH_SC | DFLASH State Change during mode transition indicator — This bit is set when the DFLASH is requested to change its power up/down state. It is cleared when the DFLASH has completed its state change.<br>0 No state change is taking place<br>1 A state change is taking place   |
| CFLASH_SC | CFLASH State Change during mode transition indicator — This bit is set when the CFLASH is requested to change its power up/down state. It is cleared when the DFLASH has completed its state change.<br>0 No state change is taking place<br>1 A state change is taking place   |

### 8.3.1.8 RESET Mode Configuration Register (ME\_RESET\_MC)

Address 0xC3FD\_C020 Access: User read, Supervisor read/write, Test read/write

|       |              |   |   |   |   |   |   |   |     |   |   |       |        |   |        |   |
|-------|--------------|---|---|---|---|---|---|---|-----|---|---|-------|--------|---|--------|---|
| R     | 0            | 0 | 0 | 0 | 0 | 0 | 0 | 0 | PDO | 0 | 0 | MVRON | DFLAON |   | CFLAON |   |
| W     | [Greyed out] |   |   |   |   |   |   |   |     |   |   |       |        |   |        |   |
| Reset | 0            | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0   | 0 | 0 | 1     | 1      | 1 | 1      | 1 |

|       |              |   |   |   |   |   |   |   |   |   |   |        |        |   |   |   |
|-------|--------------|---|---|---|---|---|---|---|---|---|---|--------|--------|---|---|---|
| R     | 0            | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |   |   | FIRCON | SYSCLK |   |   |   |
| W     | [Greyed out] |   |   |   |   |   |   |   |   |   |   |        |        |   |   |   |
| Reset | 0            | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1      | 0      | 0 | 0 | 0 |

Figure 8-9. Invalid Mode Transition Status Register (ME\_IMTS)

This register configures system behavior during RESET mode. Please see [Table 8-11](#) for details.

### 8.3.1.9 TEST Mode Configuration Register (ME\_TEST\_MC)

Address 0xC3FD\_C024 Access: User read, Supervisor read/write, Test read/write

|       |              |   |   |   |   |   |   |   |     |   |   |       |        |   |        |   |
|-------|--------------|---|---|---|---|---|---|---|-----|---|---|-------|--------|---|--------|---|
| R     | 0            | 0 | 0 | 0 | 0 | 0 | 0 | 0 | PDO | 0 | 0 | MVRON | DFLAON |   | CFLAON |   |
| W     | [Greyed out] |   |   |   |   |   |   |   |     |   |   |       |        |   |        |   |
| Reset | 0            | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0   | 0 | 0 | 1     | 1      | 1 | 1      | 1 |

|       |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|-------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
|-------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

Figure 8-10. TEST Mode Configuration Register (ME\_TEST\_MC)

This register configures system behavior during TEST mode. Please see [Table 8-11](#) for details.

#### NOTE

Byte and half-word write accesses are not allowed to this register.

### 8.3.1.10 SAFE Mode Configuration Register (ME\_SAFE\_MC)

Address 0xC3FD\_C028 Access: User read, Supervisor read/write, Test read/write

|       |   |   |   |   |   |   |   |   |     |   |   |       |        |   |        |   |
|-------|---|---|---|---|---|---|---|---|-----|---|---|-------|--------|---|--------|---|
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | PDO | 0 | 0 | MVRON | DFLAON |   | CFLAON |   |
| W     |   |   |   |   |   |   |   |   |     |   |   |       |        |   |        |   |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1   | 0 | 0 | 1     | 1      | 1 | 1      | 1 |

|       |   |   |   |   |   |   |   |   |   |   |        |        |   |   |   |   |
|-------|---|---|---|---|---|---|---|---|---|---|--------|--------|---|---|---|---|
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |   |   | FIRCON | SYSCLK |   |   |   |   |
| W     |   |   |   |   |   |   |   |   |   |   |        |        |   |   |   |   |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0      | 1      | 0 | 0 | 0 | 0 |

Figure 8-11. SAFE Mode Configuration Register (ME\_SAFE\_MC)

This register configures system behavior during SAFE mode. Please see [Table 8-11](#) for details.

**NOTE**

Byte and half-word write accesses are not allowed to this register.

### 8.3.1.11 DRUN Mode Configuration Register (ME\_DRUN\_MC)

Address 0xC3FD\_C02C Access: User read, Supervisor read/write, Test read/write

|       |   |   |   |   |   |   |   |   |     |   |   |       |        |   |        |   |
|-------|---|---|---|---|---|---|---|---|-----|---|---|-------|--------|---|--------|---|
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | PDO | 0 | 0 | MVRON | DFLAON |   | CFLAON |   |
| W     |   |   |   |   |   |   |   |   |     |   |   |       |        |   |        |   |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0   | 0 | 0 | 1     | 1      | 1 | 1      | 1 |

|       |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|-------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
|-------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

Figure 8-12. DRUN Mode Configuration Register (ME\_DRUN\_MC)

This register configures system behavior during DRUN mode. Please see [Table 8-11](#) for details.

**NOTE**

Byte and half-word write accesses are not allowed to this register.

**NOTE**

The values of CFLAON, and DFLAON are retained through STANDBY mode.



### 8.3.1.12 RUN0...3 Mode Configuration Registers (ME\_RUN0...3\_MC)

Address 0xC3FD\_C030–0xC3FD\_C03C Access: User read, Supervisor read/write, Test read/write

|       |   |   |   |   |   |   |   |   |     |   |   |        |        |   |        |   |
|-------|---|---|---|---|---|---|---|---|-----|---|---|--------|--------|---|--------|---|
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | PDO | 0 | 0 | MVFRON | DFLAON |   | CFLAON |   |
| W     |   |   |   |   |   |   |   |   |     |   |   |        |        |   |        |   |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0   | 0 | 0 | 1      | 1      | 1 | 1      | 1 |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0   | 0 | 0 | 1      | 0      | 0 | 0      | 0 |

**Figure 8-13. RUN0...3 Mode Configuration Registers (ME\_RUN0...3\_MC)**

This register configures system behavior during RUN0...3 modes. Please see [Table 8-11](#) for details.

#### NOTE

Byte and half-word write accesses are not allowed to this register.

### 8.3.1.13 HALT Mode Configuration Register (ME\_HALT\_MC)

Address 0xC3FD\_C040 Access: User read, Supervisor read/write, Test read/write

|       |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|-------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |

**Figure 8-14. HALT Mode Configuration Register (ME\_HALT\_MC)**

This register configures system behavior during HALT mode. Please see [Table 8-11](#) for details.

#### NOTE

Byte and half-word write accesses are not allowed to this register.

### 8.3.1.14 STOP Mode Configuration Register (ME\_STOP\_MC)

This register configures system behavior during STOP mode. Please see [Table 8-11](#) for details.

#### NOTE

Byte and half-word write accesses are not allowed to this register.

Table 8-11. Mode Configuration Registers (ME\_&lt;mode&gt;\_MC) Field Descriptions

| Field  | Description  |
|--------|--|
| PDO    | <b>I/O output power-down control</b> — This bit controls the output power-down of I/Os.<br>0 No automatic safe gating of I/Os used and pads power sequence driver is enabled<br>1 In SAFE/TEST modes, outputs of pads are forced to high impedance state and pads power sequence driver is disabled. The inputs are level unchanged. In STOP mode, only pad power sequence driver is disabled but the state of the output is kept.   |
| MVRON  | <b>Main voltage regulator control</b> — This bit specifies whether main voltage regulator is switched off or not while entering this mode.<br>0 Main voltage regulator is switched off<br>1 Main voltage regulator is switched on  |
| DFLAON | <b>Data flash power-down control</b> — This bit specifies the operating mode of the data flash after entering this mode.<br>00 Reserved<br>01 Data flash is in power-down mode<br>10 Data flash is in low-power mode<br>11 Data flash is in normal mode<br><b>Note:</b> If the flash memory is to be powered down in any mode, then your software must ensure that reset sources are configured as long resets in the RGM_FESS register (see <a href="#">Section 9.3.1.7, Functional Event Short Sequence Register (RGM_FESS)</a> ). |
| CFLAON | <b>Code flash power-down control</b> — This bit specifies the operating mode of the program flash after entering this mode.<br>00 Reserved<br>01 Code flash is in power-down mode<br>10 Code flash is in low-power mode<br>11 Code flash is in normal mode   |
| FIRCON | <b>fast internal RC oscillator (16 MHz) control</b><br>0 fast internal RC oscillator (16 MHz) is switched off<br>1 fast internal RC oscillator (16 MHz) is switched on   |
| SYSClk | <b>System clock switch control</b> — These bits specify the system clock to be used by the system.<br>0000 16 MHz int. RC osc.<br>0001 div. 16 MHz int. RC osc.<br>0010 4–16 MHz ext. xtal osc.<br>0011 div. ext. xtal osc.<br>0100 freq. mod. PLL<br>0101 reserved<br>0110 reserved<br>0111 reserved<br>1000 reserved<br>1001 reserved<br>1010 reserved<br>1011 reserved<br>1100 reserved<br>1101 reserved<br>1110 reserved<br>1111 system clock is disabled  |

## 8.4 Functional description

### 8.4.1 Mode transition request

The transition from one mode to another mode is normally handled by software by accessing the mode control ME\_MCTL register. But in case of special events, mode transition can be automatically managed by hardware. In order to switch from one mode to another, the application should access ME\_MCTL register twice by writing

- The first time with the value of the key (0x5AF0) into the KEY bit field and the required target mode into the TARGET\_MODE bit field, and
- The second time with the inverted value of the key (0xA50F) into the KEY bit field and the required target mode into the TARGET\_MODE bit field.

Once a valid mode transition request is detected, the target mode configuration information is loaded from the corresponding ME\_<mode>\_MC register. The mode transition request may require a number of cycles depending on the programmed configuration, and software should check the S\_CURRENT\_MODE bit field and the S\_MTRANS bit of the global status register ME\_GS to verify when the mode has been correctly entered and the transition process has completed. For a description of valid mode requests, please see [Section 8.4.5, Mode transition interrupts](#).

Any modification of the mode configuration register of the currently selected mode will not be taken into account immediately but on the next request to enter this mode. This means that transition requests such as RUN0...3 → RUN0...3, DRUN → DRUN, SAFE → SAFE, and TEST → TEST are considered valid mode transition requests. As soon as the mode request is accepted as valid, the S\_MTRANS bit is set till the status in the ME\_GS register matches the configuration programmed in the respective ME\_<mode>\_MC register.

#### NOTE

It is recommended that software poll the S\_MTRANS bit in the ME\_GS register after requesting a transition to HALT, STOP, or STANDBY modes.

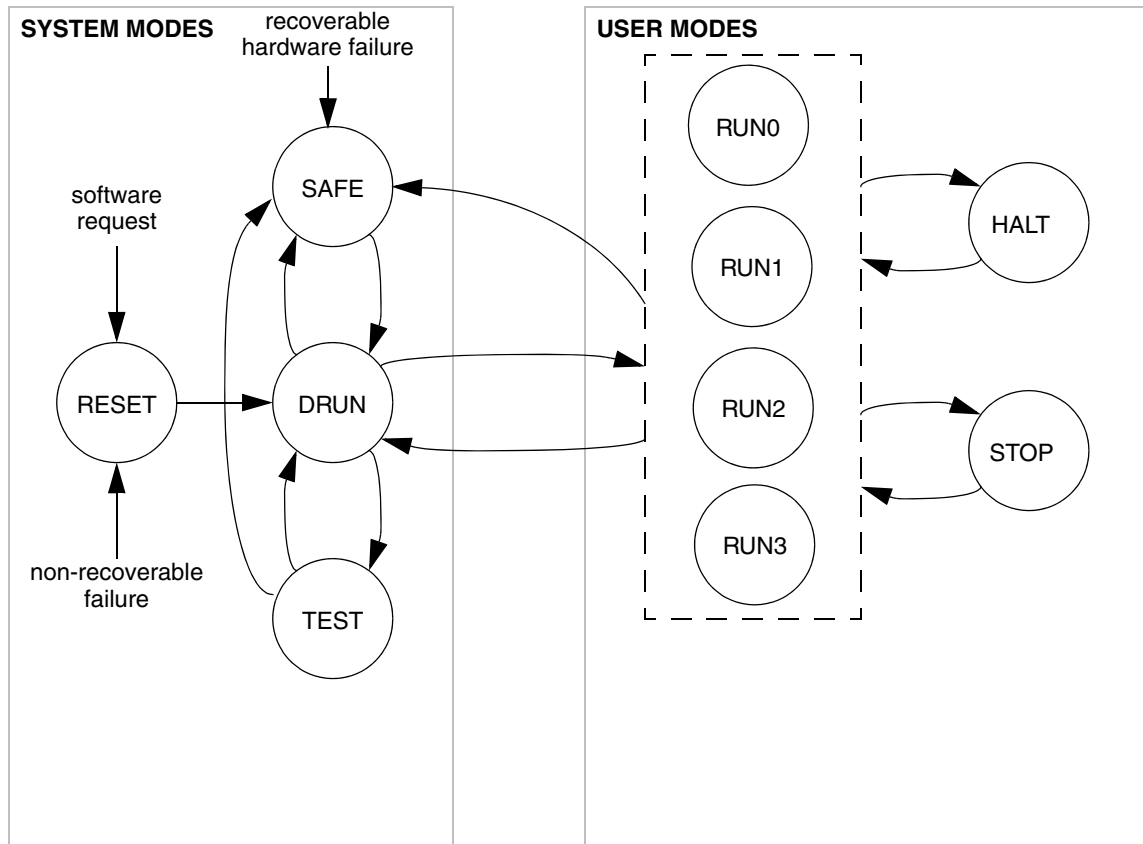


Figure 8-15. MC\_ME mode diagram

## 8.4.2 Modes details

### 8.4.2.1 RESET mode

The device enters this mode on the following events:

- From SAFE, DRUN, RUN0...3, or TEST mode when the TARGET\_MODE bit field of the ME\_MCTL register is written with 0000
- From any mode due to a system reset by the MC\_RGM because of some non-recoverable hardware failure in the system (see [Chapter 9, Reset Generation Module \(MC\\_RGM\)](#), for details)

Transition to this mode is instantaneous, and the system remains in this mode until the reset sequence is finished. The mode configuration information for this mode is provided by the ME\_RESET\_MC register. This mode has a predefined configuration, and the 16 MHz int. RC osc. is selected as the system clock.

### 8.4.2.2 DRUN mode

The device enters this mode on the following events.

- Automatically from RESET mode after completion of the reset sequence

- From RUN0...3, SAFE, or TEST mode when the TARGET\_MODE bit field of the ME\_MCTL register is written with 0011

As soon as any of the above events has occurred, a DRUN mode transition request is generated. The mode configuration information for this mode is provided by the ME\_DRUN\_MC register. In this mode, the flashes, all clock sources, and the system clock configuration can be controlled by software as required. After system reset, the software execution starts with the default configuration selecting the 16 MHz int. RC osc. as the system clock.

This mode is intended to be used by software to:

- Initialize all registers as per the system needs

#### NOTE

As flashes can be configured in low-power or power-down state in this mode, software must ensure that the code executes from RAM before changing to this mode.

### 8.4.2.3 SAFE mode

The device enters this mode on the following events:

- From DRUN, RUN0...3, or TEST mode when the TARGET\_MODE bit field of the ME\_MCTL register is written with 0010
- From any mode except RESET due to a SAFE mode request generated by the MC\_RGM because of some potentially recoverable hardware failure in the system (see [Chapter 9, Reset Generation Module \(MC\\_RGM\)](#), for details)

As soon as any of the above events has occurred, a SAFE mode transition request is generated. The mode configuration information for this mode is provided by the ME\_SAFE\_MC register. This mode has a predefined configuration, and the 16 MHz int. RC osc. is selected as the system clock.

If the SAFE mode is requested by software while some other mode transition process is ongoing, the new target mode becomes the SAFE mode regardless of other pending requests. In this case, the new mode request is not interpreted as an invalid request.

#### NOTE

If software requests to change to the SAFE mode and then requests to change back to the parent mode before the mode transition is completed, the device's final mode after mode transition will be the parent mode. However, this is not recommended software behavior. It is recommended for software to wait until the S\_MTRANS bit is cleared after requesting a change to SAFE before requesting another mode change.

As long as a SAFE event is active, the system remains in the SAFE mode and no write access is allowed to the ME\_MCTL register.

This mode is intended to be used by software to:

- Assess the severity of the cause of failure and then to either
  - Reinitialize the device via the DRUN mode, or

- Completely reset the device via the RESET mode.

If the outputs of the system I/Os need to be forced to a high impedance state upon entering this mode, the PDO bit of the ME\_SAFE\_MC register should be set. The input levels remain unchanged.

#### 8.4.2.4 TEST mode

The device enters this mode on the following events:

- From the DRUN mode when the TARGET\_MODE bit field of the ME\_MCTL register is written with 0001

As soon as any of the above events has occurred, a TEST mode transition request is generated. The mode configuration information for this mode is provided by the ME\_TEST\_MC register. Except for the main voltage regulator, all resources of the system are configurable in this mode. The system clock to the whole system can be stopped by programming the SYSCLK bit field to 1111, and in this case, the only way to exit this mode is via a device reset.

This mode is intended to be used by software to execute on-chip test routines.

#### NOTE

As flash modules can be configured to a low-power or power-down state in these modes, software must ensure that the code will execute from RAM before it changes to this mode.

#### 8.4.2.5 RUN0...3 modes

The device enters one of these modes on the following events:

- From the DRUN another RUN0...3 mode when the TARGET\_MODE bit field of the ME\_MCTL register is written with 0100...0111
- From the HALT mode by an interrupt event
- From the STOP mode by an interrupt or wakeup event

As soon as any of the above events occur, a RUN0...3 mode transition request is generated. The mode configuration information for these modes is provided by ME\_RUN0...3\_MC registers. In these modes, the flashes, all clock sources, and the system clock configuration can be controlled by software as required.

These modes are intended to be used by software to execute application routines.

#### NOTE

As flash modules can be configured to a low-power or power-down state in these modes, software must ensure that the code will execute from RAM before it changes to this mode.

#### 8.4.2.6 HALT mode

The device enters this mode on the following events:

- From one of the RUN0...3 modes when the TARGET\_MODE bit field of the ME\_MCTL register is written with 1000.

As soon as any of the above events occur, a HALT mode transition request is generated. The mode configuration information for this mode is provided by ME\_HALT\_MC register. This mode is quite configurable, and the ME\_HALT\_MC register should be programmed according to the system needs. The flashes can be put in power-down mode as needed. If there is a HALT mode request while an interrupt request is active, the device mode does not change, and an invalid mode interrupt is not generated.

This mode is intended as a first level low-power mode with

- The core clock frozen
- Only a few peripherals running

and to be used by software to wait until it is required to do something and then to react quickly (that is, within a few system clock cycles of an interrupt event).

### 8.4.2.7 STOP mode

The device enters this mode on the following events:

- From one of the RUN0...3 modes when the TARGET\_MODE bit field of the ME\_MCTL register is written with 1010.

As soon as any of the above events occur, a STOP mode transition request is generated. The mode configuration information for this mode is provided by the ME\_STOP\_MC register. This mode is fully configurable, and the ME\_STOP\_MC register should be programmed according to the system needs. The flashes can be put in power-down mode as needed. If there is a STOP mode request while any interrupt or wakeup event is active, the device mode does not change, and an invalid mode interrupt is not generated.

This can be used as an advanced low-power mode with the core clock frozen and almost all peripherals stopped.

This mode is intended as an advanced low-power mode with

- The core clock frozen
- Almost all peripherals stopped

and to be used by software to wait until it is required to do something with no need to react quickly (e.g. allow for system clock source to be restarted).

This mode can be used to stop all clock sources, thus preserving the device status. When exiting the STOP mode, the fast internal RC oscillator (16 MHz) clock is selected as the system clock until the target clock is available.

## 8.4.3 Mode transition process

The process of mode transition follows the following steps in a predefined manner depending on the current device mode and the requested target mode. In many cases of mode transition, not all steps need to be executed based on the mode control information, and some steps may not be valid according to the mode definition itself.

### 8.4.3.1 Target mode request

The target mode is requested by accessing the ME\_MCTL register with the required keys. This mode transition request by software must be a valid request satisfying a set of predefined rules to initiate the process. If the request fails to satisfy these rules, it is ignored, and the TARGET\_MODE bit field is not updated. An optional interrupt can be generated for invalid mode requests. See [Section 8.4.5, Mode transition interrupts](#), for details.

In the case of mode transitions occurring because of hardware events such as a reset, a SAFE mode request, or interrupt requests and wakeup events to exit from low-power modes, the TARGET\_MODE bit field of the ME\_MCTL register is automatically updated with the appropriate target mode. The mode change process start is indicated by the setting of the mode transition status bit S\_MTRANS of the ME\_GS register.

A RESET mode requested via the ME\_MCTL register is passed to the MC\_RGM, which generates a global system reset and initiates the reset sequence. The RESET mode request has the highest priority, and the MC\_ME is kept in the RESET mode during the entire reset sequence.

The SAFE mode request has the next highest priority after reset which can be generated by software via the ME\_MCTL register from all software running modes including DRUN, RUN0...3, and TEST or by the MC\_RGM after the detection of system hardware failures, which may occur in any mode.

### 8.4.3.2 Target mode configuration loading

On completion of the [Target mode request](#), the target mode configuration from the ME\_<target mode>\_MC register is loaded to start the resources (voltage sources, clock sources, flashes, pads, etc.) control process.

An overview of resource control possibilities for each mode is shown in [Table 8-12](#). A √ indicates that a given resource is configurable for a given mode.

**Table 8-12. MC\_ME resource control overview**

| Resource | Mode   |             |        |             |             |                |                 |            |
|----------|--------|-------------|--------|-------------|-------------|----------------|-----------------|------------|
|          | RESET  | TEST        | SAFE   | DRUN        | RUN0        | HALT           | STOP            | STANDBY    |
| FIRC     | on     | √<br>on     | on     | on          | on          | √<br>on        | √<br>on         | √<br>on    |
| CFLASH   | normal | √<br>normal | normal | √<br>normal | √<br>normal | √<br>low-power | √<br>power-down | power-down |
| DFLASH   | normal | √<br>normal | normal | √<br>normal | √<br>normal | √<br>low-power | √<br>power-down | power-down |
| MVREG    | on     | on          | on     | on          | on          | √<br>on        | √<br>on         | off        |



Table 8-12. MC\_ME resource control overview (continued)

| Resource | Mode  |          |         |      |      |      |          |         |
|----------|-------|----------|---------|------|------|------|----------|---------|
|          | RESET | TEST     | SAFE    | DRUN | RUN0 | HALT | STOP     | STANDBY |
| PDO      | off   | √<br>off | √<br>on | off  | off  | off  | √<br>off | on      |

### 8.4.3.3 Peripheral Clocks Disable

On completion of the [Target mode request](#), the MC\_ME requests each peripheral to enter its stop mode when:

The MC\_ME does not automatically request peripherals to enter their stop modes if the power domains in which they are residing are to be turned off due to a mode change. Therefore, it is software's responsibility to ensure that those peripherals that are to be powered down are configured in the MC\_ME to be frozen.

Each peripheral that may block or disrupt a communication bus to which it is connected ensures that these outputs are forced to a safe or recessive state when the device enters the SAFE mode.

### 8.4.3.4 Processor low-power mode entry

If, on completion of the [Peripheral Clocks Disable](#), the mode transition is to the HALT mode, the MC\_ME requests the processor to enter its halted state. The processor acknowledges its halt state request after completing all outstanding bus transactions.

If, on completion of the [Peripheral Clocks Disable](#), the mode transition is to the STOP mode, the MC\_ME requests the processor to enter its stopped state. The processor acknowledges its stop state request after completing all outstanding bus transactions.

### 8.4.3.5 Processor and System Memory Clock Disable

If, on completion of the [Processor low-power mode entry](#), the mode transition is to the HALT STOP mode and the processor is in its appropriate halted or stopped state, the MC\_ME disables the processor and system memory clocks to achieve further power saving.

The clocks to the processor and system memories are unaffected for all transitions between software running modes including DRUN, RUN0...3, and SAFE.

#### CAUTION

Clocks to the whole device including the processor and system memories can be disabled in TEST mode.

### 8.4.3.6 Clock sources switch-on

On completion of the [Processor low-power mode entry](#), the MC\_ME controls all clock sources that affect the system clock based on the *<clock source>ON* bits of the ME\_*<current mode>*\_MC and ME\_*<target mode>*\_MC registers. The following system clock sources are controlled at this step:

- The fast internal RC oscillator (16 MHz)

The clock sources that are required by the target mode are switched on. The duration required for the output clocks to be stable depends on the type of source, and all further steps of mode transition depending on one or more of these clocks waits for the stable status of the respective clocks. The availability status of these system clocks is updated in the S\_*<clock source>* bits of ME\_GS register.

The clock sources that need to be switched off are unaffected during this process in order to not disturb the system clock, which might require one of these clocks before switching to a different target clock.

It is also possible to automatically switch-on the FXOSC after exiting STANDBY mode, by programming ME\_DRUN\_MC[FXOSC] = 1 prior to STANDBY entry. After the chip exits STANDBY mode, your software should wait for FXOSC to be switched-on before any mode transition request in order to avoid a mode request illegal event.

### 8.4.3.7 Main voltage regulator switch-on

On completion of the [Target mode request](#), if the main voltage regulator needs to be switched on from its off state based on the MVRON bit of the ME\_*<current mode>*\_MC and ME\_*<target mode>*\_MC registers, the MC\_ME requests the MC\_PCU to power-up the regulator and waits for the output voltage stable status in order to update the S\_MVR bit of the ME\_GS register.

This step is required only during the exit of the low-power modes HALT and STOP. In this step, the fast internal RC oscillator (16 MHz) is switched on regardless of the target mode configuration, as the main voltage regulator requires the 16 MHz int. RC osc. during power-up in order to generate the voltage status.

### 8.4.3.8 Flash modules switch-on

On completion of the [Main voltage regulator switch-on](#), if a flash module needs to be switched to normal mode from its low-power or power-down mode based on the CFLAON and DFLAON bit fields of the ME\_*<current mode>*\_MC and ME\_*<target mode>*\_MC registers, the MC\_ME requests the flash to exit from its low-power/power-down mode. When the flash modules are available for access, the S\_CFLA and S\_DFLA bit fields of the ME\_GS register are updated to 11 by hardware.

If the main regulator is also off in device low-power modes, then during the exit sequence, the flash is kept in its low-power state and is switched on only when the [Main voltage regulator switch-on](#) process has completed.

#### CAUTION

It is illegal to switch the flashes from low-power mode to power-down mode and from power-down mode to low-power mode. The MC\_ME, however, does not prevent this nor does it flag it.

### 8.4.3.9 Pad outputs-on

On completion of the [Main voltage regulator switch-on](#), if the PDO bit of the ME\_<target mode>\_MC register is cleared, then

- All pad outputs are enabled to return to their previous state
- The I/O pads power sequence driver is switched on

### 8.4.3.10 Processor and memory clock enable

If the mode transition is from any of the low-power modes HALT or STOP to RUN0...3, the clocks to the processor and system memories are enabled. The process of enabling these clocks is executed only after the [Flash modules switch-on](#) process is completed.

### 8.4.3.11 Processor low-power mode exit

If the mode transition is from any of the low-power modes HALTSTOP to RUN0...3, the MC\_ME requests the processor to exit from its halted or stopped state. This step is executed only after the [Processor and memory clock enable](#) process is completed.

### 8.4.3.12 System clock switching

Based on the SYSCLK bit field of the ME\_<current mode>\_MC and ME\_<target mode>\_MC registers, if the target and current system clock configurations differ, the following method is implemented for clock switching.

- The target clock configuration for the 16 MHz int. RC osc. is effective only when the S\_FIRC bit of the ME\_GS register is set by hardware (i.e. the fast internal RC oscillator (16 MHz) has stabilized).
- If the clock is to be disabled, the SYSCLK bit field should be programmed with 1111. This is possible only in the TEST mode.

The current system clock configuration can be observed by reading the S\_SYSCLK bit field of the ME\_GS register, which is updated after every system clock switching. Until the target clock is available, the system uses the previous clock configuration.

System clock switching starts only after

- The [Clock sources switch-on](#) process has completed if the target system clock source needs to be switched on

An overview of system clock source selection possibilities for each mode is shown in [Table 8-13](#). A √ indicates that a given clock source is selectable for a given mode.

Table 8-13. MC\_ME system clock selection overview

| System Clock Source      | Mode           |                |                |                |                |                |                |                |
|--------------------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
|                          | RESET          | TEST           | SAFE           | DRUN           | RUN0           | HALT           | STOP           | STANDBY        |
| 16 MHz int. RC osc.      | √<br>(default) | √<br>(default) | √<br>(default) | √<br>(default) | √<br>(default) | √<br>(default) | √<br>(default) |                |
| system clock is disabled |                | √ <sup>1</sup> |                |                |                |                | √              | √<br>(default) |

<sup>1</sup> disabling the system clock during TEST mode will require a reset in order to exit TEST mode

#### 8.4.3.13 Pad switch-off

If the PDO bit of the ME\_<target mode>\_MC register is 1 then

- The outputs of the pads are forced to the high impedance state if the target mode is SAFE or TEST

This step is executed only after the [Peripheral Clocks Disable](#) process is completed.

#### 8.4.3.14 Clock sources switch-off

<clock source><mode>if a given clock source<clock source>.

This step is executed only after

- [System clock switching](#) process is completed in order not to lose the current system clock during mode transition.

#### 8.4.3.15 Flash switch-off

Based on the CFLAON and DFLAON bit fields of the ME\_<current mode>\_MC and ME\_<target mode>\_MC registers, if any of the flash modules is to be put in a low-power state, the MC\_ME requests the flash to enter the corresponding low-power state and waits for the deassertion of flash ready status signal. The exact low-power mode status of the flash modules is updated in the S\_CFLA and S\_DFLA bit fields of the ME\_GS register. This step is executed only when [Processor and System Memory Clock Disable](#) process is completed.

#### 8.4.3.16 Main voltage regulator switch-off

Based on the MVRON bit of the ME\_<current mode>\_MC and ME\_<target mode>\_MC registers, if the main voltage regulator is to be switched off, the MC\_ME requests it to power down and clears the availability status bit S\_MVR of the ME\_GS register.

This step is required only during the entry of low-power modes like HALT and STOP. This step is executed only after completing the following processes:

- [Flash switch-off](#)

- The device consumption is less than the predefined threshold value (i.e. the S\_DC bit of the ME\_GS register is 0).

#### 8.4.3.17 Current mode update

The current mode status bit field S\_CURRENT\_MODE of the ME\_GS register is updated with the target mode bit field TARGET\_MODE of the ME\_MCTL register when:

- All the updated status bits in the ME\_GS register match the configuration specified in the ME\_<target mode>\_MC register
- Power sequences are done
- Clock disable/enable process is finished
- Processor low-power mode (halt/stop) entry and exit processes are finished

Software can monitor the mode transition status by reading the S\_MTRANS bit of the ME\_GS register. The mode transition latency can differ from one mode to another depending on the resources' availability before the new mode request and the target mode's requirements.

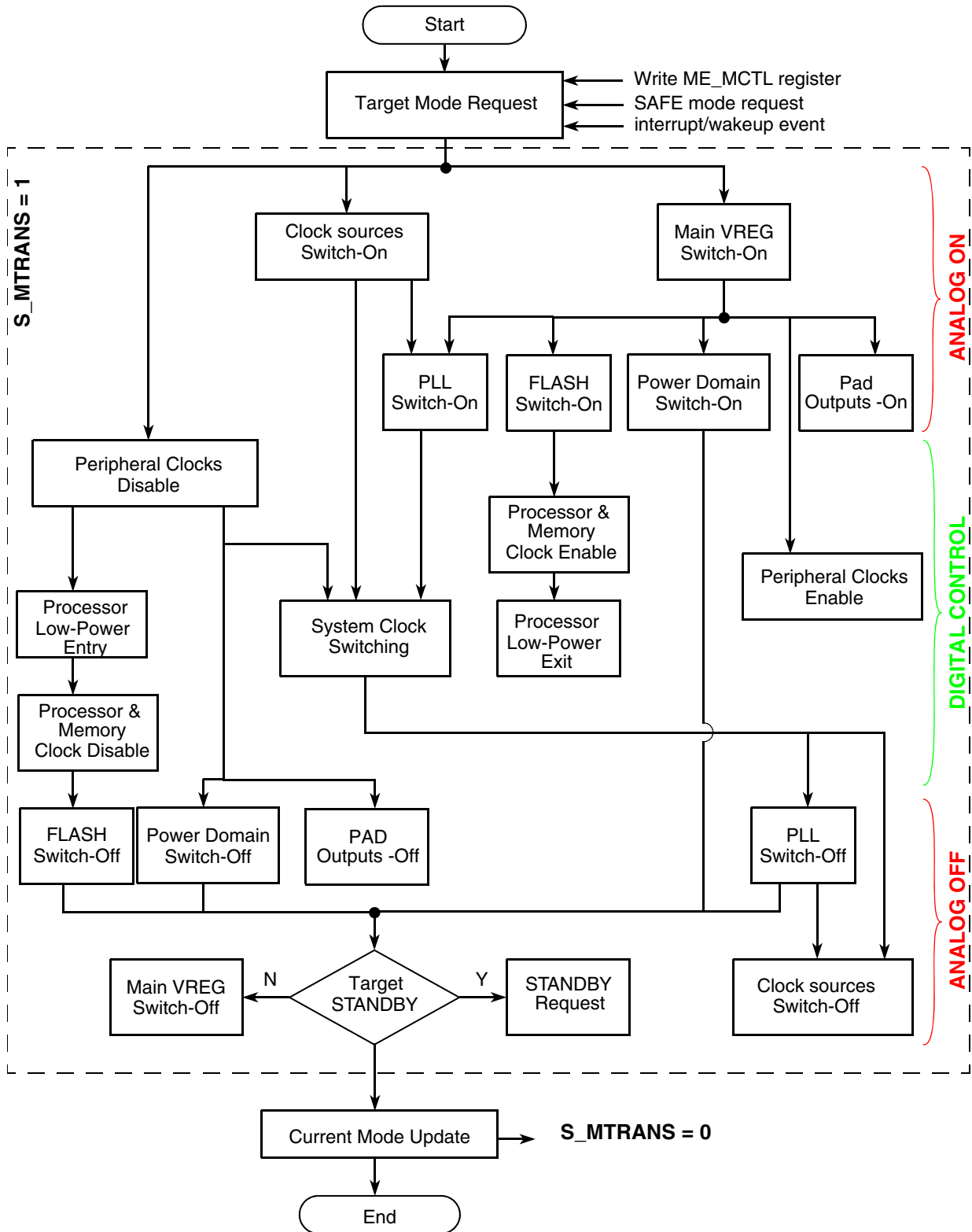


Figure 8-16. MC\_ME transition diagram

### 8.4.4 Protection of mode configuration registers

While programming the mode configuration registers `ME_<mode>_MC`, the following rules must be respected. Otherwise, the write operation is ignored and an invalid mode configuration interrupt may be generated.

- FIRC must be on if the system clock is one of the following:
  - The 16 MHz int. RC osc.
- The Configuration 00 for the CFLAON and DFLAON bit fields are reserved.
- MVREG must be on if any of the following is active:
  - CFLASH
  - DFLASH
- System clock configurations marked as reserved may not be selected.
- Configuration 1111 for the SYSCLK bit field is allowed only for the TEST mode, and only in this case may all system clock sources be turned off.

#### CAUTION

If the system clock is stopped during TEST mode, the device can exit only via a system reset.

### 8.4.5 Mode transition interrupts

The following are the three interrupts related to mode transition implemented in the MC\_ME.

#### 8.4.5.1 Invalid mode configuration interrupt

Whenever a write operation is attempted to the `ME_<mode>_MC` registers violating the protection rules mentioned in the [Section 8.4.4, Protection of mode configuration registers](#), the interrupt pending bit `I_ICONF` of the `ME_IS` register is set, and an interrupt request is generated if the mask bit `M_ICONF` of `ME_IM` register is 1.

#### 8.4.5.2 Invalid mode transition interrupt

The mode transition request is considered invalid under the following conditions:

- If the system is in the SAFE mode and the SAFE mode request from `MC_RGM` is active, and if the target mode requested is other than RESET or SAFE, then this new mode request is considered to be invalid, and the `S_SEA` bit of the `ME_IMTS` register is set.
- If the `TARGET_MODE` bit field of the `ME_MCTL` register is written with a value different from the specified mode values (i.e. a non existing mode), an invalid mode transition event is generated. When such a non existing mode is requested, the `S_NMA` bit of the `ME_IMTS` register is set. This condition is detected regardless of whether the proper key mechanism is followed while writing the `ME_MCTL` register.
- If some of the device modes are disabled as programmed in the `ME_ME` register, their respective configurations are considered reserved, and any access to the `ME_MCTL` register with those values results in an invalid mode transition request. When such a disabled mode is requested, the

S\_DMA bit of the ME\_IMTS register is set. This condition is detected regardless of whether the proper key mechanism is followed while writing the ME\_MCTL register.

- If the target mode is not a valid mode with respect to current mode, the mode request illegal status bit S\_MRI of the ME\_IMTS register is set. This condition is detected only when the proper key mechanism is followed while writing the ME\_MCTL register. Otherwise, the write operation is ignored.
- If further new mode requests occur while a mode transition is in progress (the S\_MTRANS bit of the ME\_GS register is 1), the mode transition illegal status bit S\_MTI of the ME\_IMTS register is set. This condition is detected only when the proper key mechanism is followed while writing the ME\_MCTL register. Otherwise, the write operation is ignored.

#### NOTE

As the causes of invalid mode transitions may overlap at the same time, the priority implemented for invalid mode transition status bits of the ME\_IMTS register in the order from highest to lowest is S\_SEA, S\_NMA, S\_DMA, S\_MRI, and S\_MTI.

As an exception, the mode transition request is not considered as invalid under the following conditions:

- A new request is allowed to enter the RESET or SAFE mode irrespective of the mode transition status.
- As the exit of HALT and STOP modes depends on the interrupts of the system, which can occur at any instant, these requests to return to RUN0...3 modes are always valid.
- In order to avoid any unwanted lockup of the device modes, software can abort a mode transition by requesting the parent mode if, for example, the mode transition has not completed after a software determined reasonable amount of time for whatever reason. The parent mode is the device mode before a valid mode request was made.
- Self-transition requests (e.g. RUN0 → RUN0) are not considered as invalid even when the mode transition process is active (i.e. S\_MTRANS is 1). During the low-power mode exit process, if the system is not able to enter the respective RUN0...3 mode properly (i.e. all status bits of the ME\_GS register match with configuration bits in the ME\_<mode>\_MC register), then software can only request the SAFE or RESET mode. It is not possible to request any other mode or to go back to the low-power mode again.

Whenever an invalid mode request is detected, the interrupt pending bit I\_IMODE of the ME\_IS register is set, and an interrupt request is generated if the mask bit M\_IMODE of ME\_IM register is 1.

#### 8.4.5.3 SAFE mode transition interrupt

Whenever the system enters the SAFE mode as a result of a SAFE mode request from the MC\_RGM due to a hardware failure, the interrupt pending bit I\_SAFE of the ME\_IS register is set, and an interrupt is generated if the mask bit M\_SAFE of ME\_IM register is 1.

The SAFE mode interrupt pending bit can be cleared only when the SAFE mode request is deasserted by the MC\_RGM (see [Chapter 9, Reset Generation Module \(MC\\_RGM\)](#), for details on how to clear a SAFE mode request). If the system is already in SAFE mode, any new SAFE mode request by the MC\_RGM



also sets the interrupt pending bit I\_SAFE. However, the SAFE mode interrupt pending bit is not set when the SAFE mode is entered by a software request (i.e. programming of ME\_MCTL register).

#### 8.4.5.4 Mode transition complete interrupt

Whenever the system completes a mode transition fully (i.e. the S\_MTRANS bit of ME\_GS register transits from 1 to 0), the interrupt pending bit I\_MTC of the ME\_IS register is set, and interrupt request is generated if the mask bit M\_MTC of the ME\_IM register is 1. The interrupt bit I\_MTC is not set when entering low-power modes HALT and STOP in order to avoid the same event requesting the exit of these low-power modes.

#### 8.4.6 Application example

Figure 8-17 shows an example application flow for requesting a mode change and then waiting until the mode transition has completed.

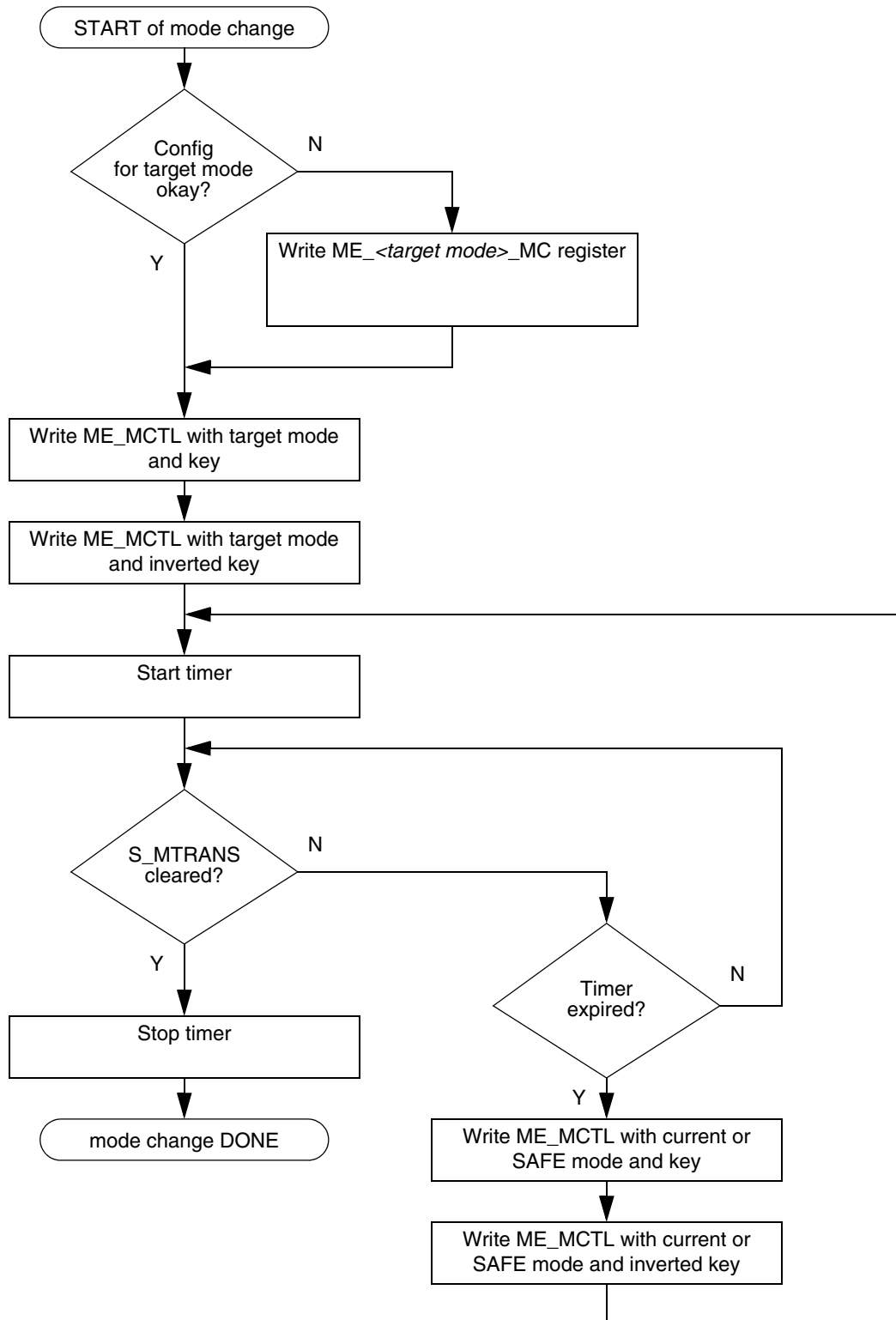


Figure 8-17. MC\_ME application example flow diagram

# Chapter 9

## Reset Generation Module (MC\_RGM)

### 9.1 Introduction

#### 9.1.1 Overview

The reset generation module (MC\_RGM) centralizes the different reset sources and manages the reset sequence of the device. It provides a register interface and the reset sequencer. The different registers are available to monitor and control the device reset sequence. The reset sequencer is a state machine that controls the different phases (PHASE0, PHASE1, PHASE2, PHASE3, and IDLE) of the reset sequence and control the reset signals generated in the system.

[Figure 9-1](#) shows the MC\_RGM block diagram.

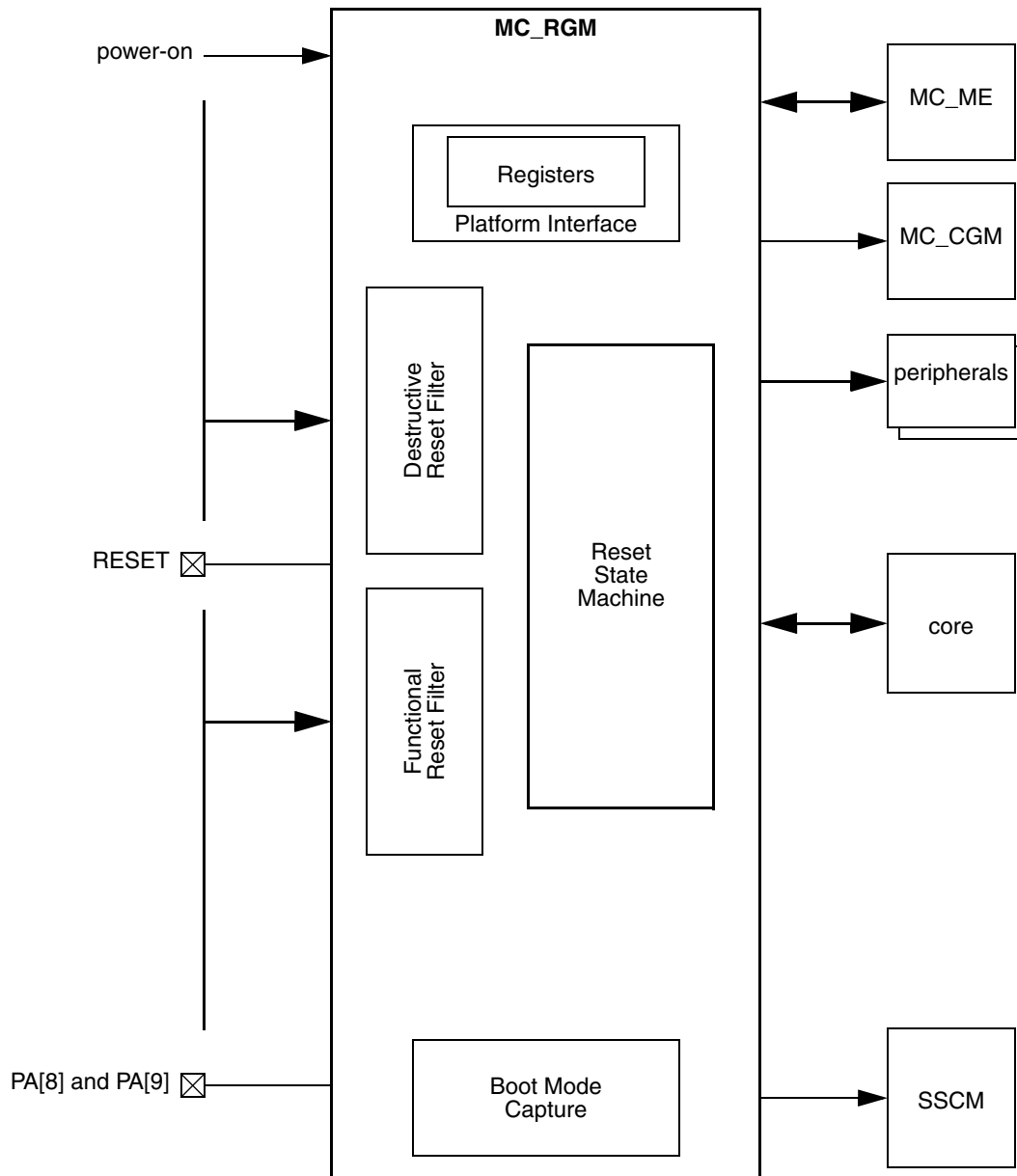


Figure 9-1. MC\_RGM block diagram

### 9.1.2 Features

The MC\_RGM contains the functionality for the following features:

- Destructive resets management
- Functional resets management
- Signalling of reset events after each reset sequence (reset status flags)
- Conversion of reset events to SAFE mode or interrupt request events [Chapter 8, Mode Entry Module \(MC\\_ME\)](#)
- Short reset sequence configuration

- Bidirectional reset behavior configuration
- [Chapter 8, Mode Entry Module \(MC\\_ME\)](#) Boot mode capture on RESET deassertion

### 9.1.3 Modes of operation

The different reset sources are organized into two families: destructive and functional.

- A destructive reset source is associated with an event related to a critical—usually hardware—error or dysfunction. When a destructive reset event occurs, the full reset sequence is applied to the device starting from PHASE0. This resets the full device ensuring a safe start-up state for both digital and analog modules. Destructive resets are
  - Power-on reset
- A functional reset source is associated with an event related to a less-critical—usually non-hardware—error or dysfunction. When a functional reset event occurs, a partial reset sequence is applied to the device starting from PHASE1. In this case, most digital modules are reset normally, while analog modules or specific digital modules' (e.g. debug modules, flash modules) state is preserved. Functional resets are
  - External reset

When a reset is triggered, the MC\_RGM state machine is activated and proceeds through the different phases (i.e. PHASE $n$  states). Each phase is associated with a particular device reset being provided to the system. A phase is completed when all corresponding phase completion gates from either the system or internal to the MC\_RGM are acknowledged. The device reset associated with the phase is then released, and the state machine proceeds to the next phase up to entering the IDLE phase. During this entire process, the MC\_ME state machine is held in RESET mode. Only at the end of the reset sequence, when the IDLE phase is reached, does the MC\_ME enter the DRUN mode.

Alternatively, it is possible for software to configure some reset source events to be converted from a reset to either a SAFE mode request issued to the MC\_ME or to an interrupt issued to the core (see [Section 9.3.1.4, Destructive Event Reset Disable Register \(RGM\\_DERD\)](#), and [Section 9.3.1.6, Destructive Event Alternate Request Register \(RGM\\_DEAR\)](#), for destructive resets, and [Section 9.3.1.3, Functional Event Reset Disable Register \(RGM\\_FERD\)](#), and [Section 9.3.1.5, Functional Event Alternate Request Register \(RGM\\_FEAR\)](#), for functional resets).

## 9.2 External signal description

The MC\_RGM interfaces to the reset pin RESET and the boot mode pins PA[8] and PA[9].

### 9.3 Memory map and register definition

Table 9-1. MC\_RGM register description

| Address     | Name     | Description                           | Size      | Access |                         |                         | Location    |
|-------------|----------|---------------------------------------|-----------|--------|-------------------------|-------------------------|-------------|
|             |          |                                       |           | Normal | Supervisor              | Test                    |             |
| 0xC3FE_4000 | RGM_FES  | Functional Event Status               | half-word | read   | read/write <sup>1</sup> | read/write <sup>1</sup> | on page 207 |
| 0xC3FE_4002 | RGM_DES  | Destructive Event Status              | half-word | read   | read/write <sup>1</sup> | read/write <sup>1</sup> | on page 208 |
| 0xC3FE_4004 | RGM_FERD | Functional Event Reset Disable        | half-word | read   | read                    | read                    | on page 209 |
| 0xC3FE_4006 | RGM_DERD | Destructive Event Reset Disable       | half-word | read   | read                    | read                    | on page 211 |
| 0xC3FE_4010 | RGM_FEAR | Functional Event Alternate Request    | half-word | read   | read                    | read                    | on page 212 |
| 0xC3FE_4012 | RGM_DEAR | Destructive Event Alternate Request   | half-word | read   | read                    | read                    | on page 213 |
| 0xC3FE_4018 | RGM_FESS | Functional Event Short Sequence       | half-word | read   | read                    | read                    | on page 214 |
| 0xC3FE_401C | RGM_FBRE | Functional Bidirectional Reset Enable | half-word | read   | read/write              | read/write              | on page 216 |

<sup>1</sup> individual bits cleared on writing 1

#### NOTE

Any access to unused registers as well as write accesses to read-only registers will:

- Not change register content
- Cause a transfer error

Table 9-2. MC\_RGM Memory Map

|             |                   |   |       |   |   |   |   |   |   |   |   |   |   |  |  |  |         |  |  |  |  |
|-------------|-------------------|---|-------|---|---|---|---|---|---|---|---|---|---|--|--|--|---------|--|--|--|--|
| 0xC3FE_4000 | RGM_FES / RGM_DES | R | F_EXR | 0 | 0 | 0 | 0 | 0 | 0 |   |   |   |   |  |  |  | CHKSTOP |  |  |  |  |
|             |                   | W | w1c   |   |   |   |   |   |   |   |   |   |   |  |  |  |         |  |  |  |  |
|             |                   | R | F_POR | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |  |  |  |         |  |  |  |  |
|             |                   | W | w1c   |   |   |   |   |   |   |   |   |   |   |  |  |  |         |  |  |  |  |

Table 9-2. MC\_RGM Memory Map (continued)

|                                   |                       |   |        |   |   |   |   |   |   |   |   |   |   |   |  |  |  |         |  |  |  |  |
|-----------------------------------|-----------------------|---|--------|---|---|---|---|---|---|---|---|---|---|---|--|--|--|---------|--|--|--|--|
| 0xC3FE_4004                       | RGM_FERD/<br>RGM_DERD | R | D_EXR  | 0 | 0 | 0 | 0 | 0 | 0 |   |   |   |   |   |  |  |  | CHKSTOP |  |  |  |  |
|                                   |                       | W |        |   |   |   |   |   |   |   |   |   |   |   |  |  |  |         |  |  |  |  |
|                                   |                       | R |        | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |  |  |  |         |  |  |  |  |
|                                   |                       | W |        |   |   |   |   |   |   |   |   |   |   |   |  |  |  |         |  |  |  |  |
| 0xC3FE_4008<br>...<br>0xC3FE_400C | reserved              |   |        |   |   |   |   |   |   |   |   |   |   |   |  |  |  |         |  |  |  |  |
| 0xC3FE_4010                       | RGM_FEAR/<br>RGM_DEAR | R | AR_EXR | 0 | 0 | 0 | 0 | 0 | 0 |   |   |   |   |   |  |  |  | CHKSTOP |  |  |  |  |
|                                   |                       | W |        |   |   |   |   |   |   |   |   |   |   |   |  |  |  |         |  |  |  |  |
|                                   |                       | R |        | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |  |  |  |         |  |  |  |  |
|                                   |                       | W |        |   |   |   |   |   |   |   |   |   |   |   |  |  |  |         |  |  |  |  |
| 0xC3FE_4014                       | reserved              |   |        |   |   |   |   |   |   |   |   |   |   |   |  |  |  |         |  |  |  |  |
| 0xC3FE_4018                       | RGM_FESS              | R | SS_EXR | 0 | 0 | 0 | 0 | 0 | 0 |   |   |   |   |   |  |  |  | CHKSTOP |  |  |  |  |
|                                   |                       | W |        |   |   |   |   |   |   |   |   |   |   |   |  |  |  |         |  |  |  |  |
| 0xC3FE_401C                       | RGM_FBRE              | R | BE_EXR | 0 | 0 | 0 | 0 | 0 | 0 |   |   |   |   |   |  |  |  | CHKSTOP |  |  |  |  |
|                                   |                       | W |        |   |   |   |   |   |   |   |   |   |   |   |  |  |  |         |  |  |  |  |
| 0xC3FE_4020<br>...<br>0xC3FE_7FFC | reserved              |   |        |   |   |   |   |   |   |   |   |   |   |   |  |  |  |         |  |  |  |  |

### 9.3.1 Register descriptions

Unless otherwise noted, all registers may be accessed as 32-bit words, 16-bit half-words, or 8-bit bytes. The bytes are ordered according to big endian. For example, the RGM\_STDBY register may be accessed as a word at address 0xC3FE\_4018, as a half-word at address 0xC3FE\_401A, or as a byte at address 0xC3FE\_401B.

### 9.3.1.1 Functional Event Status Register (RGM\_FES)

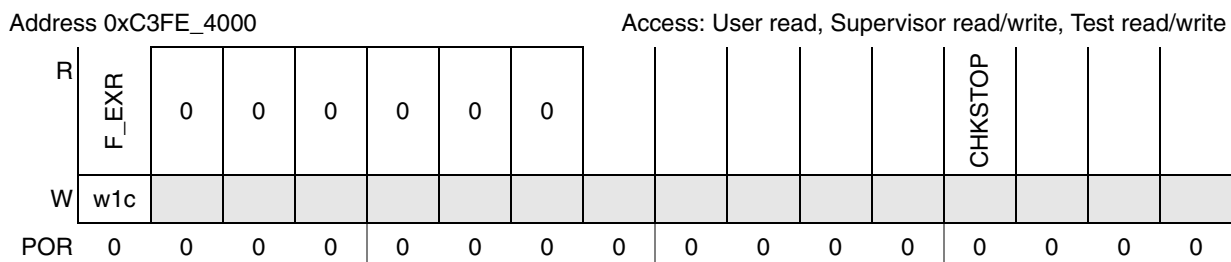


Figure 9-2. Functional Event Status Register (RGM\_FES)

This register contains the status of the last asserted functional reset sources. It can be accessed in read/write on either supervisor mode or test mode. Register bits are cleared on write 1.

Table 9-3. Functional Event Status Register (RGM\_FES) field descriptions

| Field | Description  |
|-------|--|
| F_EXR | <b>Flag for External Reset</b><br>0 No external reset event has occurred since either the last clear or the last destructive reset assertion<br>1 An external reset event has occurred |

### 9.3.1.2 Destructive Event Status Register (RGM\_DES)

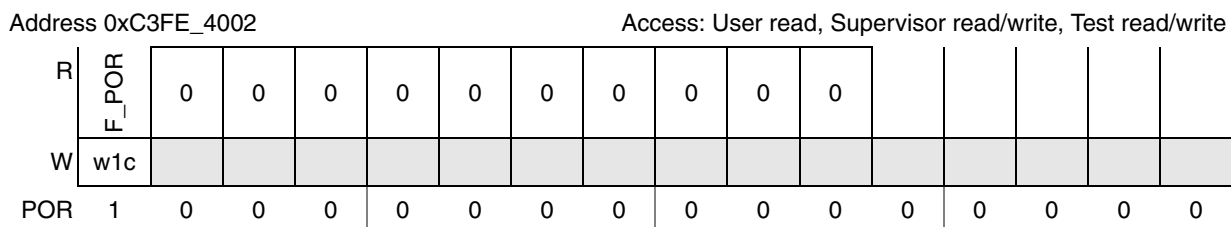


Figure 9-3. Destructive Event Status Register (RGM\_DES)

This register contains the status of the last asserted destructive reset sources. It can be accessed in read/write on either supervisor mode or test mode. Register bits are cleared on write 1.

Table 9-4. Destructive Event Status Register (RGM\_DES) field descriptions

| Field | Description  |
|-------|--|
| F_POR | <b>Flag for Power-On reset</b><br>0 No power-on event has occurred since the last clear (due to either a software clear or a low-voltage detection)<br>1 A power-on event has occurred |

**NOTE**

The F\_POR flag is also set when a low-voltage is detected on the 1.2 V supply, even if the low voltage is detected after power-on has completed.



The F\_LVD27 flag may still have the value 0 after a dip has occurred on the 2.7 V supply during a non-monotonic power-on sequence. The F\_POR flag will, however, still be set in this case as expected after each power-on sequence.

In contrast to all other reset sources, the 1.2 V low-voltage detected (power domain #0) event is captured on its deassertion. Therefore, the status bit F\_LVD12\_PD0 is also asserted on the reset's deassertion. In case an alternate event is selected, the SAFE mode or interrupt request are similarly asserted on the reset's deassertion.

### 9.3.1.3 Functional Event Reset Disable Register (RGM\_FERD)

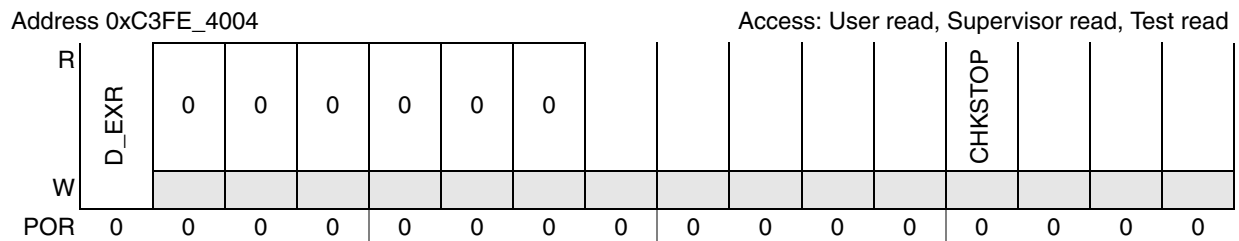


Figure 9-4. Functional Event Reset Disable Register (RGM\_FERD)

This register provides dedicated bits to disable functional reset sources. When a functional reset source is disabled, the associated functional event will trigger either a SAFE mode request or an interrupt request (see [Section 9.3.1.5, Functional Event Alternate Request Register \(RGM\\_FEAR\)](#)). It can be accessed in read/write in either supervisor mode or test mode. It can be accessed in read only in user mode. Each byte can be written only once after power-on reset.

Table 9-5. Functional Event Reset Disable Register (RGM\_FERD) field descriptions

| Field | Description   |
|-------|---|
| D_EXR | <b>Disable External Reset</b><br>0 An external reset event triggers a reset sequence<br>1 An external reset event generates a SAFE mode request |

### 9.3.1.4 Destructive Event Reset Disable Register (RGM\_DERD)

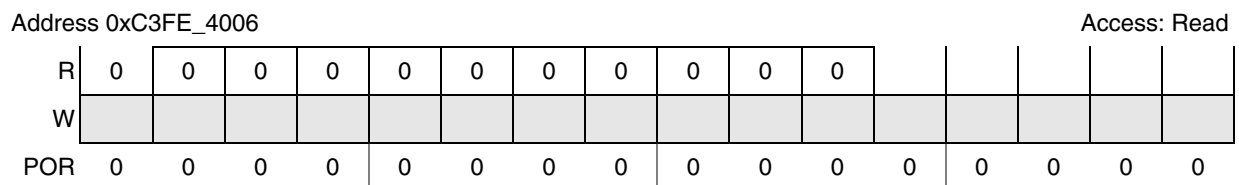


Figure 9-5. Destructive Event Reset Disable Register (RGM\_DERD)

This register provides dedicated bits to disable particular destructive reset sources. When a destructive reset source is disabled, the associated destructive event will trigger either a safe mode request or an interrupt request (see Section 9.3.1.6, Destructive Event Alternate Request Register (RGM\_DEAR)).

### 9.3.1.5 Functional Event Alternate Request Register (RGM\_FEAR)

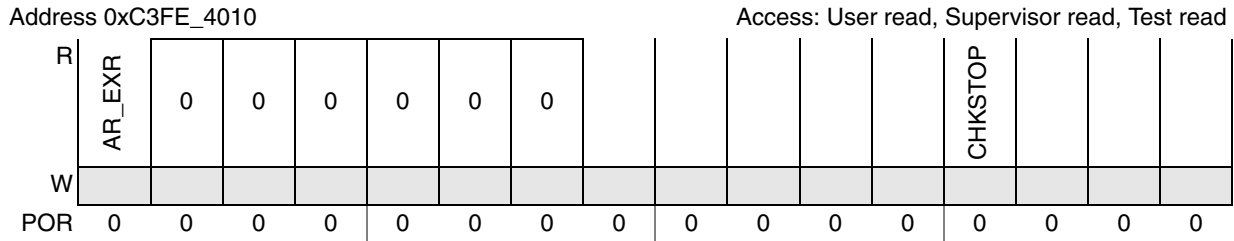


Figure 9-6. Functional Event Alternate Request Register (RGM\_FEAR)

This register defines an alternate request to be generated when a reset on a functional event has been disabled. The alternate request can be either a SAFE mode request to MC\_ME or an interrupt request to the system. It can be accessed in read/write in either supervisor mode or test mode. It can be accessed in read only in user mode.

Table 9-7. Functional Event Alternate Request Register (RGM\_FEAR) field descriptions

| Field  | Description  |
|--------|--|
| AR_EXR | <b>Alternate Request for External Reset</b><br>0 Generate a SAFE mode request on an <b>external reset</b> event if the reset is disabled<br>1 Generate an interrupt request on an <b>external reset</b> event if the reset is disabled |

### 9.3.1.6 Destructive Event Alternate Request Register (RGM\_DEAR)

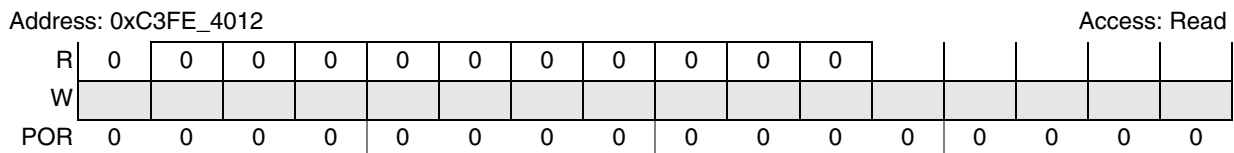


Figure 9-7. Destructive Event Alternate Request Register (RGM\_DEAR)

This register defines an alternate request to be generated when a reset on a destructive event has been disabled. The alternate request can be either a SAFE mode request to MC\_ME or an interrupt request to the system.

### 9.3.1.7 Functional Event Short Sequence Register (RGM\_FESS)

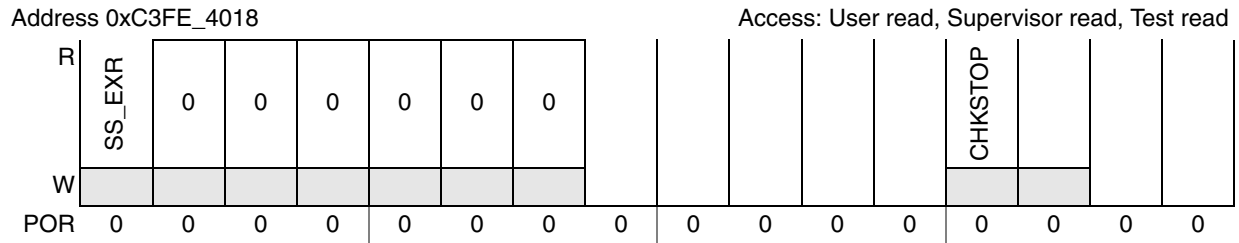


Figure 9-8. Functional Event Short Sequence Register (RGM\_FESS)

This register defines which reset sequence will be done when a functional reset sequence is triggered. The functional reset sequence can either start from PHASE1 or from PHASE3, skipping PHASE1 and PHASE2.

#### NOTE

This could be useful for fast reset sequence, for example to skip flash reset.

It can be accessed in read/write in either supervisor mode or test mode. It can be accessed in read in user mode.

Table 9-9. Functional Event Short Sequence Register (RGM\_FESS) field descriptions

| Field  | Description  |
|--------|--|
| SS_EXR | <b>Short Sequence for External Reset</b><br>0 The reset sequence triggered by an <b>external</b> reset event will start from PHASE1<br>1 The reset sequence triggered by an <b>external</b> reset event will start from PHASE3, skipping PHASE1 and PHASE2 |

#### NOTE

This register is reset on any enabled destructive or functional reset event.

### 9.3.1.8 Functional Bidirectional Reset Enable Register (RGM\_FBRE)

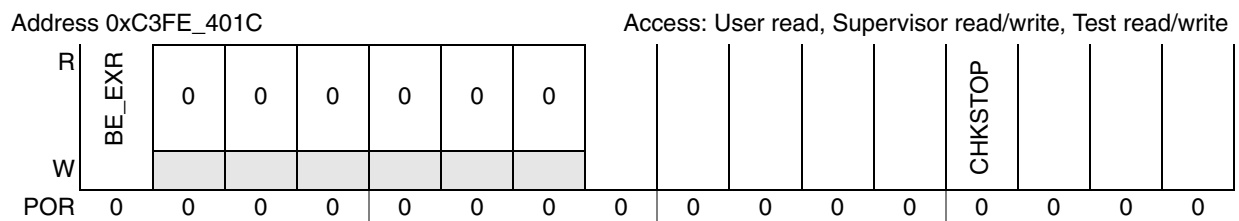


Figure 9-9. Functional Bidirectional Reset Enable Register (RGM\_FBRE)

This register enables the generation of an external reset on functional reset. It can be accessed in read/write in either supervisor mode or test mode. It can be accessed in read in user mode.

Table 9-10. Functional Bidirectional Reset Enable Register (RGM\_FBRE) field descriptions

| Field  | Description  |
|--------|--|
| BE_EXR | <b>Bidirectional Reset Enable for External Reset</b><br>0 RESET is asserted on an <b>external</b> reset event if the reset is enabled<br>1 RESET is not asserted on an <b>external</b> reset event |

## 9.4 Functional description

### 9.4.1 Reset state machine

The main role of MC\_RGM is the generation of the reset sequence, which ensures that the correct parts of the device are reset based on the reset source event. This is summarized in [Table 9-11](#)

Table 9-11. MC\_RGM Reset Implications

| Source                                   | What Gets Reset                                  | External Reset Assertion  | Boot Mode Capture         |
|--|--|---------------------------|---------------------------|
| Power-on Reset                           | All  | Yes                       | Yes                       |
| Destructive reset                        | All except some clock/reset management           | Yes                       | Yes                       |
| External reset                           | All except some clock/reset management and debug | Yes                       | Yes                       |
| Functional resets                        | All except some clock/reset management and debug | Programmable <sup>1</sup> | Programmable <sup>2</sup> |
| Shortened functional resets <sup>3</sup> | Flip-flops except some clock/reset management    | Programmable <sup>1</sup> | Programmable <sup>2</sup> |

<sup>1</sup> the assertion of the external reset is controlled via the RGM\_FBRE register

<sup>2</sup> the boot mode is captured if the external reset is asserted

<sup>3</sup> the short sequence is enabled via the RGM\_FESS register

#### NOTE

JTAG logic has its own independent reset control and is not controlled by the MC\_RGM in any way.

The reset sequence is comprised of five phases managed by a state machine, which ensures that all phases are correctly processed through waiting for a minimum duration and until all processes that need to occur during that phase have been completed before proceeding to the next phase.

The state machine used to produce the reset sequence is shown in [Figure 9-10](#).

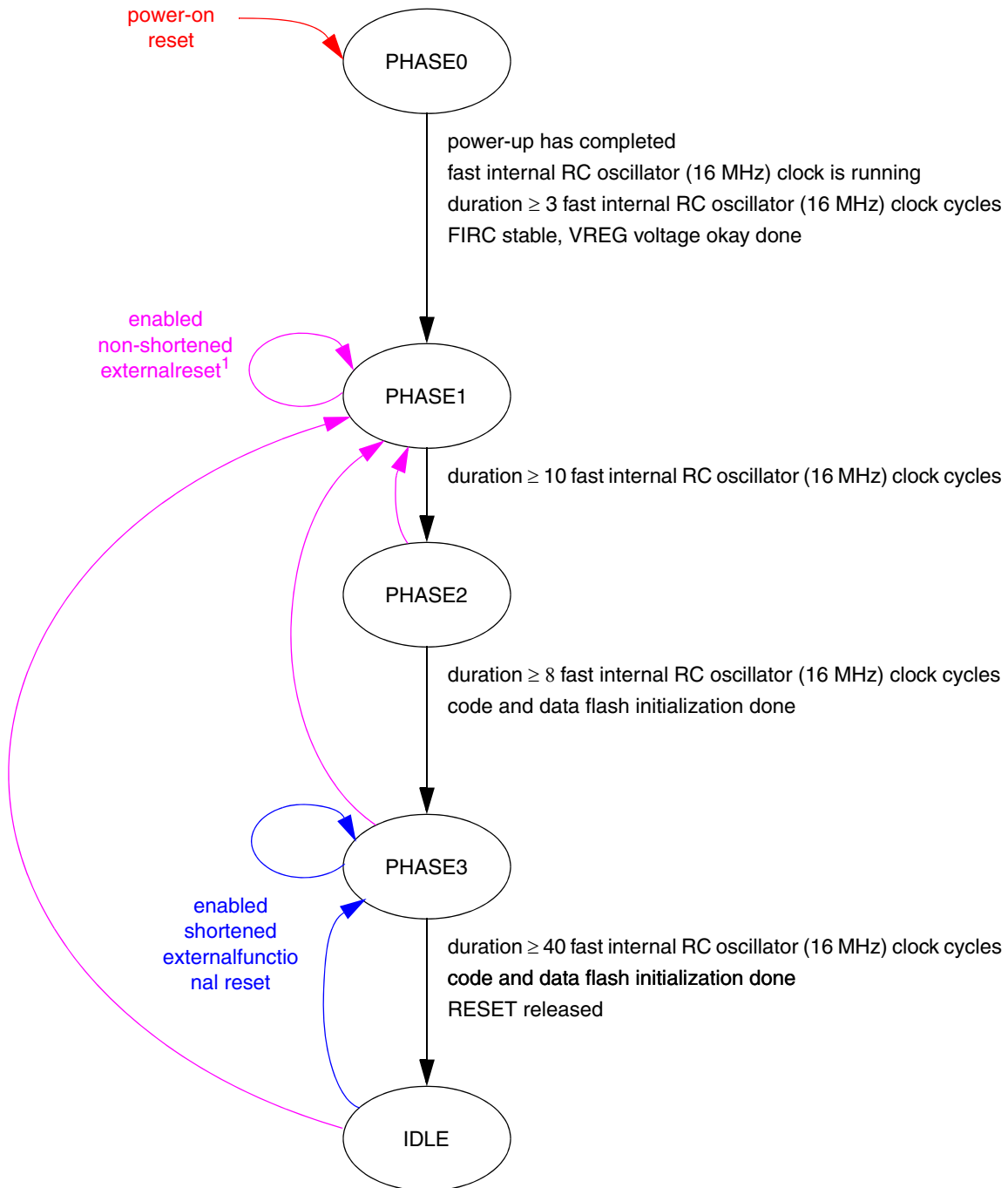


Figure 9-10. MC\_RGM State Machine

Figure 9-11 describes how the device behaves during the startup.



**Figure 9-11. Reset configuration timing**

#### **9.4.1.1 PHASE0 phase**

This phase is entered immediately from any phase on a power-on or enabled destructive reset event. The reset state machine exits PHASE0 and enters PHASE1 on verification of the following:

- Power-up has completed
- Fast internal RC oscillator (16 MHz) clock is running
- All enabled destructive resets have been processed
- All processes that need to be done in PHASE0 are completed
- A minimum of 3 fast internal RC oscillator (16 MHz) clock cycles have elapsed since power-up completion and the last enabled destructive reset event

#### **9.4.1.2 PHASE1 phase**

This phase is entered either on exit from PHASE0 or immediately from PHASE2, PHASE3, or IDLE on a non-masked external or functional reset event if it has not been configured to trigger a short sequence. The reset state machine exits PHASE1 and enters PHASE2 on verification of the following:

- All enabled, non-shortened functional resets have been processed
- A minimum of 10 fast internal RC oscillator (16 MHz) clock cycles have elapsed since the last enabled external or non-shortened functional reset event

### 9.4.1.3 PHASE2 phase

This phase is entered on exit from PHASE1. The reset state machine exits PHASE2 and enters PHASE3 on verification of the following:

- All processes that need to be done in PHASE2 are completed
- A minimum of 8 fast internal RC oscillator (16 MHz) clock cycles have elapsed since entering PHASE2

### 9.4.1.4 PHASE3 phase

This phase is entered either on exit from PHASE2 or immediately from IDLE on an enabled, shortened functional reset event. The reset state machine exits PHASE3 and enters IDLE on verification of the following:

- All processes that need to be done in PHASE3 are completed
- A minimum of 40 fast internal RC oscillator (16 MHz) clock cycles have elapsed since the last enabled, shortened functional reset event

### 9.4.1.5 IDLE phase

This is the final phase, and is entered on exit from PHASE3. When this phase is reached, the MC\_RGM releases control of the system to the platform and waits for new reset events that can trigger a reset sequence.

## 9.4.2 Destructive resets

A destructive reset indicates that an event has occurred, after which critical register or memory content can no longer be guaranteed.

The status flag associated with a given destructive reset event (RGM\_DES.F\_<destructive reset> bit) is set when the destructive reset is asserted and the power-on reset is not asserted. It is possible for multiple status bits to be set simultaneously, and it is software's responsibility to determine which reset source is the most critical for the application.

The destructive reset can be optionally disabled by writing bit RGM\_DERD.D\_<destructive reset>.

### NOTE

The RGM\_DERD register can be written only once between two power-on reset events.

The device's low-voltage detector threshold ensures that, when 1.2 V low-voltage detected (power domain #0) is enabled, the supply is sufficient to have the destructive event correctly propagated through the digital logic. Therefore, if a given destructive reset is enabled, the MC\_RGM ensures that the associated reset event will be correctly triggered to the full system. However, if the given destructive reset is disabled and the voltage goes below the digital functional threshold, functionality can no longer be ensured, and the reset may or may not be asserted.

An enabled destructive reset will trigger a reset sequence starting from the beginning of PHASE0.

### 9.4.3 External reset

The MC\_RGM manages the external reset coming from RESET. The detection of a falling edge on RESET will start the reset sequence from the beginning of PHASE1.

The status flag associated with the external reset falling edge event (RGM\_FES.F\_EXR bit) is set when the external reset is asserted and the power-on reset is not asserted.

The external reset can optionally be disabled by writing bit RGM\_FERD.D\_EXR.

#### NOTE

The RGM\_FERD register can be written only once between two power-on reset events.

An enabled external reset will normally trigger a reset sequence starting from the beginning of PHASE1. Nevertheless, the RGM\_FESS register enables the further configuring of the reset sequence triggered by the external reset. When RGM\_FESS.SS\_EXR is set, the external reset will trigger a reset sequence starting directly from the beginning of PHASE3, skipping PHASE1 and PHASE2. This can be useful especially when an external reset should not reset the flash.

### 9.4.4 Functional resets

A functional reset indicates that an event has occurred, after which it can be guaranteed that critical register and memory content is still intact.

The status flag associated with a given functional reset event (RGM\_FES.F\_<functional reset> bit) is set when the functional reset is asserted and the power-on reset is not asserted. It is possible for multiple status bits to be set simultaneously, and it is software's responsibility to determine which reset source is the most critical for the application.

The functional reset can be optionally disabled by software writing bit RGM\_FERD.D\_<functional reset>.

#### NOTE

The RGM\_FERD register can be written only once between two power-on reset events.

An enabled functional reset will normally trigger a reset sequence starting from the beginning of PHASE1. Nevertheless, the RGM\_FESS register enables the further configuring of the reset sequence triggered by a functional reset. When RGM\_FESS.SS\_<functional reset> is set, the associated functional reset will trigger a reset sequence starting directly from the beginning of PHASE3, skipping PHASE1 and PHASE2. This can be useful especially in case a functional reset should not reset the flash module.

See [Chapter 8, Mode Entry Module \(MC\\_ME\)](#), for details on the STANDBY and DRUN modes.

### 9.4.5 Alternate event generation

The MC\_RGM provides alternative events to be generated on reset source assertion. When a reset source is asserted, the MC\_RGM normally enters the reset sequence. Alternatively, it is possible for each reset



source event (except the power-on reset event) to be converted from a reset to either a SAFE mode request issued to the MC\_ME or to an interrupt request issued to the core.

Alternate event selection for a given reset source is made via the RGM\_F/DERD and RGM\_F/DEAR registers, as shown in [Table 9-12](#).

**Table 9-12. MC\_RGM alternate event selection**

| RGM_F/DERD bit value | RGM_F/DEAR bit value | Generated event   |
|----------------------|----------------------|-------------------|
| 0                    | X                    | Reset             |
| 1                    | 0                    | SAFE mode request |
| 1                    | 1                    | Interrupt request |

The alternate event is cleared by deasserting the source of the request (that is, at the reset source that caused the alternate request) and also clearing the appropriate RGM\_F/DES status bit.

#### NOTE

Alternate requests (SAFE mode as well as interrupt requests) are generated asynchronously.

#### NOTE

If a masked destructive reset event that is configured to generate a SAFE mode/interrupt request occurs during PHASE0, it is ignored, and the MC\_RGM will not send any safe mode/interrupt request to the MC\_ME. The same is true for masked functional reset events during PHASE1.

### 9.4.6 Boot mode capturing

The MC\_RGM samples PA[9:8] whenever RESET is asserted until five FIRC (16 MHz internal RC oscillator) clock cycles before its deassertion edge. The result of the sampling is used at the beginning of reset PHASE3 for boot mode selection and is retained after RESET has been deasserted for subsequent boots after reset sequences during which RESET is not asserted.

#### NOTE

In order to ensure that the boot mode is correctly captured, the application needs to apply the valid boot mode value the entire time that RESET is asserted.

RESET can be asserted as a consequence of the internal reset generation. This will force re-sampling of the boot mode pins. (See [Table 9-11](#) for details.)

This page is intentionally left blank.

# Chapter 10

## Power Control Unit (MC\_PCU)

### 10.1 Introduction

#### 10.1.1 Overview

The power control unit (MC\_PCU) is used to reduce the overall SoC power consumption. Power can be saved by disconnecting parts of the SoC from the power supply via a power switching device. The SoC is grouped into multiple parts having this capability, which are called power domains.

When a power domain is disconnected from the supply, the power consumption is reduced to zero in that domain. Any status information of such a power domain is lost. When reconnecting a power domain to the supply voltage, the domain draws an increased current until the power domain reaches its operational voltage.

Power domains are controlled on a device mode basis. For each mode, software can configure whether a power domain is connected to the supply voltage (power-up state) or disconnected (power-down state). Maximum power saving is reached by entering the STANDBY mode.

On each mode change request, the MC\_PCU evaluates the power domain settings in the power domain configuration registers and initiates a power-down or a power-up sequence for each individual power domain. The power-up/down sequences are handled by finite state machines to ensure a smooth and safe transition from one power state to the other.

Exiting the STANDBY mode can only be done via a system wakeup event as all power domains other than power domain #0 are in the power-down state.

In addition, the MC\_PCU acts as a bridge for mapping the VREG peripheral to the MC\_PCU address space.

Figure 10-1 shows the MC\_PCU block diagram.

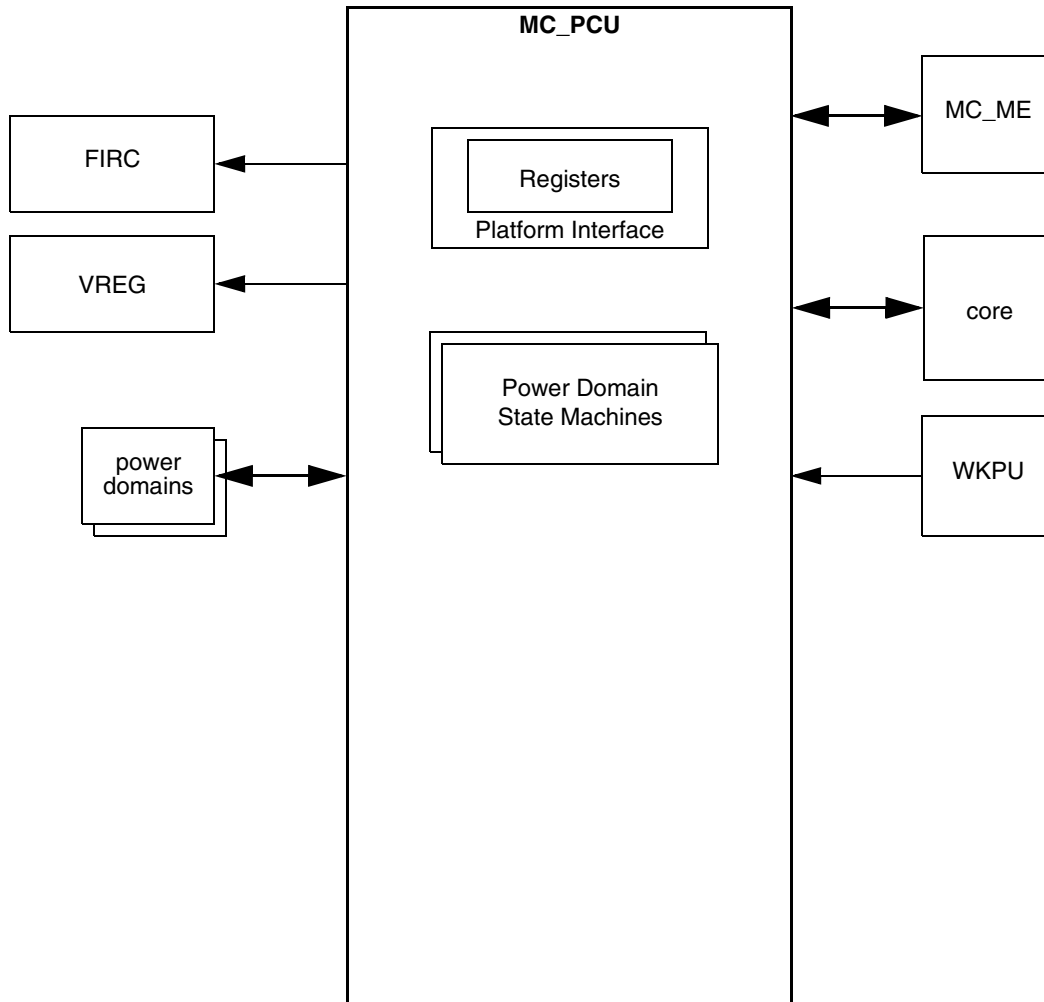


Figure 10-1. MC\_PCU Block Diagram

### 10.1.2 Features

The MC\_PCU includes the following features:

- Support for 1012345789 power domains
- Support for device modes RESET, DRUN, SAFE, TEST, RUN0...3, HALT, STOP, and STANDBY (for further mode details, please see)
- Power states updating on each mode change and on system wakeup
- A handshake mechanism for power state changes thus guaranteeing operable voltage

### 10.1.3 Modes of operation

The MC\_PCU is available in all device modes.

## 10.2 External signal description

The MC\_PCU has no connections to any external pins.

## 10.3 Memory map and register definition

Table 10-1. MC\_PCU register description

| Address     | Name        | Description                    | Size | Access |            |            | Location   |
|-------------|-------------|--------------------------------|------|--------|------------|------------|--|
|             |             |                                |      | Normal | Supervisor | Test       |  |
| 0xC3FE_8000 | PCU_PCONF0  | Power Domain #0 Configuration  | word | read   | read       | read       | <a href="#">on page 199</a>                                |
| 0xC3FE_8004 | PCU_PCONF1  | Power Domain #1 Configuration  | word | read   | read       | read       | <a href="#">on page 200</a>                                |
| 0xC3FE_8008 | PCU_PCONF2  | Power Domain #2 Configuration  | word | read   | read/write | read/write | <a href="#">on page 201</a><br><a href="#">on page 201</a> |
| 0xC3FE_800C | PCU_PCONF3  | Power Domain #3 Configuration  | word | read   | read/write | read/write | <a href="#">on page 201</a>                                |
| 0xC3FE_8010 | PCU_PCONF4  | Power Domain #4 Configuration  | word | read   | read/write | read/write | <a href="#">on page 201</a>                                |
| 0xC3FE_8014 | PCU_PCONF5  | Power Domain #5 Configuration  | word | read   | read/write | read/write | <a href="#">on page 201</a>                                |
| 0xC3FE_8018 | PCU_PCONF6  | Power Domain #6 Configuration  | word | read   | read/write | read/write | <a href="#">on page 201</a>                                |
| 0xC3FE_801C | PCU_PCONF7  | Power Domain #7 Configuration  | word | read   | read/write | read/write | <a href="#">on page 201</a>                                |
| 0xC3FE_8020 | PCU_PCONF8  | Power Domain #8 Configuration  | word | read   | read/write | read/write | <a href="#">on page 201</a>                                |
| 0xC3FE_8024 | PCU_PCONF9  | Power Domain #9 Configuration  | word | read   | read/write | read/write | <a href="#">on page 201</a>                                |
| 0xC3FE_8028 | PCU_PCONF10 | Power Domain #10 Configuration | word | read   | read/write | read/write | <a href="#">on page 201</a>                                |
| 0xC3FE_802C | PCU_PCONF11 | Power Domain #11 Configuration | word | read   | read/write | read/write | <a href="#">on page 201</a>                                |
| 0xC3FE_8030 | PCU_PCONF12 | Power Domain #12 Configuration | word | read   | read/write | read/write | <a href="#">on page 201</a>                                |
| 0xC3FE_8040 | PCU_PSTAT   | Power Domain Status Register   | word | read   | read       | read       | <a href="#">on page 202</a>                                |

### NOTE

Any access to unused registers as well as write accesses to read-only registers will:

- Not change register content

- Cause a transfer error

**Table 10-2. MC\_PCU memory map**

|   |                          |   |   |   |       |   |   |      |   |      |      |      |      |      |      |      |      |     |
|---|--------------------------|---|---|---|-------|---|---|------|---|------|------|------|------|------|------|------|------|-----|
| 0xC3FE_8000                                 | PCU_PCONF0               | R | 0 | 0 | 0     | 0 | 0 | 0    | 0 | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    |     |
|   |                          | W |   |   |       |   |   |      |   |      |      |      |      |      |      |      |      |     |
|   |                          | R | 0 | 0 | STBY0 | 0 | 0 | STOP | 0 | HALT | RUN3 | RUN2 | RUN1 | RUN0 | DRUN | SAFE | TEST | RST |
|   |                          | W |   |   |       |   |   |      |   |      |      |      |      |      |      |      |      |     |
| 0xC3FE_8004                                 | PCU_PCONF1               | R | 0 | 0 | 0     | 0 | 0 | 0    | 0 | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    |     |
|   |                          | W |   |   |       |   |   |      |   |      |      |      |      |      |      |      |      |     |
|   |                          | R | 0 | 0 | STBY0 | 0 | 0 | STOP | 0 | HALT | RUN3 | RUN2 | RUN1 | RUN0 | DRUN | SAFE | TEST | RST |
|   |                          | W |   |   |       |   |   |      |   |      |      |      |      |      |      |      |      |     |
| 0xC3FE_8008                                 | PCU_PCONF2101<br>2456789 | R | 0 | 0 | 0     | 0 | 0 | 0    | 0 | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    |     |
|   |                          | W |   |   |       |   |   |      |   |      |      |      |      |      |      |      |      |     |
|   |                          | R | 0 | 0 | STBY0 | 0 | 0 | STOP | 0 | HALT | RUN3 | RUN2 | RUN1 | RUN0 | DRUN | SAFE | TEST | RST |
|   |                          | W |   |   |       |   |   |      |   |      |      |      |      |      |      |      |      |     |
| 0xC3FE_800C                                 | PCU_PCONF1012<br>3456789 | R | 0 | 0 | 0     | 0 | 0 | 0    | 0 | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    |     |
|   |                          | W |   |   |       |   |   |      |   |      |      |      |      |      |      |      |      |     |
|   |                          | R | 0 | 0 | STBY0 | 0 | 0 | STOP | 0 | HALT | RUN3 | RUN2 | RUN1 | RUN0 | DRUN | SAFE | TEST | RST |
|   |                          | W |   |   |       |   |   |      |   |      |      |      |      |      |      |      |      |     |
| 0xC3FE_800123<br>048C<br>...<br>0xC3FE_803C | reserved                 |   |   |   |       |   |   |      |   |      |      |      |      |      |      |      |      |     |
| 0xC3FE_8040                                 | PCU_PSTAT                | R | 0 | 0 | 0     | 0 | 0 | 0    | 0 | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    |     |
|   |                          | W |   |   |       |   |   |      |   |      |      |      |      |      |      |      |      |     |
|   |                          | R | 0 | 0 | 0     | 0 | 0 | 0    | 0 | 0    | 0    | 0    | 0    | 0    | PD3  | PD2  | PD1  | PD0 |
|   |                          | W |   |   |       |   |   |      |   |      |      |      |      |      |      |      |      |     |
| 0x044<br>...<br>0x07C                       | reserved                 |   |   |   |       |   |   |      |   |      |      |      |      |      |      |      |      |     |

Table 10-2. MC\_PCU memory map (continued)

|                                   |          |
|-----------------------------------|----------|
| 0xC3FE_8080<br>...<br>0xC3FE_80FC |          |
| 0xC3FE_8100<br>...<br>0xC3FE_BFFC | reserved |

### 10.3.1 Register descriptions

All registers may be accessed as 32-bit words, 16-bit half-words, or 8-bit bytes. The bytes are ordered according to big endian. For example, the PD0 field of the PCU\_PSTAT register may be accessed as a word at address 0xC3FE\_8040, as a half-word at address 0xC3FE\_8042, or as a byte at address 0xC3FE\_8043.

#### 10.3.1.1 Power Domain #0 Configuration Register (PCU\_PCONF0)

|                     |   |   |       |   |   |      |   |      |      |      |      |      |      |      |      |     |
|---------------------|---|---|-------|---|---|------|---|------|------|------|------|------|------|------|------|-----|
| Address 0xC3FE_8000 |   | Access: User read, Supervisor read, Test read |       |   |   |      |   |      |      |      |      |      |      |      |      |     |
| R                   | 0 | 0   | 0     | 0 | 0 | 0    | 0 | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0   |
| W                   |   |   |       |   |   |      |   |      |      |      |      |      |      |      |      |     |
| Reset               | 0 | 0   | 0     | 0 | 0 | 0    | 0 | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0   |
| R                   | 0 | 0   | STBY0 | 0 | 0 | STOP | 0 | HALT | RUN3 | RUN2 | RUN1 | RUN0 | DRUN | SAFE | TEST | RST |
| W                   |   |   |       |   |   |      |   |      |      |      |      |      |      |      |      |     |
| Reset               | 0 | 0   | 1     | 0 | 0 | 1    | 0 | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1   |

Figure 10-2. Power Domain #0 Configuration Register (PCU\_PCONF0)

This register defines for power domain #0 whether it is on or off in each device mode. As power domain #0 is the always-on power domain (and includes the MC\_PCU), none of its bits are programmable. This register is available for completeness reasons.

Table 10-3. Power Domain Configuration Register field descriptions

| Field | Description   |
|-------|---|
| RST   | Power domain control during RESET mode<br>0 Power domain off<br>1 Power domain on |
| TEST  | Power domain control during TEST mode<br>0 Power domain off<br>1 Power domain on  |

**Table 10-3. Power Domain Configuration Register field descriptions (continued)**

| Field | Description   |
|-------|---|
| SAFE  | Power domain control during SAFE mode<br>0 Power domain off<br>1 Power domain on    |
| DRUN  | Power domain control during DRUN mode<br>0 Power domain off<br>1 Power domain on    |
| RUN0  | Power domain control during RUN0 mode<br>0 Power domain off<br>1 Power domain on    |
| RUN1  | Power domain control during RUN1 mode<br>0 Power domain off<br>1 Power domain on    |
| RUN2  | Power domain control during RUN2 mode<br>0 Power domain off<br>1 Power domain on    |
| RUN3  | Power domain control during RUN3 mode<br>0 Power domain off<br>1 Power domain on    |
| HALT  | Power domain control during HALT mode<br>0 Power domain off<br>1 Power domain on    |
| STOP  | Power domain control during STOP mode<br>0 Power domain off<br>1 Power domain on    |
| STBY0 | Power domain control during STANDBY mode<br>0 Power domain off<br>1 Power domain on |

### 10.3.1.2 Power Domain #1 Configuration Register (PCU\_PCONF1)

Address 0xC3FE\_8004 Access: User read, Supervisor read, Test read

|       |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|-------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| W     |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

|       |   |   |       |   |   |      |   |      |      |      |      |      |      |      |      |     |
|-------|---|---|-------|---|---|------|---|------|------|------|------|------|------|------|------|-----|
| R     | 0 | 0 | STBY0 | 0 | 0 | STOP | 0 | HALT | RUN3 | RUN2 | RUN1 | RUN0 | DRUN | SAFE | TEST | RST |
| W     |   |   |       |   |   |      |   |      |      |      |      |      |      |      |      |     |
| Reset | 0 | 0 | 0     | 0 | 0 | 1    | 0 | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1   |

**Figure 10-3. Power Domain #1 Configuration Register (PCU\_PCONF1)**

This register defines for power domain #1 whether it is on or off in each device mode. The bit field description is the same as in [Table 10-3](#). As the platform, clock generation, and mode control reside in



power domain #1, this power domain is only powered down during the STANDBY mode. Therefore, none of the bits is programmable. This register is available for completeness reasons.

The difference between PCU\_PCONF0 and PCU\_PCONF1 is the reset value of the STBY0 bit: During the STANDBY mode, power domain #1 is disconnected from the power supply, and therefore PCU\_PCONF1.STBY0 is always 0. Power domain #0 is always on, and therefore PCU\_PCONF0.STBY0 is 1.

For further details about STANDBY mode, please see [Section 10.4.4.2, STANDBY mode transition](#).

### 10.3.1.3 Power Domain #2 Configuration Register (PCU\_PCONF2)

| Address 0xC3FE_8008 |   | Access: User read/write, Supervisor read/write, Test read/write |       |   |   |       |   |       |      |      |      |      |      |      |      |     |   |
|---------------------|---|---|-------|---|---|-------|---|-------|------|------|------|------|------|------|------|-----|---|
| R                   | 0 | 0   | 0     | 0 | 0 | 0     | 0 | 0     | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0   | 0 |
| W                   |   |   |       |   |   |       |   |       |      |      |      |      |      |      |      |     |   |
| Reset               | 0 | 0   | 0     | 0 | 0 | 0     | 0 | 0     | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0   | 0 |
| R                   | 0 | 0   | STBY0 | 0 | 0 | STOP0 | 0 | HALT0 | RUN3 | RUN2 | RUN1 | RUN0 | DRUN | SAFE | TEST | RST |   |
| W                   |   |   |       |   |   |       |   |       |      |      |      |      |      |      |      |     |   |
| Reset               | 0 | 0   | 0     | 0 | 0 | 1     | 0 | 1     | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1   |   |

Figure 10-4. Power Domain #2 Configuration Register (PCU\_PCONF2)

This register defines for power domain #2 whether it is on or off in each device mode. The bit field description is the same as in [Table 10-3](#).

### 10.3.1.4 Power Domain #2...10123456789 Configuration Registers (PCU\_PCONF2...10123456789)

| Address 0xC3FE_8008 |   | Access: User read/write, Supervisor read/write, Test read/write |       |   |   |      |   |      |      |      |      |      |      |      |      |     |   |
|---------------------|---|---|-------|---|---|------|---|------|------|------|------|------|------|------|------|-----|---|
| R                   | 0 | 0   | 0     | 0 | 0 | 0    | 0 | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0   | 0 |
| W                   |   |   |       |   |   |      |   |      |      |      |      |      |      |      |      |     |   |
| Reset               | 0 | 0   | 0     | 0 | 0 | 0    | 0 | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0   | 0 |
| R                   | 0 | 0   | STBY0 | 0 | 0 | STOP | 0 | HALT | RUN3 | RUN2 | RUN1 | RUN0 | DRUN | SAFE | TEST | RST |   |
| W                   |   |   |       |   |   |      |   |      |      |      |      |      |      |      |      |     |   |
| Reset               | 0 | 0   | 0     | 0 | 0 | 1    | 0 | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1   |   |

Figure 10-5. Power Domain #2 Configuration Register (PCU\_PCONF2)

Address 0xC3FE\_800C Access: User read/write, Supervisor read/write, Test read/write

|       |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|-------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| W     |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

|       |   |   |       |   |   |      |   |      |      |      |      |      |      |      |      |     |
|-------|---|---|-------|---|---|------|---|------|------|------|------|------|------|------|------|-----|
| R     | 0 | 0 | STBY0 | 0 | 0 | STOP | 0 | HALT | RUN3 | RUN2 | RUN1 | RUN0 | DRUN | SAFE | TEST | RST |
| W     |   |   |       |   |   |      |   |      |      |      |      |      |      |      |      |     |
| Reset | 0 | 0 | 0     | 0 | 0 | 1    | 0 | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1    | 1   |

Figure 10-6. Power Domain #3 Configuration Register (PCU\_PCONF3)

These registers define for each power domain #2 through #10123456789 whether it is on or off in each device mode. The bit field description is the same as in Table 10-3.

### 10.3.1.5 Power Domain Status Register (PCU\_PSTAT)

Address 0xC3FE\_8040 Access: User read, Supervisor read, Test read

|       |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
|-------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| W     |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

|       |   |   |   |    |    |    |    |    |    |    |    |     |     |     |     |
|-------|---|---|---|----|----|----|----|----|----|----|----|-----|-----|-----|-----|
| R     | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | PD3 | PD2 | PD1 | PD0 |
| W     |   |   |   |    |    |    |    |    |    |    |    |     |     |     |     |
| Reset | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10  | 10  | 1   | 1   |

Figure 10-7. Power Domain Status Register (PCU\_PSTAT)

This register reflects the power status of all available power domains.

Table 10-4. Power Domain Status Register (PCU\_PSTAT) field descriptions

| Field | Description  |
|-------|--|
| PDn   | Power status for power domain #n<br>0 Power domain is inoperable<br>1 Power domain is operable |

## 10.4 Functional description

### 10.4.1 General

The MC\_PCU controls all available power domains on a device mode basis. The PCU\_PCONFn registers specify during which system/user modes a power domain is powered up. The power state for each individual power domain is reflected by the bits in the PCU\_PSTAT register.

On a mode change, the MC\_PCU evaluates which power domain(s) must change power state. The power state is controlled by a state machine (FSM) for each individual power domain (see [Figure 10-8](#)), which ensures a clean and safe state transition.

## 10.4.2 Reset / Power-On Reset

After any reset, the SoC will transition to the RESET mode during which all power domains are powered up (see [Chapter 8, Mode Entry Module \(MC\\_ME\)](#)). Once the reset sequence has been completed, the DRUN mode is entered and software can begin the MC\_PCU configuration.

## 10.4.3 MC\_PCU configuration

Per default, all power domains are powered in all modes other than STANDBY. Software can change the configuration for each power domain on a mode basis by programming the PCU\_PCONF<sub>n</sub> registers.

Each power domain that is powered down is held in a reset state. Read/write accesses to peripherals in those power domains will result in a transfer error.

## 10.4.4 Mode transitions

On a mode change requested by the MC\_ME, the MC\_PCU evaluates the power configurations for all power domains. It compares the settings in the PCU\_PCONF<sub>n</sub> registers for the new mode with the settings for the current mode. If the configuration for a power domain differs between the modes, a power state change request is generated. These requests are handled by a finite state machine to ensure a smooth and safe transition from one power state to another.

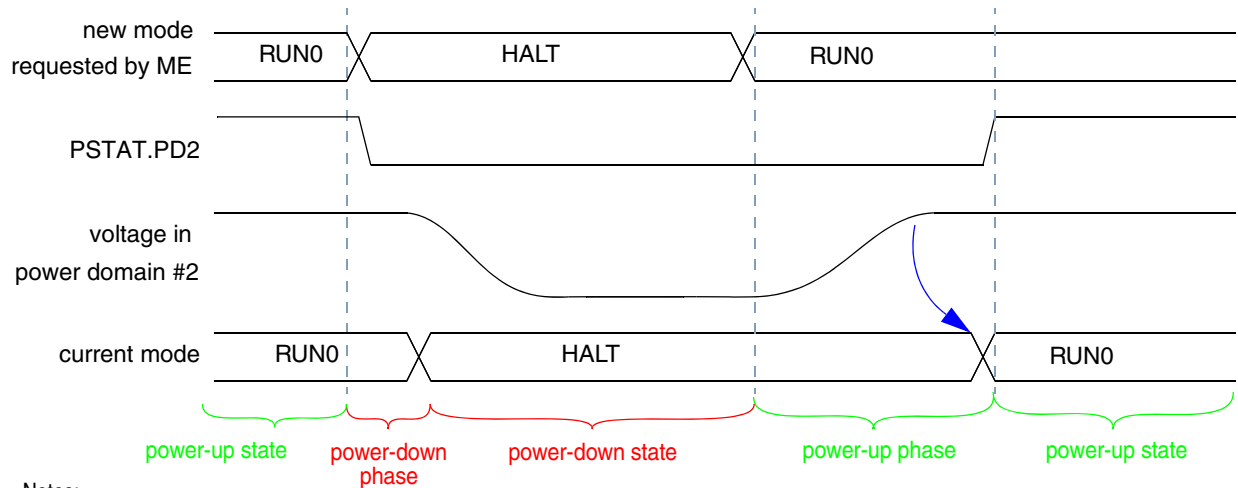
### 10.4.4.1 DRUN, SAFE, TEST, RUN0...3, HALT, and STOP mode transition

The DRUN, SAFE, TEST, RUN0...3, HALT, and STOP modes allow an increased power saving. The level of power saving is software-controllable via the settings in the PCU\_PCONF<sub>n</sub> registers for power domain #2 onwards. The settings for power domains #0 and #1 cannot be changed. Therefore, power domains #0 and #1 remain connected to the power supply for all modes beside STANDBY.

[Figure 10-8](#) shows an example for a mode transition from RUN0 to HALT and back, which will result in power domain #2 being powered down during the HALT mode. In this case, PCU\_PCONF2. HALT is programmed to be 0.

When the MC\_PCU receives the mode change request to HALT mode, it starts its power-down phase. During the power-down phase, clocks are disabled and the reset is asserted, resulting in a loss of all information for this power domain.

Then the power domain is disconnected from the power supply (power-down state).



## Notes:

Not drawn to scale; PCONF2.RUN0 = 1; PCONF2.HALT = 0

**Figure 10-8. MC\_PCU Events During Power Sequences (non-STANDBY mode)**

When the MC\_PCU receives a mode change request to RUN0, it starts its power-up phase if PCU\_PCONF2.RUN0 is 1. The power domain is reconnected to the power supply, and the voltage in power domain #2 will increase slowly. Once the voltage of power domain #2 is within an operable range, its clocks are enabled, and its resets are deasserted (power-up state).

#### NOTE

It is possible that, due to a mode change, power-up is requested before a power domain completed its power-down sequence. In this case, the information in that power domain is lost.

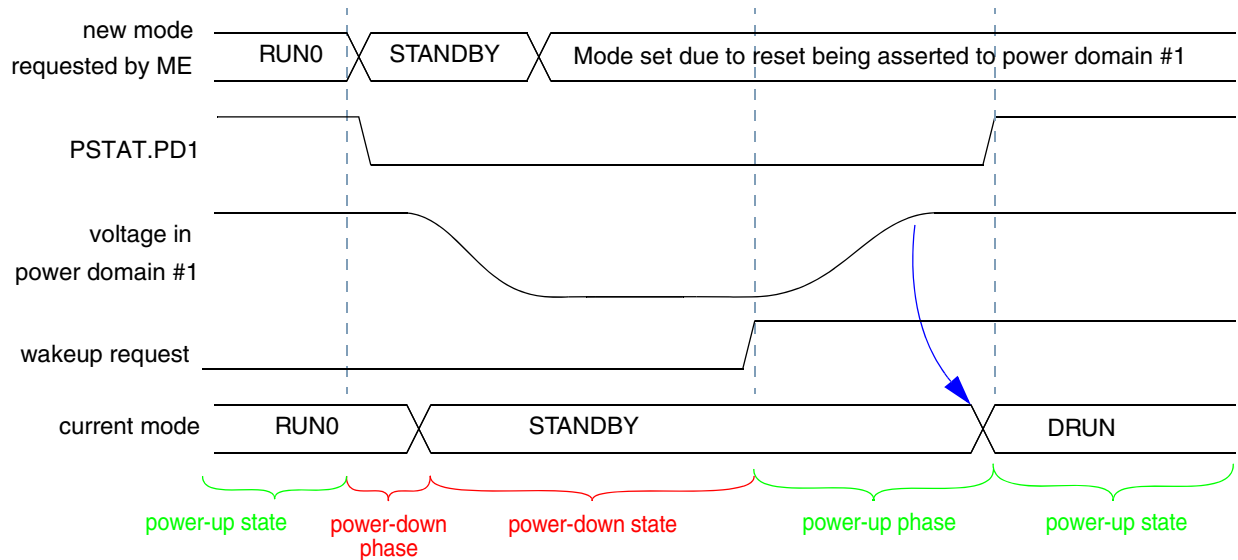
#### 10.4.4.2 STANDBY mode transition

STANDBY offers the maximum power saving. The level of power saving is software-controllable via the settings in the PCU\_PCONF<sub>n</sub> registers for power domain #2 onwards. Power domain #0 stays connected to the power supply while power domain #1 is disconnected from the power supply. Among others, power domain #1 contains the platform and the MC\_ME. Therefore this mode differs from all other user/system modes.

Once STANDBY is entered it can only be left via a system wakeup. On exiting the STANDBY mode, all power domains are powered up according to the settings in the PCU\_PCONF<sub>n</sub> registers, and the DRUN mode is entered. In DRUN mode, at least power domains #0 and #1 are powered.

Figure 10-9 shows an example for a mode transition from RUN0 to STANDBY to DRUN. All power domains that have PCU\_PCONF<sub>n</sub>.STBY0 cleared will enter power-down phase. In this example only power domain #1 will be disabled during STANDBY mode.

When the MC\_PCU receives the mode change request to STANDBY mode it starts the power down phase for power domain #1. During the power down phase, clocks are disabled and reset is asserted resulting in a loss of all information for this power domain. Then the power domain is disconnected from the power supply (power-down state).



## Notes:

Not drawn to scale; PCONF1.RUN0 = 1; PCONF1.STBY0 = 0

**Figure 10-9. MC\_PCU Events During Power Sequences (STANDBY mode)**

When the MC\_PCU receives a system wakeup request, it starts the power-up phase. The power domain is reconnected to the power supply and the voltage in power domain #1 will increase slowly. Once the voltage is in an operable range, clocks are enabled and the reset is deasserted (power-up state).

### NOTE

It is possible that due to a wakeup request, power-up is requested before a power domain completed its power-down sequence. In this case, the information in that power domain is lost.

#### 10.4.4.3 Power saving for memories during STANDBY mode

All memories that are not powered down during STANDBY mode automatically enter a power saving state. No software configuration is required to enable this power saving state. While a memory is residing in this state an increased power saving is achieved. Data in the memories is retained.

## 10.5 Initialization information

To initialize the MC\_PCU, the registers PCU\_PCONF2...3 should be programmed. After programming is done, those registers should no longer be changed.

## 10.6 Application information

### 10.6.1 STANDBY Mode Considerations

STANDBY offers maximum power saving possibility. But power is only saved during the time a power domain is disconnected from the supply. Increased power is required when a power domain is reconnected to the power supply. Additional power is required during restoring the information (for example, in the platform).

Care should be taken that the time during which the SoC is operating in STANDBY mode is significantly longer than the required time for restoring the information.

# Chapter 11

## Voltage Regulators and Power Supplies

### 11.1 Voltage regulators

The power blocks provide a 1.2 V digital supply to the internal logic of the device. The main supply is 3.3 V–5 V  $\pm 10\%$  and digital/regulated output supply is 1.2 V  $\pm 10\%$ . The voltage regulator used in MPC5606BK comprises three regulators.

- High power regulator (HPREG)
- Low power regulator (LPREG)
- Ultra low power regulator (ULPREG)

The HPREG and LPREG regulators are switched off during STANDBY mode to save consumption from the regulator itself. In STANDBY mode, the supply is provided by the ULPREG regulator.

In STOP mode, the user can configure the HPREG regulator to switch off (see [Chapter 8, Mode Entry Module \(MC\\_ME\)](#)). In this case, when current is low enough to be handled by LPREG alone, the HPREG regulator is switched off and the supply is provided by the LPREG regulator.

The internal voltage regulator requires an external capacitance (CREG) to be connected to the device in order to provide a stable low voltage digital supply to the device. Capacitances should be placed on the board as near as possible to the associated pins.

The regulator has two digital domains, one for the high power regulator (HPREG) and the low power regulator (LPREG) called “High Power domain” and another one for the ultra low power regulator (ULPREG) called “Standby domain.” For each domain there is a low voltage detector for the 1.2 V output voltage. Additionally there are two low voltage detectors for the main/input supply with different thresholds, one at the 3.3 V level and the other one at the 5 V level.

#### 11.1.1 High power regulator (HPREG)

The HPREG converts the 3.3 V–5 V input supply to a 1.2 V digital supply. For more information, see the voltage regulator electrical characteristics section of the data sheet.

The regulator can be switched off by software. Refer to the main voltage regulator control bit (MVRON) of the mode configuration registers in [Chapter 8, Mode Entry Module \(MC\\_ME\)](#).

#### 11.1.2 Low power regulator (LPREG)

The LPREG generates power for the device in the STOP mode, providing the output supply of 1.2 V. It always sees the minimum external capacitance. The control part of the regulator can be used to disable the low power regulator. It is managed by MC\_ME.

### 11.1.3 Ultra low power regulator (ULPREG)

The ULPREG generates power for the standby domain as well as a part of the main domain and might or might not see the external capacitance. The control circuit of ULPREG can be used to disable the ultra low power regulator by software: This action is managed by MC\_ME.

### 11.1.4 LVDs and POR

There are three kinds of LVD available:

1. LVD\_MAIN for the 3.3 V–5 V input supply with thresholds at approximately 3 V level<sup>1</sup>
2. LVD\_MAIN5 for the 3.3 V–5 V input supply with threshold at approximately 4.5 V level<sup>1</sup>
3. LVD\_DIG for the 1.2 V output voltage

The LVD\_MAIN and LVD\_MAIN5 sense the 3.3 V–5 V power supply for CORE, shared with IO ring supply and indicate when the 3.3 V–5 V supply is stabilized.

Two LVD\_DIGs are provided in the design. One LVD\_DIG is placed in the high power domain and senses the HPREG/LPREG output notifying that the 1.2 V output is stable. The other LVD\_DIG is placed in the standby domain and senses the standby 1.2 V supply level notifying that the 1.2 V output is stable. The reference voltage used for all LVDs is generated by the low power reference generator and is trimmed for LVD\_DIG, using the bits LP[4:7]. Therefore, during the pre-trimming period, LVD\_DIG exhibits higher thresholds, whereas during post trimming, the thresholds come in the desired range. Power-down pins are provided for LVDs. When LVDs are power-down, their outputs are pulled high.

POR is required to initialize the device during supply rise. POR works only on the rising edge of the main supply. To ensure its functioning during the following rising edge of the supply, it is reset by the output of the LVD\_MAIN block when main supply reaches below the lower voltage threshold of the LVD\_MAIN.

POR is asserted on power-up when V<sub>dd</sub> supply is above V<sub>PORUP</sub> min (refer to data sheet for details). It will be released only after V<sub>dd</sub> supply is above V<sub>PORH</sub> (refer to data sheet for details). V<sub>dd</sub> above V<sub>PORH</sub> ensures power management module including internal LVDs modules are fully functional.

### 11.1.5 VREG digital interface

The voltage regulator digital interface provides the temporization delay at initial power-up and at exit from low-power modes. A signal, indicating that Ultra Low Power domain is powered, is used at power-up to release reset to temporization counter. At exit from low-power modes, the power-down for high power regulator request signal is monitored by the digital interface and used to release reset to the temporization counter. In both cases, on completion of the delay counter, an end-of-count signal is released, it is gated with an other signal indicating main domain voltage fine in order to release the VREGOK signal. This is used by MC\_RGM to release the reset to the device. It manages other specific requirements, like the transition between high power/low power mode to ultra low power mode avoiding a voltage drop below the permissible threshold limit of 1.08 V.

The VREG digital interface also holds control register to mask 5 V LVD status coming from the voltage regulator at the power-up.

1. See section “Voltage monitor electrical characteristics” of the data sheet for detailed information about this voltage value.



## 11.1.6 Register description

The VREG\_CTL register is mapped to the MC\_PCU address space as described in [Chapter 10, Power Control Unit \(MC\\_PCU\)](#).

| Address: 0xC3FE_8080 |   |   |   |   |   |   |   |   |   |   |    | Access: User read/write |    |    |    |    |
|----------------------|---|---|---|---|---|---|---|---|---|---|----|-------------------------|----|----|----|----|
|                      | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11                      | 12 | 13 | 14 | 15 |
| R                    | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0                       | 0  | 0  | 0  | 0  |
| W                    |   |   |   |   |   |   |   |   |   |   |    |                         |    |    |    |    |
| Reset                | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0                       | 0  | 0  | 0  | 0  |

|       | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31          |
|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------------|
| R     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 5V_LVD_MASK |
| W     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |             |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 1           |

Figure 11-1. Voltage Regulator Control Register (VREG\_CTL)

Table 11-1. VREG\_CTL field descriptions

| Field       | Description   |
|-------------|---|
| 5V_LVD_MASK | Mask bit for 5 V LVD from regulator<br>This is a read/write bit and must be unmasked by writing a 1 by software to generate LVD functional reset request to MC_RGM for 5 V trip.<br>1: 5 V LVD is masked<br>0: 5 V LVD is not masked. |

## 11.2 Power supply strategy

From a power-routing perspective, the device is organized as follows.

The device provides four dedicated supply domains at package level:

1. HV (high voltage external power supply for I/Os and most analog module) — This must be provided externally through VDD\_HV/VSS\_HV power pins. Voltage values should be aligned with  $V_{DD}/V_{SS}$ . Refer to data sheet for details.
2. ADC (high voltage external power supply for ADC module) — This must be provided externally through VDD\_HV\_ADC/VSS\_HV\_ADC power pins. Voltage values should be aligned with  $V_{DD\_HV\_ADC}/V_{SS\_HV\_ADC}$ . Refer to data sheet for details.
3. BV (high voltage external power supply for voltage regulator module) — This must be provided externally through VDD\_BV\_/VSS\_BV power pins. Voltage values should be aligned with  $V_{DD}/V_{SS}$ . Refer to data sheet for details.

4. LV (low voltage internal power supply for core, FMPLL and Flash digital logic) — This is generated internally by embedded voltage regulator and provided to the core, FMPLL and Flash. Three VDD\_LV/VSS\_LV pins pairs are provided to connect the three decoupling capacitances. This is generated internally by internal voltage regulator but provided outside to connect stability capacitor. Refer to data sheet for details.

The four dedicated supply domains are further divided within the package in order to reduce as much as possible EMC and noise issues.

- HV\_IO: High voltage pad supply
- HV\_FLAn: High voltage flash memory supply
- HV\_OSCOREG<sup>1</sup>: High voltage external oscillator and regulator supply
- HV\_ADR: High voltage reference for ADC module. Supplies are further star routed to reduce impact of ADC resistive reference on ADC capacitive reference accuracy.
- HV\_ADV: High voltage supply for ADC module
- BV: High voltage supply for voltage regulator ballast. These two ballast pads are used to supply the core and the flash memory. Each pad contains two ballasts to supply 80 mA and 20 mA, respectively. Core is hence supplied through two ballasts of 80 mA capability and CFlash and DFlash through two 20 mA ballasts. The HV supply for both ballasts is shorted through double bonding.
- LV\_COR: Low voltage supply for the core. It is also used to provide supply for FMPLL through double bonding.
- LV\_FLAn: Low voltage supply for flash memory module *n*. It is supplied with dedicated ballast and shorted to LV\_COR through double bonding.
- LV\_PLL<sup>2</sup>: Low voltage supply for FMPLL

### 11.3 Power domain organization

Based on stringent requirements for current consumption in different operational modes, the device is partitioned into different power domains. Organization into these power domains primarily means separate power supplies that are separated from each other by use of power switches (switch SW1 for power domain No. 1 and switch SW2 for power domain No. 2 as shown in [Figure 11-2](#)). These different separated power supplies are hence enabling to switch off power to certain regions of the device to avoid even leakage current consumption in logic supplied by the corresponding power supply.

This device employs three primary power domains, namely PD0, PD1, and PD2. As PCU supports dynamic power down of domains based on different device mode, such a possible domain is depicted below in dotted periphery.

Power domain organization and connections to the internal regulator are depicted in [Figure 11-2](#).

1. Regulator ground is separated from oscillator ground and shorted to the LV ground through star routing

2. During production test, it is also possible to provide the VDD\_LV externally through pins by configuring regulator in bypass mode.

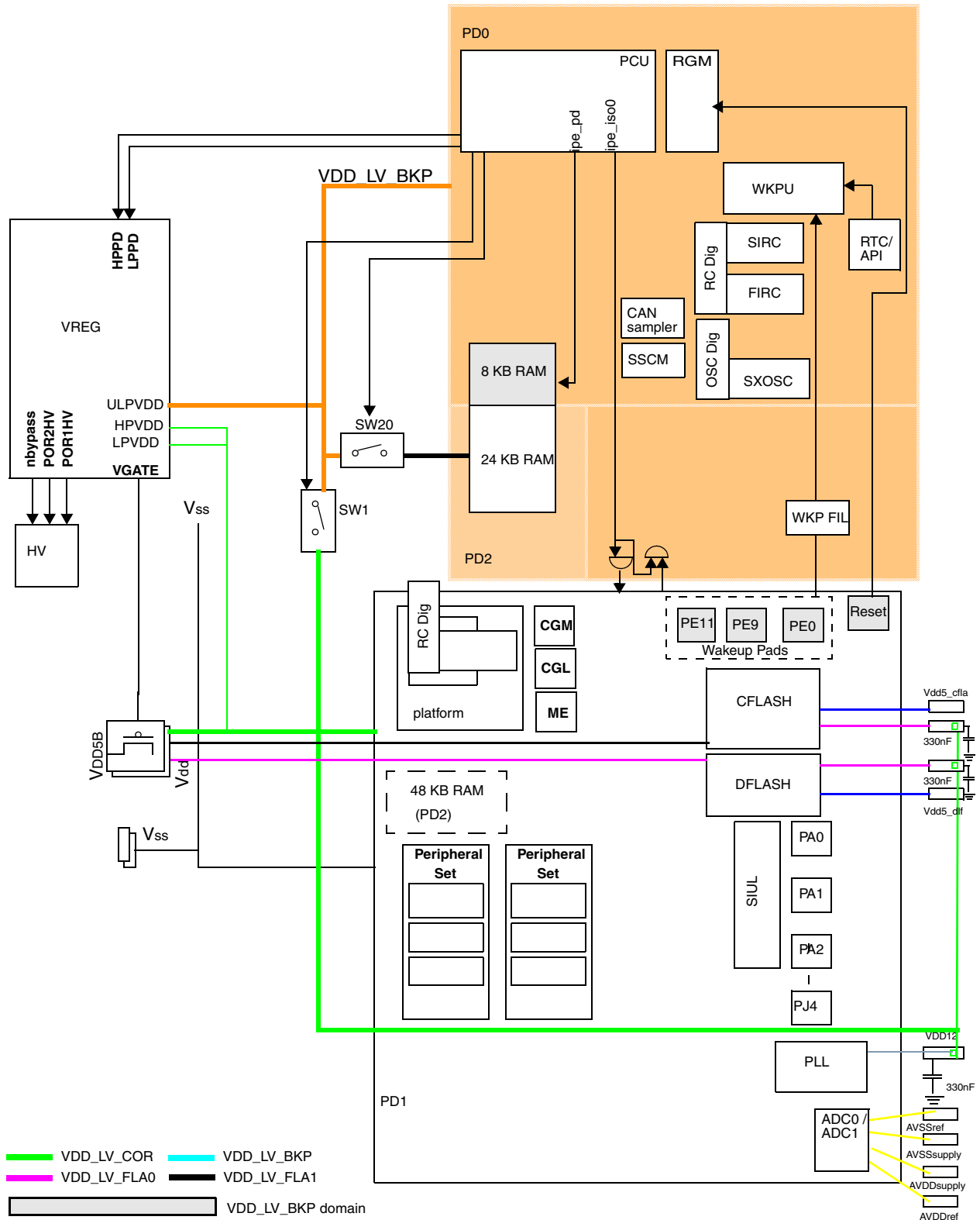


Figure 11-2. Power domain organization

This page is intentionally left blank.

# Chapter 12

## Wakeup Unit (WKPU)

### 12.1 Overview

The Wakeup Unit supports two internal sources and as many as 27 external sources that can generate interrupts or wakeup events, of which one can cause non-maskable interrupt requests or wakeup events. [Figure 12-1](#) is the block diagram of the Wakeup Unit and its interfaces to other system components.

The wakeup vector mapping is shown in [Table 12-1](#). All unused WKPU pins must use a pull resistor — either pullup (internal or external) or pulldown (external) — to ensure no leakage from floating inputs.

**Table 12-1. Wakeup vector mapping**

| Wakeup number | Port | SIU PCR#         | Port input function <sup>1</sup> (can be used in conjunction with WKPU function) | WKPU IRQ to INTC | IRQ# | WISR       | Register <sup>2</sup> bit position | Package        |                |                |
|---------------|------|------------------|--|------------------|------|------------|------------------------------------|----------------|----------------|----------------|
|               |      |                  |  |                  |      |            |                                    | 100-pin QFP    | 144-pin QFP    | 176-pin QFP    |
| WKPU0         | API  | n/a <sup>3</sup> | —  | WakeUpIRQ0       | 46   | EIF0       | 31                                 | ✓ <sup>3</sup> | ✓ <sup>3</sup> | ✓ <sup>3</sup> |
| WKPU1         | RTC  | n/a <sup>3</sup> | —  |                  |      | EIF1       | 30                                 | ✓ <sup>3</sup> | ✓ <sup>3</sup> | ✓ <sup>3</sup> |
| WKPU2         | PA1  | PCR1             | NMI  |                  |      | EIF2       | 29                                 | ✓              | ✓              | ✓              |
| WKPU3         | PA2  | PCR2             | —  |                  |      | EIF3       | 28                                 | ✓              | ✓              | ✓              |
| WKPU4         | PB1  | PCR17            | CAN0RX, LIN0RX   |                  |      | EIF4       | 27                                 | ✓              | ✓              | ✓              |
| WKPU5         | PC11 | PCR43            | CAN1RX, CAN4RX   |                  |      | EIF5       | 26                                 | ✓              | ✓              | ✓              |
| WKPU6         | PE0  | PCR64            | CAN5RX   |                  |      | EIF6       | 25                                 | ✓              | ✓              | ✓              |
| WKPU7         | PE9  | PCR73            | CAN2RX, CAN3RX   |                  |      | EIF7       | 24                                 | ✓              | ✓              | ✓              |
| WKPU8         | PB10 | PCR26            | —  |                  |      | WakeUpIRQ1 | 47                                 | EIF8           | 23             | ✓              |
| WKPU9         | PA4  | PCR4             | LIN5RX   | EIF9             | 22   |            |                                    | ✓              | ✓              | ✓              |
| WKPU10        | PA15 | PCR15            | —  | EIF10            | 21   |            |                                    | ✓              | ✓              | ✓              |
| WKPU11        | PB3  | PCR19            | LIN0RX   | EIF11            | 20   |            |                                    | ✓              | ✓              | ✓              |
| WKPU12        | PC7  | PCR39            | LIN1RX   | EIF12            | 19   |            |                                    | ✓              | ✓              | ✓              |
| WKPU13        | PC9  | PCR41            | LIN2RX   | EIF13            | 18   |            |                                    | ✓              | ✓              | ✓              |
| WKPU14        | PE11 | PCR75            | LIN3RX   | EIF14            | 17   |            |                                    | ✓              | ✓              | ✓              |
| WKPU15        | PF11 | PCR91            | LIN4RX   | EIF15            | 16   |            |                                    | x <sup>4</sup> | ✓              | ✓              |

Table 12-1. Wakeup vector mapping (continued)

| Wakeup number | Port | SIU PCR# | Port input function <sup>1</sup> (can be used in conjunction with WKPU function) | WKPU IRQ to INTC | IRQ# | WISR  | Register <sup>2</sup> bit position | Package        |                |             |
|---------------|------|----------|--|------------------|------|-------|------------------------------------|----------------|----------------|-------------|
|               |      |          |  |                  |      |       |                                    | 100-pin QFP    | 144-pin QFP    | 176-pin QFP |
| WKPU16        | PF13 | PCR93    | LIN5RX   | WakeUpIRQ2       | 48   | EIF16 | 15                                 | x <sup>4</sup> | ✓              | ✓           |
| WKPU17        | PG3  | PCR99    | —  |                  |      | EIF17 | 14                                 | x <sup>4</sup> | ✓              | ✓           |
| WKPU18        | PG5  | PCR101   | —  |                  |      | EIF18 | 13                                 | x <sup>4</sup> | ✓              | ✓           |
| WKPU19        | PA0  | PCR0     | —  |                  |      | EIF19 | 12                                 | ✓              | ✓              | ✓           |
| WKPU20        | PG7  | PCR103   | LIN6RX   |                  |      | EIF20 | 11                                 | x <sup>4</sup> | ✓              | ✓           |
| WKPU21        | PG9  | PCR105   | LIN7RX   |                  |      | EIF21 | 10                                 | x <sup>4</sup> | ✓              | ✓           |
| WKPU22        | PF9  | PCR89    | CAN2RX, CAN3RX   |                  |      | EIF22 | 9                                  | x <sup>4</sup> | ✓              | ✓           |
| WKPU23        | PI3  | PCR131   | —  |                  |      | EIF23 | 8                                  | x <sup>4</sup> | x <sup>4</sup> | ✓           |
| WKPU24        | PI1  | PCR129   | —  |                  | 49   | EIF24 | 7                                  | x <sup>4</sup> | x <sup>4</sup> | ✓           |
| WKPU25        | PB8  | PCR24    | —  |                  |      | EIF25 | 6                                  | ✓              | ✓              | ✓           |
| WKPU26        | PB9  | PCR25    | —  |                  |      | EIF26 | 5                                  | ✓              | ✓              | ✓           |
| WKPU27        | PD0  | PCR48    | —  |                  |      | EIF27 | 4                                  | ✓              | ✓              | ✓           |
| WKPU28        | PD1  | PCR49    | —  |                  |      | EIF28 | 3                                  | ✓              | ✓              | ✓           |

<sup>1</sup> This column does not contain an exhaustive list of functions on that pin. Rather, it includes peripheral communication functions (such as CAN and LINFlex Rx) that could be used to wake up the microcontroller. DSPI pins are not included because DSPI would typically be used in master mode.

<sup>2</sup> WISR, IRER, WRER, WIFEER, WIFEER, WIFER, WIPUER

<sup>3</sup> Port not required to use timer functions.

<sup>4</sup> Unavailable WKPU pins must use internal pullup enabled using WIPUER.

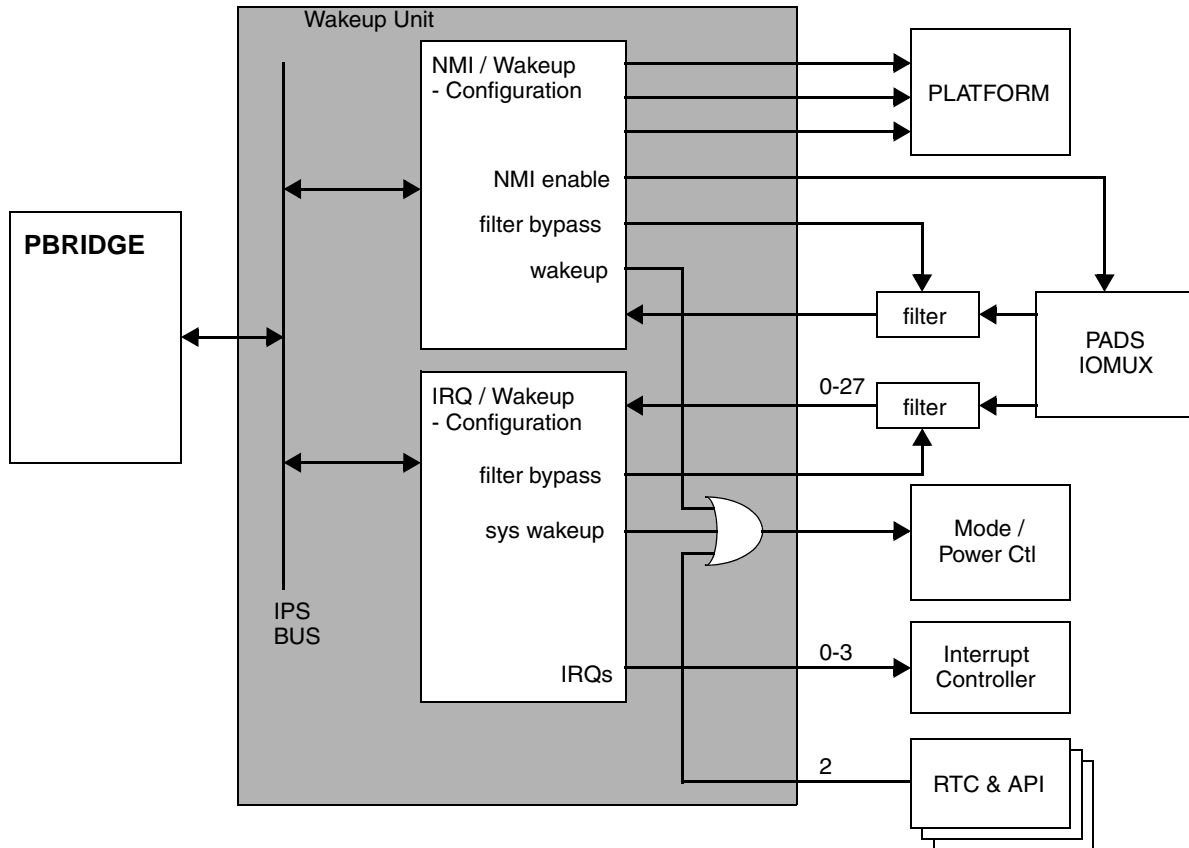


Figure 12-1. WKPU block diagram

## 12.2 Features

The Wakeup Unit supports these distinctive features:

- Non-maskable interrupt support with
  - One NMI source with bypassable glitch filter
  - Independent interrupt destination: non-maskable interrupt, critical interrupt, or machine check request
  - Edge detection
- External wakeup/interrupt support with
  - Four system interrupt vectors for as many as 29 interrupt sources
  - Analog glitch filter per each wakeup line
  - Independent interrupt mask
  - Edge detection
  - Configurable system wakeup triggering from all interrupt sources
  - Configurable pullup
- On-chip wakeup support

- Two wakeup sources
- Wakeup status mapped to same register as external wakeup/interrupt status

## 12.3 External signal description

The Wakeup Unit has 29 signal inputs that can be used as external interrupt sources in normal RUN mode or as system wakeup sources in all power down modes.

The 27 external signal inputs include one signal input that can be used as a non-maskable interrupt source in normal RUN, HALT or STOP modes or a system wakeup source in STOP or STANDBY modes. The exception is with ports PB[8] and PB[9], which have wakeup functionality in all modes except STANDBY.

### NOTE

The user should be aware that the Wake-up pins are enabled in ALL modes, therefore, the Wake-up pins should be correctly terminated to ensure minimal current consumption. Any unused Wake-up signal input should be terminated by using an external pullup or pulldown, or by internal pullup enabled at WKPU\_WIPUER. Also, care has to be taken on packages where the Wake-up signal inputs are not bonded. For these packages the user must ensure the internal pullup are enabled for those signals not bonded.

## 12.4 Memory map and register description

This section provides a detailed description of all registers accessible in the WKPU module.

### 12.4.1 Memory map

Table 12-2 shows the WKPU memory map.

Table 12-2. WKPU memory map

| Base address: 0xC3F9_4000 |  |                             |
|---------------------------|--|-----------------------------|
| Address offset            | Register name  | Location                    |
| 0x00                      | NMI Status Flag Register (NSR)                               | <a href="#">on page 217</a> |
| 0x04 – 0x07               | Reserved   |                             |
| 0x08                      | NMI Configuration Register (NCR)                             | <a href="#">on page 218</a> |
| 0x0C – 0x13               | Reserved   |                             |
| 0x14                      | Wakeup/Interrupt Status Flag Register (WISR)                 | <a href="#">on page 219</a> |
| 0x18                      | Interrupt Request Enable Register (IRER)                     | <a href="#">on page 219</a> |
| 0x1C                      | Wakeup Request Enable Register (WRER)                        | <a href="#">on page 220</a> |
| 0x20 – 0x27               | Reserved   |                             |
| 0x28                      | Wakeup/Interrupt Rising-Edge Event Enable Register (WIREER)  | <a href="#">on page 220</a> |
| 0x2C                      | Wakeup/Interrupt Falling-Edge Event Enable Register (WIFEER) | <a href="#">on page 221</a> |



Table 12-2. WKPU memory map (continued)

| Base address: 0xC3F9_4000 |  |                             |
|---------------------------|--|-----------------------------|
| Address offset            | Register name                                    | Location                    |
| 0x30                      | Wakeup/Interrupt Filter Enable Register (WIFER)  | <a href="#">on page 221</a> |
| 0x34                      | Wakeup/Interrupt Pullup Enable Register (WIPUER) | <a href="#">on page 222</a> |

**NOTE**

Reserved registers will read as 0, writes will have no effect. If SSCM\_ERROR[RAE] is enabled, a transfer error will be issued when trying to access completely reserved register space.

**12.4.2 NMI Status Flag Register (NSR)**

This register holds the non-maskable interrupt status flags.

Offset: 0x00 Access: User read/write

|       |      |       |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
|-------|------|-------|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
|       | 0    | 1     | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| R     | NIF0 | NOVF0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     | w1c  | w1c   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0    | 0     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|       | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| R     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |

Figure 12-2. NMI Status Flag Register (NSR)

Table 12-3. NSR field descriptions

| Field | Description   |
|-------|---|
| NIF0  | NMI Status Flag<br>If enabled (NREE0 or NFEE0 set), NIF0 causes an interrupt request.<br>1 An event as defined by NREE0 and NFEE0 has occurred<br>0 No event has occurred on the pad  |
| NOVF0 | NMI Overrun Status Flag<br>It will be a copy of the current NIF0 value whenever an NMI event occurs, thereby indicating to the software that an NMI occurred while the last one was not yet serviced. If enabled (NREE0 or NFEE0 set), NOVF0 causes an interrupt request.<br>1 An overrun has occurred on NMI input<br>0 No overrun has occurred on NMI input |

### 12.4.3 NMI Configuration Register (NCR)

This register holds the configuration bits for the non-maskable interrupt settings.

Offset: 0x08 Access: User read/write

|       | 0      | 1     | 2 | 3     | 4 | 5     | 6     | 7    | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|-------|--------|-------|---|-------|---|-------|-------|------|---|---|----|----|----|----|----|----|
| R     | NLOCK0 | NDSS0 |   | NWRE0 | 0 | NREE0 | NFEE0 | NFE0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |        |       |   |       |   |       |       |      |   |   |    |    |    |    |    |    |
| Reset | 0      | 0     | 0 | 0     | 0 | 0     | 0     | 0    | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| R     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |

Figure 12-3. NMI Configuration Register (NCR)

Table 12-4. NCR field descriptions

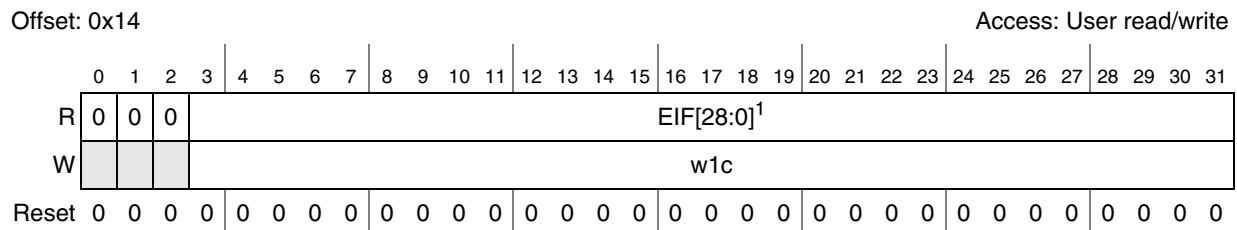
| Field  | Description   |
|--------|---|
| NLOCK0 | NMI Configuration Lock Register<br>Writing a 1 to this bit locks the configuration for the NMI until it is unlocked by a system reset. Writing a 0 has no effect.   |
| NDSS0  | NMI Destination Source Select<br>00 Non-maskable interrupt<br>01 Critical interrupt<br>10 Machine check request<br>11 Reserved—no NMI, critical interrupt, or machine check request generated   |
| NWRE0  | NMI Wakeup Request Enable<br>1 A set NIF0 bit or set NOVFO bit causes a system wakeup request<br>0 System wakeup requests from the corresponding NIF0 bit are disabled<br><b>Note:</b> Software should only enable the NMI after the IVPR/IVOR registers have been configured. This should be noted when booting from RESET or STANDBY mode as all registers will have been cleared to their reset state. |
| NREE0  | NMI Rising-edge Events Enable<br>1 Rising-edge event is enabled<br>0 Rising-edge event is disabled  |
| NFEE0  | NMI Falling-edge Events Enable<br>1 Falling-edge event is enabled<br>0 Falling-edge event is disabled   |
| NFE0   | NMI Filter Enable<br>Enable analog glitch filter on the NMI pad input.<br>1 Filter is enabled<br>0 Filter is disabled   |

**NOTE**

Writing a 0 to both NREE0 and NFEE0 disables the NMI functionality completely (that is, no system wakeup or interrupt will be generated on any pad activity)!

**12.4.4 Wakeup/Interrupt Status Flag Register (WISR)**

This register holds the wakeup/interrupt flags.



**Figure 12-4. Wakeup/Interrupt Status Flag Register (WISR)**

<sup>1</sup> EIF[24:20] and EIF[18:15] not available in 100-pin LQFP; EIF[24:23] not available in 144-pin LQFP

**Table 12-5. WISR field descriptions**

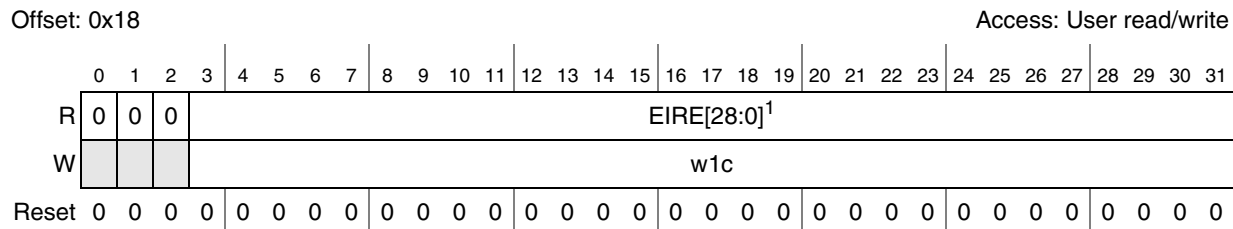
| Field  | Description  |
|--------|--|
| EIF[x] | External Wakeup/Interrupt WKPU[x] Status Flag<br>This flag can be cleared only by writing a 1. Writing a 0 has no effect. If enabled (IRER[x]), EIF[x] causes an interrupt request.<br>1 An event as defined by WIREER and WIFEER has occurred<br>0 No event has occurred on the pad |

**NOTE**

Status bits associated with on-chip wakeup sources are located to the left of the external wakeup/interrupt status bits and are read only. The wakeup for these sources must be configured and cleared at the on-chip wakeup source. Also, the configuration registers for the external interrupts/wakeups do not have corresponding bits.

**12.4.5 Interrupt Request Enable Register (IRER)**

This register is used to enable the interrupt messaging from the wakeup/interrupt pads to the interrupt controller.

**Figure 12-5. Interrupt Request Enable Register (IRER)**

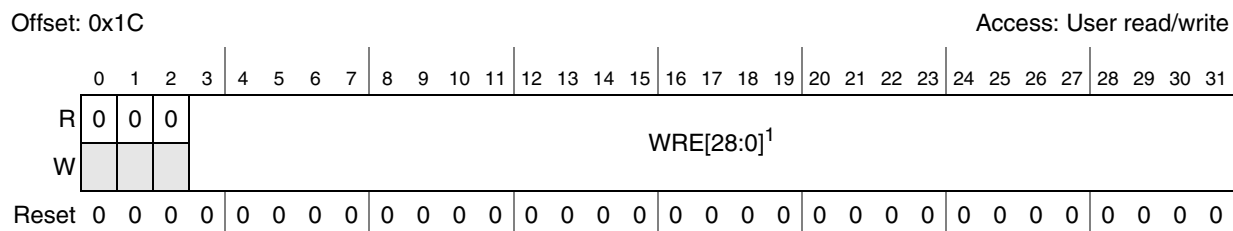
<sup>1</sup> EIRE[24:20] and EIRE[18:15] not available in 100-pin LQFP; EIRE[24:23] not available in 144-pin LQFP

**Table 12-6. IRER field descriptions**

| Field   | Description  |
|---------|--|
| EIRE[x] | External Interrupt Request Enable x<br>1 A set EIF[x] bit causes an interrupt request<br>0 Interrupt requests from the corresponding EIF[x] bit are disabled |

## 12.4.6 Wakeup Request Enable Register (WRER)

This register is used to enable the system wakeup messaging from the wakeup/interrupt pads to the mode entry and power control modules.

**Figure 12-6. Wakeup Request Enable Register (WRER)**

<sup>1</sup> WRE[24:20] and WRE[18:15] not available in 100-pin LQFP; WRE[24:23] not available in 144-pin LQFP

**Table 12-7. WRER field descriptions**

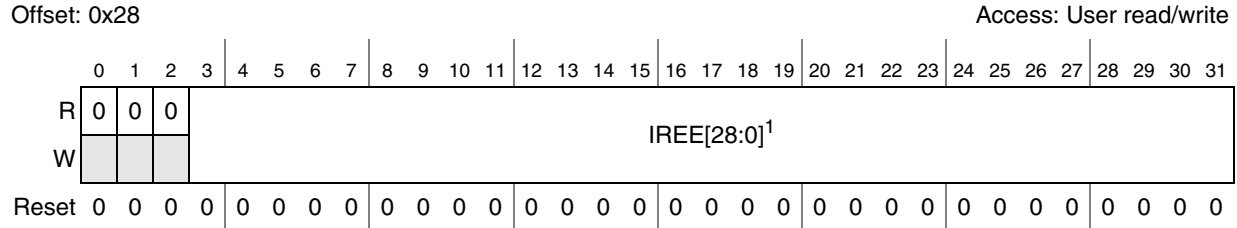
| Field  | Description  |
|--------|--|
| WRE[x] | External Wakeup Request Enable x<br>1 A set EIF[x] bit causes a system wakeup request<br>0 System wakeup requests from the corresponding EIF[x] bit are disabled |

## 12.4.7 Wakeup/Interrupt Rising-Edge Event Enable Register (WIREER)

This register is used to enable rising-edge triggered events on the corresponding wakeup/interrupt pads.

### NOTE

The RTC\_API can only be configured on the rising edge.

**Figure 12-7. Wakeup/Interrupt Rising-Edge Event Enable Register (WIREER)**

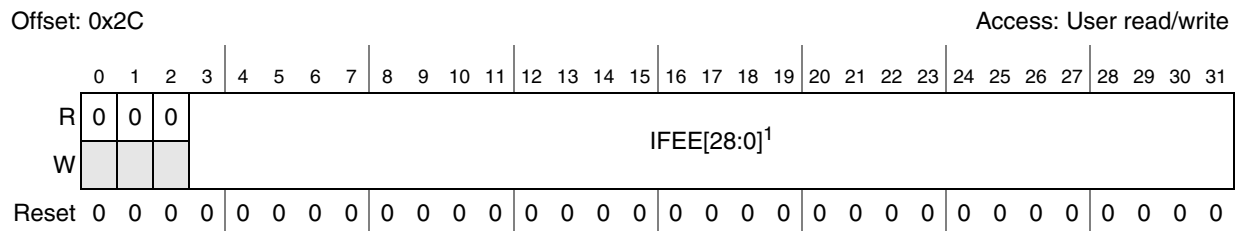
<sup>1</sup> IREE[24:20] and IREE[18:15] not available in 100-pin LQFP; IREE[24:23] not available in 144-pin LQFP

**Table 12-8. WIREER field descriptions**

| Field   | Description   |
|---------|---|
| IREE[x] | External Interrupt Rising-edge Events Enable x<br>1 Rising-edge event is enabled<br>0 Rising-edge event is disabled |

### 12.4.8 Wakeup/Interrupt Falling-Edge Event Enable Register (WIFEER)

This register is used to enable falling-edge triggered events on the corresponding wakeup/interrupt pads.

**Figure 12-8. Wakeup/Interrupt Falling-Edge Event Enable Register (WIFEER)**

<sup>1</sup> IFEE[24:20] and IFEE[18:15] not available in 100-pin LQFP; IFEE[24:23] not available in 144-pin LQFP

**Table 12-9. WIFEER field descriptions**

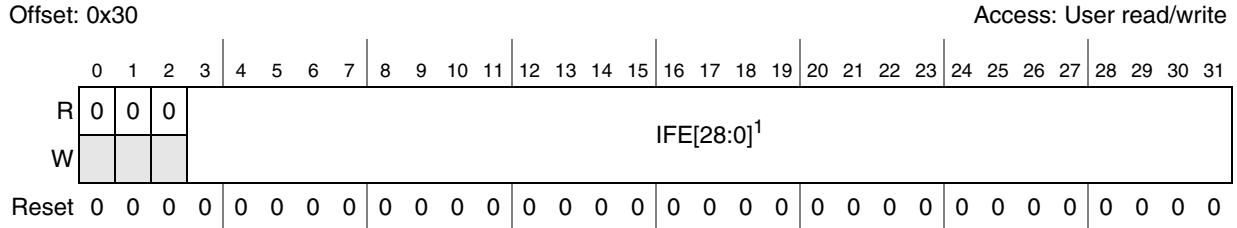
| Field   | Description  |
|---------|--|
| IFEE[x] | External Interrupt Falling-edge Events Enable x<br>1 Falling-edge event is enabled<br>0 Falling-edge event is disabled |

### 12.4.9 Wakeup/Interrupt Filter Enable Register (WIFER)

This register is used to enable an analog filter on the corresponding interrupt pads to filter out glitches on the inputs.

#### NOTE

There is no analog filter for the RTC\_API.



**Figure 12-9. Wakeup/Interrupt Filter Enable Register (WIFER)**

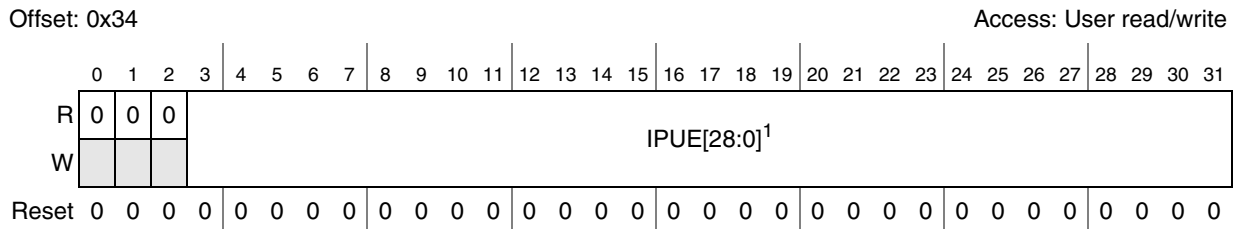
<sup>1</sup> IFE[24:20] and IFE[18:15] not available in 100-pin LQFP; IFE[24:23] not available in 144-pin LQFP

**Table 12-10. WIFER field descriptions**

| Field  | Description   |
|--------|---|
| IFE[x] | External Interrupt Filter Enable x<br>Enable analog glitch filter on the external interrupt pad input.<br>1 Filter is enabled<br>0 Filter is disabled |

### 12.4.10 Wakeup/Interrupt Pullup Enable Register (WIPUER)

This register is used to enable a pullup on the corresponding interrupt pads to pull an unconnected wakeup/interrupt input to a value of 1.



**Figure 12-10. Wakeup/Interrupt Pullup Enable Register (WIPUER)**

<sup>1</sup> IPUE[24:20] and IPUE[18:15] not available in 100-pin LQFP; IPUE[24:23] not available in 144-pin LQFP

**Table 12-11. WIPUER field descriptions**

| Field   | Description   |
|---------|---|
| IPUE[x] | External Interrupt Pullup Enable x<br>1 Pullup is enabled<br>0 Pullup is disabled |

## 12.5 Functional description

### 12.5.1 General

This section provides a complete functional description of the Wakeup Unit.

## 12.5.2 Non-maskable interrupts

The Wakeup Unit supports one non-maskable interrupt that is allocated to the following pins:

- 100-pin LQFP: Pin 7
- 144-pin LQFP: Pin 11
- 176-pin LQFP: Pin 19

The Wakeup Unit supports the generation of three types of interrupts from the NMI. The Wakeup Unit supports the capturing of a second event per NMI input before the interrupt is cleared, thus reducing the chance of losing an NMI event.

Each NMI passes through a bypassable analog glitch filter.

### NOTE

Glitch filter control and pad configuration should be done while the NMI is disabled in order to avoid erroneous triggering by glitches caused by the configuration process itself.

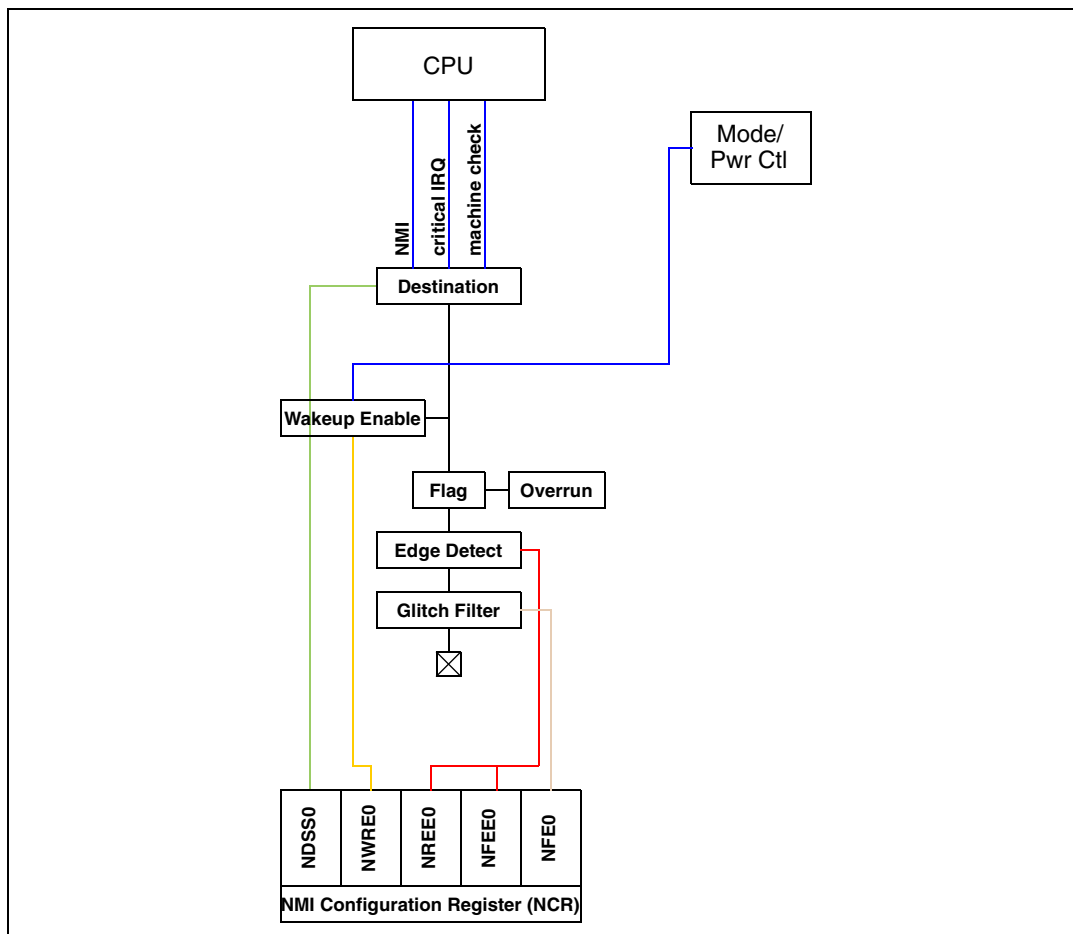


Figure 12-11. NMI pad diagram

### 12.5.2.1 NMI management

The NMI can be enabled or disabled using the single NCR register laid out to contain all configuration bits for an NMI in a single byte (see [Figure 12-3](#)). The pad defined as an NMI can be configured by the user to recognize interrupts with an active rising edge, an active falling edge or both edges being active. A setting of having both edge events disabled results in no interrupt being detected, and should not be configured.

The active NMI edge is controlled by the user through the configuration of the NREE0 and NFEE0 bits.

#### NOTE

After reset, NREE0 and NFEE0 are set to 0. Therefore, the NMI functionality is disabled after reset and must be enabled explicitly by software.

Once the pad's NMI functionality has been enabled, the pad cannot be reconfigured in the IOMUX to override or disable the NMI.

The NMI destination interrupt is controlled by the user through the configuration of the NDSS0 field. See [Table 12-4](#) for details.

An NMI supports a status flag and an overrun flag, which are located in the NSR register (see [Figure 12-2](#)). The NIF0 and NOV0 fields in this register are cleared by writing a 1 to them; this prevents inadvertent overwriting of other flags in the register. The status flag is set whenever an NMI event is detected. The overrun flag is set whenever an NMI event is detected and the status flag is set (that is, has not yet been cleared).

#### NOTE

The overrun flag is cleared by writing a 1 to the appropriate overrun bit in the NSR register. If the status bit is cleared and the overrun bit is still set, the pending interrupt will not be cleared.

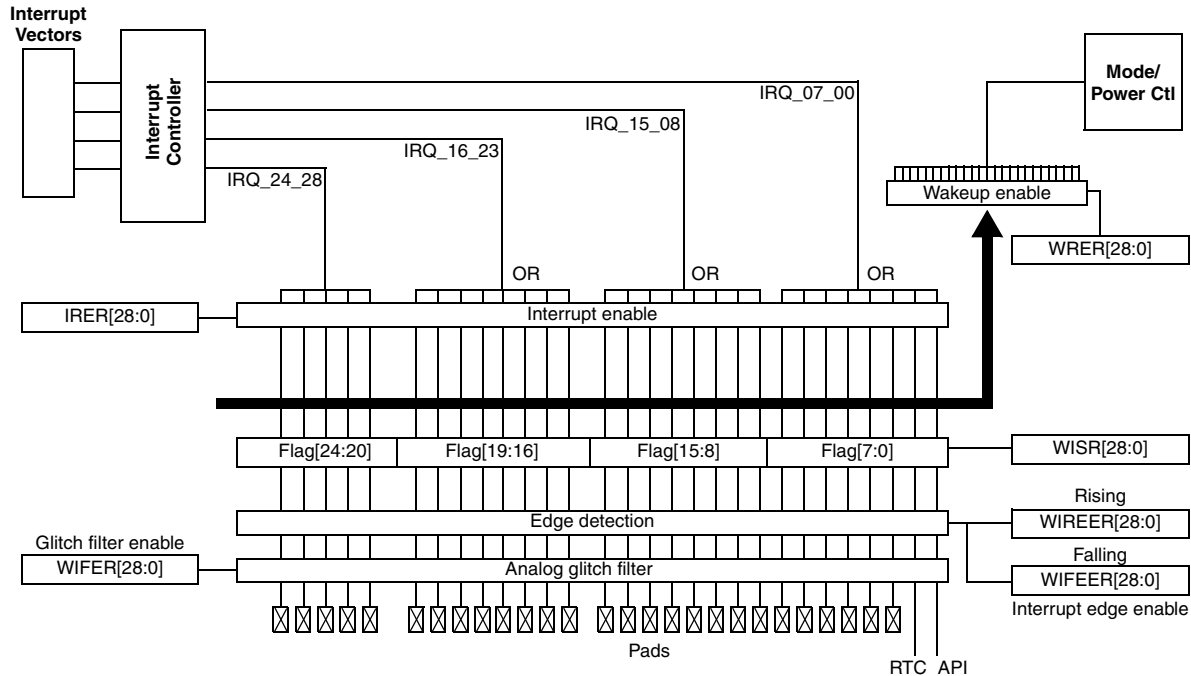
### 12.5.3 External wakeups/interrupts

The Wakeup Unit supports as many as 27 external wakeup/interrupts that can be allocated to any pad necessary at the SoC level. This allocation is fixed per SoC.

The Wakeup Unit supports as many as four interrupt vectors to the interrupt controller of the SoC. Each interrupt vector can support up to the number of external interrupt sources from the device pads with the total across all vectors being equal to the number of external interrupt sources. Each external interrupt source is assigned to exactly one interrupt vector. The interrupt vector assignment is sequential so that one interrupt vector is for external interrupt sources 0 through  $N - 1$ , the next is for  $N$  through  $N + M - 1$ , and so forth.

See [Figure 12-12](#) for an overview of the external interrupt implementation for the example of four interrupt vectors with as many as eight external interrupt sources each.





**Figure 12-12. External interrupt pad diagram**

All of the external interrupt pads within a single group have equal priority. It is the responsibility of the user software to search through the group of sources in the most appropriate way for their application.

#### NOTE

Glitch filter control and pad configuration should be done while the external interrupt line is disabled in order to avoid erroneous triggering by glitches caused by the configuration process itself.

### 12.5.3.1 External interrupt management

Each external interrupt can be enabled or disabled independently. This can be performed using a single rolled up register (Figure 12-5). A pad defined as an external interrupt can be configured by the user to recognize external interrupts with an active rising edge, an active falling edge or both edges being active.

#### NOTE

Writing a 0 to both  $IREE[x]$  and  $IFEE[x]$  disables the external interrupt functionality for that pad completely (that is, no system wakeup or interrupt will be generated on any activity on that pad)!

The active IRQ edge is controlled by the users through the configuration of the registers WIREER and WIFEER.

Each external interrupt supports an individual flag that is held in the flag register (WISR). The bits in the WISR[EIF] field are cleared by writing a 1 to them; this prevents inadvertent overwriting of other flags in the register.

## 12.5.4 On-chip wakeups

The Wakeup Unit supports two on-chip wakeup sources. It combines the on-chip wakeups with the external ones to generate a single wakeup to the system.

### 12.5.4.1 On-chip wakeup management

In order to allow software to determine the wakeup source at one location, on-chip wakeups are reported along with external wakeups in the WISR register (see [Figure 12-4](#) for details). Enabling and clearing of these wakeups are done via the on-chip wakeup source's own registers.

# Chapter 13

## Real Time Clock / Autonomous Periodic Interrupt (RTC/API)

### 13.1 Overview

The RTC/API is a free running counter used for time keeping applications. The RTC may be configured to generate an interrupt at a predefined interval independent of the mode of operation (run mode or low power mode). If in a low power mode when the RTC interval is reached, the RTC first generates a wakeup, and then asserts the interrupt request. The RTC also supports an autonomous periodic interrupt (API) function used to generate a periodic wakeup request to exit a low power mode or an interrupt request.

### 13.2 Features

Features of the RTC/API include:

- Three selectable counter clock sources
  - SIRC (128 kHz)
  - SXOSC (32 KHz)
  - FIRC (16 MHz)
- Optional  $\div 512$  prescaler and optional  $\div 32$  prescaler
- 32-bit counter
  - Supports times as long as 1.5 months with 1 ms resolution
  - Runs in all modes of operation
  - Reset when disabled by software and by POR
- 12-bit compare value to support interrupt intervals from 1 second to longer than 1 hour, with 1 second resolution
- RTC compare value changeable while counter is running
- RTC status and control register are reset only by POR
- Autonomous periodic interrupt (API)
  - 10-bit compare value to support wakeup intervals of 1.0 ms to 1 second
  - Compare value changeable while counter is running
- Configurable interrupt for RTC match, API match, and RTC rollover
- Configurable wakeup event for RTC match, API match, and RTC rollover

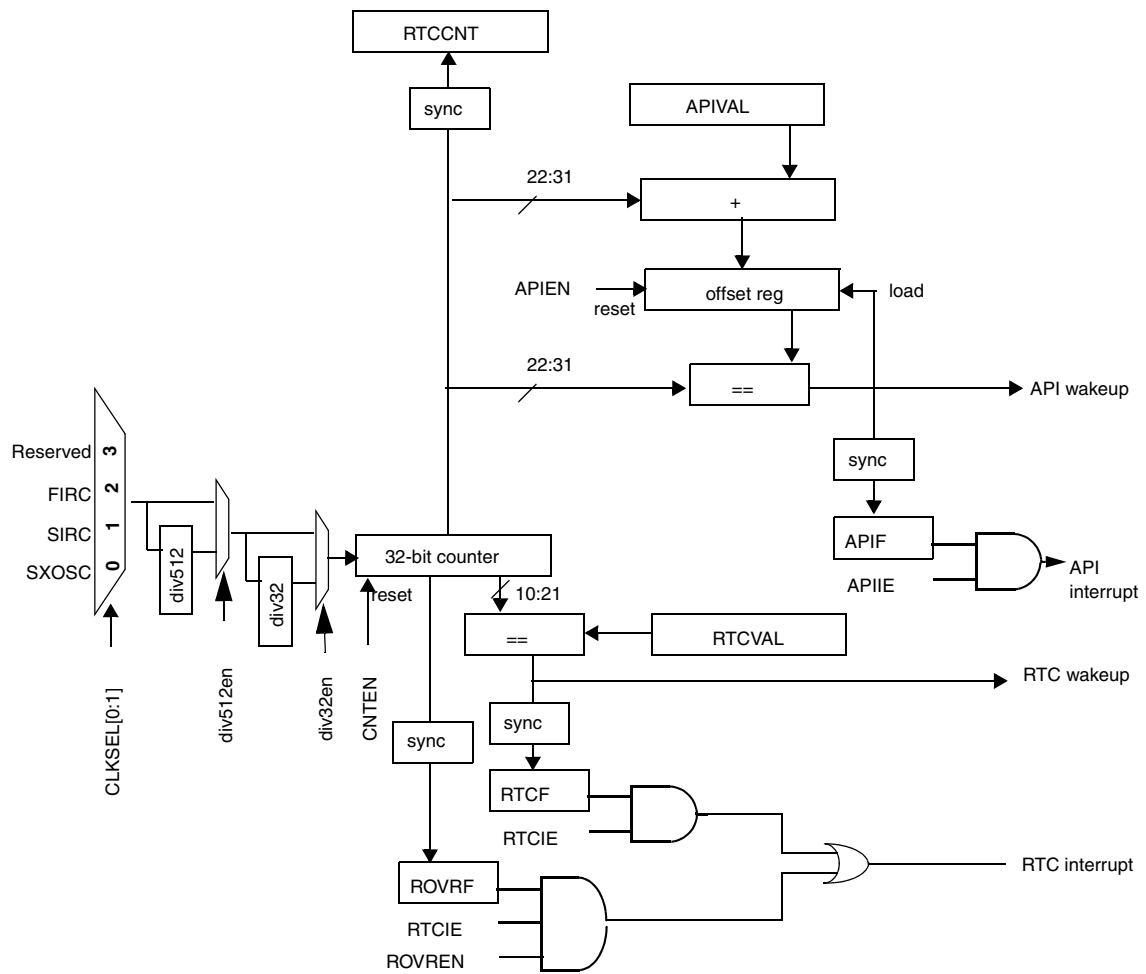


Figure 13-1. RTC/API block diagram

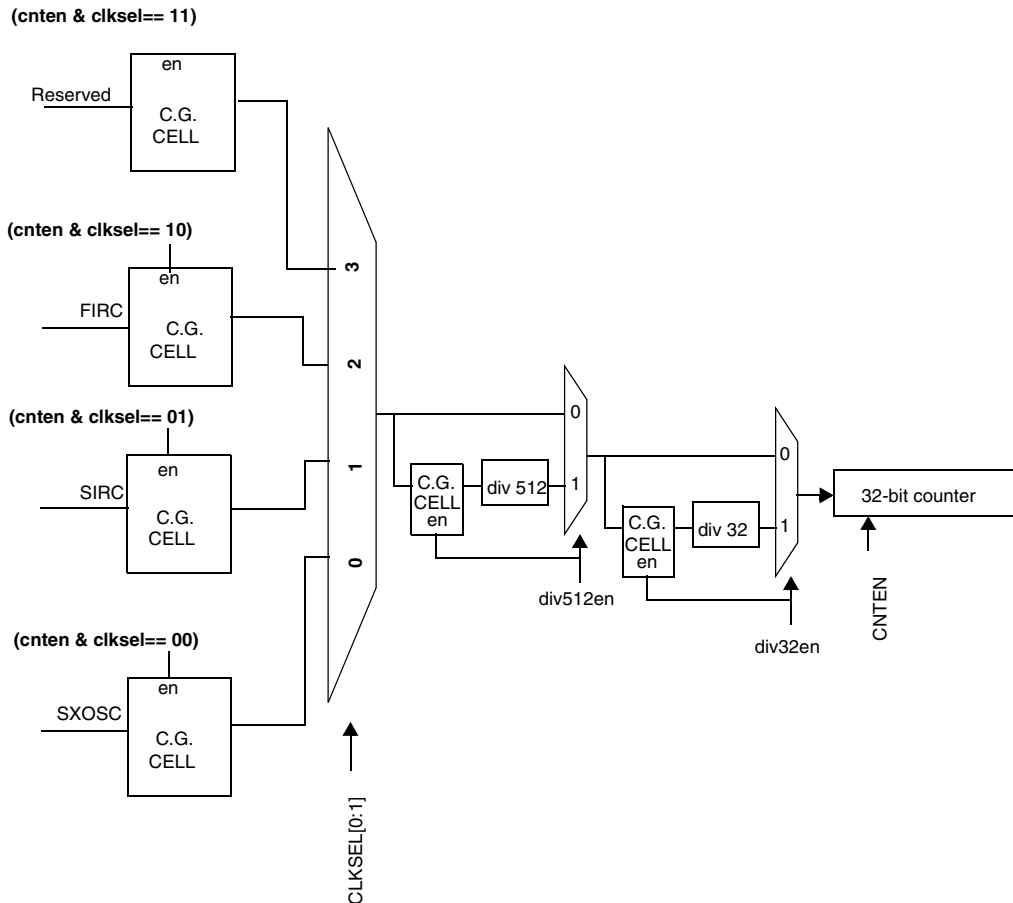


Figure 13-2. Clock gating for RTC clocks

### 13.3 Device-specific information

For MPC5606BK, the device specific information is the following:

- SXOSC, FIRC, and SIRC clocks are provided as counter clocks for the RTC. Default clock on reset is SIRC divided by 4.
- The RTC will be reset on destructive reset, with the exception of software watchdog reset.
- The RTC provides a configurable divider by 512 to be optionally used when FIRC source is selected.

### 13.4 Modes of operation

#### 13.4.1 Functional mode

There are two functional modes of operation for the RTC: normal operation and low power mode. In normal operation, all RTC registers can read or written and the input isolation is disabled. The RTC/API

and associated interrupts are optionally enabled. In low power mode, the bus interface is disabled and the input isolation is enabled. The RTC/API is enabled if enabled prior to entry into low power mode.

### 13.4.2 Debug mode

On entering into the debug mode the RTC counter freezes on the last valid count if the RTCC[FRZEN] is set. On exit from debug mode counter continues from the frozen value.

## 13.5 Register descriptions

Table 13-1 lists the RTC/API registers.

Table 13-1. RTC/API register map

| Base address: 0xC3FE_C000 |   |             |
|---------------------------|---|-------------|
| Address offset            | Register                                  | Location    |
| 0x0                       | RTC Supervisor Control Register (RTCSUPV) | on page 230 |
| 0x4                       | RTC Control Register (RTCC)               | on page 231 |
| 0x8                       | RTC Status Register (RTCS)                | on page 233 |
| 0xC                       | RTC Counter Register (RTCCNT)             | on page 234 |

### 13.5.1 RTC Supervisor Control Register (RTCSUPV)

The RTCSUPV register contains the SUPV bit, which determines whether other registers are accessible in supervisor mode or user mode.

**NOTE**

RTCSUPV register is accessible only in supervisor mode.

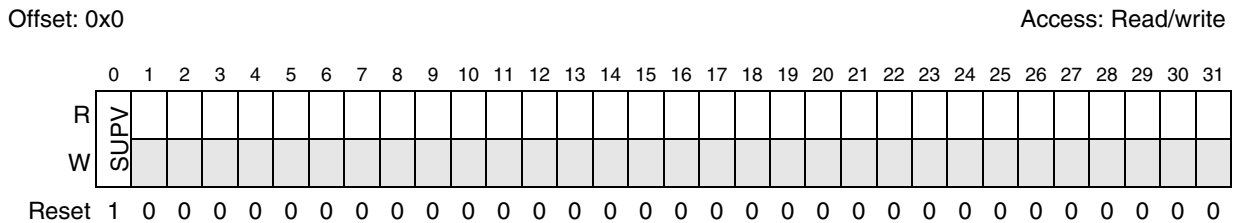


Figure 13-3. RTC Supervisor Control Register (RTCSUPV)

Table 13-2. RTCSUPV field descriptions

| Field | Description  |
|-------|--|
| SUPV  | RTC Supervisor Bit<br>0 All registers are accessible in both user as well as supervisor mode.<br>1 All other registers are accessible in supervisor mode only. |

## 13.5.2 RTC Control Register (RTCC)

The RTCC register contains:

- RTC counter enable
- RTC interrupt enable
- RTC clock source select
- RTC compare value
- API enable
- API interrupt enable
- API compare value

Offset: 0x4

Access: User read/write

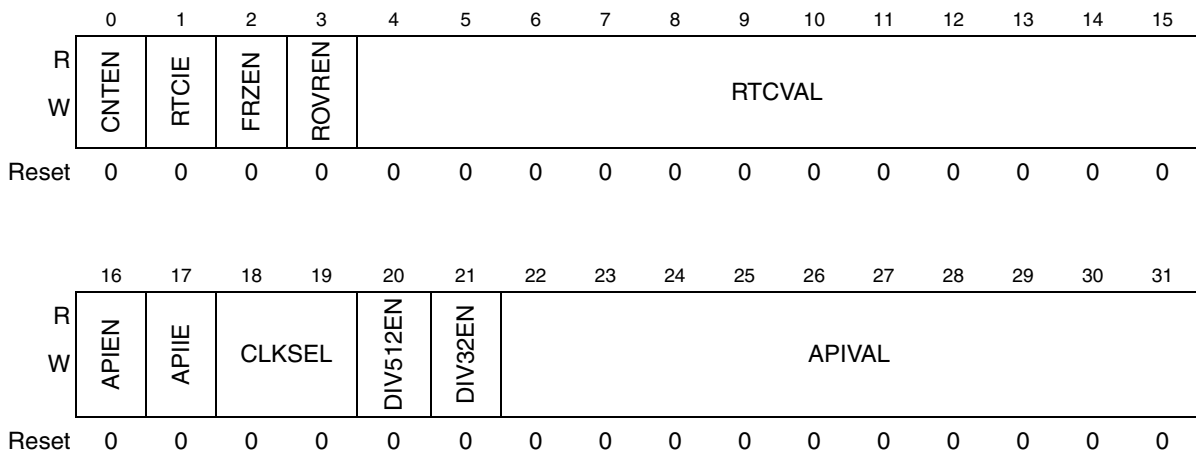


Figure 13-4. RTC Control Register (RTCC)

Table 13-3. RTCC field descriptions

| Field | Description  |
|-------|--|
| CNTEN | Counter Enable<br>The CNTEN field enables the RTC counter. Making CNTEN bit 0 has the effect of asynchronously resetting (synchronous reset negation) all the RTC and API logic. This allows for the RTC configuration and clock source selection to be updated without causing synchronization issues.<br>1 Counter enabled<br>0 Counter disabled |
| RTCIE | RTC Interrupt Enable<br>The RTCIE field enables interrupts requests to the system if RTCF is asserted.<br>1 RTC interrupts enabled<br>0 RTC interrupts disabled  |
| FRZEN | Freeze Enable<br>The counter freezes on entering the debug mode on the last valid count value if the FRZEN bit is set. After coming out of the debug mode, the counter starts from the frozen value.<br>0 Counter does not freeze in debug mode.<br>1 Counter freezes in debug mode.   |

Table 13-3. RTCC field descriptions (continued)

| Field    | Description   |
|----------|---|
| ROVREN   | Counter Roll Over Wakeup/Interrupt Enable<br>The ROVREN bit enables wakeup and interrupt requests when the RTC has rolled over from 0xFFFF_FFFF to 0x0000_0000. The RTCIE bit must also be set in order to generate an interrupt from a counter rollover.<br>1 RTC rollover wakeup/interrupt enabled<br>0 RTC rollover wakeup/interrupt disabled  |
| RTCVAL   | <b>Note:</b> RTC Compare Value<br>The RTCVAL bits are compared to bits 10:21 of the RTC counter and if match sets RTCF. RTCVAL can be updated when the counter is running.  |
| APIEN    | Autonomous Periodic Interrupt Enable<br>The APIEN bit enables the autonomous periodic interrupt function.<br>1 API enabled<br>0 API disabled  |
| APIIE    | API Interrupt Enable<br>The APIIE bit enables interrupts requests to the system if APIF is asserted.<br>1 API interrupts enabled<br>0 API interrupts disabled   |
| CLKSEL   | Clock Select<br>This field selects the clock source for the RTC. CLKSEL may only be updated when CNTEN is 0. The user should ensure that oscillator is enabled before selecting it as a clock source for RTC.<br>00 SXOSC<br>01 SIRC<br>10 FIRC<br>11 Reserved  |
| DIV512EN | Divide by 512 enable<br>The DIV512EN bit enables the 512 clock divider. DIV512EN may only be updated when CNTEN is 0.<br>0 Divide by 512 is disabled.<br>1 Divide by 512 is enabled.  |
| DIV32EN  | Divide by 32 enable<br>The DIV32EN bit enables the 32 clock divider. DIV32EN may only be updated when CNTEN is 0.<br>0 Divide by 32 is disabled.<br>1 Divide by 32 is enabled.  |
| APIVAL   | API Compare Value<br>The APIVAL field is compared with bits 22:31 of the RTC counter and if match asserts an interrupt/wakeup request. APIVAL may only be updated when APIEN is 0 or API function is undefined.<br><b>Note:</b> API functionality starts only when APIVAL is nonzero. The first API interrupt takes two more cycles because of synchronization of APIVAL to the RTC clock, and APIVAL + 1 cycles for subsequent occurrences. After that, interrupts are periodic in nature. The minimum supported value of APIVAL is 4. |



### 13.5.3 RTC Status Register (RTCS)

The RTCS register contains:

- RTC interrupt flag
- API interrupt flag
- ROLLOVR flag

Offset: 0x8

Access: User read/write

|       |   |   |      |   |   |   |   |   |   |   |    |    |    |    |    |    |
|-------|---|---|------|---|---|---|---|---|---|---|----|----|----|----|----|----|
|       | 0 | 1 | 2    | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| R     | 0 | 0 | RTCF | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |      |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0    | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       |    |    |      |    |    |       |    |    |    |    |    |    |    |    |    |    |
|-------|----|----|------|----|----|-------|----|----|----|----|----|----|----|----|----|----|
|       | 16 | 17 | 18   | 19 | 20 | 21    | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| R     | 0  | 0  | APIF | 0  | 0  | ROVRF | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |    |    |      |    |    |       |    |    |    |    |    |    |    |    |    |    |
| Reset | 0  | 0  | 0    | 0  | 0  | 0     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |

Figure 13-5. RTC Status Register (RTCS)

Table 13-4. RTCS field descriptions

| Field | Description   |
|-------|---|
| RTCF  | <p>RTC Interrupt Flag</p> <p>The RTCF bit indicates that the RTC counter has reached the counter value matching RTCVAL. RTCF is cleared by writing a 1 to RTCF. Writing a 0 to RTCF has no effect.</p> <p>1 RTC counter matches RTCVAL<br/>0 RTC counter is not equal to RTCVAL</p>   |
| APIF  | <p>API Interrupt Flag</p> <p>The APIF bit indicates that the RTC counter has reached the counter value matching API offset value. APIF is cleared by writing a 1 to APIF. Writing a 0 to APIF has no effect.</p> <p>1 API interrupt<br/>0 No API interrupt</p> <p>Note: The periodic interrupt comes after APIVAL[0:9] + 1 RTC counts</p> |
| ROVRF | <p>Counter Roll Over Interrupt Flag</p> <p>The ROVRF bit indicates that the RTC has rolled over from 0xffff_fff to 0x0000_0000. ROVRF is cleared by writing a 1 to ROVRF.</p> <p>1 RTC has rolled over<br/>0 RTC has not rolled over</p>  |

### 13.5.4 RTC Counter Register (RTCCNT)

The RTCCNT register contains the current value of the RTC counter.

Offset: 0xC

Access: Read

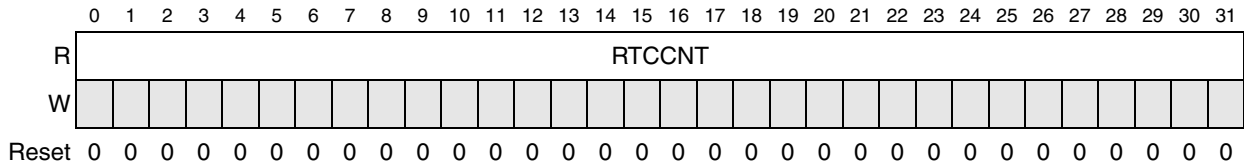


Figure 13-6. RTC Counter Register (RTCCNT)

Table 13-5. RTCCNT field descriptions

| Field  | Description  |
|--------|--|
| RTCCNT | RTC Counter Value<br>Due to the clock synchronization, the RTCCNT value may actually represent a previous counter value. |

### 13.6 RTC functional description

The RTC consists of a 32-bit free running counter enabled with the RTCC[**CNTEN**] bit (**CNTEN** when negated asynchronously resets the counter and synchronously enables the counter when enabled). The value of the counter may be read via the RTCCNT register. Note that due to the clock synchronization, the RTCCNT value may actually represent a previous counter value. The difference between the counter and the read value depends on ratio of counter clock and system clock. Maximum possible difference between the two is 6 count values.

The clock source to the counter is selected with the RTCC[**CLKSEL**] field, which gives the options for clocking the RTC/API. The output of the clock mux can be optionally divided by combination of 512 and 32 to give a 1 ms RTC/API count period for different clock sources. Note that the RTCC[**CNTEN**] bit must be disabled when the RTC/API clock source is switched.

When the counter value for counter bits 10:21 match the 12-bit value in the RTCC[**RTCVAL**] field, then the RTCS[**RTCF**] interrupt flag bit is set (after proper clock synchronization). If the RTCC[**RTCIE**] interrupt enable bit is set, then the RTC interrupt request is generated. The RTC supports interrupt requests in the range of 1 s to 4096 s (> 1 hr) with a 1 s resolution. If there is a match while in low power mode then the RTC will first generate a wakeup request to force a wakeup to run mode, then the RTCF flag will be set.

A rollover wakeup and/or interrupt can be generated when the RTC transitions from a count of 0xFFFF\_FFFF to 0x0000\_0000. The rollover flag is enabled by setting the RTCC[**ROVREN**] bit. An RTC counter rollover with this bit will cause a wakeup from low power mode. An interrupt request is generated for an RTC counter rollover when both the RTCC[**ROVREN**] and RTCC[**RTCIE**] bits are set.

All the flags and counter values are synchronized with the system clock. It is assumed that the system clock frequency is always more than or equal to the rtc\_clk used to run the counter.

## 13.7 API functional description

Setting the RTCC[APIEN] bit enables the autonomous interrupt function. The 10-bit RTCC[APIVAL] field selects the time interval for triggering an interrupt and/or wakeup event. Since the RTC is a free running counter, the APIVAL is added to the current count to calculate an offset. When the counter reaches the offset count, a interrupt and/or wakeup request is generated. Then the offset value is recalculated and again re-triggers a new request when the new value is reached. APIVAL may only be updated when APIEN is disabled. When a compare is reached, the RTCS[APIF] interrupt flag bit is set (after proper clock synchronization). If the RTCC[APIIE] interrupt enable bit is set, then the API interrupt request is generated. If there is a match while in low power mode, then the API will first generate a wakeup request to force a wakeup into normal operation, then the APIF flag will be set.

This page is intentionally left blank.

# Chapter 14

## CAN Sampler

### 14.1 Introduction

The CAN Sampler peripheral has been designed to store the first identifier of CAN message detected on the CAN bus while no precise clock (Crystal) is running at that time on the device, typically in Low Power modes (STOP, HALT or STANBY) or in RUN mode with crystal switched off.

Depending on both CAN baud rate and Low Power mode used, it is possible to catch either the first or the second CAN frame by sampling one CAN Rx port among 6 and storing all samples in internal registers.

After selection of the mode (first or second frame), the CAN Sampler stores samples of the 48 bits or skips the first frame and stores samples of the 48 bits of second frame using the 16 MHz fast internal RC oscillator and the 5-bit clock prescaler.

After completion, software must process the sampled data in order to rebuild the 48 minimal bits.

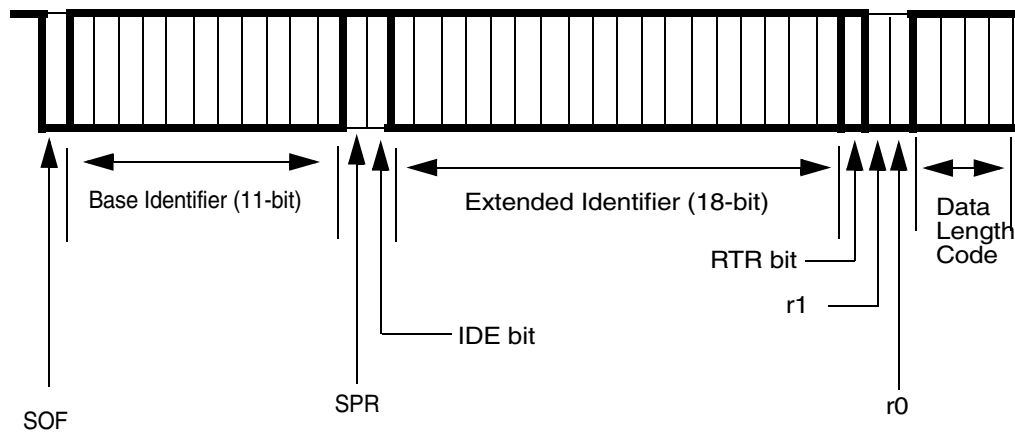


Figure 14-1. Extended CAN data frame

### 14.2 Main features

- Store 384 samples, equivalent to 48 CAN bit at 8 samples/bit
- Sample frequency from 500 kHz to 16 MHz, equivalent at 8 samples/bit to CAN baud rates of 62.5 Kbit/s to 2 Mbit/s
- User selectable CAN Rx sample port [CAN0RX–CAN5RX]
- 16 MHz fast internal RC oscillator clock
- 5-bit clock prescaler
- Configurable trigger mode (immediate, next frame)
- Flexible samples processing by software
- Very low power consumption

## 14.3 Memory map and register description

The CAN Sampler registers are listed in [Table 14-1](#).

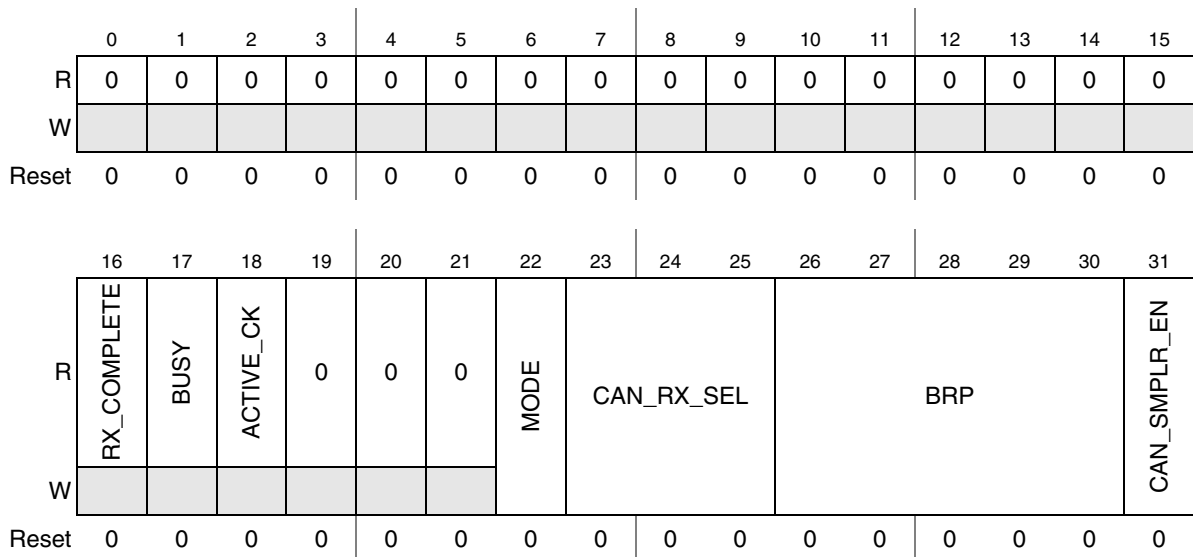
**Table 14-1. CAN Sampler memory map**

| Base address: 0xFFE7_0000 |                       |                             |
|---------------------------|-----------------------|-----------------------------|
| Address offset            | Register              | Location                    |
| 0x00                      | Control Register (CR) | <a href="#">on page 238</a> |
| 0x04–0x30                 | Sample registers 0–11 | <a href="#">on page 239</a> |

### 14.3.1 Control Register (CR)

Offset: 0x00

Access: Read/write



**Figure 14-2. Control Register (CR)**

**Table 14-2. CR field descriptions**

| Field       | Description  |
|-------------|--|
| RX_COMPLETE | 0: CAN frame has not been stored in the sample registers<br>1: CAN frame is stored in the sample registers                                 |
| BUSY        | This bit indicates the status of sampling<br>0: Sampling is complete or has not started<br>1: Sampling is ongoing                          |
| ACTIVE_CK   | This bit indicates the current clock (xmem_ck) for sample registers<br>0: ipg_clk_s is currently xmem_ck<br>1: RC_CLK is currently xmem_ck |
| MODE        | 0: Skip the first frame and sample and store the second frame (SF_MODE)<br>1: Sample and store the first frame (FF_MODE)                   |

Table 14-2. CR field descriptions (continued)

| Field        | Description   |
|--------------|---|
| CAN_RX_SEL   | These bits determine which RX bit is sampled.<br>000: Rx0 is selected<br>001: Rx1 is selected<br>010: Rx2 is selected<br>011: Rx3 is selected<br>100: Rx4 is selected<br>101: Rx5 is selected<br>110: Rx6 is selected<br>111: Rx7 is selected |
| BRP          | Baud Rate Prescaler<br>These bits are used to set the baud rate before going into standby mode<br>00000: Prescaler has 1<br>11111: Prescaler has 32   |
| CAN_SMPLR_EN | CAN SAMPLER Enable<br>This bit enables the CAN Sampler before going into standby or stop mode.<br>0: CAN Sampler is disabled<br>1: Can Sampler is enabled   |

### 14.3.2 CAN Sampler Sample Registers 0–11

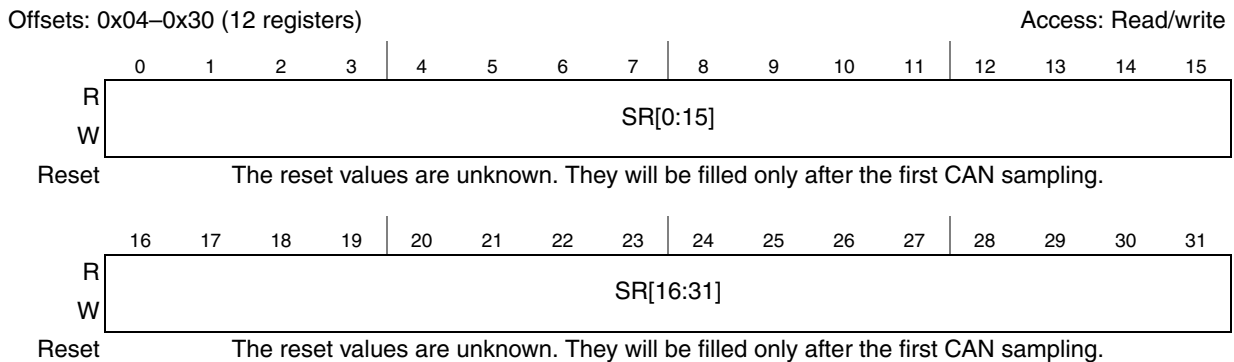


Figure 14-3. CAN Sampler Sample Registers 0–11

## 14.4 Functional description

As the CAN Sampler is driven by the 16 MHz fast internal RC oscillator to sample properly the CAN identifier, two modes are possible depending on both CAN baud rate and low power mode used:

- Immediate sampling on falling edge detection (first CAN frame): this mode is used when the fast internal RC oscillator 16 MHz is available in LP mode, for example, STOP or HALT.
- Sampling on next frame (second CAN frame): this mode is used when the fast internal RC oscillator 16 MHz is switched off in low power mode, such as STANDBY. Due to the start-up times of both the voltage regulator and the fast internal RC oscillator 16 MHz (~10 μs), the CAN sampler would miss the first bits of a CAN identifier sent at 500 kbit/s. Therefore the first identifier is ignored and the sampling is performed on the first falling edge of after interframe space.

The CAN sampler performs sampling on a user selected CAN Rx port among six Rx ports available, normally when the device is in standby or stop mode storing the samples in internal registers. The user is required to configure the baud rate to achieve eight samples per CAN nominal bit. It does not perform any sort of filtering on input samples. Thereafter the software must enable the sampler by setting CAN\_SMPLR\_EN bit in CR register. It then becomes the master controller for accessing the internal registers implemented for storing samples.

When enabled, the CAN sampler waits for a low pulse on the selected Rx line, taking it as a valid bit of the first CAN frame, and generates the RC wakeup request, which can be used to start the RC oscillator. Depending upon the mode, it stores the first 8 samples of the 48 bits on selected Rx line or skips the first frame and stores 8 bits for first 48 bits of second frame. In FF\_MODE, it samples the CAN Rx line on RC clock and stores the 8 samples of first 48 bits (384 samples). In SF\_MODE, it samples the Rx and waits for 11 consecutive dominant bits ( $11 \times 8$  samples), taking it as the end of first frame. It then waits for first low pulse on the Rx, taking it as a valid Start of Frame (SOF) of the second frame. The sampler takes 384 samples ( $48 \text{ bytes} \times 8$ ) using the RC clock (configuring 8 samples per nominal bit) of the second frame, including the SOF bit. These samples are stored in consecutive addresses of the ( $12 \times 32$ ) internal registers. RX\_COMPLETE bit is set to 1, indicating that sampling is complete.

Software should now process the sampled data by first becoming master for accessing samples internal registers by resetting CAN\_SMPLR\_EN bit. The sampler will need to be enabled again to start waiting for a new sampling routine.

#### 14.4.1 Enabling/disabling the CAN sampler

The CAN sampler is disabled on reset and the CPU is able to access the 12 registers used for storing samples. The CAN Sampler must be enabled before going into standby or stop mode by setting CAN\_SMPLR\_EN bit in the Control Register (CR) by writing 1 to this bit.

Any activity on selected Rx line, the sampler enables the 16 MHz fast internal RC oscillator. When CAN\_SMPLR\_EN is reset to 0, the sampler should at least receive three RC clock pulses to reset itself, after which the RC can be switched off.

When the software wishes to access the sample registers contents it must first reset the CAN\_SMPLR\_EN bit by writing a 0. Before accessing the register contents it must monitor Active\_CK bit for 0. When this bit is reset it can safely access the ( $12 \times 32$ ) sample registers. While shifting from normal to sample mode and vice versa, the sample register signals must be static and inactive to ensure the data is not corrupt.

#### 14.4.2 Selecting the Rx port

One Rx port can be selected per sampling routine. The port to be sampled is selected by CAN\_RX\_SEL.

**Table 14-3. Internal multiplexer correspondence**

| CAN_RX_SEL | Rx selected      |
|------------|------------------|
| 000        | CAN0RX on PB[1]  |
| 001        | CAN1RX on PC[11] |
| 010        | CAN2RX on PE[9]  |



Table 14-3. Internal multiplexer correspondence (continued)

| CAN_RX_SEL | Rx selected      |
|------------|------------------|
| 011        | CAN3RX on PE[9]  |
| 100        | CAN4RX on PC[11] |
| 101        | CAN5RX on PE[0]  |
| 110        | Rx6              |
| 111        | Rx7              |

### 14.4.3 Baud rate generation

Sampling is performed at a baud rate that is set by the software as a multiple of RC oscillator frequency of 62.5 ns (assuming RC is configured for high frequency mode, that is, 16 MHz). User must set the baud rate prescaler (BRP) such that 8 samples per bit are achieved.

Baud rate setting must be made by software before going into standby or stop mode. This is done by setting BRP bits 5:1 in Control register. The reset value of BRP is 00000 and can be set to maximum value of 11111, which gives a prescale value of BRP + 1 thus providing a BRP range of 1 to 32.

- Maximum bit rate supported for sampling is 2 Mbit/s using BRP as 1
- Minimum bit rate supported for sampling is 62.5 kbit/s using BRP as 32

For example, suppose the system is transmitting at 125 kbit/s. In this case, nominal bit period:

$$T = 1 / (125 \times 10^3) \text{ s} = 8 \times 10^{-3} \times 10^{-3} \text{ s} = 8 \mu\text{s} \quad \text{Eqn. 14-1}$$

To achieve 8 samples per bit


Sample period =  $8/8 \mu\text{s} = 1 \mu\text{s}$

BRP =  $1 \mu\text{s} / 62.5 \text{ ns} = 16$ . Thus in this case BRP = 01111

This page is intentionally left blank.

---

## —— Core platform modules ——



This page is intentionally left blank.

# Chapter 15

## e200z0h Core

### 15.1 Overview

The e200 processor family is a set of CPU cores that implement cost-efficient versions of the Power Architecture. e200 processors are designed for deeply embedded control applications that require low cost solutions rather than maximum performance.

The e200z0h processors integrate an integer execution unit, branch control unit, instruction fetch and load/store units, and a multi-ported register file capable of sustaining three read and two write operations per clock. Most integer instructions execute in a single clock cycle. Branch target prefetching is performed by the branch unit to allow single-cycle branches in some cases.

The e200z0h core is a single-issue, 32-bit Power Architecture technology VLE-only design with 32-bit general purpose registers (GPRs). All arithmetic instructions that execute in the core operate on data in the general purpose registers (GPRs).

Instead of the base Power Architecture technology support, the e200z0h core only implements the VLE (variable-length encoding) APU, providing improved code density.

### 15.2 Microarchitecture summary

The e200z0h processor utilizes a four stage pipeline for instruction execution. The Instruction Fetch (stage 1), Instruction Decode/Register file Read/Effective Address Calculation (stage 2), Execute/Memory Access (stage 3), and Register Writeback (stage 4) stages operate in an overlapped fashion, allowing single clock instruction execution for most instructions.

The integer execution unit consists of a 32-bit Arithmetic Unit (AU), a Logic Unit (LU), a 32-bit Barrel shifter (Shifter), a Mask-Insertion Unit (MIU), a Condition Register manipulation Unit (CRU), a Count-Leading-Zeros unit (CLZ), an 8x32 Hardware Multiplier array, result feed-forward hardware, and a hardware divider.

Arithmetic and logical operations are executed in a single cycle with the exception of the divide and multiply instructions. A Count-Leading-Zeros unit operates in a single clock cycle.

The Instruction Unit contains a PC incrementer and a dedicated Branch Address adder to minimize delays during change of flow operations. Sequential prefetching is performed to ensure a supply of instructions into the execution pipeline. Branch target prefetching from the BTB is performed to accelerate certain taken branches in the e200z0h. Prefetched instructions are placed into an instruction buffer with 4 entries in e200z0h, each capable of holding a single 32-bit instruction or a pair of 16-bit instructions.

Conditional branches that are not taken execute in a single clock. Branches with successful target prefetching have an effective execution time of one clock on e200z0h. All other taken branches have an execution time of two clocks.

Memory load and store operations are provided for byte, halfword, and word (32-bit) data with automatic zero or sign extension of byte and halfword load data as well as optional byte reversal of data. These

instructions can be pipelined to allow effective single cycle throughput. Load and store multiple word instructions allow low overhead context save and restore operations. The load/store unit contains a dedicated effective address adder to allow effective address generation to be optimized. Also, a load-to-use dependency does not incur any pipeline bubbles for most cases.

The Condition Register unit supports the condition register (CR) and condition register operations defined by the Power Architecture platform. The condition register consists of eight 4-bit fields that reflect the results of certain operations, such as move, integer and floating-point compare, arithmetic, and logical instructions, and provide a mechanism for testing and branching.

Vectored and autovectored interrupts are supported by the CPU. Vectored interrupt support is provided to allow multiple interrupt sources to have unique interrupt handlers invoked with no software overhead.

## 15.3 Block diagram

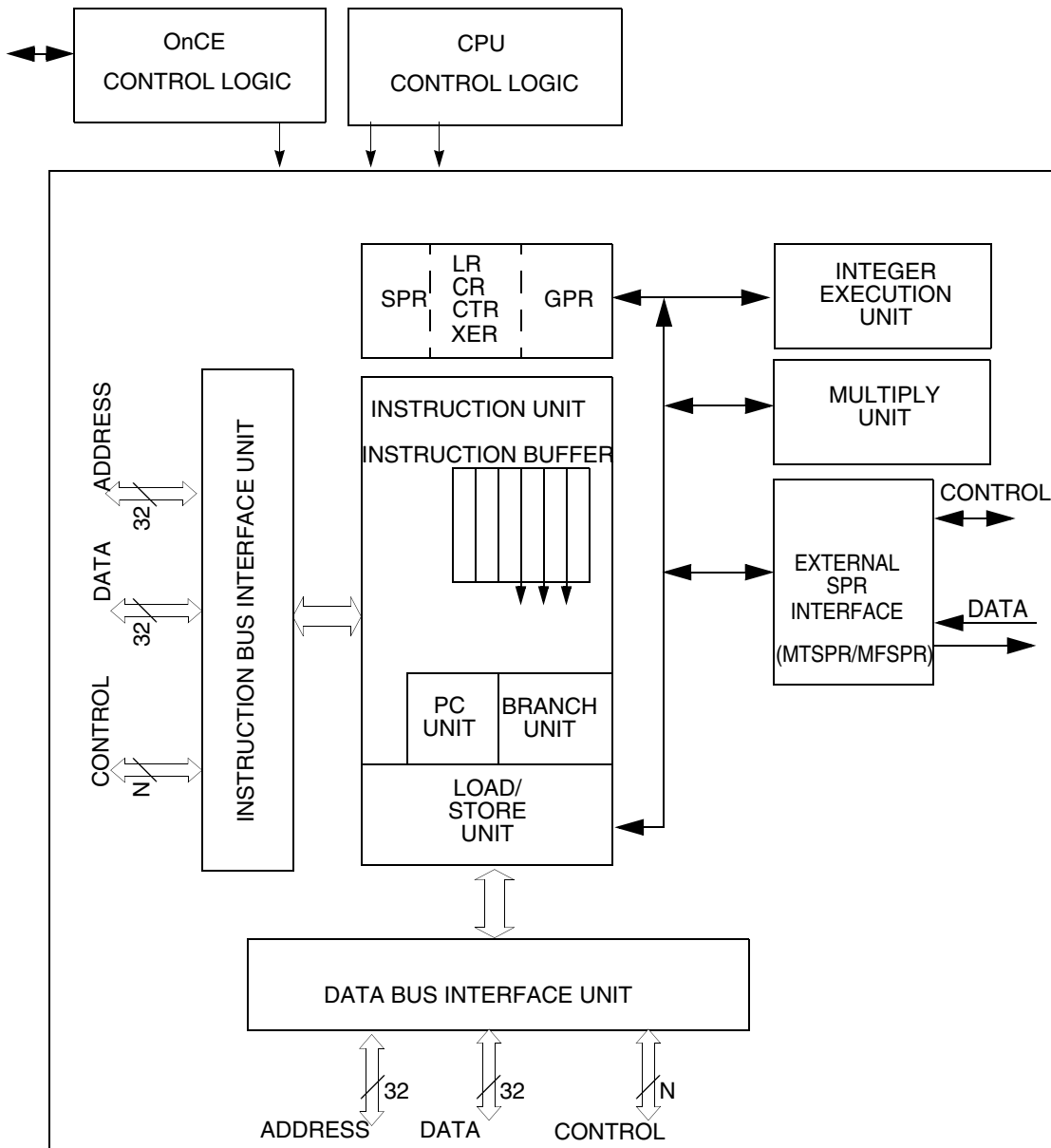


Figure 15-1. e200z0h block diagram

## 15.4 Features

The following is a list of some of the key features of the e200z0h core:

- 32-bit Power Architecture VLE-only programmer's model
- Single issue, 32-bit CPU
- Implements the VLE APU for reduced code footprint
- In-order execution and retirement

- Precise exception handling
- Branch processing unit
  - Dedicated branch address calculation adder
  - Branch acceleration using Branch Target Buffer
- Supports independent instruction and data accesses to different memory subsystems, such as SRAM and Flash memory via independent Instruction and Data bus interface units (BIUs) (e200z0h only).
- Load/store unit
  - 1 cycle load latency
  - Fully pipelined
  - Big-endian support only
  - Misaligned access support
  - Zero load-to-use pipeline bubbles for aligned transfers
- Power management
  - Low power design
  - Power saving modes: nap, sleep, and wait
  - Dynamic power management of execution units
- Testability
  - Synthesizeable, full MuxD scan design
  - ABIST/MBIST for optional memory arrays

### 15.4.1 Instruction unit features

The features of the e200 Instruction unit are:

- 32-bit instruction fetch path supports fetching of one 32-bit instruction per clock, or as many as two 16-bit VLE instructions per clock
- Instruction buffer with 4 entries in e200z0h, each holding a single 32-bit instruction, or a pair of 16-bit instructions
- Dedicated PC incrementer supporting instruction prefetches
- Branch unit with dedicated branch address adder supporting single cycle of execution of certain branches, two cycles for all others

### 15.4.2 Integer unit features

The e200 integer unit supports single cycle execution of most integer instructions:

- 32-bit AU for arithmetic and comparison operations
- 32-bit LU for logical operations
- 32-bit priority encoder for count leading zeros function
- 32-bit single cycle barrel shifter for shifts and rotates



- 32-bit mask unit for data masking and insertion
- Divider logic for signed and unsigned divide in 5 to 34 clocks with minimized execution timing
- $8 \times 32$  hardware multiplier array supports 1 to 4 cycle  $32 \times 32 \rightarrow 32$  multiply (early out)

### 15.4.3 Load/Store unit features

The e200 load/store unit supports load, store, and the load multiple / store multiple instructions:

- 32-bit effective address adder for data memory address calculations
- Pipelined operation supports throughput of one load or store operation per cycle
- 32-bit interface to memory (dedicated memory interface on e200z0h)

### 15.4.4 e200z0h system bus features

The features of the e200z0h system bus interface are as follows:

- Independent instruction and data buses
- AMBA<sup>1</sup> AHB<sup>2</sup> Lite Rev 2.0 specification with support for ARM v6 AMBA extensions
  - Exclusive access monitor
  - Byte lane strobes
  - Cache allocate support
- 32-bit address bus plus attributes and control on each bus
- 32-bit read data bus for instruction interface
- Separate uni-directional 32-bit read data bus and 32-bit write data bus for data interface
- Overlapped, in-order accesses

## 15.5 Core registers and programmer's model

This section describes the registers implemented in the e200z0h cores. It includes an overview of registers defined by the Power Architecture platform, highlighting differences in how these registers are implemented in the e200 core, and provides a detailed description of e200-specific registers. Full descriptions of the architecture-defined register set are provided in the Power Architecture specification.

The Power Architecture defines register-to-register operations for all computational instructions. Source data for these instructions are accessed from the on-chip registers or are provided as immediate values embedded in the opcode. The three-register instruction format allows specification of a target register distinct from the two source registers, thus preserving the original data for use by other instructions. Data is transferred between memory and registers with explicit load and store instructions only.

Figure 15-2, and Figure 15-1 show the e200 register set, including the registers that are accessible while in supervisor mode, and the registers that are accessible in user mode. The number to the right of the special-purpose registers (SPRs) is the decimal number used in the instruction syntax to access the register (for example, the integer exception register (XER) is SPR 1).

1. Advanced Microcontroller Bus Architecture

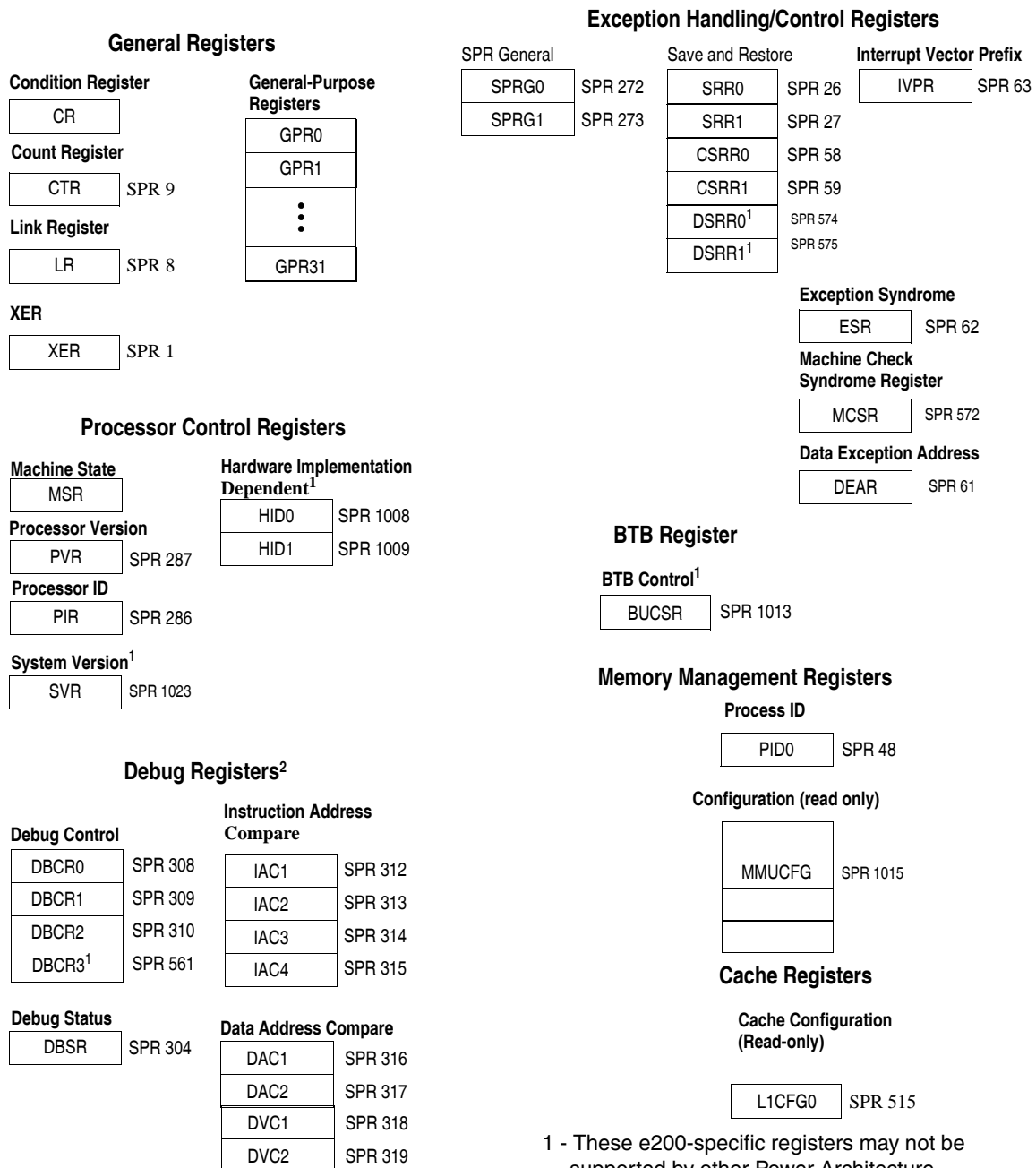
2. Advanced High Performance Bus

### NOTE

e200z0h is a 32-bit implementation of the Power Architecture specification. In this document, register bits are sometimes numbered from bit 0 (Most Significant Bit) to 31 (Least Significant Bit), rather than the Book E numbering scheme of 32:63, thus register bit numbers for some registers in Book E are 32 higher.

Where appropriate, the Book E defined bit numbers are shown in parentheses.

## SUPERVISOR Mode Program Model SPRs



1 - These e200-specific registers may not be supported by other Power Architecture processors.

2 - Optional registers defined by the Power Architecture technology

3 - Read-only registers

Figure 15-2. e200z0 SUPERVISOR Mode Program Model SPRs

This page is intentionally left blank.

# Chapter 16

## Enhanced Direct Memory Access (eDMA)

### 16.1 Device-specific features

- 16 programmable channels to support independent 8-, 16-, or 32-bit single value or block transfers
- Support of variable sized queues and circular queues
- Source and destination address registers independently configured to post-incrementor remain constant
- Each transfer initiated by peripheral, CPU, periodic timer interrupt or eDMA channel request
- Peripheral eDMA request sources possible from DSPI, I<sup>2</sup>C, 10-bit ADC, 12-bit ADC, LINFlexD, and eMIOS
- Each eDMA channel able to optionally send interrupt request to CPU on completion of single value or block transfer
- DMA transfers possible between system memories and all accessible memory mapped locations including peripheral and registers
- Programmable eDMA channel mux allows assignment of any eDMA source to any available eDMA channel with total of as many as 32 request sources
- DMA supports the following functionality:
  - Scatter Gather
  - Channel Linking
  - Inner Loop Offset
  - Arbitration
    - Fixed Group, fixed channel
    - Round Robin Group, fixed channel
    - Round Robin Group, Round Robin Channel
    - Fixed Group, Round Robin Channel
  - Channel preemption
  - Cancel channel transfer
- Interrupts – The eDMA has a single interrupt request for each implemented channel and a combined eDMA Error interrupt to flag transfer errors to the system. Each channel eDMA interrupt can be enabled or disabled and provides notification of a completed transfer. Refer to the Interrupt Vector in [Chapter 18, Interrupt Controller \(INTC\)](#), for the allocation of these interrupts.

### 16.2 Introduction

The enhanced direct memory access controller (eDMA) is a second-generation platform block capable of performing complex data movements through 16 programmable channels, with minimal intervention from the host processor. The hardware microarchitecture includes a DMA engine that performs source and destination address calculations, and the actual data movement operations, along with an SRAM-based

memory containing the transfer control descriptors (TCD) for the channels. This implementation minimizes the overall block size.

Figure 16-1 is a block diagram of the eDMA module.

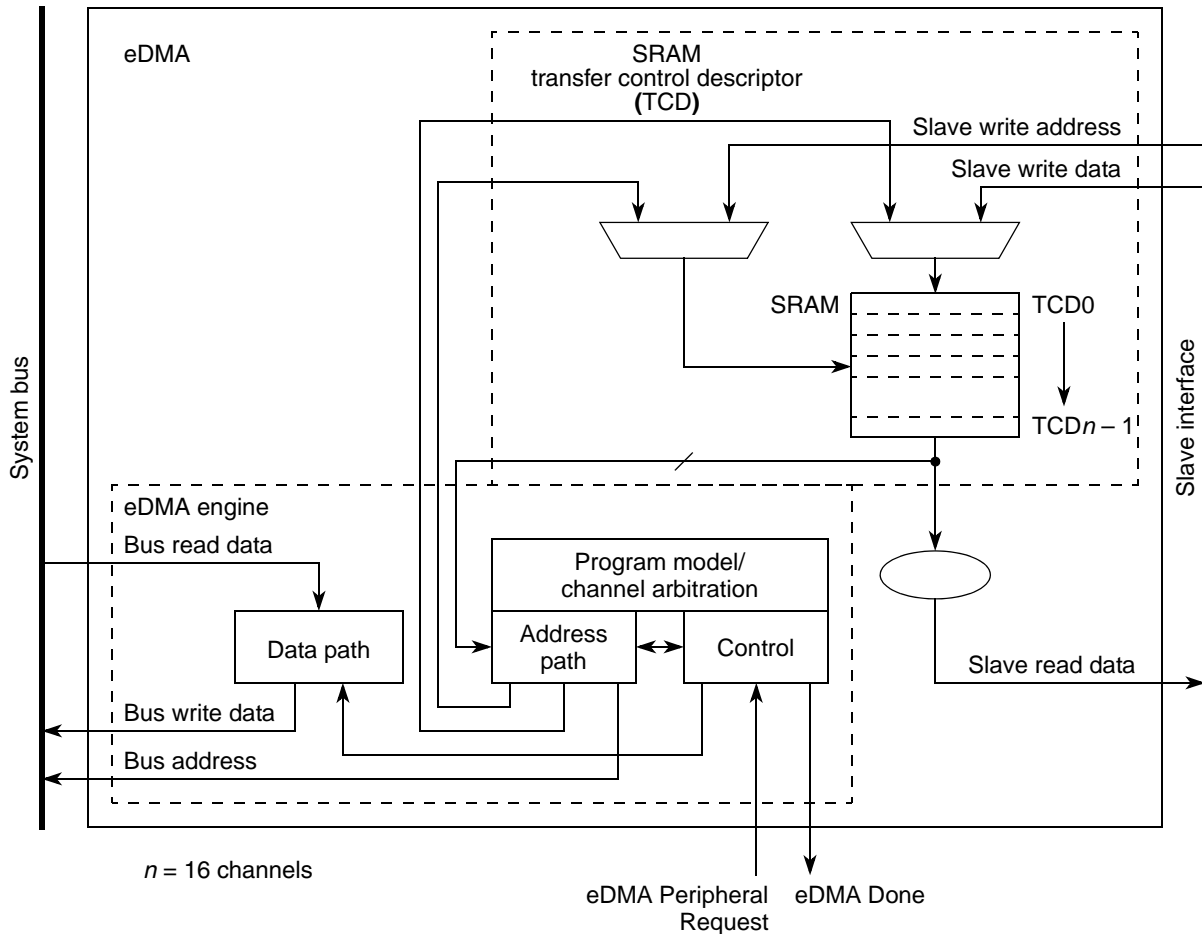


Figure 16-1. eDMA block diagram

## 16.2.1 Features

The eDMA module supports the following features:

- All data movement via dual-address transfers: read from source, write to destination
  - Programmable source, destination addresses, transfer size, plus support for enhanced addressing modes
- Transfer control descriptor organized to support two-deep, nested transfer operations
  - An *inner* data transfer loop defined by a minor byte transfer count
  - An *outer* data transfer loop defined by a major iteration count
- Channel service request via one of three methods:

- Explicit software initiation
- Initiation via a channel-to-channel linking mechanism for continuous transfers
  - Independent channel linking at end of minor loop and/or major loop
- Peripheral-paced hardware requests (one per channel)
- For all three methods, one service request per execution of the minor loop is required
- Support for fixed-priority and round-robin channel arbitration
- Channel completion reported via optional interrupt requests
  - One interrupt per channel, optionally asserted at completion of major iteration count
  - Error terminations are optionally enabled per channel, and logically summed together to form a small number of error interrupt outputs
- Support for scatter/gather eDMA processing
- Support for complex data structures
- Support to cancel transfers via software or hardware

## 16.3 Memory map and register definition

### 16.3.1 Memory map

The eDMA memory map is shown in [Table 16-1](#). The eDMA base address is 0xFFF4\_4000. The address of each register is given as an offset to the eDMA base address. Registers are listed in address order, identified by complete name and mnemonic, and list the type of accesses allowed.

The eDMA's programming model is partitioned into two regions—the first region defines a number of registers providing control functions; the second region corresponds to the local transfer control descriptor memory.

**Table 16-1. eDMA memory map**

| Base address: 0xFFF4_4000 |  |                             |
|---------------------------|--|-----------------------------|
| Address offset            | Register   | Location                    |
| 0x0000                    | EDMA_CR — eDMA control register  | <a href="#">on page 257</a> |
| 0x0004                    | EDMA_ESR — eDMA error status register                                  | <a href="#">on page 259</a> |
| 0x0008                    | Reserved   |                             |
| 0x000C                    | EDMA_ERQRL — eDMA enable request low register (channels 15–00)         | <a href="#">on page 261</a> |
| 0x0010                    | Reserved   |                             |
| 0x0014                    | EDMA_EEIRL — eDMA enable error interrupt low register (channels 15–00) | <a href="#">on page 262</a> |
| 0x0018                    | EDMA_SERQR — eDMA set enable request register                          | <a href="#">on page 263</a> |
| 0x0019                    | EDMA_CERQR — eDMA clear enable request register                        | <a href="#">on page 264</a> |
| 0x001A                    | EDMA_SEEIR — eDMA set enable error interrupt register                  | <a href="#">on page 264</a> |
| 0x001B                    | EDMA_CEEIR — eDMA clear enable error interrupt register                | <a href="#">on page 265</a> |

Table 16-1. eDMA memory map (continued)

| Base address: 0xFFF4_4000 |  |             |
|---------------------------|--|-------------|
| Address offset            | Register   | Location    |
| 0x001C                    | EDMA_CIRQR — eDMA clear interrupt request register | on page 265 |
| 0x001D                    | EDMA_CER — eDMA clear error register               | on page 266 |
| 0x001E                    | EDMA_SSBP — eDMA set start bit register            | on page 266 |
| 0x001F                    | EDMA_CDSBR — eDMA clear done status bit register   | on page 267 |
| 0x0020                    | Reserved   |             |
| 0x0024                    | EDMA_IRQRL — eDMA interrupt request low register   | on page 267 |
| 0x0028                    | Reserved   |             |
| 0x002C                    | EDMA_ERL — eDMA error low register                 | on page 268 |
| 0x0030                    | Reserved   |             |
| 0x0034                    | EDMA_HRSL — eDMA hardware request status register  | on page 269 |
| 0x0038 – 0x01FC           | Reserved   |             |
| 0x0100                    | EDMA_CPR0 — eDMA channel 0 priority register       | on page 269 |
| 0x0101                    | EDMA_CPR1 — eDMA channel 1 priority register       | on page 269 |
| 0x0102                    | EDMA_CPR2 — eDMA channel 2 priority register       | on page 269 |
| 0x0103                    | EDMA_CPR3 — eDMA channel 3 priority register       | on page 269 |
| 0x0104                    | EDMA_CPR4 — eDMA channel 4 priority register       | on page 269 |
| 0x0105                    | EDMA_CPR5 — eDMA channel 5 priority register       | on page 269 |
| 0x0106                    | EDMA_CPR6 — eDMA channel 6 priority register       | on page 269 |
| 0x0107                    | EDMA_CPR7 — eDMA channel 7 priority register       | on page 269 |
| 0x0108                    | EDMA_CPR8 — eDMA channel 8 priority register       | on page 269 |
| 0x0109                    | EDMA_CPR9 — eDMA channel 9 priority register       | on page 269 |
| 0x010A                    | EDMA_CPR10 — eDMA channel 10 priority register     | on page 269 |
| 0x010B                    | EDMA_CPR11 — eDMA channel 11 priority register     | on page 269 |
| 0x010C                    | EDMA_CPR12 — eDMA channel 12 priority register     | on page 269 |
| 0x010D                    | EDMA_CPR13 — eDMA channel 13 priority register     | on page 269 |
| 0x010E                    | EDMA_CPR14 — eDMA channel 14 priority register     | on page 269 |
| 0x010F                    | EDMA_CPR15 — eDMA channel 15 priority register     | on page 269 |
| 0x0110                    | Reserved   |             |
| 0x1000                    | TCD00 — eDMA transfer control descriptor 00        | on page 271 |
| 0x1020                    | TCD01 — eDMA transfer control descriptor 01        | on page 271 |
| 0x1040                    | TCD02 — eDMA transfer control descriptor 02        | on page 271 |



Table 16-1. eDMA memory map (continued)

| Base address: 0xFFF4_4000 |   |                             |
|---------------------------|---|-----------------------------|
| Address offset            | Register                                    | Location                    |
| 0x1060                    | TCD03 — eDMA transfer control descriptor 03 | <a href="#">on page 271</a> |
| 0x1080                    | TCD04 — eDMA transfer control descriptor 04 | <a href="#">on page 271</a> |
| 0x10A0                    | TCD05 — eDMA transfer control descriptor 05 | <a href="#">on page 271</a> |
| 0x10C0                    | TCD06 — eDMA transfer control descriptor 06 | <a href="#">on page 271</a> |
| 0x10E0                    | TCD07 — eDMA transfer control descriptor 07 | <a href="#">on page 271</a> |
| 0x1100                    | TCD08 — eDMA transfer control descriptor 08 | <a href="#">on page 271</a> |
| 0x1120                    | TCD09 — eDMA transfer control descriptor 09 | <a href="#">on page 271</a> |
| 0x1140                    | TCD10 — eDMA transfer control descriptor 10 | <a href="#">on page 271</a> |
| 0x1160                    | TCD11 — eDMA transfer control descriptor 11 | <a href="#">on page 271</a> |
| 0x1180                    | TCD12 — eDMA transfer control descriptor 12 | <a href="#">on page 271</a> |
| 0x11A0                    | TCD13 — eDMA transfer control descriptor 13 | <a href="#">on page 271</a> |
| 0x11C0                    | TCD14 — eDMA transfer control descriptor 14 | <a href="#">on page 271</a> |
| 0x11E0                    | TCD15 — eDMA transfer control descriptor 15 | <a href="#">on page 271</a> |
| 0x1200                    | Reserved                                    |                             |

## 16.3.2 Register descriptions

### 16.3.2.1 DMA Control Register (EDMA\_CR)

The 32-bit EDMA\_CR defines the basic operating configuration of the eDMA.

Arbitration among the channels can be configured to use a fixed priority or a round robin. In fixed-priority arbitration, the highest priority channel requesting service is selected to execute. The priorities are assigned by the channel priority registers (see [Section 16.3.2.16, DMA Channel n Priority \(EDMA\\_CPRn\)](#)). In round-robin arbitration mode, the channel priorities are ignored and the channels are cycled through, from channel 15 down to channel 0, without regard to priority.

See [Figure 16-2](#) and [Table 16-2](#) for the EDMA\_CR definition.

Offset: 0x0000

Access: Read/write

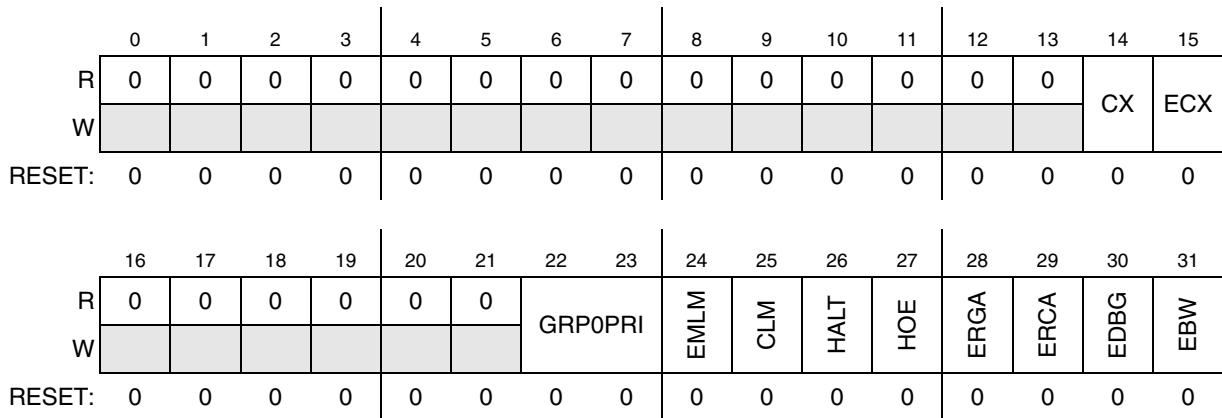


Figure 16-2. DMA Control Register (EDMA\_CR)

Table 16-2. EDMA\_CR field descriptions

| Field   | Description   |
|---------|---|
| CX      | Cancel Transfer<br>0 Normal operation.<br>1 Cancel the remaining data transfer. Stop the executing channel and force the minor loop to be finished. The cancel takes effect after the last write of the current read/write sequence. The CXFR bit clears itself after the cancel has been honored. This cancel retires the channel normally as if the minor loop was completed.   |
| ECX     | Error Cancel Transfer<br>0 Normal operation.<br>1 Cancel the remaining data transfer in the same fashion as the CX cancel transfer. Stop the executing channel and force the minor loop to be finished. The cancel takes effect after the last write of the current read/write sequence. The ECX bit clears itself after the cancel has been honored. In addition to cancelling the transfer, the ECX treats the cancel as an error condition; thus updating the EDMA_ESR register and generating an optional error interrupt (see <a href="#">Section 16.3.2.2, DMA Error Status (EDMA_ESR)</a> ). |
| GRP0PRI | Channel Group 0 Priority<br>Group 0 priority level when fixed priority group arbitration is enabled.  |
| EMLM    | Enable Minor Loop Mapping<br>0 Minor loop mapping disabled. TCDn.word2 is defined as a 32-bit nbytes field.<br>1 Minor loop mapping enabled. When set, TCDn.word2 is redefined to include individual enable fields, an offset field and the nbytes field. The individual enable fields allow the minor loop offset to be applied to the source address, the destination address, or both. The nbytes field is reduced when either offset is enabled.  |
| CLM     | Continuous Link Mode<br>0 A minor loop channel link made to itself will go through channel arbitration before being activated again.<br>1 A minor loop channel link made to itself will not go through channel arbitration before being activated again. Upon minor loop completion the channel will active again if that channel has a minor loop channel link enabled and the link channel is itself. This effectively applies the minor loop offsets and restarts the next minor loop.   |
| HALT    | Halt DMA Operations<br>0 Normal operation.<br>1 Stall the start of any new channels. Executing channels are allowed to complete. Channel execution will resume when the HALT bit is cleared.  |

Table 16-2. EDMA\_CR field descriptions (continued)

| Field | Description  |
|-------|--|
| HOE   | Halt On Error<br>0 Normal operation.<br>1 Any error will cause the HALT bit to be set. Subsequently, all service requests will be ignored until the HALT bit is cleared.   |
| ERGA  | Enable Round Robin Group Arbitration<br>0 Fixed priority arbitration is used for selection among the groups.<br>1 Round robin arbitration is used for selection among the groups.  |
| ERCA  | Enable Round Robin Channel Arbitration<br>0 Fixed priority arbitration is used for channel selection within each group.<br>1 Round robin arbitration is used for channel selection within each group.  |
| EDBG  | Enable Debug<br>0 The assertion of the device debug mode is ignored.<br>1 The assertion of the device debug mode causes the eDMA to stall the start of a new channel. Executing channels are allowed to complete. Channel execution will resume when either the device comes out of debug mode or the EDBG bit is cleared. |
| EBW   | 0 The bufferable write signal (hprot[2]) is not asserted during AMBA AHB writes.<br>1 The bufferable write signal (hprot[2]) is asserted on all AMBA AHB writes except for the last write sequence.  |

### 16.3.2.2 DMA Error Status (EDMA\_ESR)

The EDMA\_ESR provides information about the last recorded channel error. Channel errors can be caused by a configuration error (an illegal setting in the transfer control descriptor or an illegal priority register setting in fixed-arbitration mode) or an error termination to a bus master read or write cycle.

A configuration error is caused when the starting source or destination address, source or destination offsets, minor loop byte count, and the transfer size represent an inconsistent state. The addresses and offsets must be aligned on 0-modulo-transfer\_size boundaries, and the minor loop byte count must be a multiple of the source and destination transfer sizes. All source reads and destination writes must be configured to the natural boundary of the programmed transfer size respectively.

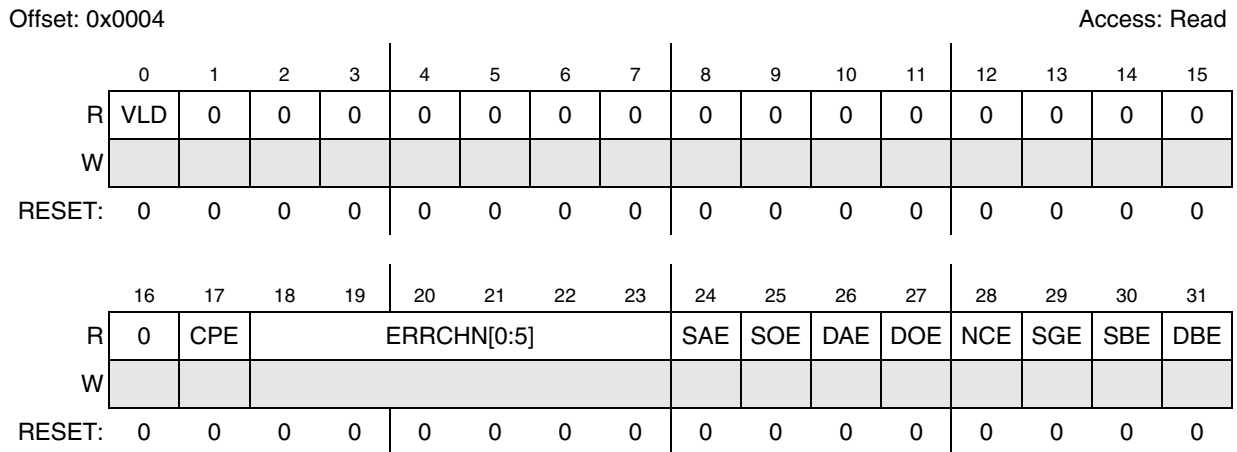
In fixed-arbitration mode, a configuration error is generated when any two channel priority levels are equal and any channel is activated. The ERRCHN field is undefined for this type of error. All channel priority levels must be unique before any service requests are made.

If a scatter-gather operation is enabled on channel completion, a configuration error is reported if the scatter-gather address (DLAST\_SGA) is not aligned on a 32-byte boundary. If minor loop channel linking is enabled on channel completion, a configuration error is reported when the link is attempted if the TCD.CITER.E\_LINK bit is not equal to the TCD.BITER.E\_LINK bit. All configuration error conditions except scatter-gather and minor loop link error are reported as the channel is activated and assert an error interrupt request if enabled. When properly enabled, a scatter-gather configuration error is reported when the scatter-gather operation begins at major loop completion. A minor loop channel link configuration error is reported when the link operation is serviced at minor loop completion.

If a system bus read or write is terminated with an error, the data transfer is immediately stopped and the appropriate bus error flag is set. In this case, the state of the channel's transfer control descriptor is updated by the DMA engine with the current source address, destination address, and minor loop byte count at the point of the fault. If a bus error occurs on the last read prior to beginning the write sequence, the write will

execute using the data captured during the bus error. If a bus error occurs on the last write prior to switching to the next read sequence, the read sequence will execute before the channel is terminated due to the destination bus error.

The occurrence of any type of error causes the DMA engine to stop the active channel and the appropriate channel bit in the eDMA error register to be asserted. At the same time, the details of the error condition are loaded into the EDMA\_ESR. The major loop complete indicators, setting the transfer control descriptor DONE flag and the possible assertion of an interrupt request, are not affected when an error is detected. After the error status has been updated, the DMA engine continues to operate by servicing the next appropriate channel. A channel that experiences an error condition is not automatically disabled. If a channel is terminated by an error and then issues another service request before the error is fixed, that channel will execute and terminate with the same error condition.



**Figure 16-3. DMA Error Status (EDMA\_ESR) Register**

**Table 16-3. EDMA\_ESR field descriptions**

| Field       | Description   |
|-------------|---|
| VLD         | Logical OR of all EDMA_ERL status bits.<br>0 No EDMA_ERL bits are set.<br>1 At least one EDMA_ERL bit is set indicating a valid error exists that has not been cleared.                                       |
| CPE         | Channel Priority Error<br>0 No channel priority error.<br>1 The last recorded error was a configuration error in the channel priorities within a group. All channel priorities within a group are not unique. |
| ERRCHN[0:5] | Error Channel Number or Cancelled Channel Number<br>The channel number of the last recorded error (excluding GPE and CPE errors) or last recorded transfer that was error cancelled.                          |
| SAE         | Source Address Error<br>0 No source address configuration error.<br>1 The last recorded error was a configuration error detected in the TCD.saddr field. TCD.saddr is inconsistent with TCD.ssize.            |
| SOE         | Source Offset Error<br>0 No source offset configuration error.<br>1 The last recorded error was a configuration error detected in the TCD.soff field. TCD.soff is inconsistent with TCD.ssize.                |

Table 16-3. EDMA\_ESR field descriptions (continued)

| Field | Description   |
|-------|---|
| DAE   | Destination Address Error<br>0 No destination address configuration error.<br>1 The last recorded error was a configuration error detected in the TCD.daddr field. TCD.daddr is inconsistent with TCD.dsize.  |
| DOE   | Destination Offset Error<br>0 No destination offset configuration error.<br>1 The last recorded error was a configuration error detected in the TCD.doff field. TCD.doff is inconsistent with TCD.dsize.  |
| NCE   | Nbytes/Citer Configuration Error<br>0 No nbytes/citer configuration error.<br>1 The last recorded error was a configuration error detected in the TCD.nbytes or TCD.citer fields. TCD.nbytes is not a multiple of TCD.ssize and TCD.dsize, or TCD.citer is equal to zero, or TCD.citer.e_link is not equal to TCD.biter.e_link.                 |
| SGE   | Scatter/Gather Configuration Error<br>0 No scatter/gather configuration error.<br>1 The last recorded error was a configuration error detected in the TCD.dlast_sga field. This field is checked at the beginning of a scatter/gather operation after major loop completion if TCD.e_sg is enabled. TCD.dlast_sga is not on a 32 byte boundary. |
| SBE   | Source Bus Error<br>0 No source bus error.<br>1 The last recorded error was a bus error on a source read.   |
| DBE   | Destination Bus Error<br>0 No destination bus error.<br>1 The last recorded error was a bus error on a destination write.   |

### 16.3.2.3 DMA Enable Request (EDMA\_ERQRL)

The EDMA\_ERQRL provides a bit map for the 16 channels to enable the request signal for each channel. EDMA\_ERQRL maps to channels 15–0.

The state of any given channel enable is directly affected by writes to this register; the state is also affected by writes to the EDMA\_SERQR, and EDMA\_CERQR registers. The EDMA\_CERQR and EDMA\_SERQR registers are provided so the request enable for a single channel can be modified without performing a read-modify-write sequence to the EDMA\_ERQRL register.

Both the eDMA request input signal and this enable request flag must be asserted before a channel's hardware service request is accepted. The state of the eDMA enable request flag does not affect a channel service request made through software or a linked channel request.

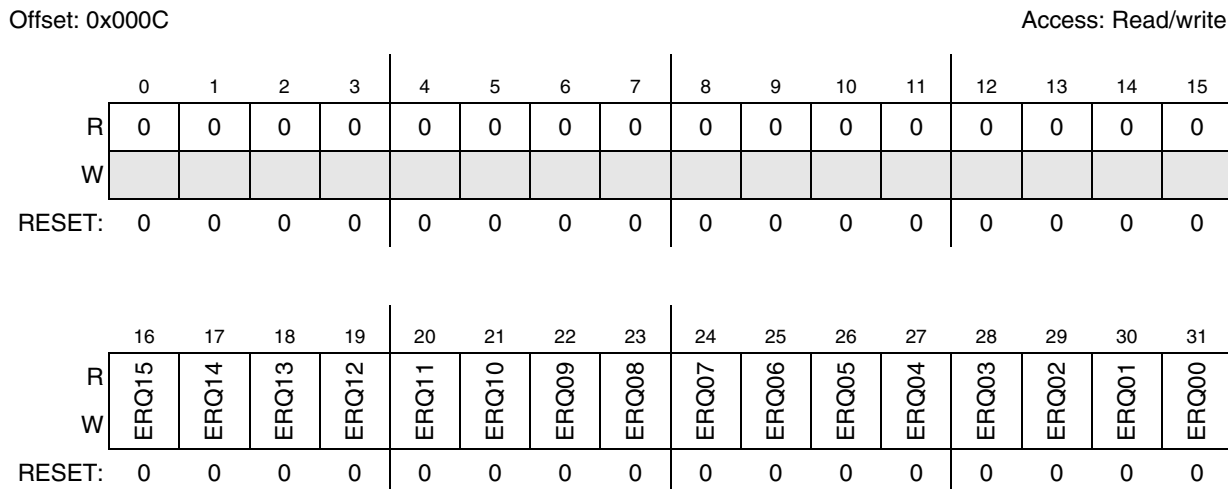


Figure 16-4. DMA Enable Request (EDMA\_ERQRL) Registers

Table 16-4. EDMA\_ERQRL field descriptions

| Field | Description  |
|-------|--|
| ERQn  | Enable eDMA Request n<br>0 The eDMA request signal for channel n is disabled.<br>1 The eDMA request signal for channel n is enabled. |

As a given channel completes the processing of its major iteration count, there is a flag in the transfer control descriptor that may affect the ending state of the EDMA\_ERQRL bit for that channel. If the TCD.d\_req bit is set, then the corresponding EDMA\_ERQRL bit is cleared, disabling the eDMA request; else if the d\_req bit is cleared, the state of the EDMA\_ERQRL bit is unaffected.

### 16.3.2.4 DMA Enable Error Interrupt (EDMA\_EEIRL)

The EDMA\_EEIRL provides a bit map for the 16 channels to enable the error interrupt signal for each channel. EDMA\_EEIRL maps to channels 15–0.

The state of any given channel’s error interrupt enable is directly affected by writes to these registers; it is also affected by writes to the EDMA\_SEEIR and EDMA\_CEEIR registers. The EDMA\_SEEIR and EDMA\_CEEIR registers are provided so that the error interrupt enable for a single channel can be modified without the performing a read-modify-write sequence to the EDMA\_EEIRL register.

Both the eDMA error indicator and this error interrupt enable flag must be asserted before an error interrupt request for a given channel is asserted. See Figure 16-5 and Table 16-5 for the EDMA\_EEIRL definition.

Offset: 0x0014

Access: Read/write

|        |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
|--------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
|        | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| R      | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W      |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| RESET: | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|        |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |
|--------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
|        | 16    | 17    | 18    | 19    | 20    | 21    | 22    | 23    | 24    | 25    | 26    | 27    | 28    | 29    | 30    | 31    |
| R      | EEI15 | EEI14 | EEI13 | EEI12 | EEI11 | EEI10 | EEI09 | EEI08 | EEI07 | EEI06 | EEI05 | EEI04 | EEI03 | EEI02 | EEI01 | EEI00 |
| W      |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |
| RESET: | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     |

Figure 16-5. DMA Enable Error Interrupt (EDMA\_EEIRL) Register

Table 16-5. EDMA\_EEIRL field descriptions

| Field | Description  |
|-------|--|
| EEIn  | Enable Error Interrupt n<br>0 The error signal for channel n does not generate an error interrupt.<br>1 The assertion of the error signal for channel n generate an error interrupt request. |

### 16.3.2.5 DMA Set Enable Request (EDMA\_SERQR)

The EDMA\_SERQR provides a simple memory-mapped mechanism to set a given bit in the EDMA\_ERQRL to enable the eDMA request for a given channel. The data value on a register write causes the corresponding bit in the EDMA\_ERQRL to be set. Setting bit 1 (SERQ[0]) provides a global set function, forcing the entire contents of EDMA\_ERQRL to be asserted. Reads of this register return all zeroes.

Offset: 0x0018

Access: Write

|        |   |      |   |   |   |   |   |   |
|--------|---|------|---|---|---|---|---|---|
|        | 0 | 1    | 2 | 3 | 4 | 5 | 6 | 7 |
| R      | 0 | 0    | 0 | 0 | 0 | 0 | 0 | 0 |
| W      |   | SERQ |   |   |   |   |   |   |
| RESET: | 0 | 0    | 0 | 0 | 0 | 0 | 0 | 0 |

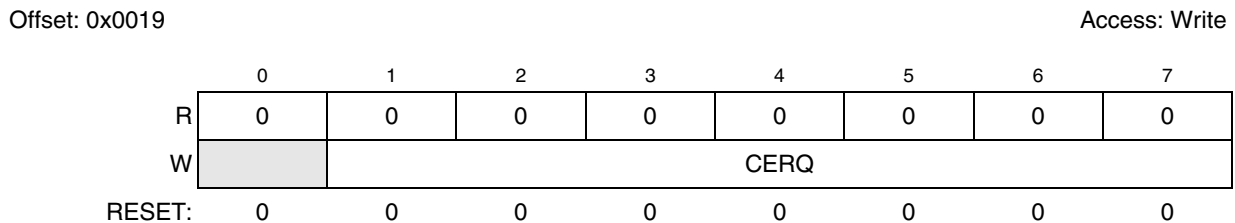
Figure 16-6. DMA Set Enable Request (EDMA\_SERQR) Register

Table 16-6. EDMA\_SERQR field descriptions

| Field | Description   |
|-------|---|
| SERQ  | Set Enable Request<br>0–15 Set the corresponding bit in EDMA_ERQRL<br>64–127 Set all bits in EDMA_ERQRL |

### 16.3.2.6 DMA Clear Enable Request (EDMA\_CERQR)

The EDMA\_CERQR provides a simple memory-mapped mechanism to clear a given bit in the EDMA\_ERQRL to disable the eDMA request for a given channel. The data value on a register write causes the corresponding bit in the EDMA\_ERQRL to be cleared. Setting bit 1 (CERQ[0]) provides a global clear function, forcing the entire contents of the EDMA\_ERQRL to be zeroed, disabling all eDMA request inputs. Reads of this register return all zeroes. See [Figure 16-7](#) and [Table 16-7](#) for the EDMA\_CERQR definition.



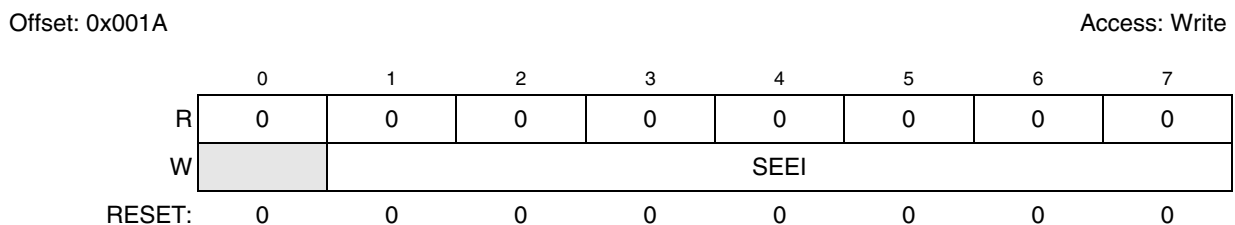
**Figure 16-7. DMA Clear Enable Request (EDMA\_CERQR) Register**

**Table 16-7. EDMA\_CERQR field descriptions**

| Field | Description   |
|-------|---|
| CERQ  | Clear Enable Request<br>0–63 Clear corresponding bit in EDMA_ERQRL<br>64–127 Clear all bits in EDMA_ERQRL |

### 16.3.2.7 DMA Set Enable Error Interrupt (EDMA\_SEEIR)

The EDMA\_SEEIR provides a memory-mapped mechanism to set a given bit in the EDMA\_EEIRL to enable the error interrupt for a given channel. The data value on a register write causes the corresponding bit in the EDMA\_EEIRL to be set. Setting bit 1 (SEEI[0]) provides a global set function, forcing the entire contents of EDMA\_EEIRL to be asserted. Reads of this register return all zeroes. See [Figure 16-8](#) and [Table 16-8](#) for the EDMA\_SEEIR definition.



**Figure 16-8. DMA Set Enable Error Interrupt (EDMA\_SEEIR) Register**

**Table 16-8. EDMA\_SEEIR field descriptions**

| Name | Description   |
|------|---|
| SEEI | Set Enable Error Interrupt<br>0–63 Set the corresponding bit in EDMA_EEIRL<br>64–127 Set all bits in EDMA_EEIRL |



### 16.3.2.8 DMA Clear Enable Error Interrupt (EDMA\_CEEIR)

The EDMA\_CEEIR provides a memory-mapped mechanism to clear a given bit in the EDMA\_EEIRL to disable the error interrupt for a given channel. The data value on a register write causes the corresponding bit in the EDMA\_EEIRL to be cleared. Setting bit 1 (CEEI[0]) provides a global clear function, forcing the entire contents of the EDMA\_EEIRL to be zeroed, disabling error interrupts for all channels. Reads of this register returns all zeroes. See [Figure 16-9](#) and [Table 16-9](#) for the EDMA\_CEEIR definition.

Offset: 0x001B

Access: Write

|        |           |   |   |   |   |   |   |   |
|--------|-----------|---|---|---|---|---|---|---|
|        | 0         | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| R      | 0         | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| W      | CEEI[0:6] |   |   |   |   |   |   |   |
| RESET: | 0         | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Figure 16-9. DMA Clear Enable Error Interrupt (EDMA\_CEEIR) Register

Table 16-9. EDMA\_CEEIR field descriptions

| Field | Description   |
|-------|---|
| CEEI  | Clear Enable Error Interrupt<br>0–63 Clear corresponding bit in EDMA_EEIRL<br>64–127 Clear all bits in EDMA_EEIRL |

### 16.3.2.9 DMA Clear Interrupt Request (EDMA\_CIRQR)

The EDMA\_CIRQR provides a memory-mapped mechanism to clear a given bit in the EDMA\_IRQRL to disable the interrupt request for a given channel. The given value on a register write causes the corresponding bit in the EDMA\_IRQRL to be cleared. Setting bit 1 (CINT[0]) provides a global clear function, forcing the entire contents of the EDMA\_IRQRL to be zeroed, disabling all eDMA interrupt requests. Reads of this register return all zeroes. See [Figure 16-10](#) and [Table 16-10](#) for the EDMA\_CIRQR definition.

Offset: 0x001C

Access: Write

|        |      |   |   |   |   |   |   |   |
|--------|------|---|---|---|---|---|---|---|
|        | 0    | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| R      | 0    | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| W      | CINT |   |   |   |   |   |   |   |
| RESET: | 0    | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

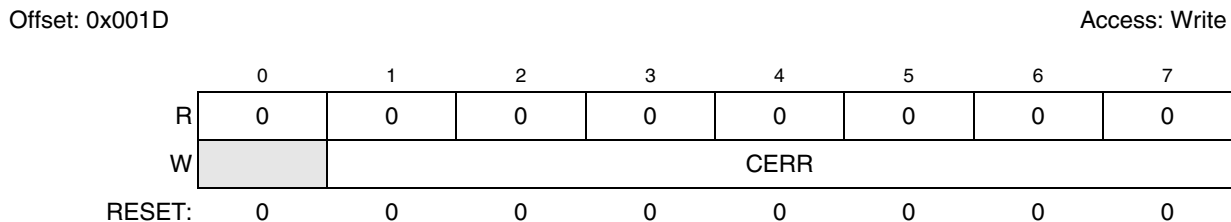
Figure 16-10. DMA Clear Interrupt Request (EDMA\_CIRQR) Fields

Table 16-10. EDMA\_CIRQR field descriptions

| Field     | Description  |
|-----------|--|
| CINT[0:6] | Clear Interrupt Request<br>0–63 Clear the corresponding bit in EDMA_IRQRL<br>64–127 Clear all bits in EDMA_IRQRL |

### 16.3.2.10 DMA Clear Error (EDMA\_CER)

The EDMA\_CER provides a memory-mapped mechanism to clear a given bit in the EDMA\_ERL to disable the error condition flag for a given channel. The given value on a register write causes the corresponding bit in the EDMA\_ERL to be cleared. Setting bit 1 (CERR[0]) provides a global clear function, forcing the entire contents of the EDMA\_ERL to be zeroed, clearing all channel error indicators. Reads of this register return all zeroes. See [Figure 16-11](#) and [Table 16-11](#) for the EDMA\_CER definition.



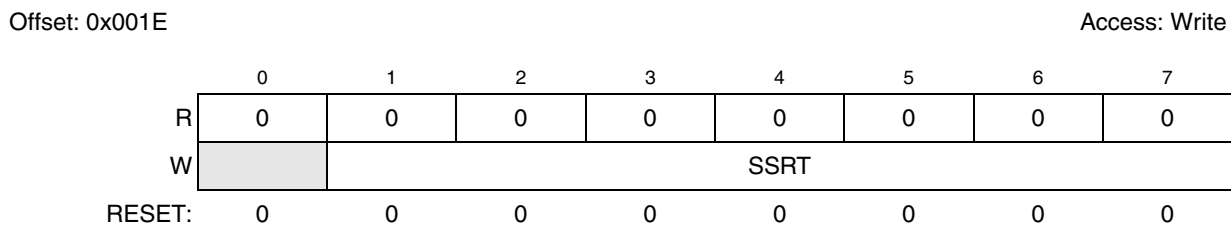
**Figure 16-11. DMA Clear Error (EDMA\_CER) Register**

**Table 16-11. EDMA\_CER field descriptions**

| Field | Description  |
|-------|--|
| CERR  | Clear Error Indicator<br>0–63 Clear corresponding bit in EDMA_ERL<br>64–127 Clear all bits in EDMA_ERL |

### 16.3.2.11 DMA Set START Bit (EDMA\_SSBR)

The EDMA\_SSBR provides a memory-mapped mechanism to set the START bit in the TCD of the given channel. The data value on a register write causes the START bit in the corresponding transfer control descriptor to be set. Setting bit 1 (SSB[0]) provides a global set function, forcing all START bits to be set. Reads of this register return all zeroes. See [Table 16-19](#) for the TCD START bit definition.



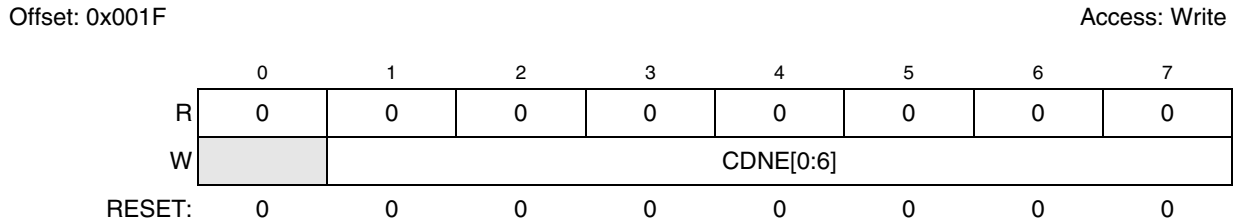
**Figure 16-12. DMA Set START Bit (EDMA\_SSBR) Register**

**Table 16-12. EDMA\_SSBR field descriptions**

| Field | Description  |
|-------|--|
| SSRT  | Set START Bit (Channel Service Request)<br>0–63 Set the corresponding channel's TCD.start<br>64–127 Set all TCD.start bits |

### 16.3.2.12 DMA Clear DONE Status (EDMA\_CDSBR)

The EDMA\_CDSBR provides a memory-mapped mechanism to clear the DONE bit in the TCD of the given channel. The data value on a register write causes the DONE bit in the corresponding transfer control descriptor to be cleared. Setting bit 1 (CDSB[0]) provides a global clear function, forcing all DONE bits to be cleared. See [Table 16-19](#) for the TCD DONE bit definition.



**Figure 16-13. DMA Clear DONE Status (EDMA\_CDSBR) Register**

**Table 16-13. EDMA\_CDSBR field descriptions**

| Field     | Description  |
|-----------|--|
| CDNE[0:6] | Clear DONE Status Bit<br>0–63 Clear the corresponding channel's DONE bit<br>64–127 Clear all TCD DONE bits |

### 16.3.2.13 DMA Interrupt Request (EDMA\_IRQRL)

The EDMA\_IRQRL provides a bit map for the 16 channels signaling the presence of an interrupt request for each channel. EDMA\_IRQRL maps to channels 15–0.

The DMA engine signals the occurrence of a programmed interrupt on the completion of a data transfer as defined in the transfer control descriptor by setting the appropriate bit in this register. The outputs of this register are directly routed to the interrupt controller (INTC). During the execution of the interrupt service routine associated with any given channel, software must clear the appropriate bit, negating the interrupt request. Typically, a write to the EDMA\_CIRQR in the interrupt service routine is used for this purpose.

The state of any given channel's interrupt request is directly affected by writes to this register; it is also affected by writes to the EDMA\_CIRQR. On writes to the EDMA\_IRQRL, a 1 in any bit position clears the corresponding channel's interrupt request. A 0 in any bit position has no affect on the corresponding channel's current interrupt status. The EDMA\_CIRQR is provided so the interrupt request for a single channel can be cleared without performing a read-modify-write sequence to the EDMA\_IRQRL. See [Figure 16-14](#) and [Table 16-14](#) for the EDMA\_IRQRL definition.

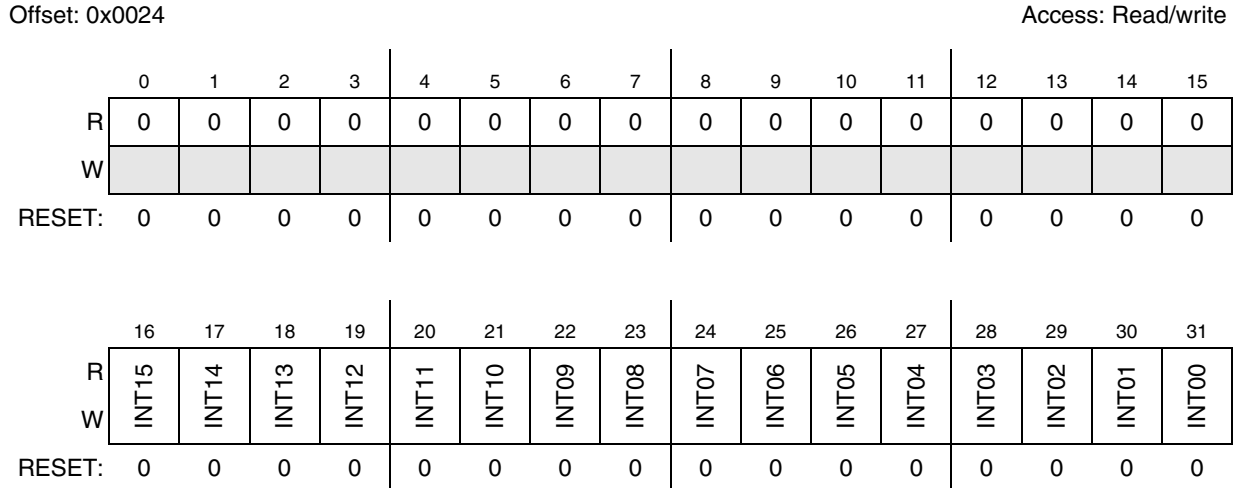


Figure 16-14. DMA Interrupt Request (EDMA\_IRQRL) Registers

Table 16-14. EDMA\_IRQRL field descriptions

| Field | Description  |
|-------|--|
| INTn  | DMA Interrupt Request n<br>0 The interrupt request for channel n is cleared.<br>1 The interrupt request for channel n is active. |

### 16.3.2.14 DMA Error (EDMA\_ERL)

The EDMA\_ERL provides a bit map for the 16 channels signaling the presence of an error for each channel. EDMA\_ERL maps to channels 15–0.

The DMA engine signals the occurrence of a error condition by setting the appropriate bit in this register. The outputs of this register are enabled by the contents of the EDMA\_EEIR, then logically summed across 16 channels to form an error interrupt request, which is then routed to the interrupt controller. During the execution of the interrupt service routine associated with any eDMA errors, it is software’s responsibility to clear the appropriate bit, negating the error interrupt request. Typically, a write to the EDMA\_CER in the interrupt service routine is used for this purpose. The normal eDMA channel completion indicators, setting the transfer control descriptor DONE flag and the possible assertion of an interrupt request, are not affected when an error is detected.

The contents of this register can also be polled. A non-zero value indicates the presence of a channel error, regardless of the state of the EDMA\_EEIR. The EDMA\_ESR[VLD] bit is a logical OR of all bits in this register, and it provides a single bit indication of any errors. The state of any given channel’s error indicators is affected by writes to this register; it is also affected by writes to the EDMA\_CER. On writes to EDMA\_ERL, a 1 in any bit position clears the corresponding channel’s error status. A 0 in any bit position has no affect on the corresponding channel’s current error status. The EDMA\_CER is provided so the error indicator for a single channel can be cleared. See [Figure 16-15](#) and [Table 16-15](#) for the EDMA\_ERL definition.

Offset: 0x002C

Access: Read/write

|        |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
|--------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
|        | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| R      | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W      |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| RESET: | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|        |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |
|--------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
|        | 16    | 17    | 18    | 19    | 20    | 21    | 22    | 23    | 24    | 25    | 26    | 27    | 28    | 29    | 30    | 31    |
| R      | ERR15 | ERR14 | ERR13 | ERR12 | ERR11 | ERR10 | ERR09 | ERR08 | ERR07 | ERR06 | ERR05 | ERR04 | ERR03 | ERR02 | ERR01 | ERR00 |
| W      |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |
| RESET: | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     |

Figure 16-15. DMA Error (EDMA\_ERL) Registers

Table 16-15. EDMA\_ERL field descriptions

| Field | Description   |
|-------|---|
| ERRn  | DMA Error n<br>0 An error in channel n has not occurred.<br>1 An error in channel n has occurred. |

### 16.3.2.15 DMA Hardware Request Status (EDMA\_HRSL)

The EDMA\_HRSL register provides a bit map for the implemented channels to show the current hardware request status for each channel. This view into the hardware request signals may be used for debug purposes.

See [Figure 16-16](#) and [Figure 16-16](#) for the EDMA\_HRSL definition.

Offset: 0x0034

Access: Read/write

|        |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
|--------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
|        | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| R      | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W      |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| RESET: | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|        |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |
|--------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
|        | 16    | 17    | 18    | 19    | 20    | 21    | 22    | 23    | 24    | 25    | 26    | 27    | 28    | 29    | 30    | 31    |
| R      | HRS15 | HRS14 | HRS13 | HRS12 | HRS11 | HRS10 | HRS09 | HRS08 | HRS07 | HRS06 | HRS05 | HRS04 | HRS03 | HRS02 | HRS01 | HRS00 |
| W      |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |
| RESET: | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 0     |

Figure 16-16. DMA Hardware Request Status (EDMA\_HRSL) Register

**Table 16-16. EDMA\_HRSL field descriptions**

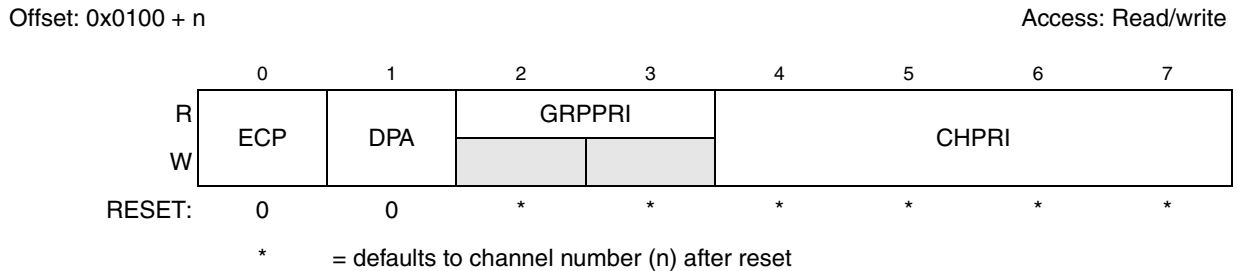
| Field | Description  |
|-------|--|
| HRSn  | DMA Hardware Request Status n<br>0 A hardware service request for channel n is not present.<br>1 A hardware service request for channel n is present.<br><br><b>Note:</b> The hardware request status reflects the state of the request as seen by the arbitration logic. Therefore, this status is affected by the EDMA_ERQRL[n] bit. |

**16.3.2.16 DMA Channel n Priority (EDMA\_CPRn)**

When the fixed-priority channel arbitration mode is enabled (EDMA\_CR[ERCA] = 0), the contents of these registers define the unique priorities associated with each channel. The channel priorities are evaluated by numeric value; that is, 0 is the lowest priority, 1 is the next higher priority, then 2, 3, etc. If software modifies channel priority values, then the software must ensure that the channel priorities contain unique values, otherwise a configuration error will be reported. The range of the priority value is limited to the values of 0 through 15.

Channel preemption is enabled on a per-channel basis by setting the ECP bit in the EDMA\_CPRn register. Channel preemption allows the executing channel’s data transfers to be temporarily suspended in favor of starting a higher priority channel. After the preempting channel has completed all its minor loop data transfers, the preempted channel is restored and resumes execution. After the restored channel completes one read/write sequence, it is again eligible for preemption. If any higher priority channel requests service, the restored channel will be suspended and the higher priority channel will be serviced. Nested preemption (attempting to preempt a preempting channel) is not supported. After a preempting channel begins execution, it cannot be preempted. Preemption is available only when fixed arbitration is selected for channel arbitration mode

A channel’s ability to preempt another channel can be disabled by setting the DPA bit in the EDMA\_CPRn register. When a channel’s preempt ability is disabled, that channel cannot suspend a lower priority channel’s data transfer; regardless of the lower priority channel’s ECP setting. This allows for a pool of low priority, large data moving channels to be defined. These low priority channels can be configured to not preempt each other, thus preventing a low priority channel from consuming the preempt slot normally available a true, high priority channel. See Figure 16-17 and Table 16-17 for the EDMA\_CPRn definition.



**Figure 16-17. DMA Channel n Priority (EDMA\_CPRn) Register**

**Table 16-17. EDMA\_CPRn field descriptions**

| Field      | Description   |
|------------|---|
| ECP        | Enable Channel Preemption<br>0 Channel n cannot be suspended by a higher priority channel's service request.<br>1 Channel n can be temporarily suspended by the service request of a higher priority channel. |
| DPA        | Disable Preempt Ability<br>0 Channel n can suspend a lower priority channel.<br>1 Channel n cannot suspend any channel, regardless of channel priority.   |
| CHPRI[0:3] | Channel n Arbitration Priority<br>Channel priority when fixed-priority arbitration is enabled.  |

### 16.3.2.17 Transfer Control Descriptor (TCD)

Each channel requires a 32-byte transfer control descriptor for defining the desired data movement operation. The channel descriptors are stored in the local memory in sequential order: channel 0, channel 1,... channel 15. The definitions of the TCD are presented as eight 32-bit values. [Table 16-18](#) is a field list of the basic TCD structure.

**Table 16-18. TCDn 32-bit memory structure**

| eDMA offset            | TCDn field   |  |
|------------------------|--|--|
| 0x1000+(32 × n)+0x0000 | Source address (saddr)   |  |
| 0x1000+(32 × n)+0x0004 | Transfer attributes  | Signed source address offset (soff)      |
| 0x1000+(32 × n)+0x0008 | Inner minor byte count (nbytes)  |  |
| 0x1000+(32 × n)+0x000C | Last source address adjustment (slast)                                   |  |
| 0x1000+(32 × n)+0x0010 | Destination address (daddr)  |  |
| 0x1000+(32 × n)+0x0014 | Current major iteration count (citer)                                    | Signed destination address offset (doff) |
| 0x1000 (32 × n) 0x0018 | Last destination address adjustment / scatter-gather address (dlast_sga) |  |
| 0x1000+(32 × n)+0x001C | Beginning major iteration count (biter)                                  | Channel control/status                   |

[Figure 16-18](#) and [Table 16-19](#) define the fields of the TCD<sub>n</sub> structure.

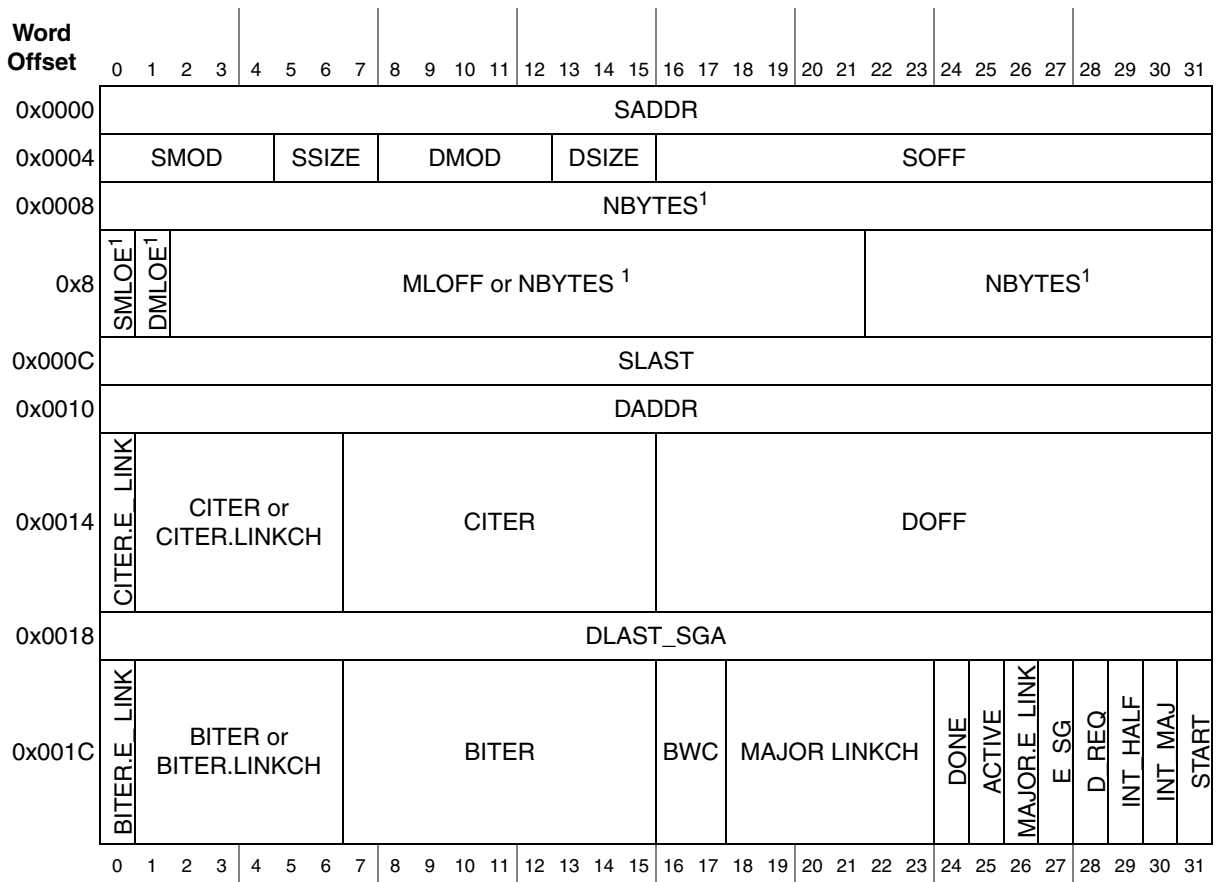


Figure 16-18. TCD structure

<sup>1</sup> The fields implemented in Word 2 depend on whether EDMA\_CR(EMLM) is set to 0 or 1. See Table 16-2.

**NOTE**

The TCD structures for the eDMA channels shown in Figure 16-18 are implemented in internal SRAM. These structures are not initialized at reset; therefore, all channel TCD parameters must be initialized by the application code before activating that channel.



Table 16-19. TCD<sub>n</sub> field descriptions

| Bits / Word Offset [n:n] | Name          | Description   |
|--------------------------|---------------|---|
| 0–31 / 0x0 [0:31]        | SADDR [0:31]  | Source address. Memory address pointing to the source data. Word 0x0, bits 0–31.  |
| 32–36 / 0x4 [0:4]        | SMOD [0:4]    | Source address modulo.<br>0 Source address modulo feature is disabled.<br>non-0 This value defines a specific address range that is specified to be the value after SADDR + SOFF calculation is performed or the original register value. The setting of this field provides the ability to easily implement a circular data queue. For data queues requiring power-of-2 size bytes, the queue should start at a 0-modulo-size address and the SMOD field should be set to the appropriate value for the queue, freezing the desired number of upper address bits. The value programmed into this field specifies the number of lower address bits that are allowed to change. For this circular queue application, the SOFF is typically set to the transfer size to implement post-increment addressing with the SMOD function constraining the addresses to a 0-modulo-size range. |
| 37–39 / 0x4 [5:7]        | SSIZE [0:2]   | Source data transfer size.<br>000 8-bit<br>001 16-bit<br>010 32-bit<br>011 Reserved<br>100 16-byte (32-bit, 4-beat, WRAP4 burst)<br>101 32-byte (32-bit, 8-beat, WRAP8 burst)<br>110 Reserved<br>111 Reserved<br>The attempted specification of a reserved encoding causes a configuration error.   |
| 40–44 / 0x4 [8:12]       | DMOD [0:4]    | Destination address modulo. See the SMOD[0:5] definition.   |
| 45–47 / 0x4 [13:15]      | DSIZE [0:2]   | Destination data transfer size. See the SSIZE[0:2] definition.  |
| 48–63 / 0x4 [16:31]      | SOFF [0:15]   | Source address signed offset. Sign-extended offset applied to the current source address to form the next-state value as each source read is completed.   |
| 64–95 / 0x8 [0:31]       | NBYTES [0:31] | Inner minor byte transfer count. Number of bytes to be transferred in each service request of the channel. As a channel is activated, the contents of the appropriate TCD is loaded into the DMA engine, and the appropriate reads and writes performed until the complete byte transfer count has been transferred. This is an indivisible operation and cannot be stalled or halted. After the minor count is exhausted, the current values of the SADDR and DADDR are written back into the local memory, the major iteration count is decremented and restored to the local memory. If the major iteration count is completed, additional processing is performed.<br><b>Note:</b> The NBYTES value of 0x0000_0000 is interpreted as 0x1_0000_0000, thus specifying a 4 GB transfer.  |

Table 16-19. TCDn field descriptions (continued)

| Bits / Word Offset [n:n] | Name                                      | Description   |
|--------------------------|---|---|
| 64–95 / 0x8 [0:31]       | NBYTES <sup>1</sup><br>[0:31]             | Inner minor byte transfer count. Number of bytes to be transferred in each service request of the channel. As a channel is activated, the contents of the appropriate TCD is loaded into the eDMA engine, and the appropriate reads and writes performed until the complete byte transfer count has been transferred. This is an indivisible operation and cannot be stalled or halted. Once the minor count is exhausted, the current values of the SADDR and DADDR are written back into the local memory, the major iteration count is decremented and restored to the local memory. If the major iteration count is completed, additional processing is performed.<br><b>Note:</b> The NBYTES value of 0x0000_0000 is interpreted as 0x1_0000_0000, thus specifying a 4 Gbyte transfer. |
| 64<br>0x8 [0]            | SMLOE <sup>1</sup><br>0                   | Source minor loop offset enable<br>This flag selects whether the minor loop offset is applied to the source address upon minor loop completion.<br><br>0 The minor loop offset is not applied to the saddr.<br>1 The minor loop offset is applied to the saddr.   |
| 65<br>0x8 [1]            | DMLOE <sup>1</sup><br>1                   | Destination minor loop offset enable<br>This flag selects whether the minor loop offset is applied to the destination address upon minor loop completion.<br><br>0 The minor loop offset is not applied to the daddr.<br>1 The minor loop offset is applied to the daddr.   |
| 66–85<br>0x8 [2:21]      | MLOFF or<br>NBYTES <sup>1</sup><br>[0:19] | Inner minor byte transfer count or Minor loop offset<br>If both SMLOE and DMLOE are cleared, this field is part of the byte transfer count.<br><br>If either SMLOE or DMLOE are set, this field represents a sign-extended offset applied to the source or destination address to form the next-state value after the minor loop is completed.  |
| 86–95 / 0x8 [22:31]      | NBYTES <sup>1</sup>                       | Inner minor byte transfer count. Number of bytes to be transferred in each service request of the channel. As a channel is activated, the contents of the appropriate TCD is loaded into the eDMA engine, and the appropriate reads and writes performed until the complete byte transfer count has been transferred. This is an indivisible operation and cannot be stalled or halted. Once the minor count is exhausted, the current values of the SADDR and DADDR are written back into the local memory, the major iteration count is decremented and restored to the local memory. If the major iteration count is completed, additional processing is performed.<br><b>Note:</b> The NBYTES value of 0x0000_0000 is interpreted as 0x1_0000_0000, thus specifying a 4 GByte transfer. |
| 96–127 / 0xC [0:31]      | SLAST<br>[0:31]                           | Last source address adjustment. Adjustment value added to the source address at the completion of the outer major iteration count. This value can be applied to restore the source address to the initial value, or adjust the address to reference the next data structure.  |
| 128–159 / 0x10 [0:31]    | DADDR<br>[0:31]                           | Destination address. Memory address pointing to the destination data.   |

Table 16-19. TCDn field descriptions (continued)

| Bits /<br>Word Offset<br>[n:n] | Name  | Description   |
|--------------------------------|---|---|
| 160 /<br>0x14 [0]              | CITER.E_LINK                                  | <p>Enable channel-to-channel linking on minor loop completion. As the channel completes the inner minor loop, this flag enables the linking to another channel, defined by CITER.LINKCH[0:5]. The link target channel initiates a channel service request via an internal mechanism that sets the TCD.START bit of the specified channel. If channel linking is disabled, the CITER value is extended to 15 bits in place of a link channel number. If the major loop is exhausted, this link mechanism is suppressed in favor of the MAJOR.E_LINK channel linking.</p> <p>0 The channel-to-channel linking is disabled.<br/>1 The channel-to-channel linking is enabled.</p> <p><b>Note:</b> This bit must be equal to the BITER.E_LINK bit otherwise a configuration error will be reported.</p>  |
| 161–166 /<br>0x14 [1:6]        | CITER<br>[0:5]<br>or<br>CITER.LINKCH<br>[0:5] | <p>Current major iteration count or link channel number.</p> <p>If channel-to-channel linking is disabled (TCD.CITER.E_LINK = 0), then</p> <ul style="list-style-type: none"> <li>No channel-to-channel linking (or chaining) is performed after the inner minor loop is exhausted. TCD bits [161:175] are used to form a 15-bit CITER field.</li> </ul> <p>Otherwise,</p> <ul style="list-style-type: none"> <li>After the minor loop is exhausted, the DMA engine initiates a channel service request at the channel defined by CITER.LINKCH[0:5] by setting that channel's TCD.START bit.</li> </ul>   |
| 167–175 /<br>0x14 [7:15]       | CITER<br>[6:14]                               | <p>Current major iteration count. This 9- or 15-bit count represents the current major loop count for the channel. It is decremented each time the minor loop is completed and updated in the transfer control descriptor memory. After the major iteration count is exhausted, the channel performs a number of operations (for example, final source and destination address calculations), optionally generating an interrupt to signal channel completion before reloading the CITER field from the beginning iteration count (BITER) field.</p> <p><b>Note:</b> When the CITER field is initially loaded by software, it must be set to the same value as that contained in the BITER field.</p> <p><b>Note:</b> If the channel is configured to execute a single service request, the initial values of BITER and CITER should be 0x0001.</p> |
| 176–191 /<br>0x14 [16:31]      | DOFF<br>[0:15]                                | <p>Destination address signed Offset. Sign-extended offset applied to the current destination address to form the next-state value as each destination write is completed.</p>  |

Table 16-19. TCDn field descriptions (continued)

| Bits / Word Offset [n:n] | Name                                   | Description   |
|--------------------------|--|---|
| 192–223 / 0x18 [0:31]    | DLAST_SGA [0:31]                       | <p>Last destination address adjustment or the memory address for the next transfer control descriptor to be loaded into this channel (scatter-gather). If scatter-gather processing for the channel is disabled (TCD.E_SG = 0) then</p> <ul style="list-style-type: none"> <li>Adjustment value added to the destination address at the completion of the outer major iteration count.</li> </ul> <p>This value can be applied to restore the destination address to the initial value, or adjust the address to reference the next data structure. Otherwise,</p> <ul style="list-style-type: none"> <li>This address points to the beginning of a 0-modulo-32 byte region containing the next transfer control descriptor to be loaded into this channel. This channel reload is performed as the major iteration count completes. The scatter-gather address must be 0-modulo-32 byte, otherwise a configuration error is reported.</li> </ul>                   |
| 224 / 0x1C [0]           | BITER.E_LINK                           | <p>Enables channel-to-channel linking on minor loop complete. As the channel completes the inner minor loop, this flag enables the linking to another channel, defined by BITER.LINKCH[0:5]. The link target channel initiates a channel service request via an internal mechanism that sets the TCD.START bit of the specified channel. If channel linking is disabled, the BITER value is extended to 15 bits in place of a link channel number. If the major loop is exhausted, this link mechanism is suppressed in favor of the MAJOR.E_LINK channel linking.</p> <p>0 The channel-to-channel linking is disabled.<br/>1 The channel-to-channel linking is enabled.</p> <p><b>Note:</b> When the TCD is first loaded by software, this field must be set equal to the corresponding CITER field, otherwise a configuration error will be reported. As the major iteration count is exhausted, the contents of this field is reloaded into the CITER field.</p> |
| 225–230 / 0x1C [1:6]     | BITER [0:5]<br>or<br>BITER.LINKCH[0:5] | <p>Starting major iteration count or link channel number. If channel-to-channel linking is disabled (TCD.BITER.E_LINK = 0), then</p> <ul style="list-style-type: none"> <li>No channel-to-channel linking (or chaining) is performed after the inner minor loop is exhausted. TCD bits [225:239] are used to form a 15-bit BITER field.</li> </ul> <p>Otherwise,</p> <ul style="list-style-type: none"> <li>After the minor loop is exhausted, the DMA engine initiates a channel service request at the channel, defined by BITER.LINKCH[0:5], by setting that channel's TCD.START bit.</li> </ul> <p><b>Note:</b> When the TCD is first loaded by software, this field must be set equal to the corresponding CITER field, otherwise a configuration error will be reported. As the major iteration count is exhausted, the contents of this field is reloaded into the CITER field.</p>  |
| 231–239 / 0x1C [7:15]    | BITER [6:14]                           | <p>Starting major iteration count. As the transfer control descriptor is first loaded by software, this field must be equal to the value in the CITER field. As the major iteration count is exhausted, the contents of this field are reloaded into the CITER field.</p> <p><b>Note:</b> If the channel is configured to execute a single service request, the initial values of BITER and CITER should be 0x0001.</p>   |

Table 16-19. TCDn field descriptions (continued)

| Bits / Word Offset [n:n] | Name               | Description  |
|--------------------------|--------------------|--|
| 240–241 / 0x1C [16:17]   | BWC [0:1]          | Bandwidth control. This 2-bit field provides a mechanism to effectively throttle the amount of bus bandwidth consumed by the eDMA. In general, as the eDMA processes the inner minor loop, it continuously generates read/write sequences until the minor count is exhausted. This field forces the eDMA to stall after the completion of each read/write access to control the bus request bandwidth seen by the system bus crossbar switch (XBAR).<br>00 No DMA engine stalls<br>01 Reserved<br>10 DMA engine stalls for 4 cycles after each r/w<br>11 DMA engine stalls for 8 cycles after each r/w |
| 242–247 / 0x1C [18:23]   | MAJOR.LINKCH [0:5] | Link channel number.<br>If channel-to-channel linking on major loop complete is disabled (TCD.MAJOR.E_LINK = 0) then, <ul style="list-style-type: none"> <li>No channel-to-channel linking (or chaining) is performed after the outer major loop counter is exhausted.</li> </ul> Otherwise <ul style="list-style-type: none"> <li>After the major loop counter is exhausted, the DMA engine initiates a channel service request at the channel defined by MAJOR.LINKCH[0:5] by setting that channel's TCD.START bit.</li> </ul>   |
| 248 / 0x1C [24]          | DONE               | Channel done. This flag indicates the eDMA has completed the outer major loop. It is set by the DMA engine as the CITER count reaches zero; it is cleared by software or hardware when the channel is activated (when the DMA engine has begun processing the channel, not when the first data transfer occurs).<br><b>Note:</b> This bit must be cleared to write the MAJOR.E_LINK or E_SG bits.  |
| 249 / 0x1C [25]          | ACTIVE             | Channel active. This flag signals the channel is currently in execution. It is set when channel service begins, and is cleared by the DMA engine as the inner minor loop completes or if any error condition is detected.  |
| 250 / 0x1C [26]          | MAJOR.E_LINK       | Enable channel-to-channel linking on major loop completion. As the channel completes the outer major loop, this flag enables the linking to another channel, defined by MAJOR.LINKCH[0:5]. The link target channel initiates a channel service request via an internal mechanism that sets the TCD.START bit of the specified channel.<br><b>NOTE:</b> To support the dynamic linking coherency model, this field is forced to zero when written to while the TCD.DONE bit is set.<br>0 The channel-to-channel linking is disabled.<br>1 The channel-to-channel linking is enabled.                    |

Table 16-19. TCDn field descriptions (continued)

| Bits / Word Offset [n:n] | Name     | Description   |
|--------------------------|----------|---|
| 251 / 0x1C [27]          | E_SG     | Enable scatter-gather processing. As the channel completes the outer major loop, this flag enables scatter-gather processing in the current channel. If enabled, the DMA engine uses DLAST_SGA as a memory pointer to a 0-modulo-32 address containing a 32-byte data structure that is loaded as the transfer control descriptor into the local memory.<br>NOTE: To support the dynamic scatter-gather coherency model, this field is forced to zero when written to while the TCD.DONE bit is set.<br>0 The current channel's TCD is normal format.<br>1 The current channel's TCD specifies a scatter gather format. The DLAST_SGA field provides a memory pointer to the next TCD to be loaded into this channel after the outer major loop completes its execution.                                  |
| 252 / 0x1C [28]          | D_REQ    | Disable hardware request. If this flag is set, the eDMA hardware automatically clears the corresponding EDMA_ERQRL bit when the current major iteration count reaches zero.<br>0 The channel's EDMA_ERQRL bit is not affected.<br>1 The channel's EDMA_ERQRL bit is cleared when the outer major loop is complete.  |
| 253 / 0x1C [29]          | INT_HALF | Enable an interrupt when major counter is half complete. If this flag is set, the channel generates an interrupt request by setting the appropriate bit in the EDMA_ERQRL when the current major iteration count reaches the halfway point. Specifically, the comparison performed by the eDMA engine is $(CITER == (BITER >> 1))$ . This halfway point interrupt request is provided to support double-buffered (also known as ping-pong) schemes, or other types of data movement where the processor needs an early indication of the transfer's progress. $CITER = BITER = 1$ with INT_HALF enabled will generate an interrupt as it satisfies the equation $(CITER == (BITER >> 1))$ after a single activation.<br>0 The half-point interrupt is disabled.<br>1 The half-point interrupt is enabled. |
| 254 / 0x1C [30]          | INT_MAJ  | Enable an interrupt when major iteration count completes. If this flag is set, the channel generates an interrupt request by setting the appropriate bit in the EDMA_ERQRL when the current major iteration count reaches zero.<br>0 The end-of-major loop interrupt is disabled.<br>1 The end-of-major loop interrupt is enabled.  |
| 255 / 0x1C [31]          | START    | Channel start. If this flag is set the channel is requesting service. The eDMA hardware automatically clears this flag after the channel begins execution.<br>0 The channel is not explicitly started.<br>1 The channel is explicitly started via a software initiated service request.   |

<sup>1</sup> The fields implemented at 0x8 depend on whether EDMA\_CR(EMLM) is set to 0 or 1. Refer to [Table 16-2](#).

## 16.4 Functional description

This section provides an overview of the microarchitecture and functional operation of the eDMA block.

The eDMA module is partitioned into two major modules: the DMA engine and the transfer control descriptor local memory. The DMA engine is further partitioned into four submodules, which are detailed below.

- DMA engine
  - Address path: This module implements registered versions of two channel transfer control descriptors: channel x and channel y, and is responsible for all the master bus address calculations. All the implemented channels provide the same functionality. This hardware structure allows the data transfers associated with one channel to be preempted after the completion of a read/write sequence if a higher priority channel service request is asserted while the first channel is active. After a channel is activated, it runs until the minor loop is completed unless preempted by a higher priority channel. This capability provides a mechanism (optionally enabled by EDMA\_CPR $n$ [ECP]) where a large data move operation can be preempted to minimize the time another channel is blocked from execution.
 

When another channel is activated, the contents of its transfer control descriptor is read from the local memory and loaded into the registers of the other address path channel{x,y}. After the inner minor loop completes execution, the address path hardware writes the new values for the TCD $n$ .{SADDR, DADDR, CITER} back into the local memory. If the major iteration count is exhausted, additional processing is performed, including the final address pointer updates, reloading the TCD $n$ .CITER field, and a possible fetch of the next TCD $n$  from memory as part of a scatter-gather operation.
  - Data path: This module implements the actual bus master read/write datapath. It includes 32 bytes of register storage (matching the maximum transfer size) and the necessary mux logic to support any required data alignment. The system read data bus is the primary input, and the system write data bus is the primary output.
 

The address and data path modules directly support the two-stage pipelined system bus. The address path module represents the 1st stage of the bus pipeline (the address phase), while the data path module implements the second stage of the pipeline (the data phase).
  - Program model/channel arbitration: This module implements the first section of eDMA's programming model and also the channel arbitration logic. The programming model registers are connected to the slave bus (not shown). The eDMA peripheral request inputs and eDMA interrupt request outputs are also connected to this module (via the control logic).
  - Control: This module provides all the control functions for the DMA engine. For data transfers where the source and destination sizes are equal, the DMA engine performs a series of source read, destination write operations until the number of bytes specified in the inner minor loop byte count has been moved.
 

A minor loop interaction is defined as the number of bytes to transfer ( $n$ bytes) divided by the transfer size. Transfer size is defined as:

```
if (SSIZE < DSIZE)
  transfer size = destination transfer size (# of bytes)
else
  transfer size = source transfer size (# of bytes)
```

Minor loop TCD variables are SOFF, SMOD, DOFF, DMOD, NBYTES, SADDR, DADDR, BWC, ACTIVE, and START. Major loop TCD variables are DLAST, SLAST, CITER, BITER, DONE, D\_REQ, INT\_MAJ, MAJOR\_LNKCH, and INT\_HALF.

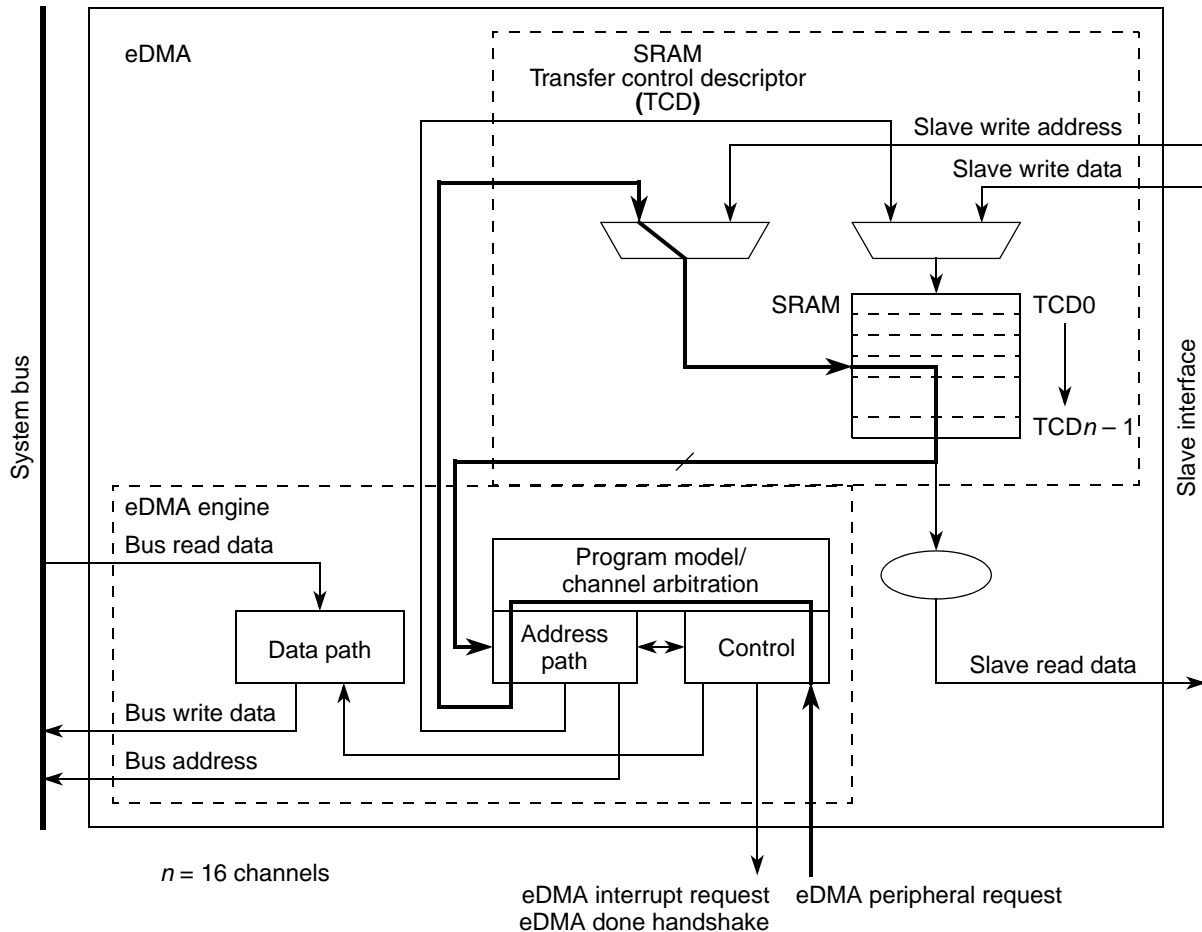
For descriptors where the sizes are not equal, multiple access of the smaller size data are required for each reference of the larger size. For example, if the source size references 16-bit data and the destination is 32-bit data, two reads are performed, then one 32-bit write.

- TCD local memory
  - Memory controller: This logic implements the required dual-ported controller, handling accesses from both the DMA engine as well as references from the slave bus. As noted earlier, in the event of simultaneous accesses, the DMA engine is given priority and the slave transaction is stalled. The hooks to a BIST controller for the local TCD memory are included in this module.
  - Memory array: The TCD is implemented using a single-ported, synchronous compiled RAM memory array.

### 16.4.1 eDMA basic data flow

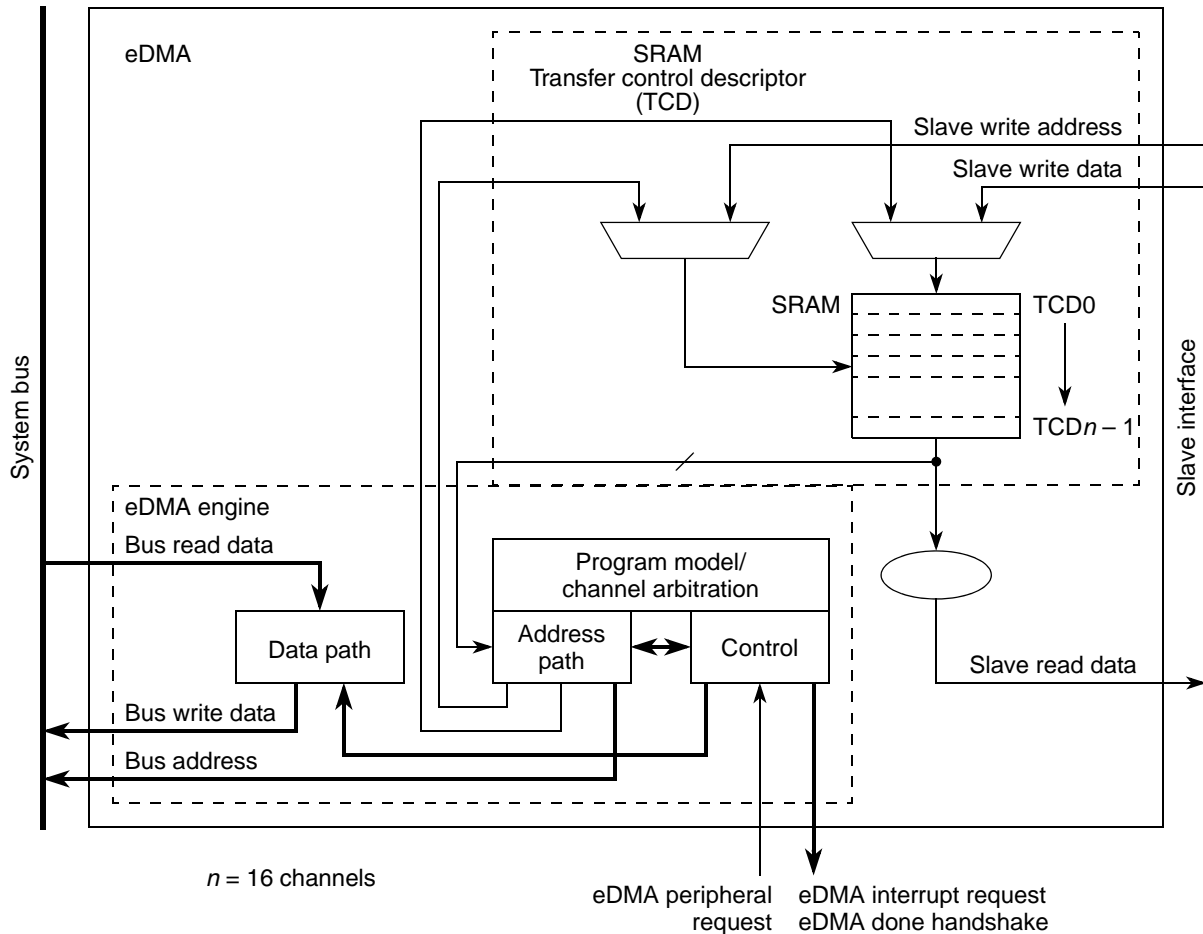
The eDMA transfers data based on a two-deep, nested flow. The basic flow of a data transfer can be partitioned into three segments. As shown in [Figure 16-19](#), the first segment involves the channel service request. In the diagram, this example uses the assertion of the eDMA peripheral request signal to request service for channel  $n$ . Channel service request via software and the TCDn.START bit follows the same basic flow as an eDMA peripheral request. The eDMA peripheral request input signal is registered internally and then routed to through the DMA engine, first through the control module, then into the program model/channel arbitration module. In the next cycle, the channel arbitration is performed using the fixed-priority or round-robin algorithm. After the arbitration is complete, the activated channel number is sent through the address path and converted into the required address to access the TCD local memory. Next, the TCD memory is accessed and the required descriptor read from the local memory and loaded into the DMA engine address path channel{x,y} registers. The TCD memory is organized 64 bits in width to minimize the time needed to fetch the activated channel's descriptor and load it into the eDMA engine address path channel{x,y} registers.





**Figure 16-19. eDMA operation, part 1**

In the second part of the basic data flow as shown in [Figure 16-20](#), the modules associated with the data transfer (address path, data path, and control) sequence through the required source reads and destination writes to perform the actual data movement. The source reads are initiated and the fetched data is temporarily stored in the data path module until it is gated onto the system bus during the destination write. This source read/destination write processing continues until the inner minor byte count has been transferred. The eDMA done handshake signal is asserted at the end of the minor byte count transfer.



**Figure 16-20. eDMA operation, part 2**

After the inner minor byte count has been moved, the final phase of the basic data flow is performed. In this segment, the address path logic performs the required updates to certain fields in the channel's TCD; for example, SADDR, DADDR, CITER. If the outer major iteration count is exhausted, then there are additional operations performed. These include the final address adjustments and reloading of the BITER field into the CITER. Additionally, assertion of an optional interrupt request occurs at this time, as does a possible fetch of a new TCD from memory using the scatter-gather address pointer included in the descriptor. The updates to the TCD memory and the assertion of an interrupt request are shown in [Figure 16-21](#).

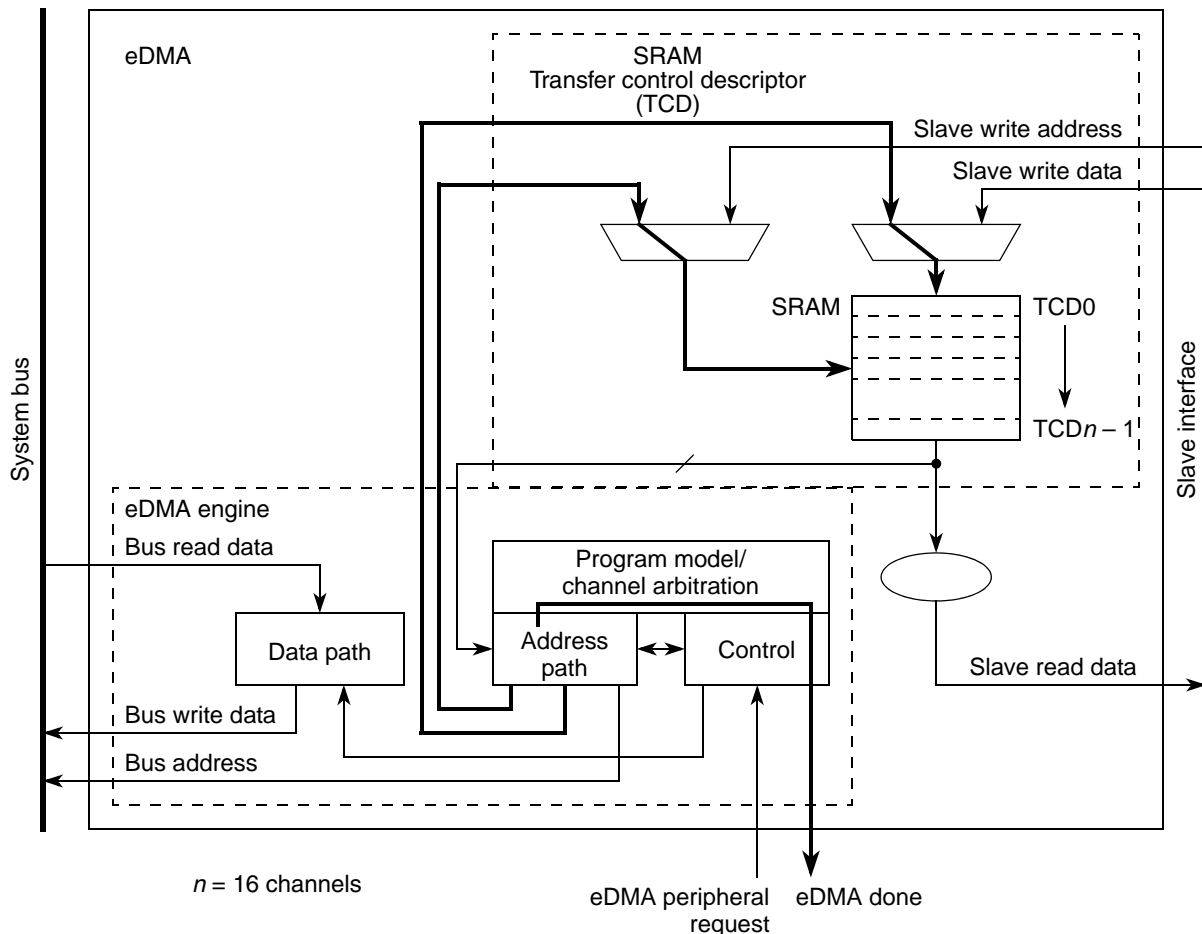


Figure 16-21. eDMA operation, part 3

## 16.5 Initialization / application information

### 16.5.1 eDMA initialization

A typical initialization of the eDMA has the following sequence:

1. Write the EDMA\_CR if a configuration other than the default is desired.
2. Write the channel priority levels into the EDMA\_CPR $n$  registers if a configuration other than the default is desired.
3. Enable error interrupts in the EDMA\_EEIRL and/or EDMA\_EEIRH registers if desired.
4. Write the 32-byte TCD for each channel that may request service.
5. Enable any hardware service requests via the EDMA\_ERQRH and/or EDMA\_ERQRL registers.
6. Request channel service by software (setting the TCD.START bit) or by hardware (slave device asserting its DMA peripheral request signal).

After any channel requests service, a channel is selected for execution based on the arbitration and priority levels written into the programmer's model. The DMA engine will read the entire TCD, including the

primary transfer control parameter shown in [Table 16-20](#), for the selected channel into its internal address path module. As the TCD is being read, the first transfer is initiated on the system bus unless a configuration error is detected. Transfers from the source (as defined by the source address, TCD.SADDR) to the destination (as defined by the destination address, TCD.DADDR) continue until the specified number of bytes (TCD.NBYTES) have been transferred. When the transfer is complete, the DMA engine's local TCD.SADDR, TCD.DADDR, and TCD.CITER are written back to the main TCD memory and any minor loop channel linking is performed, if enabled. If the major loop is exhausted, further post processing is executed; for example, interrupts, major loop channel linking, and scatter-gather operations, if enabled.

**Table 16-20. TCD primary control and status fields**

| TCD field name | Description  |
|----------------|--|
| START          | Control bit to start channel when using a software initiated DMA service (Automatically cleared by hardware)   |
| ACTIVE         | Status bit indicating the channel is currently in execution  |
| DONE           | Status bit indicating major loop completion (cleared by software when using a software initiated DMA service)  |
| D_REQ          | Control bit to disable DMA request at end of major loop completion when using a hardware-initiated DMA service |
| BWC            | Control bits for throttling bandwidth control of a channel   |
| E_SG           | Control bit to enable scatter-gather feature   |
| INT_HALF       | Control bit to enable interrupt when major loop is half complete   |
| INT_MAJ        | Control bit to enable interrupt when major loop completes  |

[Figure 16-22](#) shows how each DMA request initiates one minor loop transfer (iteration) without CPU intervention. DMA arbitration can occur after each minor loop, and one level of minor loop DMA preemption is allowed. The number of minor loops in a major loop is specified by the beginning iteration count (biter).

| Example memory array |   |            | Current major loop iteration count (CITER) |            |            |
|----------------------|---|------------|--|------------|------------|
| DMA request          |   | Minor loop | Major loop                                 | 3          |            |
|                      |   |            |  |            |            |
|                      | ⋮ |            |  |            |            |
|                      |   |            |  |            |            |
| DMA request          |   | Minor loop |  | Major loop | 2          |
|                      |   |            |  |            |            |
|                      | ⋮ |            |  |            |            |
|                      |   |            |  |            |            |
| DMA request          |   | Minor loop |  |            | Major loop |
|                      |   |            |  |            |            |
|                      | ⋮ |            |  |            |            |
|                      |   |            |  |            |            |

Figure 16-22. Example of multiple loop iterations

Figure 16-23 lists the memory array terms and how the TCD settings interrelate.

|  |                                       |   |  |
|--|---------------------------------------|---|--|
| xADDR:<br>(Starting address)   | xSIZE:<br>(Size of one data transfer) | Minor loop<br>(NBYTES in minor loop, often the same value as xSIZE) | Offset (xOFF): Number of bytes added to current address after each transfer (Often the same value as xSIZE)  |
| ⋮  | ⋮                                     | Minor loop  | Each DMA source (S) and destination (D) has its own: <ul style="list-style-type: none"> <li>• Address (xADDR)</li> <li>• Size (xSIZE)</li> <li>• Offset (xOFF)</li> <li>• Modulo (xMOD)</li> <li>• Last address adjustment (xLAST) where x = S or D</li> </ul> |
| xLAST: Number of bytes added to current address after major loop (typically used to loop back) | ⋮                                     | Last minor loop   | Peripheral queues typically have size and offset equal to NBYTES   |

Figure 16-23. Memory array terms

## 16.5.2 DMA programming errors

The DMA performs various tests on the transfer control descriptor to verify consistency in the descriptor data. Most programming errors are reported on a per-channel basis with the exception of channel-priority error, or EDMA\_ESR[CPE].

For all error types other than channel-priority errors, the channel number causing the error is recorded in the EDMA\_ESR. If the error source is not removed before the next activation of the problem channel, the error will be detected and recorded again.

If priority levels are not unique, the highest (channel) priority that has an active request is selected, but the lowest numbered (channel) with that priority is selected by arbitration and executed by the DMA engine. The hardware service request handshake signals, error interrupts, and error reporting are associated with the selected channel.

### 16.5.3 DMA request assignments

The assignments between the DMA requests from the modules to the channels of the eDMA are shown in [Table 16-21](#). The source column is written in C language syntax. The syntax is `module_instance.register[bit]`.

**Table 16-21. DMA request summary for eDMA**

| DMA Request               | Channel | Source                     | Description               |
|---------------------------|---------|----------------------------|---------------------------|
| DMA_MUX_CHCONFIG0_SOURCE  | 0       | DMA_MUX.CHCONFIG0[SOURCE]  | DMA MUX channel 0 source  |
| DMA_MUX_CHCONFIG1_SOURCE  | 1       | DMA_MUX.CHCONFIG1[SOURCE]  | DMA MUX channel 1 source  |
| DMA_MUX_CHCONFIG2_SOURCE  | 2       | DMA_MUX.CHCONFIG2[SOURCE]  | DMA MUX channel 2 source  |
| DMA_MUX_CHCONFIG3_SOURCE  | 3       | DMA_MUX.CHCONFIG3[SOURCE]  | DMA MUX channel 3 source  |
| DMA_MUX_CHCONFIG4_SOURCE  | 4       | DMA_MUX.CHCONFIG4[SOURCE]  | DMA MUX channel 4 source  |
| DMA_MUX_CHCONFIG5_SOURCE  | 5       | DMA_MUX.CHCONFIG5[SOURCE]  | DMA MUX channel 5 source  |
| DMA_MUX_CHCONFIG6_SOURCE  | 6       | DMA_MUX.CHCONFIG6[SOURCE]  | DMA MUX channel 6 source  |
| DMA_MUX_CHCONFIG7_SOURCE  | 7       | DMA_MUX.CHCONFIG7[SOURCE]  | DMA MUX channel 7 source  |
| DMA_MUX_CHCONFIG8_SOURCE  | 8       | DMA_MUX.CHCONFIG8[SOURCE]  | DMA MUX channel 8 source  |
| DMA_MUX_CHCONFIG9_SOURCE  | 9       | DMA_MUX.CHCONFIG9[SOURCE]  | DMA MUX channel 9 source  |
| DMA_MUX_CHCONFIG10_SOURCE | 10      | DMA_MUX.CHCONFIG10[SOURCE] | DMA MUX channel 10 source |
| DMA_MUX_CHCONFIG11_SOURCE | 11      | DMA_MUX.CHCONFIG11[SOURCE] | DMA MUX channel 11 source |
| DMA_MUX_CHCONFIG12_SOURCE | 12      | DMA_MUX.CHCONFIG12[SOURCE] | DMA MUX channel 12 source |
| DMA_MUX_CHCONFIG13_SOURCE | 13      | DMA_MUX.CHCONFIG13[SOURCE] | DMA MUX channel 13 source |
| DMA_MUX_CHCONFIG14_SOURCE | 14      | DMA_MUX.CHCONFIG14[SOURCE] | DMA MUX channel 14 source |
| DMA_MUX_CHCONFIG15_SOURCE | 15      | DMA_MUX.CHCONFIG15[SOURCE] | DMA MUX channel 15 source |

### 16.5.4 DMA arbitration mode considerations

#### 16.5.4.1 Fixed-channel arbitration

In this mode, the channel service request from the highest priority channel is selected to execute. Preemption is available in this scenario only.

#### 16.5.4.2 Round-robin channel arbitration

In this mode, channels are serviced starting with the highest channel number and rotating through to the lowest channel number without regard to the assigned channel priority levels.

## 16.5.5 DMA transfer

### 16.5.5.1 Single request

To perform a simple transfer of  $n$  bytes of data with one activation, set the major loop to 1 (TCD.CITER = TCD.BITER = 1). The data transfer will begin after the channel service request is acknowledged and the channel is selected to execute. After the transfer is complete, the TCD.DONE bit will be set and an interrupt will be generated if properly enabled.

For example, the following TCD entry is configured to transfer 16 bytes of data. The eDMA is programmed for one iteration of the major loop transferring 16 bytes per iteration. The source memory has a byte-wide memory port located at 0x1000. The destination memory has a word wide port located at 0x2000. The address offsets are programmed in increments to match the size of the transfer; 1 byte for the source and 4 bytes for the destination. The final source and destination addresses are adjusted to return to their beginning values.

```
TCD.CITER = TCD.BITER = 1
TCD.NBYTES = 16
TCD.SADDR = 0x1000
TCD.SOFF = 1
TCD.SSIZE = 0
TCD.SLAST = -16
TCD.DADDR = 0x2000
TCD.DOFF = 4
TCD.DSIZE = 2
TCD.DLAST_SGA = -16
TCD.INT_MAJ = 1
TCD.START = 1 (Must be written last after all other fields have been initialized)
All other TCD fields = 0
```

This would generate the following sequence of events:

1. Slave write to the TCD.START bit requests channel service.
2. The channel is selected by arbitration for servicing.
3. eDMA engine writes: TCD.DONE = 0, TCD.START = 0, TCD.ACTIVE = 1.
4. eDMA engine reads: channel TCD data from local memory to internal register file.
5. The source to destination transfers are executed as follows:
  - a) read\_byte(0x1000), read\_byte(0x1001), read\_byte(0x1002), read\_byte(0x1003)
  - b) write\_word(0x2000) → first iteration of the minor loop
  - c) read\_byte(0x1004), read\_byte(0x1005), read\_byte(0x1006), read\_byte(0x1007)
  - d) write\_word(0x2004) → second iteration of the minor loop
  - e) read\_byte(0x1008), read\_byte(0x1009), read\_byte(0x100a), read\_byte(0x100b)
  - f) write\_word(0x2008) → third iteration of the minor loop

- g) read\_byte(0x100c), read\_byte(0x100d), read\_byte(0x100e), read\_byte(0x100f)
- h) write\_word(0x200c) → last iteration of the minor loop → major loop complete
- 6. eDMA engine writes: TCD.SADDR = 0x1000, TCD.DADDR = 0x2000, TCD.CITER = 1 (TCD.BITER).
- 7. eDMA engine writes: TCD.ACTIVE = 0, TCD.DONE = 1, EDMA\_IRQR<sub>n</sub> = 1.
- 8. The channel retires.

The eDMA goes idle or services the next channel.

### 16.5.5.2 Multiple requests

The next example is the same as previous, excepting transferring 32 bytes via two hardware requests. The only fields that change are the major loop iteration count and the final address offsets. The eDMA is programmed for two iterations of the major loop transferring 16 bytes per iteration. After the channel's hardware requests are enabled in the EDMA\_ERQR, channel service requests are initiated by the slave device (ERQR should be set after TCD). Note that TCD.START = 0.

```
TCD.CITER = TCD.BITER = 2
TCD.NBYTES = 16
TCD.SADDR = 0x1000
TCD.SOFF = 1
TCD.SSIZE = 0
TCD.SLAST = -32
TCD.DADDR = 0x2000
TCD.DOFF = 4
TCD.DSIZE = 2
TCD.DLAST_SGA = -32
TCD.INT_MAJ = 1
TCD.START = 0 (Must be written last after all other fields have been initialized)
All other TCD fields = 0
```

This generates the following sequence of events:

1. First hardware (eDMA peripheral request) request for channel service.
2. The channel is selected by arbitration for servicing.
3. eDMA engine writes: TCD.DONE = 0, TCD.START = 0, TCD.ACTIVE = 1.
4. eDMA engine reads: channel TCD data from local memory to internal register file.
5. The source to destination transfers are executed as follows:
  - a) read\_byte(0x1000), read\_byte(0x1001), read\_byte(0x1002), read\_byte(0x1003)
  - b) write\_word(0x2000) → first iteration of the minor loop
  - c) read\_byte(0x1004), read\_byte(0x1005), read\_byte(0x1006), read\_byte(0x1007)
  - d) write\_word(0x2004) → second iteration of the minor loop



- e) read\_byte(0x1008), read\_byte(0x1009), read\_byte(0x100a), read\_byte(0x100b)
  - f) write\_word(0x2008) → third iteration of the minor loop
  - g) read\_byte(0x100c), read\_byte(0x100d), read\_byte(0x100e), read\_byte(0x100f)
  - h) write\_word(0x200c) → last iteration of the minor loop
6. eDMA engine writes: TCD.SADDR = 0x1010, TCD.DADDR = 0x2010, TCD.CITER = 1.
  7. eDMA engine writes: TCD.ACTIVE = 0.
  8. The channel retires → one iteration of the major loop.

The eDMA goes idle or services the next channel.

9. Second hardware (eDMA peripheral request) requests channel service.
10. The channel is selected by arbitration for servicing.
11. eDMA engine writes: TCD.DONE = 0, TCD.START = 0, TCD.ACTIVE = 1.
12. eDMA engine reads: channel TCD data from local memory to internal register file.
13. The source to destination transfers are executed as follows:
  - a) read\_byte(0x1010), read\_byte(0x1011), read\_byte(0x1012), read\_byte(0x1013)
  - b) write\_word(0x2010) → first iteration of the minor loop
  - c) read\_byte(0x1014), read\_byte(0x1015), read\_byte(0x1016), read\_byte(0x1017)
  - d) write\_word(0x2014) → second iteration of the minor loop
  - e) read\_byte(0x1018), read\_byte(0x1019), read\_byte(0x101a), read\_byte(0x101b)
  - f) write\_word(0x2018) → third iteration of the minor loop
  - g) read\_byte(0x101c), read\_byte(0x101d), read\_byte(0x101e), read\_byte(0x101f)
  - h) write\_word(0x201c) → last iteration of the minor loop → major loop complete
14. eDMA engine writes: TCD.SADDR = 0x1000, TCD.DADDR = 0x2000, TCD.CITER = 2 (TCD.BITER).
15. eDMA engine writes: TCD.ACTIVE = 0, TCD.DONE = 1, EDMA\_IRQR<sub>n</sub> = 1.
16. The channel retires → major loop complete.

The eDMA goes idle or services the next channel.

### 16.5.5.3 Modulo feature

The modulo feature of the eDMA provides the ability to implement a circular data queue in which the size of the queue is a power of two. MOD is a 5-bit field for both the source and destination in the TCD and specifies which lower address bits are allowed to increment from their original value after the address + offset calculation. All upper address bits remain the same as in the original value. A setting of 0 for this field disables the modulo feature.

Table 16-22 shows how the transfer addresses are specified based on the setting of the MOD field. Here a circular buffer is created where the address wraps to the original value while the 28 upper address bits (0x1234567x) retain their original value. In this example the source address is set to 0x12345670, the offset is set to 4 bytes and the mod field is set to 4, allowing for a 2<sup>4</sup> byte (16-byte) size queue.

Table 16-22. Modulo Feature Example

| Transfer Number | Address    |
|-----------------|------------|
| 1               | 0x12345670 |
| 2               | 0x12345674 |
| 3               | 0x12345678 |
| 4               | 0x1234567C |
| 5               | 0x12345670 |
| 6               | 0x12345674 |

## 16.5.6 TCD status

### 16.5.6.1 Minor loop complete

There are two methods to test for minor loop completion when using software initiated service requests. The first method is to read the TCD.CITER field and test for a change. Another method may be extracted from the sequence below. The second method is to test the TCD.START bit AND the TCD.ACTIVE bit. The minor loop complete condition is indicated by both bits reading zero after the TCD.START was written to a 1. Polling the TCD.ACTIVE bit may be inconclusive because the active status may be missed if the channel execution is short in duration.

The TCD status bits execute the following sequence for a software activated channel:

1. TCD.START = 1, TCD.ACTIVE = 0, TCD.DONE = 0 (channel service request via software).
2. TCD.START = 0, TCD.ACTIVE = 1, TCD.DONE = 0 (channel is executing).
3. TCD.START = 0, TCD.ACTIVE = 0, TCD.DONE = 0 (channel has completed the minor loop and is idle), or
4. TCD.START = 0, TCD.ACTIVE = 0, TCD.DONE = 1 (channel has completed the major loop and is idle).

The best method to test for minor loop completion when using hardware initiated service requests is to read the TCD.CITER field and test for a change. The hardware request and acknowledge handshakes signals are not visible in the programmer's model.

The TCD status bits execute the following sequence for a hardware activated channel:

1. eDMA peripheral request asserts (channel service request via hardware).
2. TCD.START = 0, TCD.ACTIVE = 1, TCD.DONE = 0 (channel is executing).
3. TCD.START = 0, TCD.ACTIVE = 0, TCD.DONE = 0 (channel has completed the minor loop and is idle), or
4. TCD.START = 0, TCD.ACTIVE = 0, TCD.DONE = 1 (channel has completed the major loop and is idle).

For both activation types, the major loop complete status is explicitly indicated via the TCD.DONE bit.

The TCD.START bit is cleared automatically when the channel begins execution, regardless of how the channel was activated.

### 16.5.6.2 Active channel TCD reads

The eDMA will read back the true TCD.SADDR, TCD.DADDR, and TCD.NBYTES values if read while a channel is executing. The true values of the SADDR, DADDR, and NBYTES are the values the eDMA engine is currently using in its internal register file and not the values in the TCD local memory for that channel. The addresses (SADDR and DADDR) and NBYTES (decrements to 0 as the transfer progresses) can give an indication of the progress of the transfer. All other values are read back from the TCD local memory.

### 16.5.6.3 Preemption status

Preemption is available only when fixed arbitration is selected for channel-arbitration mode. A preemptable situation is one in which a preempt-enabled channel is running and a higher priority request becomes active. When the eDMA engine is not operating in fixed-channel arbitration mode, the determination of the relative priority of the actively running and the outstanding requests become undefined. Channel priorities are treated as equal (or more exactly, constantly rotating) when round-robin arbitration mode is selected.

The TCD.ACTIVE bit for the preempted channel remains asserted throughout the preemption. The preempted channel is temporarily suspended while the preempting channel executes one iteration of the major loop. Two TCD.ACTIVE bits set at the same time in the overall TCD map indicates a higher priority channel is actively preempting a lower priority channel.

## 16.5.7 Channel linking

Channel linking (or chaining) is a mechanism in which one channel sets the TCD.START bit of another channel (or itself), thus initiating a service request for that channel. This operation is automatically performed by the eDMA engine at the conclusion of the major or minor loop when properly enabled.

The minor loop channel linking occurs at the completion of the minor loop (or one iteration of the major loop). The TCD.CITER.E\_LINK field are used to determine whether a minor loop link is requested. When enabled, the channel link is made after each iteration of the minor loop except for the last. When the major loop is exhausted, only the major loop channel link fields are used to determine if a channel link should be made. For example, with the initial fields of:

```
TCD.CITER.E_LINK = 1
TCD.CITER.LINKCH = 0xC
TCD.CITER value = 0x4
TCD.MAJOR.E_LINK = 1
TCD.MAJOR.LINKCH = 0x7
```

will execute as:

1. Minor loop done → set channel 12 TCD.START bit
2. Minor loop done → set channel 12 TCD.START bit

3. Minor loop done → set channel 12 TCD.START bit
4. Minor loop done, major loop done → set channel 7 TCD.START bit

When minor loop linking is enabled (TCD.CITER.E\_LINK = 1), the TCD.CITER field uses a nine bit vector to form the current iteration count.

When minor loop linking is disabled (TCD.CITER.E\_LINK = 0), the TCD.CITER field uses a 15-bit vector to form the current iteration count. The bits associated with the TCD.CITER.LINKCH field are concatenated onto the CITER value to increase the range of the CITER.

#### NOTE

After configuration, the TCD.CITER.E\_LINK bit and the TCD.BITER.E\_LINK bit must be equal or a configuration error will be reported. The CITER and BITER vector widths must be equal to calculate the major loop, halfway done interrupt point.

[Table 16-23](#) summarizes how a DMA channel can link to another DMA channel, i.e, use another channel's TCD, at the end of a loop.

**Table 16-23. Channel linking parameters**

| Desired Link Behavior     | TCD Control Field Name | Description   |
|---------------------------|------------------------|---|
| Link at end of minor loop | citer.e_link           | Enable channel-to-channel linking on minor loop completion (current iteration). |
|                           | citer.linkch           | Link channel number when linking at end of minor loop (current iteration).      |
| Link at end of major loop | major.e_link           | Enable channel-to-channel linking on major loop completion.                     |
|                           | major.linkch           | Link channel number when linking at end of major loop.                          |

## 16.5.8 Dynamic programming

### 16.5.8.1 Dynamic channel linking

Dynamic channel linking is the process of setting the TCD.major.e\_link bit during channel execution. This bit is read from the TCD local memory at the end of channel execution, thus allowing the user to enable the feature during channel execution.

Because the user is allowed to change the configuration during execution, a coherency model is needed. Consider the scenario where the user attempts to execute a dynamic channel link by enabling the TCD.major.e\_link bit at the same time the eDMA engine is retiring the channel. The TCD.major.e\_link would be set in the programmer's model, but it would be unclear whether the actual link was made before the channel retired.

The coherency model in [Table 16-24](#) is recommended when executing a dynamic channel link request.

**Table 16-24. Coherency model for a dynamic channel link request**

| Step | Action   |
|------|--|
| 1    | Write 1b to the TCD.major.e_link bit.  |
| 2    | Read back the TCD.major.e_link bit.  |
| 3    | Test the TCD.major.e_link request status: <ul style="list-style-type: none"> <li>• If TCD.major.e_link = 1b, the dynamic link attempt was successful.</li> <li>• If TCD.major.e_link = 0b, the attempted dynamic link did not succeed (the channel was already retiring).</li> </ul> |

For this request, the TCD local memory controller forces the TCD.major.e\_link bit to zero on any writes to a channel's TCD.word7 after that channel's TCD.done bit is set, indicating the major loop is complete.

**NOTE**

The user must clear the TCD.done bit before writing the TCD.major.e\_link bit. The TCD.done bit is cleared automatically by the eDMA engine after a channel begins execution.

**16.5.8.2 Dynamic scatter/gather**

Dynamic scatter/gather is the process of setting the TCD.e\_sg bit during channel execution. This bit is read from the TCD local memory at the end of channel execution, thus allowing the user to enable the feature during channel execution.

Because the user is allowed to change the configuration during execution, a coherency model is needed. Consider the scenario where the user attempts to execute a dynamic scatter/gather operation by enabling the TCD.e\_sg bit at the same time the eDMA engine is retiring the channel. The TCD.e\_sg would be set in the programmer's model, but it would be unclear whether the actual scatter/gather request was honored before the channel retired.

Two methods for this coherency model are shown in the following subsections. Method 1 has the advantage of reading the major.linkch field and the e\_sg bit with a single read. For both dynamic channel linking and scatter/gather requests, the TCD local memory controller forces the TCD.major.e\_link and TCD.e\_sg bits to zero on any writes to a channel's TCD.word7 if that channel's TCD.done bit is set indicating the major loop is complete.

**NOTE**

The user must clear the TCD.done bit before writing the TCD.major.e\_link or TCD.e\_sg bits. The TCD.done bit is cleared automatically by the eDMA engine after a channel begins execution.

**16.5.8.2.1 Method 1 (channel not using major loop channel linking)**

For a channel not using major loop channel linking, the coherency model in [Table 16-25](#) may be used for a dynamic scatter/gather request.

When the TCD.major.e\_link bit is zero, the TCD.major.linkch field is not used by the eDMA. In this case, the TCD.major.linkch bits may be used for other purposes. This method uses the TCD.major.linkch field as a TCD identification (ID).

**Table 16-25. Coherency model for method 1**

| Step | Action   |
|------|--|
| 1    | When the descriptors are built, write a unique TCD ID in the TCD.major.linkch field for each TCD associated with a channel using dynamic scatter/gather.   |
| 2    | Write 1b to the TCD.d_req bit.<br><b>Note:</b> Should a dynamic scatter/gather attempt fail, setting the d_req bit will prevent a future hardware activation of this channel. This stops the channel from executing with a destination address (daddr) that was calculated using a scatter/gather address (written in the next step) instead of a dlast final offset value.  |
| 3    | Write the TCD.dlast_sga field with the scatter/gather address.   |
| 4    | Write 1b to the TCD.e_sg bit.  |
| 5    | Read back the 16 bit TCD control/status field.   |
| 6    | Test the TCD.e_sg request status and TCD.major.linkch value: <ul style="list-style-type: none"> <li>• If e_sg = 1b, the dynamic link attempt was successful.</li> <li>• If e_sg = 0b and the major.linkch (ID) did not change, the attempted dynamic link did not succeed (the channel was already retiring).</li> <li>• If e_sg = 0b and the major.linkch (ID) changed, the dynamic link attempt was successful (the new TCD's e_sg value cleared the e_sg bit).</li> </ul> |

### 16.5.8.2.2 Method 2 (channel using major loop linking)

For a channel using major loop channel linking, the coherency model in [Table 16-26](#) may be used for a dynamic scatter/gather request. This method uses the TCD.dlast\_sga field as a TCD identification (ID).

**Table 16-26. Coherency model for method 2**

| Step | Action  |
|------|---|
| 1    | Write 1b to the TCD.d_req bit.<br><b>Note:</b> Should a dynamic scatter/gather attempt fail, setting the d_req bit will prevent a future hardware activation of this channel. This stops the channel from executing with a destination address (daddr) that was calculated using a scatter/gather address (written in the next step) instead of a dlast final offset value.   |
| 2    | Write the TCD.dlast_sga field with the scatter/gather address.  |
| 3    | Write 1b to the TCD.e_sg bit.   |
| 4    | Read back the TCD.e_sg bit.   |
| 5    | Test the TCD.e_sg request status: <ul style="list-style-type: none"> <li>• If e_sg = 1b, the dynamic link attempt was successful.</li> <li>• If e_sg = 0b, read the 32 bit TCD dlast_sga field.</li> <li>• If e_sg = 0b and the dlast_sga did not change, the attempted dynamic link did not succeed (the channel was already retiring).</li> <li>• If e_sg = 0b and the dlast_sga changed, the dynamic link attempt was successful (the new TCD's e_sg value cleared the e_sg bit).</li> </ul> |

# Chapter 17

## eDMA Channel Multiplexer (DMA\_MUX)

### 17.1 Introduction

The eDMA channel multiplexer (DMA\_MUX) allows the routing of 59 DMA sources (slots) to 16 eDMA channels. This is illustrated in [Figure 17-1](#).

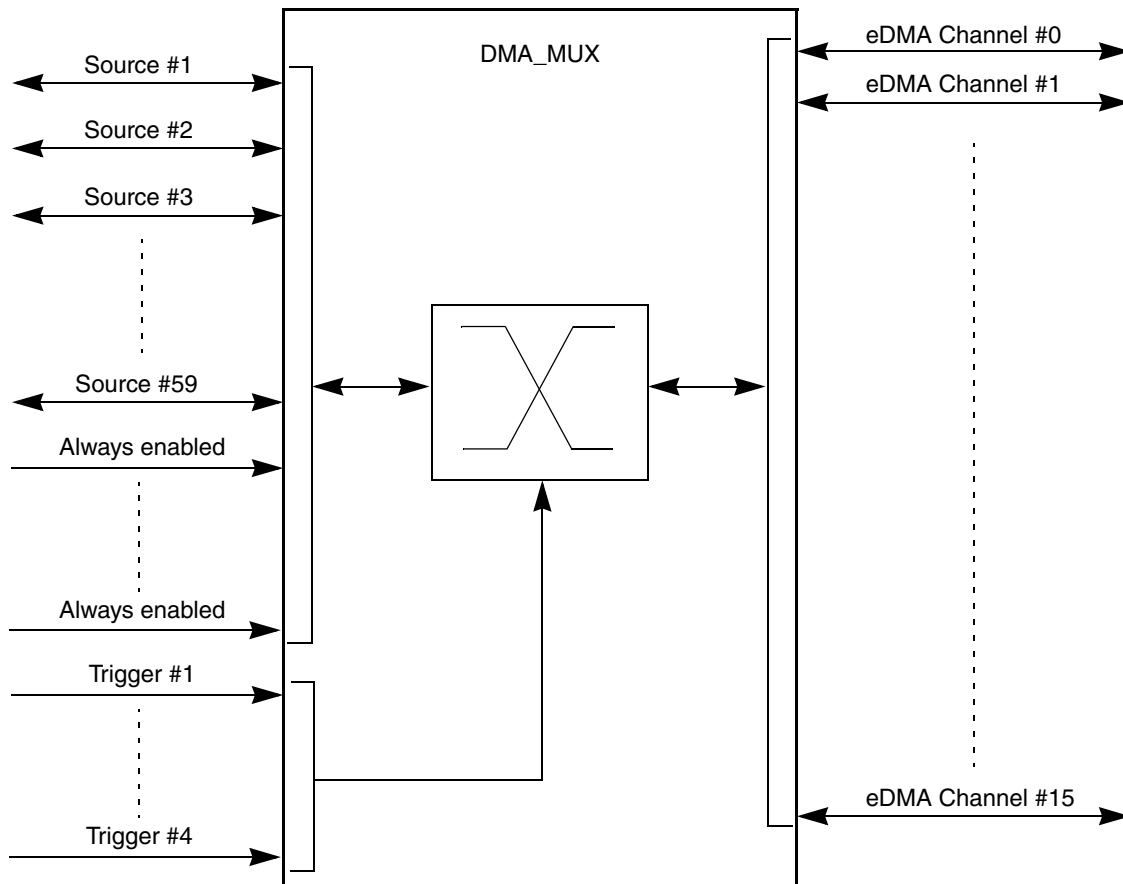


Figure 17-1. DMA\_MUX block diagram

### 17.2 Features

The DMA\_MUX has these major features:

- 16 independently selectable eDMA channel routers
  - Four channels with normal or periodic triggering capability
  - 12 channels with normal capability
- Capability to assign each channel router to one of 59 possible peripheral DMA sources, four always enabled sources, or one always disabled source
- Three modes of operation:

- Disabled
- Normal
- Periodic Trigger

## 17.3 Modes of operation

The following operation modes are available:

- **Disabled Mode** — In this mode, the eDMA channel is disabled. Since disabling and enabling of eDMA channels is done primarily via the eDMA configuration registers, this mode is used mainly as the reset state for an eDMA channel in the DMA\_MUX. It may also be used to temporarily suspend an eDMA channel while reconfiguration of the system takes place (for example, changing the period of an eDMA trigger).
- **Normal Mode** — In this mode, an eDMA source (such as DSPI\_0\_TX or DSPI\_0\_RX example) is routed directly to the specified eDMA channel. The operation of the DMA\_MUX in this mode is completely transparent to the system.
- **Periodic Trigger Mode** — In this mode, an eDMA source may only request an eDMA transfer (such as when a transmit buffer becomes empty or a receive buffer becomes full) periodically. The period is configured in the registers of the Periodic Interrupt Timer (PIT).

eDMA channels 0–3 may be used in all three modes, but channels 4–15 may only be configured to disabled or normal mode.

## 17.4 External signal description

The DMA\_MUX has no external pins.

## 17.5 Memory map and register definition

Table 17-1 shows the memory map for the DMA\_MUX. Note that all addresses are offsets; the absolute address may be computed by adding the specified offset to the base address of the DMA\_MUX.

**Table 17-1. DMA\_MUX memory map**

| Base address: 0xFFFD_C000 |  |                             |
|---------------------------|--|-----------------------------|
| Address offset            | Register                               | Location                    |
| 0x0                       | Channel #0 Configuration (CHCONFIG0)   | <a href="#">on page 297</a> |
| 0x1                       | Channel #1 Configuration (CHCONFIG1)   | <a href="#">on page 297</a> |
| ...                       | ...                                    | ...                         |
| 0xF                       | Channel #15 Configuration (CHCONFIG15) | <a href="#">on page 297</a> |

All registers are accessible via 8-, 16-, or 32-bit accesses. However, 16-bit accesses must be aligned to 16-bit boundaries, and 32-bit accesses must be aligned to 32-bit boundaries. As an example, CHCONFIG0 through CHCONFIG3 are accessible by a 32-bit read/write to address Base + 0x00, but performing a 32-bit access to address Base + 0x01 is illegal.



## 17.5.1 Channel configuration registers (CHCONFIG $n$ )

Each of the total of 16 eDMA channels can be independently enabled/disabled and associated with 1 of the 28 peripheral eDMA sources + 1 of the four always enabled eDMA sources in the system.

Offset:  $0x0 + n$  (16 registers)

Access: User read/write



**Figure 17-2. Channel Configuration Registers (CHCONFIG $n$ )**

**Table 17-2. CHCONFIG $n$  field descriptions**

| Field  | Description   |
|--------|---|
| ENBL   | eDMA Channel Enable<br>ENBL enables the eDMA channel.<br>0 eDMA channel is disabled. This mode is primarily used during configuration of the DMA_MUX.<br>The eDMA has separate channel enables/disables, which should be used to disable or reconfigure an eDMA channel.<br>1 eDMA channel is enabled                                 |
| TRIG   | eDMA Channel Trigger Enable (for triggered channels only)<br>TRIG enables the periodic trigger capability for the eDMA channel.<br>0 Periodic triggering is disabled. If periodic triggering is disabled, and the ENBL bit is set, the DMA_MUX will simply route the specified source to the eDMA channel.<br>1 Triggering is enabled |
| SOURCE | eDMA Channel Source (slot)<br>SOURCE specifies which eDMA source, if any, is routed to a particular eDMA channel. Please see <a href="#">Table 17-4</a> for DMA_MUX inputs mapping.   |

**Table 17-3. Channel and trigger enabling**

| ENBL | TRIG | Function   | Mode                  |
|------|------|--|-----------------------|
| 0    | X    | eDMA channel is disabled                                 | Disabled Mode         |
| 1    | 0    | eDMA channel is enabled with no triggering (transparent) | Normal Mode           |
| 1    | 1    | eDMA channel is enabled with triggering                  | Periodic Trigger Mode |

### NOTE

Setting multiple CHCONFIG registers with the same Source value results in unpredictable behavior.

### NOTE

Before changing the trigger or source settings an eDMA channel must be disabled via the CHCONFIG $n$ [ENBL] bit.

## 17.6 DMA\_MUX inputs

### 17.6.1 DMA\_MUX peripheral sources

Table 17-4. eDMA channel mapping

| DMA_MUX channel | Module  | eDMA requesting module | DMA_MUX input #    |
|-----------------|---------|------------------------|--------------------|
| 0               | —       | Always disabled        | —                  |
| 1               | DSPI 0  | DSPI_0 TX              | DMA_MUX Source #1  |
| 2               | DSPI 0  | DSPI_0 RX              | DMA_MUX Source #2  |
| 3               | DSPI 1  | DSPI_1 TX              | DMA_MUX Source #3  |
| 4               | DSPI 1  | DSPI_1 RX              | DMA_MUX Source #4  |
| 5               | DSPI 2  | DSPI_2 TX              | DMA_MUX Source #5  |
| 6               | DSPI 2  | DSPI_2 RX              | DMA_MUX Source #6  |
| 7               | DSPI 3  | DSPI_3 TX              | DMA_MUX Source #7  |
| 8               | DSPI 3  | DSPI_3 RX              | DMA_MUX Source #8  |
| 9               | DSPI 4  | DSPI_4 TX              | DMA_MUX Source #9  |
| 10              | DSPI 4  | DSPI_4 RX              | DMA_MUX Source #10 |
| 11              | DSPI 5  | DSPI_5 TX              | DMA_MUX Source #11 |
| 12              | DSPI 5  | DSPI_5 RX              | DMA_MUX Source #12 |
| 13              | —       | —                      | DMA_MUX Source #13 |
| 14              | —       | —                      | DMA_MUX Source #14 |
| 15              | —       | —                      | DMA_MUX Source #15 |
| 16              | —       | —                      | DMA_MUX Source #16 |
| 17              | eMIOS 0 | EMIOS0_CH0             | DMA_MUX Source #17 |
| 18              | eMIOS 0 | EMIOS0_CH1             | DMA_MUX Source #18 |
| 19              | eMIOS 0 | EMIOS0_CH9             | DMA_MUX Source #19 |
| 20              | eMIOS 0 | EMIOS0_CH18            | DMA_MUX Source #20 |
| 21              | eMIOS 0 | EMIOS0_CH25            | DMA_MUX Source #21 |
| 22              | eMIOS 0 | EMIOS0_CH26            | DMA_MUX Source #22 |
| 23              | eMIOS 1 | EMIOS1_CH0             | DMA_MUX Source #23 |
| 24              | eMIOS 1 | EMIOS1_CH9             | DMA_MUX Source #24 |
| 25              | eMIOS 1 | EMIOS1_CH17            | DMA_MUX Source #25 |
| 26              | eMIOS 1 | EMIOS1_CH18            | DMA_MUX Source #26 |
| 27              | eMIOS 1 | EMIOS1_CH25            | DMA_MUX Source #27 |
| 28              | eMIOS 1 | EMIOS1_CH26            | DMA_MUX Source #28 |

Table 17-4. eDMA channel mapping (continued)

| DMA_MUX channel | Module           | eDMA requesting module | DMA_MUX input #    |
|-----------------|------------------|------------------------|--------------------|
| 29              | ADC 0            | ADC0_EOC               | DMA_MUX Source #29 |
| 30              | ADC 1            | ADC1_EOC               | DMA_MUX Source #30 |
| 31              | I <sup>2</sup> C | IIC_RX                 | DMA_MUX Source #31 |
| 32              | I <sup>2</sup> C | IIC_TX                 | DMA_MUX Source #32 |
| 33              | LINFLEX 0        | LINFLEX0_RX            | DMA_MUX Source #33 |
| 34              | LINFLEX 0        | LINFLEX0_TX            | DMA_MUX Source #34 |
| 35              | LINFLEX 1        | LINFLEX1_RX            | DMA_MUX Source #35 |
| 36              | LINFLEX 1        | LINFLEX1_TX            | DMA_MUX Source #36 |
| 37              | —                | —                      | DMA_MUX Source #37 |
| 38              | —                | —                      | DMA_MUX Source #38 |
| 39              | —                | —                      | DMA_MUX Source #39 |
| 40              | —                | —                      | DMA_MUX Source #40 |
| 41              | —                | —                      | DMA_MUX Source #41 |
| 42              | —                | —                      | DMA_MUX Source #42 |
| 43              | —                | —                      | DMA_MUX Source #43 |
| 44              | —                | —                      | DMA_MUX Source #44 |
| 45              | —                | —                      | DMA_MUX Source #45 |
| 46              | —                | —                      | DMA_MUX Source #46 |
| 47              | —                | —                      | DMA_MUX Source #47 |
| 48              | —                | —                      | DMA_MUX Source #48 |
| 49              | —                | —                      | DMA_MUX Source #49 |
| 50              | —                | —                      | DMA_MUX Source #50 |
| 51              | —                | —                      | DMA_MUX Source #51 |
| 52              | —                | —                      | DMA_MUX Source #52 |
| 53              | —                | —                      | DMA_MUX Source #53 |
| 54              | —                | —                      | DMA_MUX Source #54 |
| 55              | —                | —                      | DMA_MUX Source #55 |
| 56              | —                | —                      | DMA_MUX Source #56 |
| 57              | —                | —                      | DMA_MUX Source #57 |
| 58              | —                | —                      | DMA_MUX Source #58 |

Table 17-4. eDMA channel mapping (continued)

| DMA_MUX channel | Module | eDMA requesting module | DMA_MUX input #    |
|-----------------|--------|------------------------|--------------------|
| 59              | —      | —                      | DMA_MUX Source #59 |
| 60              | —      | ALWAYS ENABLED         | DMA_MUX Source #60 |
| 61              | —      | ALWAYS ENABLED         | DMA_MUX Source #61 |
| 62              | —      | ALWAYS ENABLED         | DMA_MUX Source #62 |
| 63              | —      | ALWAYS ENABLED         | DMA_MUX Source #63 |

## 17.6.2 DMA\_MUX periodic trigger inputs

Table 17-5. DMA\_MUX periodic trigger inputs

| DMA_MUX trigger input | PIT channel |
|-----------------------|-------------|
| Trigger #1            | PIT0        |
| Trigger #2            | PIT1        |
| Trigger #3            | PIT4        |
| Trigger #4            | PIT5        |

## 17.7 Functional description

The primary purpose of the DMA\_MUX is to provide flexibility in the system's use of the available eDMA channels. As such, configuration of the DMA\_MUX is intended to be a static procedure done during execution of the system boot code. However, if the procedure outlined in [Section 17.8.2, Enabling and configuring sources](#), is followed, the configuration of the DMA\_MUX may be changed during the normal operation of the system.

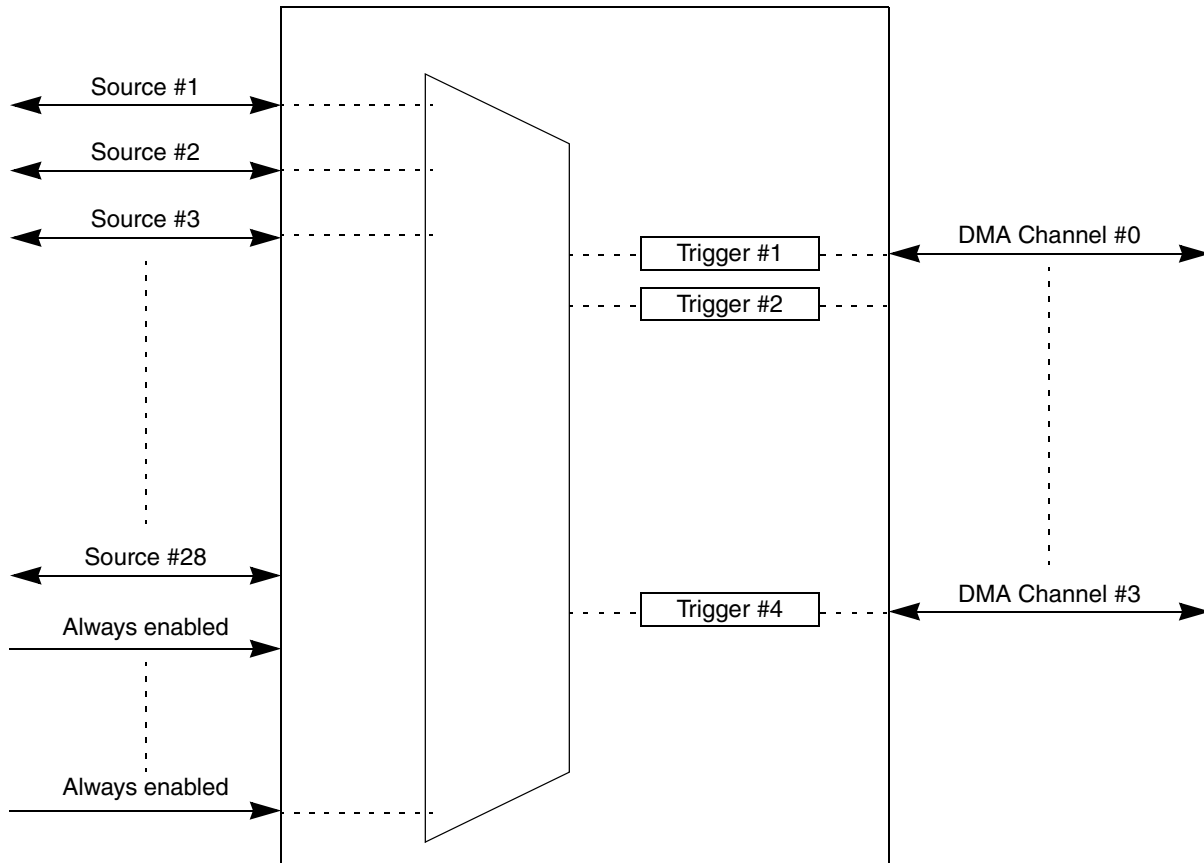
Functionally, the DMA\_MUX channels may be divided into two classes: Channels that implement the normal routing functionality plus periodic triggering capability, and channels, that implement only the normal routing functionality.

### 17.7.1 eDMA channels with periodic triggering capability

Besides the normal routing functionality, the first four channels of the DMA\_MUX provide a special periodic triggering capability that can be used to provide an automatic mechanism to transmit bytes, frames or packets at fixed intervals without the need for processor intervention. The trigger is generated by the periodic interrupt timer (PIT); as such, the configuration of the periodic triggering interval is done via configuration registers in the PIT. Please see [Section 27.5, Periodic Interrupt Timer \(PIT\)](#), for more information on this topic.

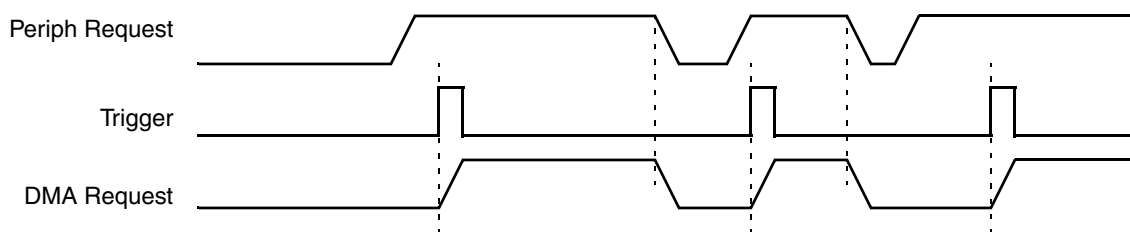
**NOTE**

Because of the dynamic nature of the system (such as eDMA channel priorities, bus arbitration, or interrupt service routine lengths), the number of clock cycles between a trigger and the actual eDMA transfer cannot be guaranteed.



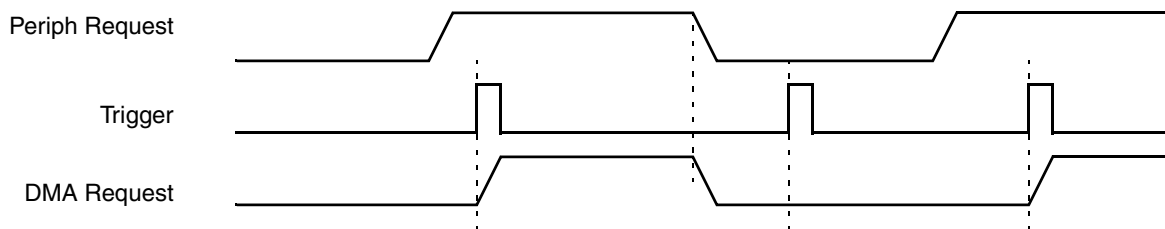
**Figure 17-3. DMA\_MUX channel 0-3 block diagram**

The eDMA channel triggering capability allows the system to schedule regular eDMA transfers, usually on the transmit side of certain peripherals, without the intervention of the processor. This trigger works by gating the request from the peripheral to the eDMA until a trigger event has been seen. This is illustrated in [Figure 17-4](#).



**Figure 17-4. DMA\_MUX channel triggering: Normal operation**

Once the eDMA request has been serviced, the peripheral will negate its request, effectively resetting the gating mechanism until the peripheral reasserts its request AND the next trigger event is seen. This means that if a trigger is seen, but the peripheral is not requesting a transfer, that triggered will be ignored. This situation is illustrated in [Figure 17-5](#).



**Figure 17-5. DMA\_MUX channel triggering: Ignored trigger**

This triggering capability may be used with any peripheral that supports eDMA transfers, and is most useful for periodically polling external devices on a particular bus.

As an example, the transmit side of a DSPI is assigned to an eDMA channel with a trigger, as described above. Once set up, the SPI will request eDMA transfers (presumably from memory) as long as its transmit buffer is empty. By using a trigger on this channel, the DSPI transfers can be automatically performed every 5  $\mu$ s (as an example). On the receive side of the SPI, the SPI and eDMA can be configured to transfer receive data into memory, effectively implementing a method to periodically read data from external devices and transfer the results into memory without processor intervention.

A more detailed description of the capability of each trigger (such as resolution, or range of values) may be found in [Chapter 27, Timers](#).

### 17.7.2 eDMA channels with no triggering capability

Channels 4–15 of the DMA\_MUX provide the normal routing functionality as described in [Section 17.3, Modes of operation](#).

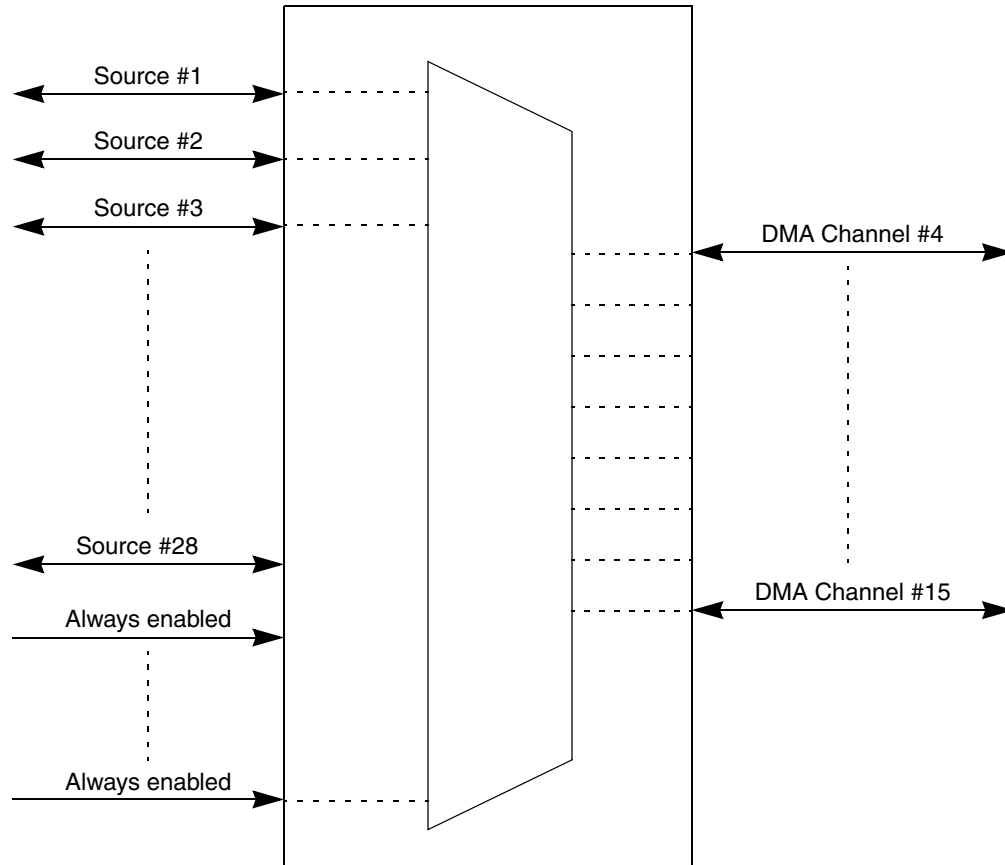


Figure 17-6. DMA\_MUX channel 4–15 block diagram

## 17.8 Initialization/Application information

### 17.8.1 Reset

The reset state of each individual bit is shown in [Section 17.5, Memory map and register definition](#). In summary, after reset, all channels are disabled and must be explicitly enabled before use.

### 17.8.2 Enabling and configuring sources

#### 17.8.2.1 Enabling a source with periodic triggering

The following describes how to enable a source with periodic triggering:

1. Determine with which eDMA channel the source will be associated. Remember that only the first four eDMA channels have periodic triggering capability.
2. Clear the ENBL and TRIG bits of the eDMA channel.
3. Ensure that the eDMA channel is properly configured in the eDMA. The eDMA channel may be enabled at this point.

4. In the PIT, configure the corresponding timer.
5. Select the source to be routed to the eDMA channel. Write to the corresponding CHCONFIG register, ensuring that the ENBL and TRIG bits are set.

**Example 17-1. Configure source #3 Transmit for use with eDMA Channel 2, with periodic triggering capability**

1. Write 0x00 to CHCONFIG2 (Base Address + 0x02)
2. Configure Channel 2 in the eDMA, including enabling the channel
3. Configure Timer 4 in the Periodic Interrupt Timer (PIT) for the desired trigger interval
4. Write 0xC3 to CHCONFIG2 (Base Address + 0x02)

The following code example illustrates steps #1 and #4 above:

```
In File registers.h:
#define DMAMUX_BASE_ADDR    0xFC084000 /* Example only ! */
/* Following example assumes char is 8 bits */
volatile unsigned char *CHCONFIG2 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0002);

In File main.c:
#include "registers.h"
:
:
*CHCONFIG2 = 0x00;
*CHCONFIG2 = 0xC3;
```

### 17.8.2.2 Enabling a source without periodic triggering

The following describes how to enable a source without periodic triggering:

1. Determine with which eDMA channel the source will be associated. Remember that only eDMA channels 0–3 have periodic triggering capability.
2. Clear the ENBL and TRIG bits of the eDMA channel.
3. Ensure that the eDMA channel is properly configured in the eDMA. The eDMA channel may be enabled at this point.
4. Select the source to be routed to the eDMA channel. Write to the corresponding CHCONFIG register, ensuring that the ENBL is set and the TRIG bit is cleared.

**Example 17-2. Configure source #5 Transmit for use with eDMA Channel 2, without periodic triggering capability**

1. Write 0x00 to CHCONFIG2 (Base Address + 0x02)
2. Configure Channel 2 in the eDMA, including enabling the channel
3. Write 0x85 to CHCONFIG2 (Base Address + 0x02)

The following code example illustrates steps #1 and #3 above:

```
In File registers.h:
#define DMAMUX_BASE_ADDR    0xFC084000 /* Example only ! */
/* Following example assumes char is 8 bits */
volatile unsigned char *CHCONFIG2 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0002);

In File main.c:
```



```
#include "registers.h"
:
:
*CHCONFIG2 = 0x00;
*CHCONFIG2 = 0x85;
```

### 17.8.2.3 Disabling a source

A particular eDMA source may be disabled by not writing the corresponding source value into any of the CHCONFIG registers. Additionally, some module specific configuration may be necessary. Please refer to the appropriate section for more details.

### 17.8.2.4 Switching the source of an eDMA channel

The following describes how to switch the source of an eDMA channel:

1. Disable the eDMA channel in the eDMA and reconfigure the channel for the new source.
2. Clear the ENBL and TRIG bits of the eDMA channel.
3. Select the source to be routed to the eDMA channel. Write to the corresponding CHCONFIG register, ensuring that the ENBL and TRIG bits are set.

---

#### Example 17-3. Switch eDMA Channel 8 from source #5 transmit to source #7 transmit

---

1. In the eDMA configuration registers, disable eDMA channel 8 and reconfigure it to handle the transfers to peripheral slot 7. This example assumes channel 8 doesn't have triggering capability.
2. Write 0x00 to CHCONFIG8 (Base Address + 0x08)
3. Write 0x87 to CHCONFIG8 (Base Address + 0x08).

The following code example illustrates steps #2 and #3 above:

```
In File registers.h:
#define DMAMUX_BASE_ADDR      0xFC084000 /* Example only ! */
/* Following example assumes char is 8 bits */
volatile unsigned char *CHCONFIG8 = (volatile unsigned char *) (DMAMUX_BASE_ADDR+0x0008);

In File main.c:
#include "registers.h"
:
:
*CHCONFIG8 = 0x00;
*CHCONFIG8 = 0x87;
```

This page is intentionally left blank.

# Chapter 18

## Interrupt Controller (INTC)

### 18.1 Introduction

The INTC provides priority-based preemptive scheduling of interrupt service requests (ISRs). This scheduling scheme is suitable for statically scheduled hard real-time systems. The INTC supports 204 interrupt requests. It is targeted to work with a Power Architecture technology processor and automotive powertrain applications where the ISRs nest to multiple levels, but it also can be used with other processors and applications.

For high priority interrupt requests in these target applications, the time from the assertion of the peripheral's interrupt request from the peripheral to when the processor is performing useful work to service the interrupt request needs to be minimized. The INTC supports this goal by providing a unique vector for each interrupt request source. It also provides 16 priorities so that lower priority ISRs do not delay the execution of higher priority ISRs. Since each individual application will have different priorities for each source of interrupt request, the priority of each interrupt request is configurable.

When multiple tasks share a resource, coherent accesses to that resource need to be supported. The INTC supports the priority ceiling protocol for coherent accesses. By providing a modifiable priority mask, the priority can be raised temporarily so that all tasks that share the resource cannot preempt each other.

Multiple processors can assert interrupt requests to each other through software configurable interrupt requests. These same software configurable interrupt requests also can be used to break the work involved in servicing an interrupt request into a high priority portion and a low priority portion. The high priority portion is initiated by a peripheral interrupt request, but then the ISR can assert a software configurable interrupt request to finish the servicing in a lower priority ISR. Therefore these software configurable interrupt requests can be used instead of the peripheral ISR scheduling a task through the RTOS.

### 18.2 Features

- Supports 196 peripheral and 8 software-configurable interrupt request sources
- Unique 9-bit vector per interrupt source
- Each interrupt source can be programmed to one of 16 priorities
- Preemption
  - Preemptive prioritized interrupt requests to processor
  - ISR at a higher priority preempts ISRs or tasks at lower priorities
  - Automatic pushing or popping of preempted priority to or from a LIFO
  - Ability to modify the ISR or task priority; modifying the priority can be used to implement the priority ceiling protocol for accessing shared resources.
- Low latency – 3 clocks from receipt of interrupt request from peripheral to interrupt request to processor

Table 18-1. Interrupt sources available

| Interrupt sources (204)             | Number available |
|-------------------------------------|------------------|
| Software                            | 8                |
| ECSM                                | 1                |
| eDMA                                | 17               |
| Software Watchdog (SWT)             | 1                |
| STM                                 | 4                |
| Flash/SRAM ECC (SEC-DED)            | 2                |
| Real Time Counter (RTC/API)         | 2                |
| System Integration Unit Lite (SIUL) | 3                |
| WKPU                                | 4                |
| MC_ME                               | 4                |
| MC_RGM                              | 1                |
| FXOSC                               | 1                |
| SXOSC                               | 1                |
| PIT                                 | 8                |
| ADC_0                               | 2                |
| ADC_1                               | 2                |
| FlexCAN_0                           | 8                |
| FlexCAN_1                           | 8                |
| FlexCAN_2                           | 8                |
| FlexCAN_3                           | 8                |
| FlexCAN_4                           | 8                |
| FlexCAN_5                           | 8                |
| LINFlex_0                           | 3                |
| LINFlex_1                           | 3                |
| LINFlex_2                           | 3                |
| LINFlex_3                           | 3                |
| LINFlex_4                           | 3                |
| LINFlex_5                           | 3                |
| LINFlex_6                           | 3                |
| LINFlex_7                           | 3                |
| DSPI_0                              | 5                |
| DSPI_1                              | 5                |
| DSPI_2                              | 5                |

Table 18-1. Interrupt sources available (continued)

| Interrupt sources (204)                    | Number available |
|--|------------------|
| DSPI_3                                     | 5                |
| DSPI_4                                     | 5                |
| DSPI_5                                     | 5                |
| I2C_0                                      | 1                |
| Enhanced Modular I/O Subsystem 0 (eMIOS_0) | 16               |
| eMIOS_1                                    | 16               |

## 18.3 Block diagram

Figure 18-1 provides a block diagram of the INTC.

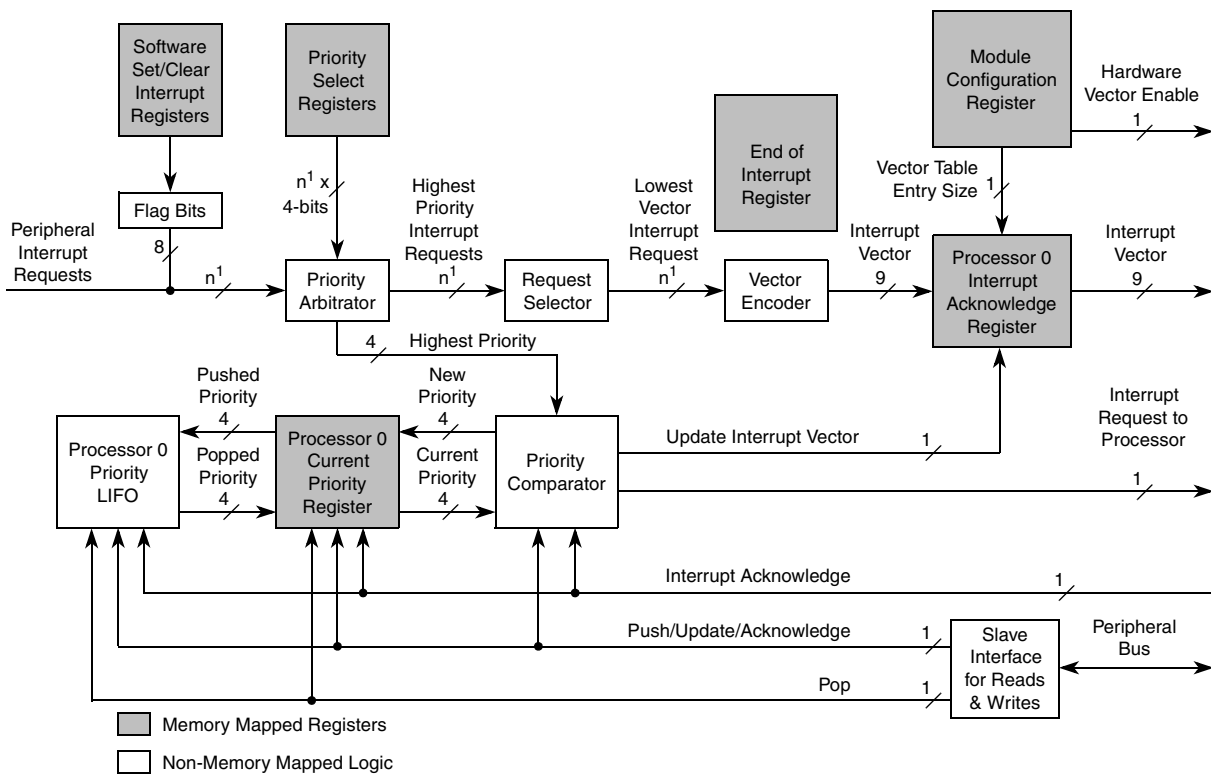


Figure 18-1. INTC block diagram

## 18.4 Modes of operation

### 18.4.1 Normal mode

In normal mode, the INTC has two handshaking modes with the processor: software vector mode and hardware vector mode.

### 18.4.1.1 Software vector mode

In software vector mode, software, that is the interrupt exception handler, must read a register in the INTC to obtain the vector associated with the interrupt request to the processor. The INTC will use software vector mode for a given processor when its associated HVEN bit in INTC\_MCR is negated. The hardware vector enable signal to processor 0 or processor 1 is driven as negated when its associated HVEN bit is negated. The vector is read from INC\_IACKR. Reading the INTC\_IACKR negates the interrupt request to the associated processor. Even if a higher priority interrupt request arrived while waiting for this interrupt acknowledge, the interrupt request to the processor will negate for at least one clock. The reading also pushes the PRI value in INTC\_CPR onto the associated LIFO and updates PRI in the associated INTC\_CPR with the new priority.

Furthermore, the interrupt vector to the processor is driven as all 0s. The interrupt acknowledge signal from the associated processor is ignored.

### 18.4.1.2 Hardware vector mode

In hardware vector mode, the hardware is the interrupt vector signal from the INTC in conjunction with a processor with the capability use that vector. In hardware vector mode, this hardware causes the first instruction to be executed in handling the interrupt request to the processor to be specific to that vector. Therefore the interrupt exception handler is specific to a peripheral or software configurable interrupt request rather than being common to all of them. The INTC uses hardware vector mode for a given processor when the associated HVEN bit in the INTC\_MCR is asserted. The hardware vector enable signal to the associated processor is driven as asserted. When the interrupt request to the associated processor asserts, the interrupt vector signal is updated. The value of that interrupt vector is the unique vector associated with the preempting peripheral or software configurable interrupt request. The vector value matches the value of the INTVEC field in the INTC\_IACKR field in the INTC\_IACKR, depending on which processor was assigned to handle a given interrupt source.

The processor negates the interrupt request to the processor driven by the INTC by asserting the interrupt acknowledge signal for one clock. Even if a higher priority interrupt request arrived while waiting for the interrupt acknowledge, the interrupt request to the processor will negate for at least one clock.

The assertion of the interrupt acknowledge signal for a given processor pushes the associated PRI value in the associated INTC\_CPR register onto the associated LIFO and updates the associated PRI in the associated INTC\_CPR register with the new priority. This pushing of the PRI value onto the associated LIFO and updating PRI in the associated INTC\_CPR does not occur when the associated interrupt acknowledge signal asserts and INTC\_SSCIR0\_3–INTC\_SSCIR4\_7 is written at a time such that the PRI value in the associated INTC\_CPR register would need to be pushed and the previously last pushed PRI value would need to be popped simultaneously. In this case, PRI in the associated INTC\_CPR is updated with the new priority, and the associated LIFO is neither pushed or popped.

### 18.4.1.3 Debug mode

The INTC operation in debug mode is identical to its operation in normal mode.

### 18.4.1.4 Stop mode

The INTC supports STOP mode. The INTC can have its clock input disabled at any time by the clock driver on the device. While its clocks are disabled, the INTC registers are not accessible.

The INTC requires clocking in order for a peripheral interrupt request to generate an interrupt request to the processor. Since the INTC is not clocked in STOP mode, peripheral interrupt requests cannot be used as a wakeup source, unless the device supports that interrupt request as a wakeup source.

## 18.5 Memory map and register description

### 18.5.1 Module memory map

Table 18-2 shows the INTC memory map.

Table 18-2. INTC memory map

| Base address: 0xFFF4_8000 |   |                             |
|---------------------------|---|-----------------------------|
| Address offset            | Register  | Location                    |
| 0x0000                    | INTC Module Configuration Register (INTC_MCR)                             | <a href="#">on page 312</a> |
| 0x0004                    | Reserved  |                             |
| 0x0008                    | INTC Current Priority Register for Processor (INTC_CPR)                   | <a href="#">on page 312</a> |
| 0x000C                    | Reserved  |                             |
| 0x0010                    | INTC Interrupt Acknowledge Register (INTC_IACKR)                          | <a href="#">on page 314</a> |
| 0x0014                    | Reserved  |                             |
| 0x0018                    | INTC End-of-Interrupt Register (INTC_EOIR)                                | <a href="#">on page 315</a> |
| 0x001C                    | Reserved  |                             |
| 0x0020–0x0027             | INTC Software Set/Clear Interrupt Registers (INTC_SSCIR0_3–INTC_SSCIR4_7) | <a href="#">on page 316</a> |
| 0x0028–0x003C             | Reserved  |                             |
| 0x0040–0x00D0             | INTC Priority Select Registers (INTC_PSR0_3–INTC_PSR232_233) <sup>1</sup> | <a href="#">on page 317</a> |

<sup>1</sup> The PRI fields are reserved for peripheral interrupt requests whose vectors are labeled Reserved in [Figure 18-3](#).

### 18.5.2 Register description

With exception of the INTC\_SSCIR $n$  and INTC\_PSR $n$ , all registers are 32 bits in width. Any combination of accessing the 4 bytes of a register with a single access is supported, provided that the access does not cross a register boundary. These supported accesses include types and sizes of 8 bits, aligned 16 bits, misaligned 16 bits to the middle 2 bytes, and aligned 32 bits.

Although INTC\_SSCIR $n$  and INTC\_PSR $n$  are 8 bits wide, they can be accessed with a single 16-bit or 32-bit access, provided that the access does not cross a 32-bit boundary.

In software vector mode, the side effects of a read of INTC\_IACKR are the same regardless of the size of the read. In either software or hardware vector mode, the size of a write to either INTC\_SSCIR0\_3–INTC\_SSCIR4\_7 or INTC\_EOIR does not affect the operation of the write.

### 18.5.2.1 INTC Module Configuration Register (INTC\_MCR)

The module configuration register is used to configure options of the INTC.

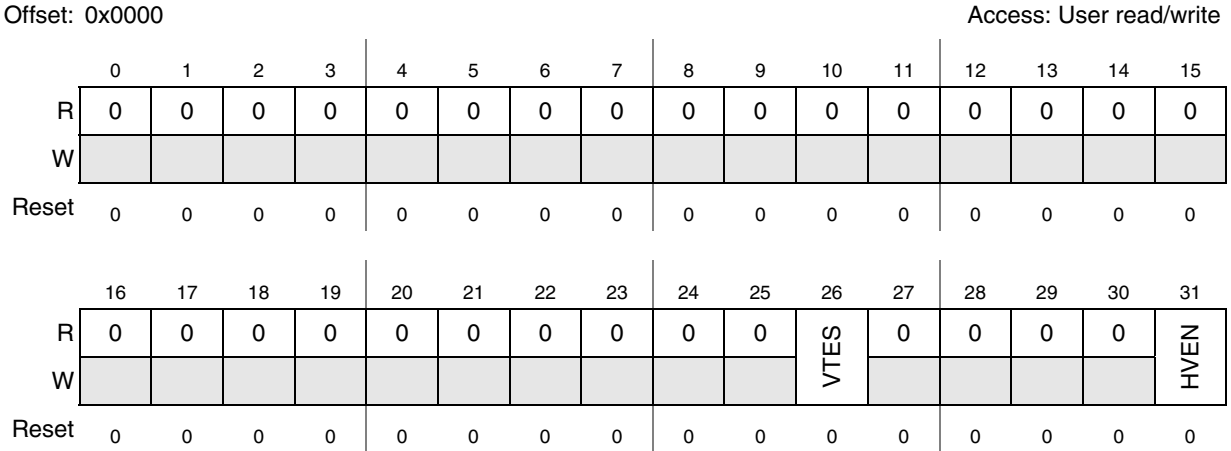


Figure 18-2. INTC Module Configuration Register (INTC\_MCR)

Table 18-3. INTC\_MCR field descriptions

| Field | Description   |
|-------|---|
| VTES  | Vector table entry size.<br>Controls the number of 0s to the right of INTVEC in <a href="#">Section 18.5.2.3, INTC Interrupt Acknowledge Register (INTC_IACKR)</a> . If the contents of INTC_IACKR are used as an address of an entry in a vector table as in software vector mode, then the number of rightmost 0s will determine the size of each vector table entry. VTES impacts software vector mode operation but also affects INTC_IACKR[INTVEC] position in both hardware vector mode and software vector mode.<br>0 4 bytes<br>1 8 bytes |
| HVEN  | Hardware vector enable.<br>Controls whether the INTC is in hardware vector mode or software vector mode. Refer to <a href="#">Section 18.4, Modes of operation</a> , for the details of the handshaking with the processor in each mode.<br>0 Software vector mode<br>1 Hardware vector mode  |

### 18.5.2.2 INTC Current Priority Register for Processor (INTC\_CPR)



Offset: 0x0008

Access: User read/write

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |   |   |   |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |   |   |   |
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |   |   |   |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |   |   |   |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 1  | 1 | 1 | 1 |

Figure 18-3. INTC Current Priority Register (INTC\_CPR)

Table 18-4. INTC\_CPR field descriptions

| Field | Description  |
|-------|--|
| PRI   | Priority<br>PRI is the priority of the currently executing ISR according to the field values defined in <a href="#">Table 18-5</a> . |

The INTC\_CPR masks any peripheral or software configurable interrupt request set at the same or lower priority as the current value of the INTC\_CPR[PRI] field from generating an interrupt request to the processor. When the INTC interrupt acknowledge register (INTC\_IACKR) is read in software vector mode or the interrupt acknowledge signal from the processor is asserted in hardware vector mode, the value of PRI is pushed onto the LIFO, and PRI is updated with the priority of the preempting interrupt request. When the INTC end-of-interrupt register (INTC\_EOIR) is written, the LIFO is popped into the INTC\_CPR's PRI field.

The masking priority can be raised or lowered by writing to the PRI field, supporting the PCP. Refer to [Section 18.7.5, Priority ceiling protocol](#).

#### NOTE

A store to modify the PRI field that closely precedes or follows an access to a shared resource can result in a non-coherent access to that resource. Refer to [Section 18.7.5.2, Ensuring coherency](#), for example code to ensure coherency.

Table 18-5. PRI values

| PRI  | Meaning                      |
|------|------------------------------|
| 1111 | Priority 15—highest priority |
| 1110 | Priority 14                  |
| 1101 | Priority 13                  |
| 1100 | Priority 12                  |
| 1011 | Priority 11                  |
| 1010 | Priority 10                  |
| 1001 | Priority 9                   |
| 1000 | Priority 8                   |
| 0111 | Priority 7                   |

Table 18-5. PRI values (continued)

| PRI  | Meaning                    |
|------|----------------------------|
| 0110 | Priority 6                 |
| 0101 | Priority 5                 |
| 0100 | Priority 4                 |
| 0011 | Priority 3                 |
| 0010 | Priority 2                 |
| 0001 | Priority 1                 |
| 0000 | Priority 0—lowest priority |

### 18.5.2.3 INTC Interrupt Acknowledge Register (INTC\_IACKR)

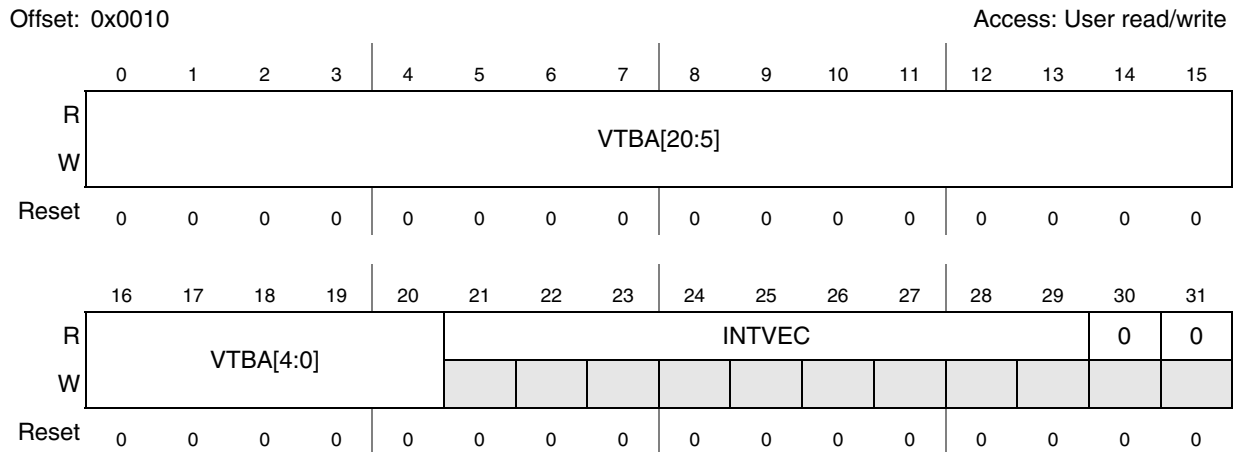


Figure 18-4. INTC Interrupt Acknowledge Register (INTC\_IACKR) when INTC\_MCR[VTES] = 0

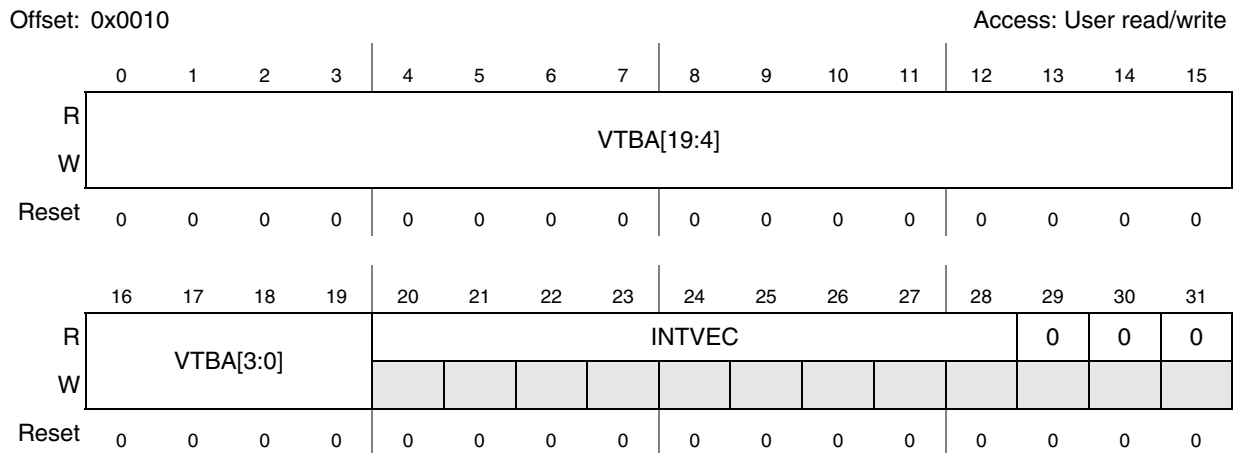


Figure 18-5. INTC Interrupt Acknowledge Register (INTC\_IACKR) when INTC\_MCR[VTES] = 1

Table 18-6. INTC\_IACKR field descriptions

| Field  | Description  |
|--------|--|
| VTBA   | Vector Table Base Address<br>Can be the base address of a vector table of addresses of ISRs.   |
| INTVEC | Interrupt Vector<br>It is the vector of the peripheral or software configurable interrupt request that caused the interrupt request to the processor. When the interrupt request to the processor asserts, the INTVEC is updated, whether the INTC is in software or hardware vector mode. |

The interrupt acknowledge register provides a value that can be used to load the address of an ISR from a vector table. The vector table can be composed of addresses of the ISRs specific to their respective interrupt vectors.

In software vector mode, the INTC\_IACKR has side effects from reads. Therefore, it must not be speculatively read while in this mode. The side effects are the same regardless of the size of the read. Reading the INTC\_IACKR does not have side effects in hardware vector mode.

#### 18.5.2.4 INTC End-of-Interrupt Register (INTC\_EOIR)

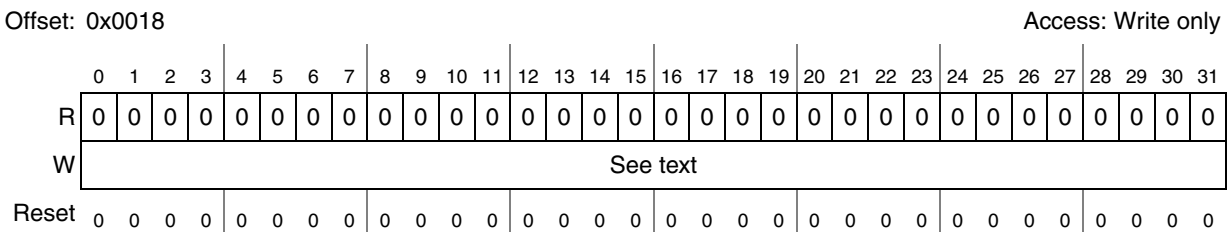
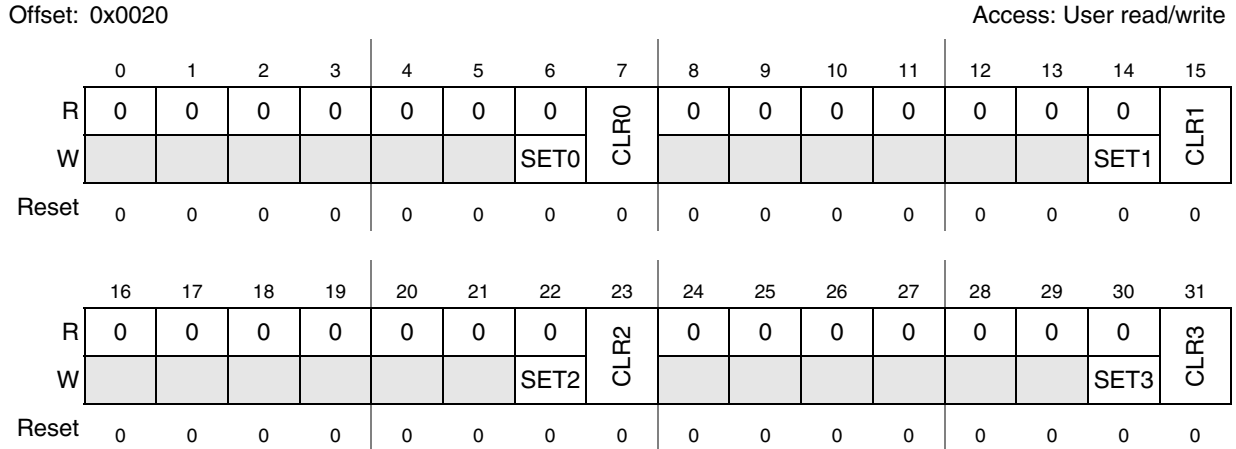


Figure 18-6. INTC End-of-Interrupt Register (INTC\_EOIR)

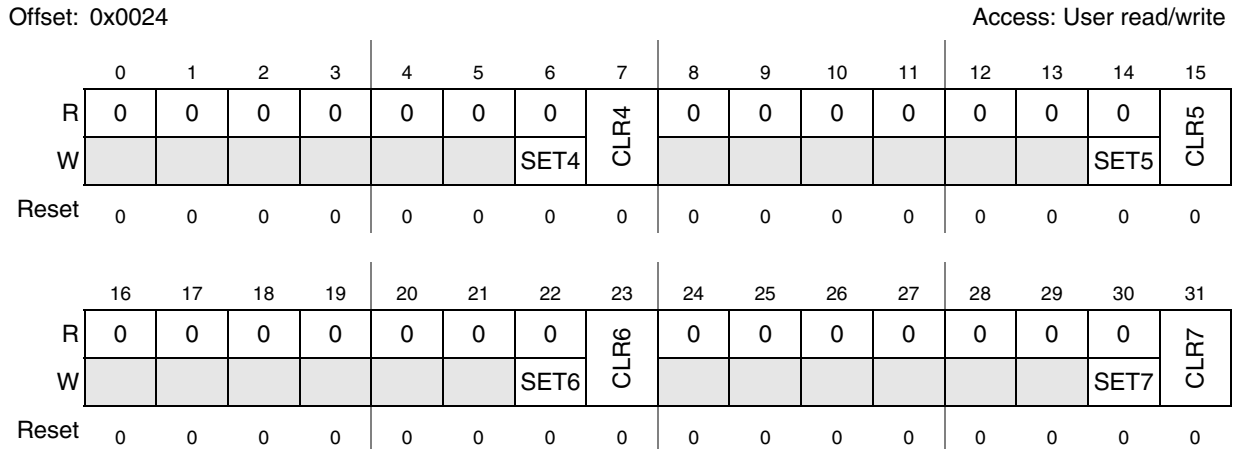
Writing to the end-of-interrupt register signals the end of the servicing of the interrupt request. When the INTC\_EOIR is written, the priority last pushed on the LIFO is popped into INTC\_CPR. An exception to this behavior is described in [Section 18.4.1.2, Hardware vector mode](#). The values and size of data written to the INTC\_EOIR are ignored. The values and sizes written to this register neither update the INTC\_EOIR contents or affect whether the LIFO pops. For possible future compatibility, write four bytes of all 0s to the INTC\_EOIR.

Reading the INTC\_EOIR has no effect on the LIFO.

### 18.5.2.5 INTC Software Set/Clear Interrupt Registers (INTC\_SSCIR0\_3–INTC\_SSCIR4\_7)



**Figure 18-7. INTC Software Set/Clear Interrupt Register 0–3 (INTC\_SSCIR[0:3])**



**Figure 18-8. INTC Software Set/Clear Interrupt Register 4–7 (INTC\_SSCIR[4:7])**

**Table 18-7. INTC\_SSCIR[0:7] field descriptions**

| Field            | Description  |
|------------------|--|
| SETx             | Set Flag Bits<br>Writing a 1 sets the corresponding CLR <sub>x</sub> bit. Writing a 0 has no effect. Each SET <sub>x</sub> always will be read as a 0.   |
| CLR <sub>x</sub> | Clear Flag Bits<br>CLR <sub>x</sub> is the flag bit. Writing a 1 to CLR <sub>x</sub> clears it provided that a 1 is not written simultaneously to its corresponding SET <sub>x</sub> bit. Writing a 0 to CLR <sub>x</sub> has no effect.<br>0 Interrupt request not pending within INTC<br>1 Interrupt request pending within INTC |

The software set/clear interrupt registers support the setting or clearing of software configurable interrupt request. These registers contain eight independent sets of bits to set and clear a corresponding flag bit by

software. Excepting being set by software, this flag bit behaves the same as a flag bit set within a peripheral. This flag bit generates an interrupt request within the INTC like a peripheral interrupt request. Writing a 1 to SET $x$  will leave SET $x$  unchanged at 0 but sets CLR $x$ . Writing a 0 to SET $x$  has no effect. CLR $x$  is the flag bit. Writing a 1 to CLR $x$  clears it. Writing a 0 to CLR $x$  has no effect. If a 1 is written simultaneously to a pair of SET $x$  and CLR $x$  bits, CLR $x$  will be asserted, regardless of whether CLR $x$  was asserted before the write.

### 18.5.2.6 INTC Priority Select Registers (INTC\_PSR0\_3–INTC\_PSR232\_233)

Offset: 0x0040 Access: User read/write

|       |   |   |   |   |      |   |   |   |   |   |    |    |      |    |    |    |
|-------|---|---|---|---|------|---|---|---|---|---|----|----|------|----|----|----|
|       | 0 | 1 | 2 | 3 | 4    | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12   | 13 | 14 | 15 |
| R     | 0 | 0 | 0 | 0 | PRI0 |   |   |   | 0 | 0 | 0  | 0  | PRI1 |    |    |    |
| W     |   |   |   |   |      |   |   |   |   |   |    |    |      |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0    | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0    | 0  | 0  | 0  |

|       |    |    |    |    |      |    |    |    |    |    |    |    |      |    |    |    |
|-------|----|----|----|----|------|----|----|----|----|----|----|----|------|----|----|----|
|       | 16 | 17 | 18 | 19 | 20   | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28   | 29 | 30 | 31 |
| R     | 0  | 0  | 0  | 0  | PRI2 |    |    |    | 0  | 0  | 0  | 0  | PRI3 |    |    |    |
| W     |    |    |    |    |      |    |    |    |    |    |    |    |      |    |    |    |
| Reset | 0  | 0  | 0  | 0  | 0    | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0    | 0  | 0  | 0  |

Figure 18-9. INTC Priority Select Register 0–3 (INTC\_PSR[0:3])

Offset: 0x0128 Access: User read/write

|       |   |   |   |   |        |   |   |   |   |   |    |    |        |    |    |    |
|-------|---|---|---|---|--------|---|---|---|---|---|----|----|--------|----|----|----|
|       | 0 | 1 | 2 | 3 | 4      | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12     | 13 | 14 | 15 |
| R     | 0 | 0 | 0 | 0 | PRI232 |   |   |   | 0 | 0 | 0  | 0  | PRI233 |    |    |    |
| W     |   |   |   |   |        |   |   |   |   |   |    |    |        |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0      | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0      | 0  | 0  | 0  |

|       |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|       | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| R     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |

Figure 18-10. INTC Priority Select Register 232–233 (INTC\_PSR[232:233])

Table 18-8. INTC\_PSR0\_3–INTC\_PSR232\_233 field descriptions

| Field | Description  |
|-------|--|
| PRI   | Priority Select<br>PRI $x$ selects the priority for interrupt requests. See <a href="#">Section 18.6, Functional description</a> . |

Table 18-9. INTC Priority Select Register address offsets

| INTC_PSR <sub>x</sub> _x | Offset address | INTC_PSR <sub>x</sub> _x | Offset address |
|--------------------------|----------------|--------------------------|----------------|
| INTC_PSR0_3              | 0x0040         | INTC_PSR120_123          | 0x00B8         |
| INTC_PSR4_7              | 0x0044         | INTC_PSR124_127          | 0x00BC         |
| INTC_PSR8_11             | 0x0048         | INTC_PSR128_131          | 0x00C0         |
| INTC_PSR12_15            | 0x004C         | INTC_PSR132_135          | 0x00C4         |
| INTC_PSR16_19            | 0x0050         | INTC_PSR136_139          | 0x00C8         |
| INTC_PSR20_23            | 0x0054         | INTC_PSR140_143          | 0x00CC         |
| INTC_PSR24_27            | 0x0058         | INTC_PSR144_147          | 0x00D0         |
| INTC_PSR28_31            | 0x005C         | INTC_PSR148_151          | 0x00D4         |
| INTC_PSR32_35            | 0x0060         | INTC_PSR152_155          | 0x00D8         |
| INTC_PSR36_39            | 0x0064         | INTC_PSR156_159          | 0x00DC         |
| INTC_PSR40_43            | 0x0068         | INTC_PSR160_163          | 0x00E0         |
| INTC_PSR44_47            | 0x006C         | INTC_PSR164_167          | 0x00E4         |
| INTC_PSR48_51            | 0x0070         | INTC_PSR168_171          | 0x00E8         |
| INTC_PSR52_55            | 0x0074         | INTC_PSR172_175          | 0x00EC         |
| INTC_PSR56_59            | 0x0078         | INTC_PSR176_179          | 0x00F0         |
| INTC_PSR60_63            | 0x007C         | INTC_PSR180_183          | 0x00F4         |
| INTC_PSR64_67            | 0x0080         | INTC_PSR184_187          | 0x00F8         |
| INTC_PSR68_71            | 0x0084         | INTC_PSR188_191          | 0x00FC         |
| INTC_PSR72_75            | 0x0088         | INTC_PSR192_195          | 0x0100         |
| INTC_PSR76_79            | 0x008C         | INTC_PSR196_199          | 0x0104         |
| INTC_PSR80_83            | 0x0090         | INTC_PSR200_203          | 0x0108         |
| INTC_PSR84_87            | 0x0094         | INTC_PSR204_207          | 0x010C         |
| INTC_PSR88_91            | 0x0098         | INTC_PSR208_211          | 0x0110         |
| INTC_PSR92_95            | 0x009C         | INTC_PSR212_215          | 0x0114         |
| INTC_PSR96_99            | 0x00A0         | INTC_PSR216_219          | 0x0118         |
| INTC_PSR100_103          | 0x00A4         | INTC_PSR220_223          | 0x011C         |
| INTC_PSR104_107          | 0x00A8         | INTC_PSR224_227          | 0x0120         |
| INTC_PSR108_111          | 0x00AC         | INTC_PSR228_231          | 0x0124         |
| INTC_PSR112_115          | 0x00B0         | INTC_PSR232_233          | 0x0128         |
| INTC_PSR116_119          | 0x00B4         |                          |                |

## 18.6 Functional description

The functional description involves the areas of interrupt request sources, priority management, and handshaking with the processor.

### NOTE

The INTC has no spurious vector support. Therefore, if an asserted peripheral or software settable interrupt request, whose  $PRI_n$  value in INTC\_PSR0–INTC\_PSR233 is higher than the PRI value in INTC\_CPR, negates before the interrupt request to the processor for that peripheral or software settable interrupt request is acknowledged, the interrupt request to the processor still can assert or will remain asserted for that peripheral or software settable interrupt request. In this case, the interrupt vector will correspond to that peripheral or software settable interrupt request. Also, the PRI value in the INTC\_CPR will be updated with the corresponding  $PRI_n$  value in INTC\_PSR $n$ . Furthermore, clearing the peripheral interrupt request's enable bit in the peripheral or, alternatively, setting its mask bit has the same consequences as clearing its flag bit. Setting its enable bit or clearing its mask bit while its flag bit is asserted has the same effect on the INTC as an interrupt event setting the flag bit.

**Table 18-10. Interrupt vector table**

| IRQ #                                      | Offset | Size (bytes) | Interrupt  | Module   |
|--|--------|--------------|--|----------|
| <b>Section A (Core Section)</b>            |        |              |  |          |
| —  | 0x0000 | 16           | Critical Input (INTC software vector mode) / NMI | Core     |
| —  | 0x0010 | 16           | Machine check / NMI                              | Core     |
| —  | 0x0020 | 16           | Data Storage                                     | Core     |
| —  | 0x0030 | 16           | Instruction Storage                              | Core     |
| —  | 0x0040 | 16           | External Input (INTC software vector mode)       | Core     |
| —  | 0x0050 | 16           | Alignment  | Core     |
| —  | 0x0060 | 16           | Program  | Core     |
| —  | 0x0070 | 16           | Reserved   | Core     |
| —  | 0x0080 | 16           | System call                                      | Core     |
| —  | 0x0090 | 96           | Unused   | Core     |
| —  | 0x00F0 | 16           | Debug  | Core     |
| —  | 0x0100 | 1792         | Unused   | Core     |
| <b>Section B (On-Platform Peripherals)</b> |        |              |  |          |
| 0  | 0x0800 | 4            | Software configurable flag 0                     | Software |

Table 18-10. Interrupt vector table (continued)

| IRQ # | Offset | Size (bytes) | Interrupt  | Module   |
|-------|--------|--------------|--|----------|
| 1     | 0x0804 | 4            | Software configurable flag 1   | Software |
| 2     | 0x0808 | 4            | Software configurable flag 2   | Software |
| 3     | 0x080C | 4            | Software configurable flag 3   | Software |
| 4     | 0x0810 | 4            | Software configurable flag 4   | Software |
| 5     | 0x0814 | 4            | Software configurable flag 5   | Software |
| 6     | 0x0818 | 4            | Software configurable flag 6   | Software |
| 7     | 0x081C | 4            | Software configurable flag 7   | Software |
| 8     | 0x0820 | 4            | Reserved   |          |
| 9     | 0x0824 | 4            | Platform Flash Bank 0 Abort  <br>Platform Flash Bank 0 Stall  <br>Platform Flash Bank 1 Abort  <br>Platform Flash Bank 1 Stall | ECSM     |
| 10    | 0x0828 | 4            | Combined Error   | eDMA     |
| 11    | 0x082C | 4            | Channel 0  | eDMA     |
| 12    | 0x0830 | 4            | Channel 1  | eDMA     |
| 13    | 0x0834 | 4            | Channel 2  | eDMA     |
| 14    | 0x0838 | 4            | Channel 3  | eDMA     |
| 15    | 0x083C | 4            | Channel 4  | eDMA     |
| 16    | 0x0840 | 4            | Channel 5  | eDMA     |
| 17    | 0x0844 | 4            | Channel 6  | eDMA     |
| 18    | 0x0848 | 4            | Channel 7  | eDMA     |
| 19    | 0x084C | 4            | Channel 8  | eDMA     |
| 20    | 0x0850 | 4            | Channel 9  | eDMA     |
| 21    | 0x0854 | 4            | Channel 10   | eDMA     |
| 22    | 0x0858 | 4            | Channel 11   | eDMA     |
| 23    | 0x085C | 4            | Channel 12   | eDMA     |
| 24    | 0x0860 | 4            | Channel 13   | eDMA     |
| 25    | 0x0864 | 4            | Channel 14   | eDMA     |
| 26    | 0x0868 | 4            | Channel 15   | eDMA     |
| 27    | 0x086C | 4            | Reserved   |          |
| 28    | 0x0870 | 4            | Timeout  | SWT      |
| 29    | 0x0874 | 4            | Reserved   |          |
| 30    | 0x0878 | 4            | Match on channel 0   | STM      |



Table 18-10. Interrupt vector table (continued)

| IRQ #            | Offset | Size (bytes) | Interrupt  | Module                             |
|------------------|--------|--------------|--|------------------------------------|
| 31               | 0x087C | 4            | Match on channel 1   | STM                                |
| 32               | 0x0880 | 4            | Match on channel 2   | STM                                |
| 33               | 0x0884 | 4            | Match on channel 3   | STM                                |
| 34               | 0x0888 | 4            | Reserved   |                                    |
| 35               | 0x088C | 4            | ECC_DBD_PlatformFlash  <br>ECC_DBD_PlatformRAM                       | Platform ECC Double Bit Detection  |
| 36               | 0x0890 | 4            | ECC_SBC_PlatformFlash  <br>ECC_SBC_PlatformRAM                       | Platform ECC Single Bit Correction |
| 37               | 0x0894 | 4            | Reserved   |                                    |
| <b>Section C</b> |        |              |  |                                    |
| 38               | 0x0898 | 4            | RTC  | RTC/API                            |
| 39               | 0x089C | 4            | API  | RTC/API                            |
| 40               | 0x08A0 | 4            | Reserved   |                                    |
| 41               | 0x08A4 | 4            | SIU External IRQ_0   | SIUL                               |
| 42               | 0x08A8 | 4            | SIU External IRQ_1   | SIUL                               |
| 43               | 0x08AC | 4            | SIU External IRQ_2   | SIUL                               |
| 44               | 0x08B0 | 4            | Reserved   |                                    |
| 45               | 0x08B4 | 4            | Reserved   |                                    |
| 46               | 0x08B8 | 4            | WakeUp_IRQ_0   | WKPU                               |
| 47               | 0x08BC | 4            | WakeUp_IRQ_1   | WKPU                               |
| 48               | 0x08C0 | 4            | WakeUp_IRQ_2   | WKPU                               |
| 49               | 0x08C4 | 4            | WakeUp_IRQ_3   | WKPU                               |
| 50               | 0x08C8 | 4            | Reserved   |                                    |
| 51               | 0x08CC | 4            | Safe Mode Interrupt  | MC_ME                              |
| 52               | 0x08D0 | 4            | Mode Transition Interrupt  | MC_ME                              |
| 53               | 0x08D4 | 4            | Invalid Mode Interrupt   | MC_ME                              |
| 54               | 0x08D8 | 4            | Invalid Mode Config  | MC_ME                              |
| 55               | 0x08DC | 4            | Reserved   |                                    |
| 56               | 0x08E0 | 4            | Functional and destructive reset alternate event interrupt (ipi_int) | MC_RGM                             |
| 57               | 0x08E4 | 4            | FXOSC counter expired (ipi_int_osc)                                  | FXOSC                              |
| 58               | 0x08E8 | 4            | Reserved   |                                    |
| 59               | 0x08EC | 4            | PITimer Channel 0  | PIT                                |

Table 18-10. Interrupt vector table (continued)

| IRQ # | Offset | Size (bytes) | Interrupt   | Module    |
|-------|--------|--------------|---|-----------|
| 60    | 0x08F0 | 4            | PITimer Channel 1   | PIT       |
| 61    | 0x08F4 | 4            | PITimer Channel 2   | PIT       |
| 62    | 0x08F8 | 4            | ADC_EOC   | ADC_0     |
| 64    | 0x0900 | 4            | ADC_WD  | ADC_0     |
| 63    | 0x08FC | 4            | Reserved  |           |
| 65    | 0x0904 | 4            | FlexCAN_ESR[ERR_INT]  | FlexCAN_0 |
| 66    | 0x0908 | 4            | FlexCAN_ESR_BOFF  <br>FlexCAN_Transmit_Warning  <br>FlexCAN_Receive_Warning | FlexCAN_0 |
| 67    | 0x090C | 4            | Reserved  |           |
| 68    | 0x0910 | 4            | FlexCAN_BUF_00_03   | FlexCAN_0 |
| 69    | 0x0914 | 4            | FlexCAN_BUF_04_07   | FlexCAN_0 |
| 70    | 0x0918 | 4            | FlexCAN_BUF_08_11   | FlexCAN_0 |
| 71    | 0x091C | 4            | FlexCAN_BUF_12_15   | FlexCAN_0 |
| 72    | 0x0920 | 4            | FlexCAN_BUF_16_31   | FlexCAN_0 |
| 73    | 0x0924 | 4            | FlexCAN_BUF_32_63   | FlexCAN_0 |
| 74    | 0x0928 | 4            | DSPI_SR[TFUF]<br>DSPI_SR[RFOF]  | DSPI_0    |
| 75    | 0x092C | 4            | DSPI_SR[EOQF]   | DSPI_0    |
| 76    | 0x0930 | 4            | DSPI_SR[TFFF]   | DSPI_0    |
| 77    | 0x0934 | 4            | DSPI_SR[TCF]  | DSPI_0    |
| 78    | 0x0938 | 4            | DSPI_SR[RFDF]   | DSPI_0    |
| 79    | 0x093C | 4            | LINFlex_RXI   | LINFlex_0 |
| 80    | 0x0940 | 4            | LINFlex_TXI   | LINFlex_0 |
| 81    | 0x0944 | 4            | LINFlex_ERR   | LINFlex_0 |
| 82    | 0x0948 | 4            | ADC_EOC   | ADC_1     |
| 83    | 0x094C | 4            | Reserved  |           |
| 84    | 0x0950 | 4            | ADC_WD  | ADC_1     |
| 85    | 0x0954 | 4            | FlexCAN_ESR[ERR_INT]  | FlexCAN_1 |
| 86    | 0x0958 | 4            | FlexCAN_ESR_BOFF  <br>FlexCAN_Transmit_Warning  <br>FlexCAN_Receive_Warning | FlexCAN_1 |
| 87    | 0x095C | 4            | Reserved  |           |
| 88    | 0x0960 | 4            | FlexCAN_BUF_00_03   | FlexCAN_1 |

Table 18-10. Interrupt vector table (continued)

| IRQ # | Offset | Size (bytes) | Interrupt   | Module    |
|-------|--------|--------------|---|-----------|
| 89    | 0x0964 | 4            | FlexCAN_BUF_04_07   | FlexCAN_1 |
| 90    | 0x0968 | 4            | FlexCAN_BUF_08_11   | FlexCAN_1 |
| 91    | 0x096C | 4            | FlexCAN_BUF_12_15   | FlexCAN_1 |
| 92    | 0x0970 | 4            | FlexCAN_BUF_16_31   | FlexCAN_1 |
| 93    | 0x0974 | 4            | FlexCAN_BUF_32_63   | FlexCAN_1 |
| 94    | 0x0978 | 4            | DSPI_SR[TFUF]<br>DSPI_SR[RFOF]  | DSPI_1    |
| 95    | 0x097C | 4            | DSPI_SR[EOQF]   | DSPI_1    |
| 96    | 0x0980 | 4            | DSPI_SR[TFFF]   | DSPI_1    |
| 97    | 0x0984 | 4            | DSPI_SR[TCF]  | DSPI_1    |
| 98    | 0x0988 | 4            | DSPI_SR[RFDF]   | DSPI_1    |
| 99    | 0x098C | 4            | LINFlex_RXI   | LINFlex_1 |
| 100   | 0x0990 | 4            | LINFlex_TXI   | LINFlex_1 |
| 101   | 0x0994 | 4            | LINFlex_ERR   | LINFlex_1 |
| 102   | 0x0998 | 4            | Reserved  |           |
| 103   | 0x099C | 4            | Reserved  |           |
| 104   | 0x09A0 | 4            | Reserved  |           |
| 105   | 0x09A4 | 4            | FlexCAN_[ERR_INT]   | FlexCAN_2 |
| 106   | 0x09A8 | 4            | FlexCAN_ESR_BOFF  <br>FlexCAN_Transmit_Warning  <br>FlexCAN_Receive_Warning | FlexCAN_2 |
| 107   | 0x09AC | 4            | Reserved  |           |
| 108   | 0x09B0 | 4            | FlexCAN_BUF_00_03   | FlexCAN_2 |
| 109   | 0x09B4 | 4            | FlexCAN_BUF_04_07   | FlexCAN_2 |
| 110   | 0x09B8 | 4            | FlexCAN_BUF_08_11   | FlexCAN_2 |
| 111   | 0x09BC | 4            | FlexCAN_BUF_12_15   | FlexCAN_2 |
| 112   | 0x09C0 | 4            | FlexCAN_BUF_16_31   | FlexCAN_2 |
| 113   | 0x09C4 | 4            | FlexCAN_BUF_32_63   | FlexCAN_2 |
| 114   | 0x09C8 | 4            | DSPI_SR[TFUF]<br>DSPI_SR[RFOF]  | DSPI_2    |
| 115   | 0x09CC | 4            | DSPI_SR[EOQF]   | DSPI_2    |
| 116   | 0x09D0 | 4            | DSPI_SR[TFFF]   | DSPI_2    |
| 117   | 0x09D4 | 4            | DSPI_SR[TCF]  | DSPI_2    |
| 118   | 0x09D8 | 4            | DSPI_SR[RFDF]   | DSPI_2    |

Table 18-10. Interrupt vector table (continued)

| IRQ # | Offset | Size (bytes) | Interrupt                                   | Module    |
|-------|--------|--------------|---|-----------|
| 119   | 0x09DC | 4            | LINFlex_RXI                                 | LINFlex_2 |
| 120   | 0x09E0 | 4            | LINFlex_TXI                                 | LINFlex_2 |
| 121   | 0x09E4 | 4            | LINFlex_ERR                                 | LINFlex_2 |
| 122   | 0x09E8 | 4            | LINFlex_RXI                                 | LINFlex_3 |
| 123   | 0x09EC | 4            | LINFlex_TXI                                 | LINFlex_3 |
| 124   | 0x09F0 | 4            | LINFlex_ERR                                 | LINFlex_3 |
| 125   | 0x09F4 | 4            | I2C_SR[IBAL]<br>I2C_SR[TCF]<br>I2C_SR[IAAS] | I2C_0     |
| 126   | 0x09F8 | 4            | Reserved                                    |           |
| 127   | 0x09FC | 4            | PITimer Channel 3                           | PIT       |
| 128   | 0x0A00 | 4            | PITimer Channel 4                           | PIT       |
| 129   | 0x0A04 | 4            | PITimer Channel 5                           | PIT       |
| 130   | 0x0A08 | 4            | PITimer Channel 6                           | PIT       |
| 131   | 0x0A0C | 4            | PITimer Channel 7                           | PIT       |
| 132   | 0x0A10 | 4            | Reserved                                    |           |
| 133   | 0x0A14 | 4            | Reserved                                    |           |
| 134   | 0x0A18 | 4            | Reserved                                    |           |
| 135   | 0x0A1C | 4            | Reserved                                    |           |
| 136   | 0x0A20 | 4            | Reserved                                    |           |
| 137   | 0x0A24 | 4            | Reserved                                    |           |
| 138   | 0x0A28 | 4            | Reserved                                    |           |
| 139   | 0x0A2C | 4            | Reserved                                    |           |
| 140   | 0x0A30 | 4            | Reserved                                    |           |
| 141   | 0x0A34 | 4            | EMIOS_GFR[F0,F1]                            | eMIOS_0   |
| 142   | 0x0A38 | 4            | EMIOS_GFR[F2,F3]                            | eMIOS_0   |
| 143   | 0x0A3C | 4            | EMIOS_GFR[F4,F5]                            | eMIOS_0   |
| 144   | 0x0A40 | 4            | EMIOS_GFR[F6,F7]                            | eMIOS_0   |
| 145   | 0x0A44 | 4            | EMIOS_GFR[F8,F9]                            | eMIOS_0   |
| 146   | 0x0A48 | 4            | EMIOS_GFR[F10,F11]                          | eMIOS_0   |
| 147   | 0x0A4C | 4            | EMIOS_GFR[F12,F13]                          | eMIOS_0   |
| 148   | 0x0A50 | 4            | EMIOS_GFR[F14,F15]                          | eMIOS_0   |
| 149   | 0x0A54 | 4            | EMIOS_GFR[F16,F17]                          | eMIOS_0   |

Table 18-10. Interrupt vector table (continued)

| IRQ #                                      | Offset | Size (bytes) | Interrupt   | Module    |
|--|--------|--------------|---|-----------|
| 150  | 0x0A58 | 4            | EMIOS_GFR[F18,F19]  | eMIOS_0   |
| 151  | 0x0A5C | 4            | EMIOS_GFR[F20,F21]  | eMIOS_0   |
| 152  | 0x0A60 | 4            | EMIOS_GFR[F22,F23]  | eMIOS_0   |
| 153  | 0x0A64 | 4            | EMIOS_GFR[F24,F25]  | eMIOS_0   |
| 154  | 0x0A68 | 4            | EMIOS_GFR[F26,F27]  | eMIOS_0   |
| 155  | 0x0A6C | 4            | EMIOS_GFR[F28,F29]  | eMIOS_0   |
| 156  | 0x0A70 | 4            | EMIOS_GFR[F30,F31]  | eMIOS_0   |
| <b>Section D (Device specific vectors)</b> |        |              |   |           |
| 157  | 0x0A74 | 4            | EMIOS_GFR[F0,F1]  | eMIOS_1   |
| 158  | 0x0A78 | 4            | EMIOS_GFR[F2,F3]  | eMIOS_1   |
| 159  | 0x0A7C | 4            | EMIOS_GFR[F4,F5]  | eMIOS_1   |
| 160  | 0x0A80 | 4            | EMIOS_GFR[F6,F7]  | eMIOS_1   |
| 161  | 0x0A84 | 4            | EMIOS_GFR[F8,F9]  | eMIOS_1   |
| 162  | 0x0A88 | 4            | EMIOS_GFR[F10,F11]  | eMIOS_1   |
| 163  | 0x0A8C | 4            | EMIOS_GFR[F12,F13]  | eMIOS_1   |
| 164  | 0x0A90 | 4            | EMIOS_GFR[F14,F15]  | eMIOS_1   |
| 165  | 0x0A94 | 4            | EMIOS_GFR[F16,F17]  | eMIOS_1   |
| 166  | 0x0A98 | 4            | EMIOS_GFR[F18,F19]  | eMIOS_1   |
| 167  | 0x0A9C | 4            | EMIOS_GFR[F20,F21]  | eMIOS_1   |
| 168  | 0x0AA0 | 4            | EMIOS_GFR[F22,F23]  | eMIOS_1   |
| 169  | 0x0AA4 | 4            | EMIOS_GFR[F24,F25]  | eMIOS_1   |
| 170  | 0x0AA8 | 4            | EMIOS_GFR[F26,F27]  | eMIOS_1   |
| 171  | 0x0AAC | 4            | EMIOS_GFR[F28,F29]  | eMIOS_1   |
| 172  | 0x0AB0 | 4            | EMIOS_GFR[F30,F31]  | eMIOS_1   |
| 173  | 0x0AB4 | 4            | FlexCAN_ESR   | FlexCAN_3 |
| 174  | 0x0AB8 | 4            | FlexCAN_ESR_BOFF  <br>FlexCAN_Transmit_Warning  <br>FlexCAN_Receive_Warning | FlexCAN_3 |
| 175  | 0x0ABC | 4            | Reserved  |           |
| 176  | 0x0AC0 | 4            | FlexCAN_BUF_0_3   | FlexCAN_3 |
| 177  | 0x0AC4 | 4            | FlexCAN_BUF_4_7   | FlexCAN_3 |
| 178  | 0x0AC8 | 4            | FlexCAN_BUF_8_11  | FlexCAN_3 |
| 179  | 0x0ACC | 4            | FlexCAN_BUF_12_15   | FlexCAN_3 |

Table 18-10. Interrupt vector table (continued)

| IRQ # | Offset | Size (bytes) | Interrupt   | Module    |
|-------|--------|--------------|---|-----------|
| 180   | 0x0AD0 | 4            | FlexCAN_BUF_16_31   | FlexCAN_3 |
| 181   | 0x0AD4 | 4            | FlexCAN_BUF_32_63   | FlexCAN_3 |
| 182   | 0x0AD8 | 4            | DSPI_SR[TFUF]<br>DSPI_SR[RFOF]  | DSPI_3    |
| 183   | 0x0ADC | 4            | DSPI_SR[EOQF]   | DSPI_3    |
| 184   | 0x0AE0 | 4            | DSPI_SR[TFFF]   | DSPI_3    |
| 185   | 0x0AE4 | 4            | DSPI_SR[TCF]  | DSPI_3    |
| 186   | 0x0AE8 | 4            | DSPI_SR[RDFD]   | DSPI_3    |
| 187   | 0x0AEC | 4            | LINFlex_RXI   | LINFlex_4 |
| 188   | 0x0AF0 | 4            | LINFlex_TXI   | LINFlex_4 |
| 189   | 0x0AF4 | 4            | LINFlex_ERR   | LINFlex_4 |
| 190   | 0x0AF8 | 4            | FlexCAN_ESR   | FlexCAN_4 |
| 191   | 0x0AFC | 4            | FlexCAN_ESR_BOFF  <br>FlexCAN_Transmit_Warning  <br>FlexCAN_Receive_Warning | FlexCAN_4 |
| 192   | 0x0B00 | 4            | Reserved  |           |
| 193   | 0x0B04 | 4            | FlexCAN_BUF_0_3   | FlexCAN_4 |
| 194   | 0x0B08 | 4            | FlexCAN_BUF_4_7   | FlexCAN_4 |
| 195   | 0x0B0C | 4            | FlexCAN_BUF_8_11  | FlexCAN_4 |
| 196   | 0x0B10 | 4            | FlexCAN_BUF_12_15   | FlexCAN_4 |
| 197   | 0x0B14 | 4            | FlexCAN_BUF_16_31   | FlexCAN_4 |
| 198   | 0x0B18 | 4            | FlexCAN_BUF_32_63   | FlexCAN_4 |
| 199   | 0x0B1C | 4            | LINFlex_RXI   | LINFlex_5 |
| 200   | 0x0B20 | 4            | LINFlex_TXI   | LINFlex_5 |
| 201   | 0x0B24 | 4            | LINFlex_ERR   | LINFlex_5 |
| 202   | 0x0B28 | 4            | FlexCAN_ESR   | FlexCAN_5 |
| 203   | 0x0B2C | 4            | FlexCAN_ESR_BOFF  <br>FlexCAN_Transmit_Warning  <br>FlexCAN_Receive_Warning | FlexCAN_5 |
| 204   | 0x0B30 | 4            | Reserved  |           |
| 205   | 0x0B34 | 4            | FlexCAN_BUF_0_3   | FlexCAN_5 |
| 206   | 0x0B38 | 4            | FlexCAN_BUF_4_7   | FlexCAN_5 |
| 207   | 0x0B3C | 4            | FlexCAN_BUF_8_11  | FlexCAN_5 |
| 208   | 0x0B40 | 4            | FlexCAN_BUF_12_15   | FlexCAN_5 |

Table 18-10. Interrupt vector table (continued)

| IRQ # | Offset | Size (bytes) | Interrupt                      | Module    |
|-------|--------|--------------|--------------------------------|-----------|
| 209   | 0x0B44 | 4            | FlexCAN_BUF_16_31              | FlexCAN_5 |
| 210   | 0x0B48 | 4            | FlexCAN_BUF_32_63              | FlexCAN_5 |
| 211   | 0x0B4C | 4            | DSPI_SR[TFUF]<br>DSPI_SR[RFOF] | DSPI_4    |
| 212   | 0x0B50 | 4            | DSPI_SR[EOQF]                  | DSPI_4    |
| 213   | 0x0B54 | 4            | DSPI_SR[TFFF]                  | DSPI_4    |
| 214   | 0x0B58 | 4            | DSPI_SR[TCF]                   | DSPI_4    |
| 215   | 0x0B5C | 4            | DSPI_SR[RFDF]                  | DSPI_4    |
| 216   | 0x0B60 | 4            | LINFlex_RXI                    | LINFlex_6 |
| 217   | 0x0B64 | 4            | LINFlex_TXI                    | LINFlex_6 |
| 218   | 0x0B68 | 4            | LINFlex_ERR                    | LINFlex_6 |
| 219   | 0x0B6C | 4            | DSPI_SR[TFUF]<br>DSPI_SR[RFOF] | DSPI_5    |
| 220   | 0x0B70 | 4            | DSPI_SR[EOQF]                  | DSPI_5    |
| 221   | 0x0B74 | 4            | DSPI_SR[TFFF]                  | DSPI_5    |
| 222   | 0x0B78 | 4            | DSPI_SR[TCF]                   | DSPI_5    |
| 223   | 0x0B7C | 4            | DSPI_SR[RFDF]                  | DSPI_5    |
| 224   | 0x0B80 | 4            | LINFlex_RXI                    | LINFlex_7 |
| 225   | 0x0B84 | 4            | LINFlex_TXI                    | LINFlex_7 |
| 226   | 0x0B88 | 4            | LINFlex_ERR                    | LINFlex_7 |
| 227   | 0x0B8C | 4            | Reserved                       |           |
| 228   | 0x0B90 | 4            | Reserved                       |           |
| 229   | 0x0B94 | 4            | Reserved                       |           |
| 230   | 0x0B98 | 4            | Reserved                       |           |
| 231   | 0x0B9C | 4            | Reserved                       |           |
| 232   | 0x0BA0 | 4            | Reserved                       |           |
| 233   | 0x0BA4 | 4            | 32KXOSC counter expired        | SXOSC     |

## 18.6.1 Interrupt request sources

The INTC has two types of interrupt requests, peripheral and software configurable. These interrupt requests can assert on any clock cycle.

### 18.6.1.1 Peripheral interrupt requests

An interrupt event in a peripheral's hardware sets a flag bit that resides in the peripheral. The interrupt request from the peripheral is driven by that flag bit.

The time from when the peripheral starts to drive its peripheral interrupt request to the INTC to the time that the INTC starts to drive the interrupt request to the processor is three clocks.

External interrupts are handled by the SIU (see [Section 20.6.3, External interrupts](#)).

### 18.6.1.2 Software configurable interrupt requests

An interrupt request is triggered by software by writing a 1 to a SETx bit in INTC\_SSCIR0\_3–INTC\_SSCIR4\_7. This write sets the corresponding flag bit, CLR<sub>x</sub>, resulting in the interrupt request. The interrupt request is cleared by writing a 1 to the CLR<sub>x</sub> bit.

The time from the write to the SETx bit to the time that the INTC starts to drive the interrupt request to the processor is four clocks.

### 18.6.1.3 Unique vector for each interrupt request source

Each peripheral and software configurable interrupt request is assigned a hardwired unique 9-bit vector. Software configurable interrupts 0–7 are assigned vectors 0–7 respectively. The peripheral interrupt requests are assigned vectors 8 to as high as needed to include all the peripheral interrupt requests. The peripheral interrupt request input ports at the boundary of the INTC block are assigned specific hardwired vectors within the INTC (see [Table 18-1](#)).

## 18.6.2 Priority management

The asserted interrupt requests are compared to each other based on their PRI<sub>x</sub> values set in the INTC Priority Select Registers (INTC\_PSR0\_3–INTC\_PSR232\_233). The result is compared to PRI in the associated INTC\_CPR. The results of those comparisons manage the priority of the ISR executed by the associated processor. The associated LIFO also assists in managing that priority.

### 18.6.2.1 Current priority and preemption

The priority arbitrator, selector, encoder, and comparator subblocks shown in [Figure 18-1](#) compare the priority of the asserted interrupt requests to the current priority. If the priority of any asserted peripheral or software configurable interrupt request is higher than the current priority for a given processor, then the interrupt request to the processor is asserted. Also, a unique vector for the preempting peripheral or software configurable interrupt request is generated for INTC interrupt acknowledge register (INTC\_IACKR), and if in hardware vector mode, for the interrupt vector provided to the processor.

#### 18.6.2.1.1 Priority arbitrator subblock

The priority arbitrator subblock for each processor compares all the priorities of all of the asserted interrupt requests assigned to that processor, both peripheral and software configurable. The output of the priority arbitrator subblock is the highest of those priorities assigned to a given processor. Also, any interrupt



requests that have this highest priority are output as asserted interrupt requests to the associated request selector subblock.

#### 18.6.2.1.2 Request selector subblock

If only one interrupt request from the associated priority arbitrator subblock is asserted, then it is passed as asserted to the associated vector encoder subblock. If multiple interrupt requests from the associated priority arbitrator subblock are asserted, the only the one with the lowest vector is passed as asserted to the associated vector encoder subblock. The lower vector is chosen regardless of the time order of the assertions of the peripheral or software configurable interrupt requests.

#### 18.6.2.1.3 Vector encoder subblock

The vector encoder subblock generates the unique 9-bit vector for the asserted interrupt request from the request selector subblock for the associated processor.

#### 18.6.2.1.4 Priority Comparator subblock

The priority comparator subblock compares the highest priority output from the priority arbitrator subblock with PRI in INTC\_CPR. If the priority comparator subblock detects that this highest priority is higher than the current priority, then it asserts the interrupt request to the associated processor. This interrupt request to the processor asserts whether this highest priority is raised above the value of PRI in INTC\_CPR or the PRI value in INTC\_CPR is lowered below this highest priority. This highest priority then becomes the new priority, which will be written to PRI in INTC\_CPR when the interrupt request to the processor is acknowledged. Interrupt requests whose PRI<sub>n</sub> in INTC\_PSR<sub>n</sub> are zero will not cause a preemption because their PRI<sub>n</sub> will not be higher than PRI in INTC\_CPR.

### 18.6.2.2 Last-In First-Out (LIFO)

The LIFO stores the preempted PRI values from the INTC\_CPR. Therefore, because these priorities are stacked within the INTC, if interrupts need to be enabled during the ISR, at the beginning of the interrupt exception handler the PRI value in the INTC\_CPR does not need to be loaded from the INTC\_CPR and stored onto the context stack. Likewise at the end of the interrupt exception handler, the priority does not need to be loaded from the context stack and stored into the INTC\_CPR.

The PRI value in the INTC\_CPR is pushed onto the LIFO when the INTC\_IACKR is read in software vector mode or the interrupt acknowledge signal from the processor is asserted in hardware vector mode. The priority is popped into PRI in the INTC\_CPR whenever the INTC\_EOIR is written.

Although the INTC supports 16 priorities, an ISR executing with PRI in the INTC\_CPR equal to 15 will not be preempted. Therefore, the LIFO supports the stacking of 15 priorities. However, the LIFO is only 14 entries deep. An entry for a priority of 0 is not needed because of how pushing onto a full LIFO and popping an empty LIFO are treated. If the LIFO is pushed 15 or more times than it is popped, the priorities first pushed are overwritten. A priority of 0 would be an overwritten priority. However, the LIFO will pop 0s if it is popped more times than it is pushed. Therefore, although a priority of 0 was overwritten, it is regenerated with the popping of an empty LIFO.

The LIFO is not memory mapped.

## 18.6.3 Handshaking with processor

### 18.6.3.1 Software vector mode handshaking

This section describes handshaking in software vector mode.

#### 18.6.3.1.1 Acknowledging interrupt request to processor

A timing diagram of the interrupt request and acknowledge handshaking in software vector mode, along with the handshaking near the end of the interrupt exception handler, is shown in [Figure 18-11](#). The INTC examines the peripheral and software configurable interrupt requests. When it finds an asserted peripheral or software configurable interrupt request with a higher priority than PRI in the associated INTC\_CPR, it asserts the interrupt request to the processor. The INTVEC field in the associated INTC\_IACKR is updated with the preempting interrupt request's vector when the interrupt request to the processor is asserted. The INTVEC field retains that value until the next time the interrupt request to the processor is asserted. The rest of the handshaking is described in [Section 18.4.1.1, Software vector mode](#).

#### 18.6.3.1.2 End of interrupt exception handler

Before the interrupt exception handling completes, INTC end-of-interrupt register (INTC\_EOIR) must be written. When written, the associated LIFO is popped so the preempted priority is restored into PRI of the INTC\_CPR. Before it is written, the peripheral or software configurable flag bit must be cleared so that the peripheral or software configurable interrupt request is negated.

#### NOTE

To ensure proper operation across all Power Architecture MCUs, execute an MBAR or MSYNC instruction between the access to clear the flag bit and the write to the INTC\_EOIR.

When returning from the preemption, the INTC does not search for the peripheral or software settable interrupt request whose ISR was preempted. Depending on how much the ISR progressed, that interrupt request may no longer even be asserted. When PRI in INTC\_CPR is lowered to the priority of the preempted ISR, the interrupt request for the preempted ISR or any other asserted peripheral or software settable interrupt request at or below that priority will not cause a preemption. Instead, after the restoration of the preempted context, the processor will return to the instruction address that it was to next execute before it was preempted. This next instruction is part of the preempted ISR or the interrupt exception handler's prolog or epilog.

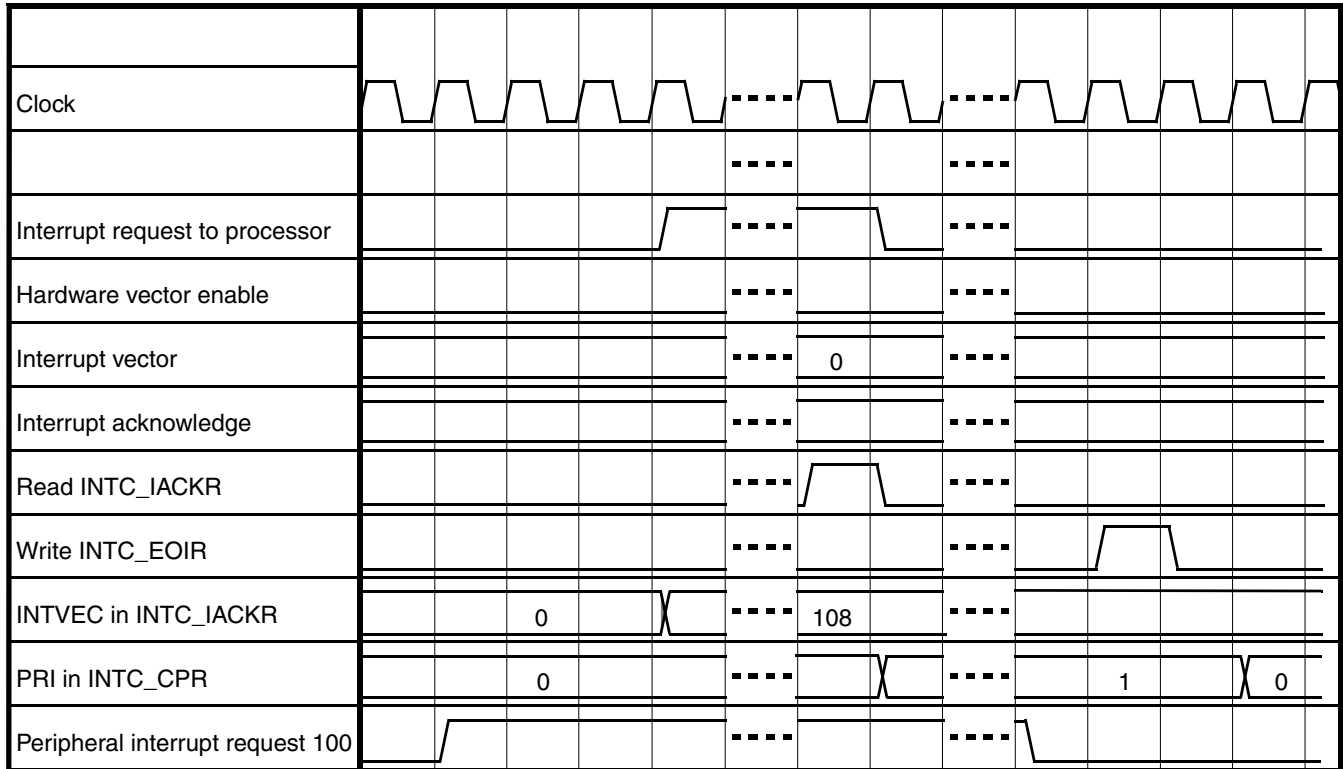


Figure 18-11. Software vector mode handshaking timing diagram

### 18.6.3.2 Hardware vector mode handshaking

A timing diagram of the interrupt request and acknowledge handshaking in hardware vector mode, along with the handshaking near the end of the interrupt exception handler, is shown in [Figure 18-12](#). As in software vector mode, the INTC examines the peripheral and software settable interrupt requests, and when it finds an asserted one with a higher priority than PRI in INTC\_CPR, it asserts the interrupt request to the processor. The INTVEC field in the INTC\_IACKR is updated with the preempting peripheral or software settable interrupt request's vector when the interrupt request to the processor is asserted. The INTVEC field retains that value until the next time the interrupt request to the processor is asserted. In addition, the value of the interrupt vector to the processor matches the value of the INTVEC field in the INTC\_IACKR. The rest of the handshaking is described in [Section 18.7.2.2, Hardware vector mode](#).

The handshaking near the end of the interrupt exception handler, that is the writing to the INTC\_EOIR, is the same as in software vector mode. Refer to [Section 18.6.3.1.2, End of interrupt exception handler](#).

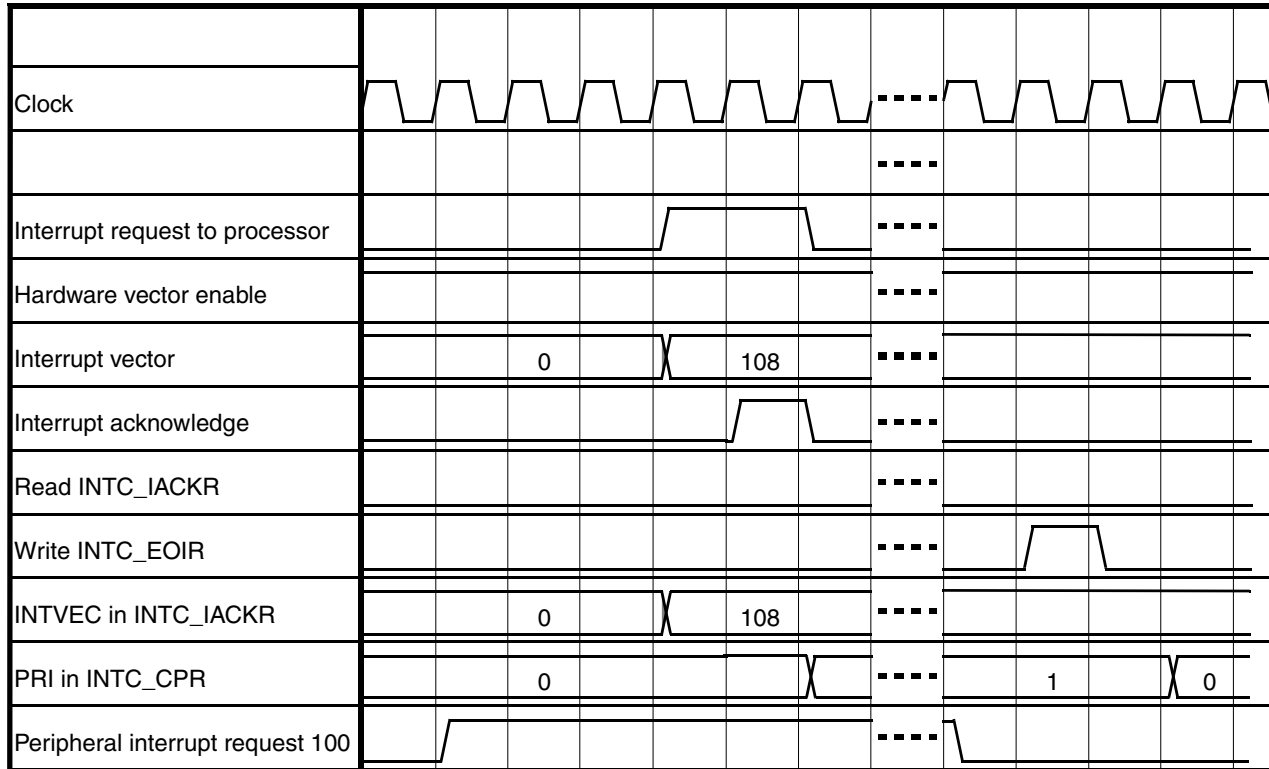


Figure 18-12. Hardware vector mode handshaking timing diagram

## 18.7 Initialization/application information

### 18.7.1 Initialization flow

After exiting reset, all of the  $PRI_n$  fields in INTC priority select registers (INTC\_PSR0–INTC\_PSR233) will be zero, and PRI in INTC current priority register (INTC\_CPR) will be 15. These reset values will prevent the INTC from asserting the interrupt request to the processor. The enable or mask bits in the peripherals are reset such that the peripheral interrupt requests are negated. An initialization sequence for allowing the peripheral and software settable interrupt requests to cause an interrupt request to the processor is:

```
interrupt_request_initialization:
    configure VTES and HVEN in INTC_MCR
    configure VTBA in INTC_IACKR
    raise the  $PRI_n$  fields in INTC_PSR $n$ 
    set the enable bits or clear the mask bits for the peripheral interrupt requests
    lower PRI in INTC_CPR to zero
    enable processor recognition of interrupts
```

### 18.7.2 Interrupt exception handler

These example interrupt exception handlers use Power Architecture assembly code.

### 18.7.2.1 Software vector mode

```

interrupt_exception_handler:
code to create stack frame, save working register, and save SRR0 and SRR1
lis    r3,INTC_IACKR@ha      # form adjusted upper half of INTC_IACKR address
lwz    r3,INTC_IACKR@l(r3)  # load INTC_IACKR, which clears request to processor
lwz    r3,0x0(r3)           # load address of ISR from vector table
wrteei 1                     # enable processor recognition of interrupts

code to save rest of context required by e500 EABI

mtrlr  r3                    # move INTC_IACKR contents into link register
blr    # branch to ISR; link register updated with epilg
      # address

epilog:
code to restore most of context required by e500 EABI

# Popping the LIFO after the restoration of most of the context and the disabling of processor
# recognition of interrupts eases the calculation of the maximum stack depth at the cost of
# postponing the servicing of the next interrupt request.
mbar                                # ensure store to clear flag bit has completed
lis    r3,INTC_EOIR@ha          # form adjusted upper half of INTC_EOIR address
li     r4,0x0                   # form 0 to write to INTC_EOIR
wrteei 0                          # disable processor recognition of interrupts
stw    r4,INTC_EOIR@l(r3)      # store to INTC_EOIR, informing INTC to lower priority

code to restore SRR0 and SRR1, restore working registers, and delete stack frame

rfi

vector_table_base_address:
address of ISR for interrupt with vector 0
address of ISR for interrupt with vector 1
.
.
.
address of ISR for interrupt with vector 510
address of ISR for interrupt with vector 511

ISRx:
code to service the interrupt event
code to clear flag bit which drives interrupt request to INTC

blr # return to epilg

```

### 18.7.2.2 Hardware vector mode

This interrupt exception handler is useful with processor and system bus implementations that support a hardware vector. This example assumes that each `interrupt_exception_handlerx` only has space for four instructions, and therefore a branch to `interrupt_exception_handler_continuedx` is needed.

```

interrupt_exception_handlerx:
b interrupt_exception_handler_continuedx# 4 instructions available, branch to continue

```

```

interrupt_exception_handler_continuedx:
code to create stack frame, save working register, and save SRR0 and SRR1

wrteei    1                # enable processor recognition of interrupts

code to save rest of context required by e500 EABI

bl        ISRx             # branch to ISR for interrupt with vector x

epilog:
code to restore most of context required by e500 EABI

# Popping the LIFO after the restoration of most of the context and the disabling of processor
# recognition of interrupts eases the calculation of the maximum stack depth at the cost of
# postponing the servicing of the next interrupt request.
mbar                    # ensure store to clear flag bit has completed
lis     r3,INTC_EOIR@ha # form adjusted upper half of INTC_EOIR address
li      r4,0x0          # form 0 to write to INTC_EOIR
wrteei    0                # disable processor recognition of interrupts
stw     r4,INTC_EOIR@l(r3) # store to INTC_EOIR, informing INTC to lower priority

code to restore SRR0 and SRR1, restore working registers, and delete stack frame

rfi

ISRx:
code to service the interrupt event
code to clear flag bit which drives interrupt request to INTC
blr                    # branch to epilog

```

### 18.7.3 ISR, RTOS, and task hierarchy

The RTOS and all of the tasks under its control typically execute with PRI in INTC current priority register (INTC\_CPR) having a value of 0. The RTOS will execute the tasks according to whatever priority scheme that it may have, but that priority scheme is independent and has a lower priority of execution than the priority scheme of the INTC. In other words, the ISRs execute above INTC\_CPR priority 0 and outside the control of the RTOS, the RTOS executes at INTC\_CPR priority 0, and while the tasks execute at different priorities under the control of the RTOS, they also execute at INTC\_CPR priority 0.

If a task shares a resource with an ISR and the PCP is being used to manage that shared resource, then the task's priority can be elevated in the INTC\_CPR while the shared resource is being accessed.

An ISR whose PRI<sub>*n*</sub> in INTC priority select registers (INTC\_PSR0–INTC\_PSR233) has a value of 0 will not cause an interrupt request to the processor, even if its peripheral or software settable interrupt request is asserted. For a peripheral interrupt request, not setting its enable bit or disabling the mask bit will cause it to remain negated, which consequently also will not cause an interrupt request to the processor. Since the ISRs are outside the control of the RTOS, this ISR will not run unless called by another ISR or the interrupt exception handler, perhaps after executing another ISR.

## 18.7.4 Order of execution

An ISR with a higher priority can preempt an ISR with a lower priority, regardless of the unique vectors associated with each of their peripheral or software configurable interrupt requests. However, if multiple peripheral or software configurable interrupt requests are asserted, more than one has the highest priority, and that priority is high enough to cause preemption, the INTC selects the one with the lowest unique vector regardless of the order in time that they asserted. However, the ability to meet deadlines with this scheduling scheme is no less than if the ISRs execute in the time order that their peripheral or software configurable interrupt requests asserted.

The example in [Table 18-11](#) shows the order of execution of both ISRs with different priorities and the same priority.

**Table 18-11. Order of ISR execution example**

| Step No. | Step description  | Code Executing at End of Step |                     |        |        |        | Interrupt exception handler | PRI in INTC_CPR at End of Step |
|----------|---|-------------------------------|---------------------|--------|--------|--------|-----------------------------|--------------------------------|
|          |   | RTOS                          | ISR108 <sup>1</sup> | ISR208 | ISR308 | ISR408 |                             |                                |
| 1        | RTOS at priority 0 is executing.  | X                             |                     |        |        |        |                             | 0                              |
| 2        | Peripheral interrupt request 100 at priority 1 asserts. Interrupt taken.                                |                               | X                   |        |        |        |                             | 1                              |
| 3        | Peripheral interrupt request 400 at priority 4 is asserts. Interrupt taken.                             |                               |                     |        |        | X      |                             | 4                              |
| 4        | Peripheral interrupt request 300 at priority 3 is asserts.  |                               |                     |        |        | X      |                             | 4                              |
| 5        | Peripheral interrupt request 200 at priority 3 is asserts.  |                               |                     |        |        | X      |                             | 4                              |
| 6        | ISR408 completes. Interrupt exception handler writes to INTC_EOIR.                                      |                               |                     |        |        |        | X                           | 1                              |
| 7        | Interrupt taken. ISR208 starts to execute, even though peripheral interrupt request 300 asserted first. |                               |                     | X      |        |        |                             | 3                              |
| 8        | ISR208 completes. Interrupt exception handler writes to INTC_EOIR.                                      |                               |                     |        |        |        | X                           | 1                              |
| 9        | Interrupt taken. ISR308 starts to execute.  |                               |                     |        | X      |        |                             | 3                              |
| 10       | ISR308 completes. Interrupt exception handler writes to INTC_EOIR.                                      |                               |                     |        |        |        | X                           | 1                              |
| 11       | ISR108 completes. Interrupt exception handler writes to INTC_EOIR.                                      |                               |                     |        |        |        | X                           | 0                              |
| 12       | RTOS continues execution.   | X                             |                     |        |        |        |                             | 0                              |

<sup>1</sup> ISR108 executes for peripheral interrupt request 100 because the first eight ISRs are for software configurable interrupt requests.

## 18.7.5 Priority ceiling protocol

### 18.7.5.1 Elevating priority

The PRI field in INTC\_CPR is elevated in the OSEK PCP to the ceiling of all of the priorities of the ISRs that share a resource. This protocol allows coherent accesses of the ISRs to that shared resource.

For example, ISR1 has a priority of 1, ISR2 has a priority of 2, and ISR3 has a priority of 3. They share the same resource. Before ISR1 or ISR2 can access that resource, they must raise the PRI value in INTC\_CPR to 3, the ceiling of all of the ISR priorities. After they release the resource, the PRI value in INTC\_CPR can be lowered. If they do not raise their priority, ISR2 can preempt ISR1, and ISR3 can preempt ISR1 or ISR2, possibly corrupting the shared resource. Another possible failure mechanism is deadlock if the higher priority ISR needs the lower priority ISR to release the resource before it can continue, but the lower priority ISR cannot release the resource until the higher priority ISR completes and execution returns to the lower priority ISR.

Using the PCP instead of disabling processor recognition of all interrupts eliminates the time when accessing a shared resource that all higher priority interrupts are blocked. For example, while ISR3 cannot preempt ISR1 while it is accessing the shared resource, all of the ISRs with a priority higher than 3 can preempt ISR1.

### 18.7.5.2 Ensuring coherency

A scenario can cause non-coherent accesses to the shared resource. For example, ISR1 and ISR2 are both running on the same core and both share a resource. ISR1 has a lower priority than ISR2. ISR1 is executing and writes to the INTC\_CPR. The instruction following this store is a store to a value in a shared coherent data block. Either immediately before or at the same time as the first store, the INTC asserts the interrupt request to the processor because the peripheral interrupt request for ISR2 has asserted. As the processor is responding to the interrupt request from the INTC, and as it is aborting transactions and flushing its pipeline, it is possible that both stores will be executed. ISR2 thereby thinks that it can access the data block coherently, but the data block has been corrupted.

OSEK uses the GetResource and ReleaseResource system services to manage access to a shared resource. To prevent corruption of a coherent data block, modifications to PRI in INTC\_CPR can be made by those system services with the code sequence:

```
disable processor recognition of interrupts
PRI modification
enable processor recognition of interrupts
```

## 18.7.6 Selecting priorities according to request rates and deadlines

The selection of the priorities for the ISRs can be made using rate monotonic scheduling (RMS) or a superset of it, deadline monotonic scheduling (DMS). In RMS, the ISRs that have higher request rates have higher priorities. In DMS, if the deadline is before the next time the ISR is requested, then the ISR is assigned a priority according to the time from the request for the ISR to the deadline, not from the time of the request for the ISR to the next request for it.



For example, ISR1 executes every 100  $\mu$ s, ISR2 executes every 200  $\mu$ s, and ISR3 executes every 300  $\mu$ s. ISR1 has a higher priority than ISR, 2 which has a higher priority than ISR3; however, if ISR3 has a deadline of 150  $\mu$ s, then it has a higher priority than ISR2.

The INTC has 16 priorities, which may be less than the number of ISRs. In this case, the ISRs should be grouped with other ISRs that have similar deadlines. For example, a priority could be allocated for every time the request rate doubles. ISRs with request rates around 1 ms would share a priority, ISRs with request rates around 500  $\mu$ s would share a priority, ISRs with request rates around 250  $\mu$ s would share a priority, etc. With this approach, a range of ISR request rates of  $2^{16}$  could be included, regardless of the number of ISRs.

Reducing the number of priorities reduces the processor's ability to meet its deadlines. However, reducing the number of priorities can reduce the size and latency through the interrupt controller. It also allows easier management of ISRs with similar deadlines that share a resource. They do not need to use the PCP to access the shared resource.

## 18.7.7 Software configurable interrupt requests

The software configurable interrupt requests can be used in two ways. They can be used to schedule a lower priority portion of an ISR, and they may also be used by processors to interrupt other processors in a multiple processor system.

### 18.7.7.1 Scheduling a lower priority portion of an ISR

A portion of an ISR needs to be executed at the  $PRI_x$  value in the INTC Priority Select Registers (INTC\_PSR0\_3–INTC\_PSR232\_233), which becomes the  $PRI$  value in INTC\_CPR with the interrupt acknowledge. The ISR, however, can have a portion that does not need to be executed at this higher priority. Therefore, executing the later portion that does not need to be executed at this higher priority can prevent the execution of ISRs that do not have a higher priority than the earlier portion of the ISR, but do have a higher priority than what the later portion of the ISR needs. This preemptive scheduling inefficiency reduces the processor's ability to meet its deadlines.

One option is for the ISR to complete the earlier higher priority portion, but then schedule through the RTOS a task to execute the later lower priority portion. However, some RTOSs can require a large amount of time for an ISR to schedule a task. Therefore, a second option is for the ISR, after completing the higher priority portion, to set a  $SET_x$  bit in INTC\_SSCIR0\_3–INTC\_SSCIR4\_7. Writing a 1 to  $SET_x$  causes a software configurable interrupt request. This software configurable interrupt request will usually have a lower  $PRI_x$  value in the INTC\_PSR $_x_x$  and will not cause preemptive scheduling inefficiencies. After generating a software settable interrupt request, the higher priority ISR completes. The lower priority ISR is scheduled according to its priority. Execution of the higher priority ISR is not resumed after the completion of the lower priority ISR.

### 18.7.7.2 Scheduling an ISR on another processor

Because the  $SET_x$  bits in the INTC\_SSCIR $_x_x$  are memory mapped, processors in multiple-processor systems can schedule ISRs on the other processors. One application is that one processor wants to command another processor to perform a piece of work and the initiating processor does not need to use

the results of that work. If the initiating processor is concerned that the processor executing the software configurable ISR has not completed the work before asking it to again execute the ISR, it can check if the corresponding CLR<sub>x</sub> bit in INTC\_SSCIR<sub>x\_x</sub> is asserted before again writing a 1 to the SET<sub>x</sub> bit.

Another application is the sharing of a block of data. For example, a first processor has completed accessing a block of data and wants a second processor to then access it. Furthermore, after the second processor has completed accessing the block of data, the first processor again wants to access it. The accesses to the block of data must be done coherently. To do this, the first processor writes a 1 to a SET<sub>x</sub> bit on the second processor. After accessing the block of data, the second processor clears the corresponding CLR<sub>x</sub> bit and then writes 1 to a SET<sub>x</sub> bit on the first processor, informing it that it can now access the block of data.

### 18.7.8 Lowering priority within an ISR

A common method for avoiding preemptive scheduling inefficiencies with an ISR whose work spans multiple priorities (see [Section 18.7.7.1, Scheduling a lower priority portion of an ISR](#)) is to lower the current priority. However, the INTC has a LIFO whose depth is determined by the number of priorities.

#### NOTE

Lowering the PRI value in INTC\_CPR within an ISR to below the ISR's corresponding PRI value in the INTC Priority Select Registers (INTC\_PSR0\_3–INTC\_PSR232\_233) allows more preemptions than the LIFO depth can support.

Therefore, the INTC does not support lowering the current priority within an ISR as a way to avoid preemptive scheduling inefficiencies.

### 18.7.9 Negating an interrupt request outside of its ISR

#### 18.7.9.1 Negating an interrupt request as a side effect of an ISR

Some peripherals have flag bits that can be cleared as a side effect of servicing a peripheral interrupt request. For example, reading a specific register can clear the flag bits and their corresponding interrupt requests. This clearing as a side effect of servicing a peripheral interrupt request can cause the negation of other peripheral interrupt requests besides the peripheral interrupt request whose ISR presently is executing. This negating of a peripheral interrupt request outside of its ISR can be a desired effect.

#### 18.7.9.2 Negating multiple interrupt requests in one ISR

An ISR can clear other flag bits besides its own. One reason that an ISR clears multiple flag bits is because it serviced those flag bits, and therefore the ISRs for these flag bits do not need to be executed.

#### 18.7.9.3 Proper setting of interrupt request priority

Whether an interrupt request negates outside its own ISR due to the side effect of an ISR execution or the intentional clearing a flag bit, the priorities of the peripheral or software configurable interrupt requests for these other flag bits must be selected properly. Their PRL<sub>x</sub> values in the INTC Priority Select Registers

(INTC\_PSR0\_3–INTC\_PSR232\_233) must be selected to be at or lower than the priority of the ISR that cleared their flag bits. Otherwise, those flag bits can cause the interrupt request to the processor to assert. Furthermore, the clearing of these other flag bits also has the same timing relationship to the writing to INTC\_SSCIR0\_3–INTC\_SSCIR4\_7 as the clearing of the flag bit that caused the present ISR to be executed (see [Section 18.6.3.1.2, End of interrupt exception handler](#)).

A flag bit whose enable bit or mask bit negates its peripheral interrupt request can be cleared at any time, regardless of the peripheral interrupt request's PRL<sub>x</sub> value in INTC\_PSR<sub>x\_x</sub>.

### 18.7.10 Examining LIFO contents

In normal mode, the user does not need to know the contents of the LIFO. He may not even know how deeply the LIFO is nested. However, if he wants to read the contents, such as in debug mode, they are not memory mapped. The contents can be read by popping the LIFO and reading the PRI field in either INTC\_CPR. The code sequence is:

```
pop_lifo:
store to INTC_EOIR
load INTC_CPR, examine PRI, and store onto stack
if PRI is not zero or value when interrupts were enabled, branch to pop_lifo
```

When the examination is complete, the LIFO can be restored using this code sequence:

```
push_lifo:
load stacked PRI value and store to INTC_CPR
load INTC_IACKR
if stacked PRI values are not depleted, branch to push_lifo
```

This page is intentionally left blank.

# Chapter 19

## Crossbar Switch (XBAR)

### 19.1 Introduction

This chapter describes the multi-port crossbar switch (XBAR), which supports simultaneous connections between three master ports and three slave ports. XBAR supports a 32-bit address bus width and a 32-bit data bus width at all master and slave ports.

The crossbar of MPC5606BK is the same as the one of all other PPC55xx and PPC56xx products except that it cannot be configured by software and that it has a hard-wired configuration.

### 19.2 Block diagram

Figure 19-1 shows a block diagram of the crossbar switch.

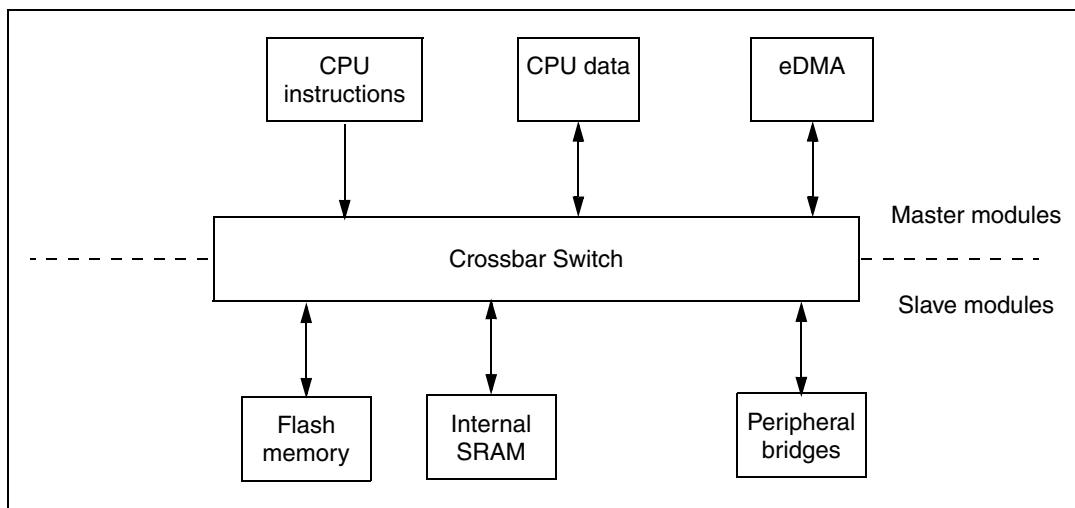


Figure 19-1. XBAR block diagram

Table 19-1 gives the crossbar switch port for each master and slave, and the assigned and fixed ID number for each master. The table shows the master ID numbers as they relate to the master port numbers.

Table 19-1. XBAR switch ports for MPC5606BK

| Module                       | Port   |                | Physical master ID |
|------------------------------|--------|----------------|--------------------|
|                              | Type   | Logical number |                    |
| e200z0 core–CPU instructions | Master | 0              | 0                  |
| e200z0 core–CPU data         | Master | 1              | 0                  |
| eDMA                         | Master | 2              | 1                  |
| Flash memory                 | Slave  | 0              | —                  |
| Internal SRAM                | Slave  | 2              | —                  |
| Peripheral bridges           | Slave  | 7              | —                  |

## 19.3 Overview

The XBAR allows for concurrent transactions to occur from any master port to any slave port. It is possible for all master ports and slave ports to be in use at the same time as a result of independent master requests. If a slave port is simultaneously requested by more than one master port, arbitration logic selects the higher priority master and grants it ownership of the slave port. All other masters requesting that slave port are stalled until the higher priority master completes its transactions.

Requesting masters are granted access based on a fixed priority.

## 19.4 Features

- Three master ports:
  - Core: e200z0 core instructions
  - Core: e200z0 core data
  - eDMA
- Three slave ports
  - Flash (refer to [Chapter 30, Flash Memory](#), for information on accessing flash memory)
  - Internal SRAM
  - Peripheral bridges
- 32-bit address, 32-bit data paths
- Fully concurrent transfers between independent master and slave ports
- Fixed priority scheme and fixed parking strategy

## 19.5 Modes of operation

### 19.5.1 Normal mode

In normal mode, the XBAR provides the logic that controls crossbar switch configuration.

### 19.5.2 Debug mode

The XBAR operation in debug mode is identical to operation in normal mode.

## 19.6 Functional description

This section describes the functionality of the XBAR in more detail.

### 19.6.1 Overview

The main goal of the XBAR is to increase overall system performance by allowing multiple masters to communicate concurrently with multiple slaves. To maximize data throughput, it is essential to keep arbitration delays to a minimum.

This section examines data throughput from the point of view of masters and slaves, detailing when the XBAR stalls masters, or inserts bubbles on the slave side.

## 19.6.2 General operation

When a master makes an access to the XBAR from an idle master state, the access is taken immediately by the XBAR. If the targeted slave port of the access is available (that is, the requesting master is currently granted ownership of the slave port), the access is immediately presented on the slave port. It is possible to make single clock (zero wait state) accesses through the XBAR by a granted master. If the targeted slave port of the access is busy or parked on a different master port, the requesting master receives wait states until the targeted slave port can service the master request. The latency in servicing the request depends on each master's priority level and the responding slave's access time.

Because the XBAR appears to be simply another slave to the master device, the master device has no indication that it owns the slave port it is targeting. While the master does not have control of the slave port it is targeting, it is wait-stated.

A master is given control of a targeted slave port only after a previous access to a different slave port has completed, regardless of its priority on the newly targeted slave port. This prevents deadlock from occurring when a master has the following conditions:

- Outstanding request to slave port A that has a long response time
- Pending access to a different slave port B
- Lower priority master also makes a request to the different slave port B.

In this case, the lower priority master is granted bus ownership of slave port B after a cycle of arbitration, assuming the higher priority master slave port A access is not terminated.

After a master has control of the slave port it is targeting, the master remains in control of that slave port until it gives up the slave port by running an IDLE cycle, leaves that slave port for its next access, or loses control of the slave port to a higher priority master with a request to the same slave port. However, because all masters run a fixed-length burst transfer to a slave port, it retains control of the slave port until that transfer sequence is completed.

When a slave bus is idled by the XBAR, it is parked on the master that did the last transfer.

## 19.6.3 Master ports

A master access is taken if the slave port to which the access decodes is either currently servicing the master or is parked on the master. In this case, the XBAR is completely transparent and the master access is immediately transmitted on the slave bus and no arbitration delays are incurred. A master access stall if the access decodes to a slave port that is busy serving another master, parked on another master.

If the slave port is currently parked on another master, and no other master is requesting access to the slave port, then only one clock of arbitration is incurred. If the slave port is currently serving another master of a lower priority and the master has a higher priority than all other requesting masters, then the master gains control over the slave port as soon as the data phase of the current access is completed. If the slave port is currently servicing another master of a higher priority, then the master gains control of the slave port after

the other master releases control of the slave port if no other higher priority master is also waiting for the slave port.

A master access is responded to with an error if the access decodes to a location not occupied by a slave port. This is the only time the XBAR directly responds with an error response. All other error responses received by the master are the result of error responses on the slave ports being passed through the XBAR.

## 19.6.4 Slave ports

The goal of the XBAR with respect to the slave ports is to keep them 100% saturated when masters are actively making requests. To do this the XBAR must not insert any bubbles onto the slave bus unless absolutely necessary.

There is only one instance when the XBAR forces a bubble onto the slave bus when a master is actively making a request. This occurs when a handoff of bus ownership occurs and there are no wait states from the slave port. A requesting master that does not own the slave port is granted access after a one clock delay.

## 19.6.5 Priority assignment

Each master port is assigned a fixed 3-bit priority level (hard-wired priority). The following table shows the priority levels assigned to each master (the lowest has highest priority).

**Table 19-2. Hardwired bus master priorities**

| Module                       | Port   |          | Priority level |
|------------------------------|--------|----------|----------------|
|                              | Type   | Master # |                |
| e200z0 core—CPU instructions | Master | 0        | 7              |
| e200z0 core—CPU data         | Master | 1        | 6              |
| eDMA                         | Master | 2        | 5              |

## 19.6.6 Arbitration

XBAR supports only a fixed-priority comparison algorithm.

### 19.6.6.1 Fixed priority operation

When operating in fixed-priority arbitration mode, each master is assigned a unique priority level in the XBAR\_MPR. If two masters both request access to a slave port, the master with the higher priority in the selected priority register gains control over the slave port.

Any time a master makes a request to a slave port, the slave port checks to see if the new requesting master's priority level is higher than that of the master that currently has control over the slave port (if any). The slave port does an arbitration check at every clock edge to ensure that the proper master (if any) has control of the slave port.



If the new requesting master's priority level is higher than that of the master that currently has control of the slave port, the higher priority master is granted control at the termination of any currently pending access, assuming the pending transfer is not part of a burst transfer.

A new requesting master must wait until the end of the fixed-length burst transfer, before it is granted control of the slave port. But if the new requesting master's priority level is lower than that of the master that currently has control of the slave port, the new requesting master is forced to wait until the master that currently has control of the slave port is finished accessing the current slave port.

#### **19.6.6.1.1 Parking**

If no master is currently requesting the slave port, the slave port is parked. The slave port parks always to the last master (park-on-last). When parked on the last master, the slave port is passing that master's signals through to the slave bus. When the master accesses the slave port again, no other arbitration penalties are incurred except that a one clock arbitration penalty is incurred for each access request to the slave port made by another master port. All other masters pay a one clock penalty.

This page is intentionally left blank.

# Chapter 20

## System Integration Unit Lite (SIUL)

### 20.1 Introduction

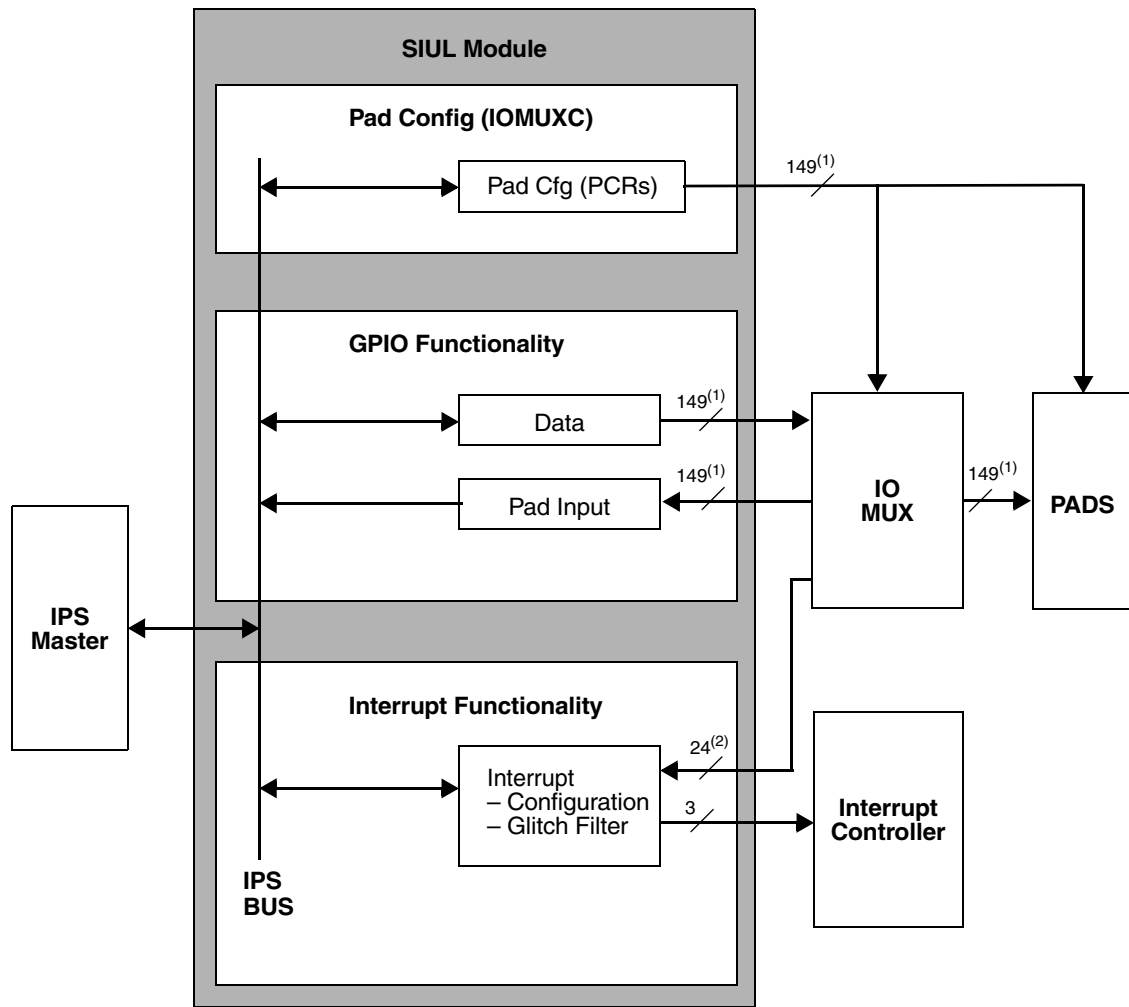
This chapter describes the System Integration Unit Lite (SIUL), which manages the pads and their configuration. It controls the multiplexing of the alternate functions used on all pads and is responsible for the management of the external interrupts to the device.

### 20.2 Overview

The System Integration Unit Lite (SIUL) controls the MCU pad configuration, ports, general-purpose input and output (GPIO) signals, and external interrupts with trigger event configuration. [Figure 20-1](#) provides a block diagram of the SIUL and its interfaces to other system components.

The module provides the capability to configure, read, and write to the device's general-purpose I/O pads that can be configured as either inputs or outputs.

- When a pad is configured as an input, the state of the pad (logic high or low) is obtained by reading an associated data input register.
- When a pad is configured as an output, the value driven onto the pad is determined by writing to an associated data output register. Enabling the input buffers when a pad is configured as an output allows the actual state of the pad to be read.
- To enable monitoring of an output pad value, the pad can be configured as both output and input so the actual pad value can be read back and compared with the expected value.



Notes:

- <sup>1</sup> 149 GPIOs in 176-pin LQFP and 208 BGA, as many as 121 GPIOs in 144-pin LQFP and as many as 77 GPIOs in 100-pin LQFP
- <sup>2</sup> 24 EIRQs in 144-pin LQFP, 176-pin LQFP and 208 BGA; as many as 20 EIRQs in 100-pin LQFP

Figure 20-1. System Integration Unit Lite block diagram

## 20.3 Features

The System Integration Unit Lite supports these distinctive features:

- GPIO
  - GPIO function on as many as 149 I/O pins
  - Dedicated input and output registers for most GPIO pins<sup>1</sup>
- External interrupts
  - Three interrupt vectors dedicated to 24 external interrupts
  - 24 programmable digital glitch filters
  - Independent interrupt mask
  - Edge detection
- System configuration
  - Pad configuration control

## 20.4 External signal description

Most device pads support multiple device functions. Pad configuration registers are provided to enable selection between GPIO and other signals. These other signals, also referred to as alternate functions, are typically peripheral functions.

GPIO pads are grouped in ports, with each port containing as many as 16 pads. With appropriate configuration, all pins in a port can be read or written to in parallel with a single R/W access.

### NOTE

In order to use GPIO port functionality, all pads in the port must be configured as GPIO rather than as alternate functions.

Table 20-1 lists the external pins configurable via the SIUL.

**Table 20-1. SIUL signal properties**

| GPIO[0:148] category | Name  | I/O direction    | Function  |
|----------------------|---|------------------|---|
| System configuration | GPIO[0:148] <sup>1</sup>  | I/O <sup>2</sup> | General-purpose input/output  |
| External interrupt   | PA[3], PA[6:8], PA[11:12], PA[14], PC[2:5], PC[12], PC[14:15], PE[2], PE[4], PE[6:7], PE[10], PE[12], PE[14], PF[15], PG[1], PG[8] <sup>3</sup> | Input            | Pins with External Interrupt Request functionality. Please see <a href="#">Chapter 4, Signal description</a> , for details. |

<sup>1</sup> GPIO[0–26], GPIO[28–59], and GPIO[61–122] in 144-pin LQFP; GPIO[0–26], GPIO[28–59], GPIO[61–76], and GPIO[121–122] in 100-pin LQFP

<sup>2</sup> Most, but not all GPIO pads can be configured as inputs or outputs but some, e.g., analog pins with GPIO function, are only configurable as inputs.

<sup>3</sup> PE[14], PF[15], PG[1], and PG[8] not available in 100-pin LQFP

1. Some device pins, e.g., analog pins, do not have both input and output functionality.

## 20.4.1 Detailed signal descriptions

### 20.4.1.1 General-purpose I/O pins (GPIO[0:148])<sup>1</sup>

The GPIO pins provide general-purpose input and output functions. The GPIO pins are generally multiplexed with other I/O pin functions. Each GPIO input and output is separately controlled by an input (GPDIn<sub>n</sub>) or output (GPDO<sub>n</sub>) register.

### 20.4.1.2 External interrupt request input pins (EIRQ[0:23])<sup>2</sup>

The EIRQ[0:23] pins are connected to the SIUL inputs. Rising- or falling-edge events are enabled by setting the corresponding bits in the SIUL\_IREER or the SIUL\_IFEER register.

---

1. GPIO[0–26], GPIO[28–59], and GPIO[61–122] in 144-pin LQFP; GPIO[0–26], GPIO[28–59], GPIO[61–76], and GPIO[121–122] in 100-pin LQFP

2. EIRQ[0:11] plus EIRQ[16:23] in 100-pin LQFP

## 20.5 Memory map and register description

This section provides a detailed description of all registers accessible in the SIUL module.

### 20.5.1 SIUL memory map

Table 20-2 gives an overview of the SIUL registers implemented.

**Table 20-2. SIUL memory map**

| Base address: 0xC3F9_0000 |  |                             |
|---------------------------|--|-----------------------------|
| Address offset            | Register   | Location                    |
| 0x0000                    | Reserved   |                             |
| 0x0004                    | MCU ID Register #1 (MIDR1)   | <a href="#">on page 353</a> |
| 0x0008                    | MCU ID Register #2 (MIDR2)   | <a href="#">on page 354</a> |
| 0x000C–0x0013             | Reserved   |                             |
| 0x0014                    | Interrupt Status Flag Register (ISR)                               | <a href="#">on page 355</a> |
| 0x0018                    | Interrupt Request Enable Register (IRER)                           | <a href="#">on page 356</a> |
| 0x001C–0x00               | Reserved   |                             |
| 0x0028                    | Interrupt Rising-Edge Event Enable Register (IREER)                | <a href="#">on page 356</a> |
| 0x002C                    | Interrupt Falling-Edge Event Enable Register (IFEER)               | <a href="#">on page 357</a> |
| 0x0030                    | Interrupt Filter Enable Register (IFER)                            | <a href="#">on page 358</a> |
| 0x0034–0x003F             | Reserved   |                             |
| 0x0040–0x0168             | Pad Configuration Registers (PCR0–PCR148) <sup>1</sup>             | <a href="#">on page 359</a> |
| 0x016A–0x04FF             | Reserved   |                             |
| 0x0500–0x053C             | Pad Selection for Multiplexed Inputs Registers (PSMI0_3–PSMI60_63) | <a href="#">on page 361</a> |
| 0x0540–0x05FF             | Reserved   |                             |
| 0x0600–0x06A0             | GPIO Pad Data Output Registers (GPDO0_3–GPDO148_151) <sup>2</sup>  | <a href="#">on page 366</a> |
| 0x06A4–0x07FF             | Reserved   |                             |
| 0x0800–0x08A0             | GPIO Pad Data Input Registers (GPDIO_3–GPDIO148_151) <sup>3</sup>  | <a href="#">on page 367</a> |
| 0x08A4–0x0BFF             | Reserved   |                             |
| 0x0C00–0x0C10             | Parallel GPIO Pad Data Out Registers (PGPDO0 – PGPDO4)             | <a href="#">on page 367</a> |
| 0x0C14–0x0C3F             | Reserved   |                             |
| 0x0C40–0x0C50             | Parallel GPIO Pad Data In Registers (PGPDIO – PGPDIO4)             | <a href="#">on page 368</a> |
| 0x0C54–0x0C7F             | Reserved   |                             |
| 0x0C80–0x0CA4             | Masked Parallel GPIO Pad Data Out Register (MPGPDO0–MPGPDO9)       | <a href="#">on page 369</a> |
| 0x0CA8–0x0FFF             | Reserved   |                             |

Table 20-2. SIUL memory map (continued)

| Base address: 0xC3F9_0000 |  |             |
|---------------------------|--|-------------|
| Address offset            | Register   | Location    |
| 0x1000–0x105C             | Interrupt Filter Maximum Counter Registers (IFMC0–IFMC23) <sup>4</sup> | on page 371 |
| 0x1060–0x107C             | Reserved   |             |
| 0x1080                    | Interrupt Filter Clock Prescaler Register (IFCPR)                      | on page 371 |
| 0x1084–0x3FFF             | Reserved   |             |

<sup>1</sup> PCR0–26, PCR28–59, and PCR61–122 in 144-pin LQFP; PCR0–26, PCR28–59, PCR61–76, and PCR121–122 in 100-pin LQFP—all remaining registers are reserved.

<sup>2</sup> GPDO0–26, GPDO28–59, and GPDO61–122 in 144-pin LQFP; GPDO0–26, GPDO28–59, GPDO61–76, and GPDO121–122 in 100-pin LQFP—all remaining registers are reserved.

<sup>3</sup> GPDIO–26, GPDIO28–59, and GPDIO61–122 in 144-pin LQFP; GPDIO–26, GPDIO28–59, GPDIO61–76, and GPDIO121–122 in 100-pin LQFP—all remaining registers are reserved.

<sup>4</sup> IFMC0–11 plus IFMC16–23 in 100-pin LQFP—all remaining registers are reserved.

### NOTE

A transfer error will be issued when trying to access completely reserved register space.

## 20.5.2 Register protection

Individual registers in System Integration Unit Lite can be protected from accidental writes using the Register Protection module. The following registers can be protected:

- Interrupt Request Enable Register (IRER)
- Interrupt Rising-Edge Event Enable Register (IREER)
- Interrupt Falling-Edge Event Enable Register (IFEER)
- Interrupt Filter Enable Register (IFER),
- Pad Configuration Registers (PCR0–PCR148). Note that only the following registers can be protected:
  - PCR[0:15] (Port A)
  - PCR[16:19] (Port B[0:3])
  - PCR[34:47] (Port C[2:15])
- Pad Selection for Multiplexed Inputs Registers (PSMI0\_3–PSMI60\_63)
- Interrupt Filter Maximum Counter Registers (IFMC0–IFMC23). Note that only IFMC[0:15] can be protected.
- Interrupt Filter Clock Prescaler Register (IFCPR)

See [Chapter 32, Register Protection](#), for more details.



## 20.5.3 Register descriptions

### 20.5.3.1 MCU ID Register #1 (MIDR1)

This register holds identification information about the device.

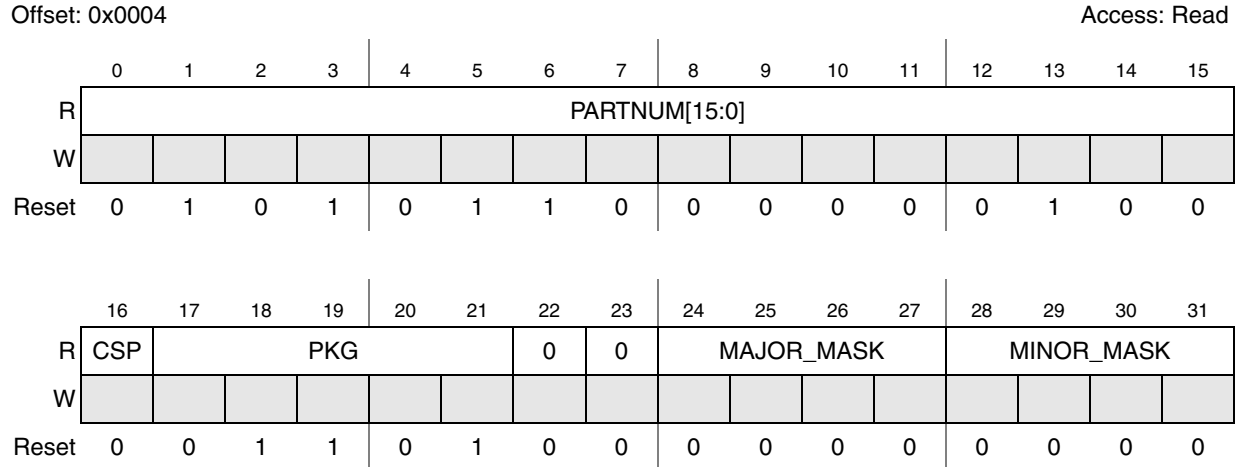


Figure 20-2. MCU ID Register #1 (MIDR1)

Table 20-3. MIDR1 field descriptions

| Field         | Description   |
|---------------|---|
| PARTNUM[15:0] | MCU Part Number, lower 16 bits<br>Device part number of the MCU.<br>0101_0110_0000_0001: 128 KB<br>0101_0110_0000_0010: 256 KB<br>0101_0110_0000_0011: 320/384 KB<br>0101_0110_0000_0100: 512 KB<br>0101_0110_0000_0101: 768 KB<br>0101_0110_0000_0110: 1 MB<br>0101_0110_0000_0111: 1.5 MB<br>For the full part number this field needs to be combined with MIDR2[PARTNUM[23:16]]. |
| CSP           | Always reads back 0   |
| PKG           | Package Settings<br>Can be read by software to determine the package type that is used for the particular device as described below. Any values not explicitly specified are reserved.<br>0b01001: 100-pin LQFP<br>0b01101: 144-pin LQFP<br>0b10000: 208 BGA<br>0b10001: 176-pin LQFP   |
| MAJOR_MASK    | Major Mask Revision<br>Counter starting at 0x0. Incremented each time there is a resynthesis.   |
| MINOR_MASK    | Minor Mask Revision<br>Counter starting at 0x0. Incremented each time a mask change is done.  |

### 20.5.3.2 MCU ID Register #2 (MIDR2)

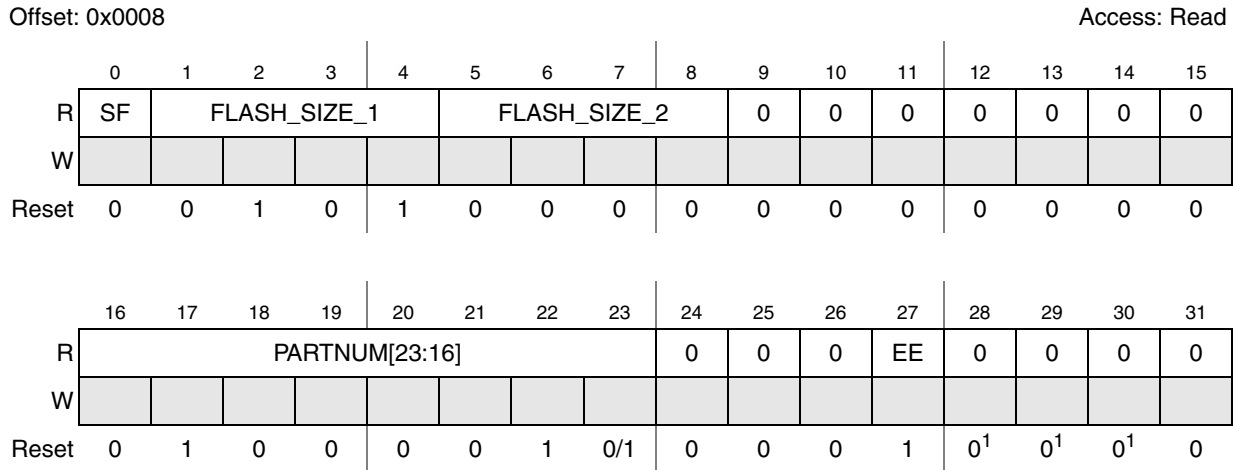


Figure 20-3. MCU ID Register #2 (MIDR2)

<sup>1</sup> Static bit fixed in hardware

Table 20-4. MIDR2 field descriptions

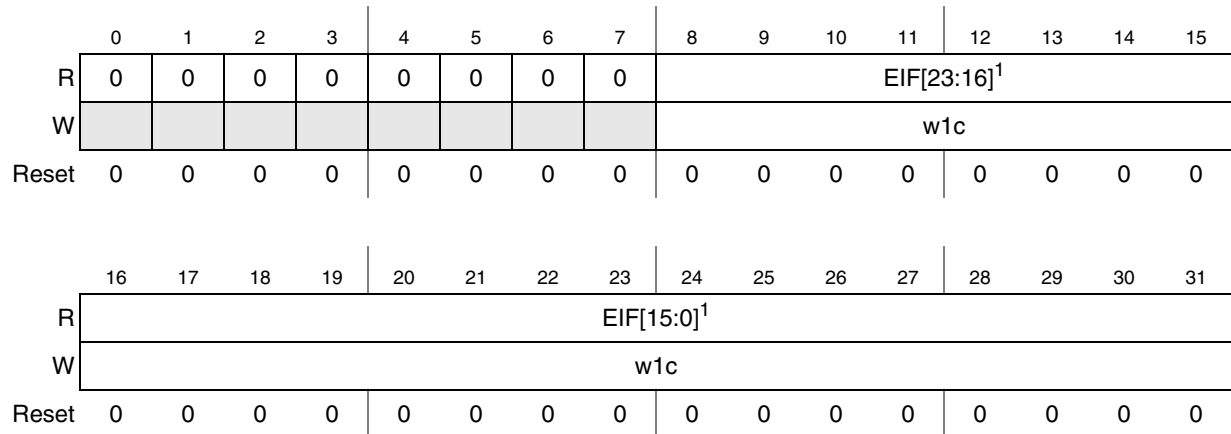
| Field              | Description  |
|--------------------|--|
| SF                 | Manufacturer<br>0 Freescale<br>1 Reserved  |
| FLASH_SIZE_1       | Coarse granularity for Flash memory size<br>Total flash memory size = FLASH_SIZE_1 + FLASH_SIZE_2<br>0011 128 KB<br>0100 256 KB<br>0101 512 KB<br>0110 1 MB  |
| FLASH_SIZE_2       | Fine granularity for Flash memory size<br>Total flash memory size = FLASH_SIZE_1 + FLASH_SIZE_2<br>0000 0 × (FLASH_SIZE_1 / 8)<br>0010 2 × (FLASH_SIZE_1 / 8)<br>0100 4 × (FLASH_SIZE_1 / 8)   |
| PARTNUM<br>[23:16] | MCU Part Number, upper 8 bits containing the ASCII character within the MCU part number<br>0x42: Character B (Body controller)<br>0x43: Character C (Gateway)<br><br>For the full part number this field needs to be combined with MIDR1[PARTNUM[15:0]]. |
| EE                 | Data Flash present<br>0 No Data Flash is present<br>1 Data Flash is present  |

### 20.5.3.3 Interrupt Status Flag Register (ISR)

This register holds the interrupt flags.

Offset: 0x0014

Access: User read/write



**Figure 20-4. Interrupt Status Flag Register (ISR)**

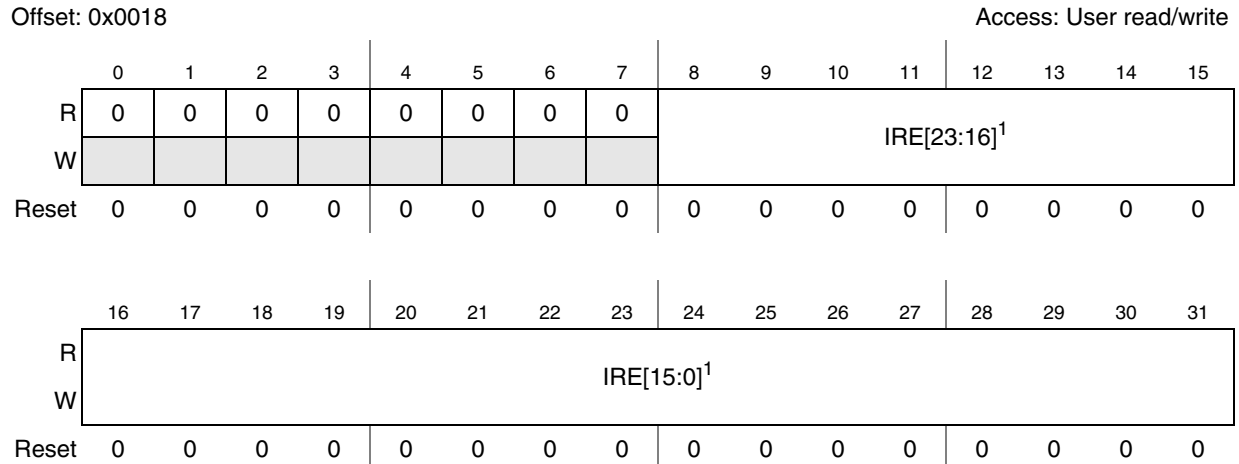
<sup>1</sup> 20 flags in 100-pin LQFP: EIF[23:16] plus EIF[11:0] (register bits 16–19 reserved).

**Table 20-5. ISR field descriptions**

| Field  | Description  |
|--------|--|
| EIF[x] | <p>External Interrupt Status Flag x</p> <p>This flag can be cleared only by writing a 1. Writing a 0 has no effect. If enabled (IRER[x]), EIF[x] causes an interrupt request.</p> <p>0 No interrupt event has occurred on the pad</p> <p>1 An interrupt event as defined by IREER[x] and IFEER[x] has occurred</p> |

### 20.5.3.4 Interrupt Request Enable Register (IRER)

This register is used to enable the interrupt messaging to the interrupt controller.



**Figure 20-5. Interrupt Request Enable Register (IRER)**

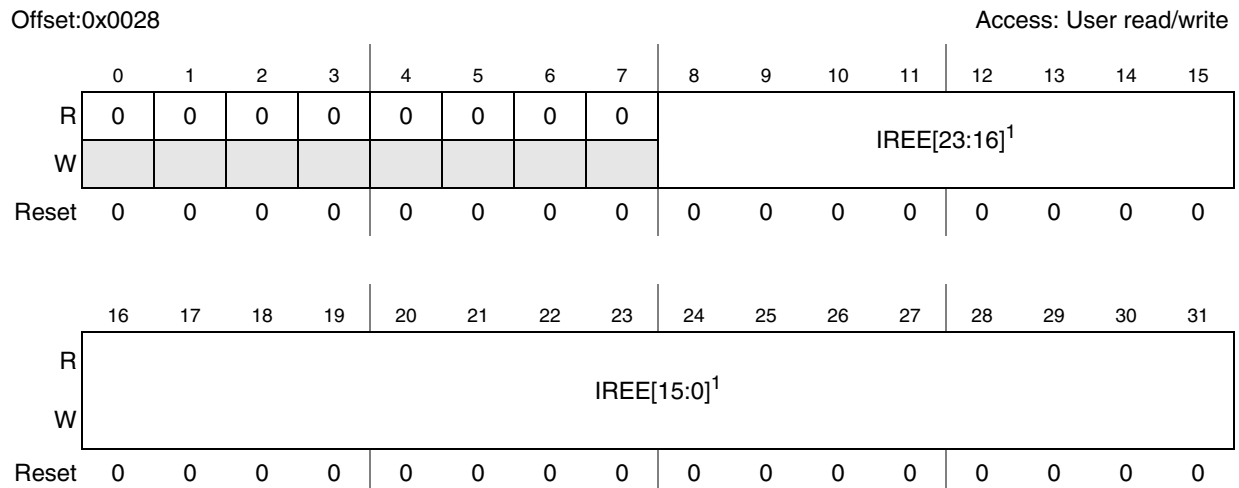
<sup>1</sup> 20 enable requests in 100-pin LQFP: IRE[23:16] plus IRE[11:0] (register bits 16–19 reserved).

**Table 20-6. IRER field descriptions**

| Field  | Description  |
|--------|--|
| IRE[x] | External Interrupt Request Enable x<br>0 Interrupt requests from the corresponding ISR(EIF[x]) bit are disabled.<br>1 Interrupt requests from the corresponding ISR(EIF[x]) bit are enabled. |

### 20.5.3.5 Interrupt Rising-Edge Event Enable Register (IREER)

This register is used to enable rising-edge triggered events on the corresponding interrupt pads.



**Figure 20-6. Interrupt Rising-Edge Event Enable Register (IREER)**

<sup>1</sup> 20 enable events in 100-pin LQFP: IREE[23:16] plus IREE[11:0] (register bits 16–19 reserved).

Table 20-7. IREER field descriptions

| Field   | Description  |
|---------|--|
| IREE[x] | Enable rising-edge events to cause the ISR(EIF[x]) bit to be set.<br>0 Rising-edge event is disabled<br>1 Rising-edge event is enabled |

### 20.5.3.6 Interrupt Falling-Edge Event Enable Register (IFEER)

This register is used to enable falling-edge triggered events on the corresponding interrupt pads.

Offset: 0x002C Access: User read/write

|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8                        | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|-------|---|---|---|---|---|---|---|---|--------------------------|---|----|----|----|----|----|----|
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | IFEE[23:16] <sup>1</sup> |   |    |    |    |    |    |    |
| W     |   |   |   |   |   |   |   |   |                          |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0                        | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       | 16                      | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|-------|-------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| R     | IFEE[15:0] <sup>1</sup> |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| W     |                         |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| Reset | 0                       | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |

Figure 20-7. Interrupt Falling-Edge Event Enable Register (IFEER)

<sup>1</sup> 20 enabling events in 100-pin LQFP: IFEE[23:16] plus IFEE[11:0] (register bits 16–19 reserved).

Table 20-8. IFEER field descriptions

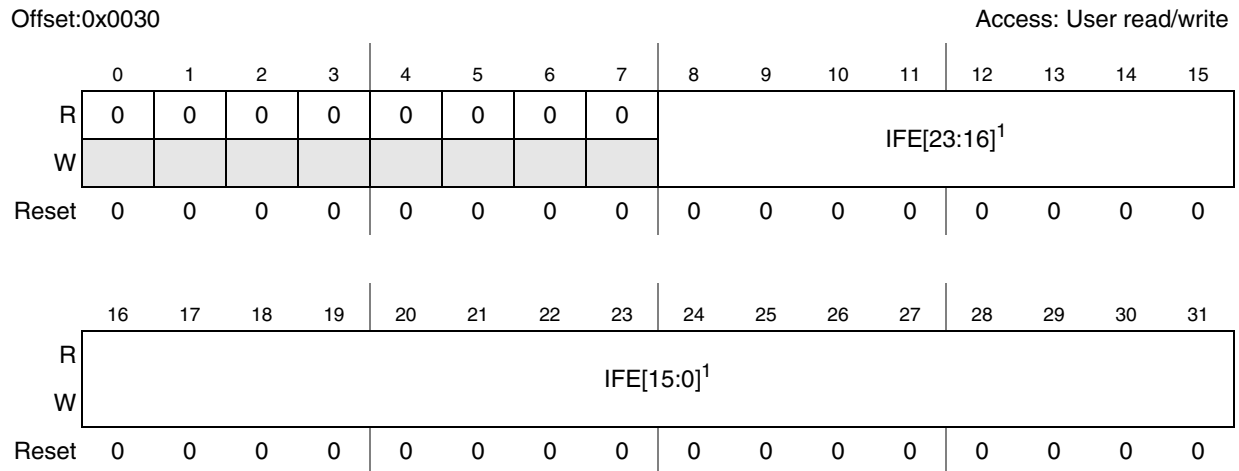
| Field   | Description   |
|---------|---|
| IFEE[x] | Enable falling-edge events to cause the ISR(EIF[x]) bit to be set.<br>0 Falling-edge event is disabled<br>1 Falling-edge event is enabled |

#### NOTE

If both the IREER[IREE] and IFEER[IFEE] bits are cleared for the same interrupt source, the interrupt status flag for the corresponding external interrupt will never be set. If IREER[IREE] and IFEER[IFEE] bits are set for the same source the interrupts are triggered by both rising edge events and falling edge events.

### 20.5.3.7 Interrupt Filter Enable Register (IFER)

This register is used to enable a digital filter counter on the corresponding interrupt pads to filter out glitches on the inputs.



**Figure 20-8. Interrupt Filter Enable Register (IFER)**

<sup>1</sup> 20 bits in 100-pin LQFP: IFE[23:16] plus IFE[11:0] (register bits 16–19 reserved).

**Table 20-9. IFER field descriptions**

| Field  | Description   |
|--------|---|
| IFE[x] | Enable digital glitch filter on the interrupt pad input<br>0 Filter is disabled<br>1 Filter is enabled<br>See the IFMC field descriptions in <a href="#">Table 20-20</a> for details on how the filter works. |

### 20.5.3.8 Pad Configuration Registers (PCR0–PCR148)

The Pad Configuration Registers allow configuration of the static electrical and functional characteristics associated with I/O pads. Each PCR controls the characteristics of a single pad.

Please note that input and output peripheral muxing are separate.

- For output pads:
  - Select the appropriate alternate function in Pad Config Register (PCR)
  - OBE is not required for functions other than GPIO
- For input pads:
  - Select the feature location from PSMI register
  - Set the IBE bit in the appropriate PCR
- For normal GPIO (not alternate function):
  - Configure PCR
  - Read from GPDI or write to GPDO

Offsets: Base + 0x0040 (PCR0)(149 registers)  
 Base + 0x0042 (PCR1)  
 ...  
 Base + 0x0168 (PCR148)

Access: User read/write

|       |   |                |     |   |                |     |                |                |   |   |     |    |    |     |                |                |
|-------|---|----------------|-----|---|----------------|-----|----------------|----------------|---|---|-----|----|----|-----|----------------|----------------|
|       | 0 | 1              | 2   | 3 | 4              | 5   | 6              | 7              | 8 | 9 | 10  | 11 | 12 | 13  | 14             | 15             |
| R     | 0 | SMC            | APC |   | PA[1:0]        | OBE | IBE            |                | 0 | 0 | ODE | 0  | 0  | SRC | WPE            | WPS            |
| W     |   |                |     |   |                |     |                |                |   |   |     |    |    |     |                |                |
| Reset | 0 | 0 <sup>1</sup> | 0   | 0 | 0 <sup>1</sup> | 0   | 0 <sup>2</sup> | 0 <sup>3</sup> | 0 | 0 | 0   | 0  | 0  | 0   | 0 <sup>3</sup> | 1 <sup>4</sup> |

Figure 20-9. Pad Configuration Registers (PCR<sub>x</sub>)

- <sup>1</sup> SMC and PA[1] are 1 for JTAG pads
- <sup>2</sup> OBE is 1 for TDO
- <sup>3</sup> IBE and WPE are 1 for TCK, TMS, TDI, FAB, and ABS
- <sup>4</sup> WPS is 0 for input only pad with analog feature and FAB

**NOTE**

16- and 32-bit accesses to the PCR<sub>x</sub> registers are supported.

In addition to the bit map above, the following Table 20-11 describes the PCR depending on the pad type (pad types are defined in the “Pad types” section of this reference manual). The bits in shaded fields are not implemented for the particular I/O type. The PA field selecting the number of alternate functions may or may not be present depending on the number of alternate functions actually mapped on the pad.

Table 20-10. PCR bit implementation by pad type

| Pad type   | PCR bit No. |     |     |   |         |     |     |   |   |   |     |    |    |     |     |     |
|--|-------------|-----|-----|---|---------|-----|-----|---|---|---|-----|----|----|-----|-----|-----|
|  | 0           | 1   | 2   | 3 | 4       | 5   | 6   | 7 | 8 | 9 | 10  | 11 | 12 | 13  | 14  | 15  |
| S, M, F (Pad with GPIO and digital alternate function) |             | SMC | APC |   | PA[1:0] | OBE | IBE |   |   |   | ODE |    |    | SRC | WPE | WPS |
| Pad with slew rate control                             |             | SMC | APC |   | PA[1:0] | OBE | IBE |   |   |   | ODE |    |    | SRC | WPE | WPS |
| J (Pad with GPIO and analog functionality)             |             | SMC | APC |   | PA[1:0] | OBE | IBE |   |   |   | ODE |    |    | SRC | WPE | WPS |
| I (Pad dedicated to ADC)                               |             | SMC | APC |   | PA[1:0] | OBE | IBE |   |   |   | ODE |    |    | SRC | WPE | WPS |

Table 20-11. PCR<sub>x</sub> field descriptions

| Field | Description  |
|-------|--|
| SMC   | Safe Mode Control.<br>This bit supports the overriding of the automatic deactivation of the output buffer of the associated pad upon entering SAFE mode of the device.<br>0 In SAFE mode, the output buffer of the pad is disabled.<br>1 In SAFE mode, the output buffer remains functional. |

Table 20-11. PCRx field descriptions (continued)

| Field   | Description   |
|---------|---|
| APC     | Analog Pad Control.<br>This bit enables the usage of the pad as analog input.<br>0 Analog input path from the pad is gated and cannot be used<br>1 Analog input path switch can be enabled by the ADC   |
| PA[1:0] | Pad Output Assignment<br>This field is used to select the function that is allowed to drive the output of a multiplexed pad.<br>00 Alternative Mode 0 — GPIO<br>01 Alternative Mode 1 — See <a href="#">Chapter 4, Signal description</a><br>10 Alternative Mode 2 — See <a href="#">Chapter 4, Signal description</a><br>11 Alternative Mode 3 — See <a href="#">Chapter 4, Signal description</a><br><br><b>Note:</b> Number of bits depends on the actual number of actual alternate functions. Please see data sheet. |
| OBE     | Output Buffer Enable<br>This bit enables the output buffer of the pad in case the pad is in GPIO mode.<br>0 Output buffer of the pad is disabled when PA[1:0] = 00<br>1 Output buffer of the pad is enabled when PA[1:0] = 00   |
| IBE     | Input Buffer Enable<br>This bit enables the input buffer of the pad.<br>0 Input buffer of the pad is disabled<br>1 Input buffer of the pad is enabled   |
| ODE     | Open Drain Output Enable<br>This bit controls output driver configuration for the pads connected to this signal. Either open drain or push/pull driver configurations can be selected. This feature applies to output pads only.<br>0 Pad configured for push/pull output<br>1 Pad configured for open drain  |
| SRC     | Slew Rate Control<br>This field controls the slew rate of the associated pad when it is slew rate selectable. Its usage is the following:<br>0 Pad configured as slow (default)<br>1 Pad is configured as medium or fast (depending on the pad)<br><b>Note:</b> PC[1] (TDO pad) is medium only. By default SRC = 0, and writing 1 has no effect.  |
| WPE     | Weak Pull Up/Down Enable<br>This bit controls whether the weak pull up/down devices are enabled/disabled for the pad connected to this signal.<br>0 Weak pull device disabled for the pad<br>1 Weak pull device enabled for the pad   |
| WPS     | Weak Pull Up/Down Select<br>This bit controls whether weak pull up or weak pull down devices are used for the pads connected to this signal when weak pull up/down devices are enabled.<br>0 Weak pulldown selected<br>1 Weak pullup selected   |

### 20.5.3.9 Pad Selection for Multiplexed Inputs Registers (PSMI0\_3–PSMI60\_63)

In some cases, a peripheral input signal can be selected from more than one pin. For example, the CAN1\_RXD signal can be selected on three different pins: PC[3], PC[11], and PF[15]. Only one can be active at a time. To select the pad to be used as input to the peripheral:



- Select the signal via the pad’s PCR register using the PA field.
- Specify the pad to be used via the appropriate PSMI field.

Offsets:0x0500–0x053C (16 registers)

Access: User read/write

|       |   |   |   |   |         |   |   |   |   |   |    |    |         |    |    |    |
|-------|---|---|---|---|---------|---|---|---|---|---|----|----|---------|----|----|----|
|       | 0 | 1 | 2 | 3 | 4       | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12      | 13 | 14 | 15 |
| R     | 0 | 0 | 0 | 0 | PADSEL0 |   |   |   | 0 | 0 | 0  | 0  | PADSEL1 |    |    |    |
| W     |   |   |   |   |         |   |   |   |   |   |    |    |         |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0       | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0       | 0  | 0  | 0  |

|       |    |    |    |    |         |    |    |    |    |    |    |    |         |    |    |    |
|-------|----|----|----|----|---------|----|----|----|----|----|----|----|---------|----|----|----|
|       | 16 | 17 | 18 | 19 | 20      | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28      | 29 | 30 | 31 |
| R     | 0  | 0  | 0  | 0  | PADSEL2 |    |    |    | 0  | 0  | 0  | 0  | PADSEL3 |    |    |    |
| W     |    |    |    |    |         |    |    |    |    |    |    |    |         |    |    |    |
| Reset | 0  | 0  | 0  | 0  | 0       | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0       | 0  | 0  | 0  |

Figure 20-10. Pad Selection for Multiplexed Inputs Register (PSMI0\_3)

Table 20-12. PSMI0\_3 field descriptions

| Field  | Description  |
|--|--|
| PADSEL0–3,<br>PADSEL4–7,<br>...<br>PADSEL60–63 | Pad Selection Bits<br>Each PADSEL field selects the pad currently used for a certain input function. See <a href="#">Table 20-13</a> . |

In order to multiplex different pads to the same peripheral input, the SIUL provides a register that controls the selection between the different sources.

Table 20-13. Peripheral input pin selection

| PSMI registers | PADSEL fields | SIUL address offset | Function / Peripheral | Mapping <sup>1</sup>                                   |
|----------------|---------------|---------------------|-----------------------|--|
| PSMI0_3        | PADSEL0       | 0x500               | CAN1RX / FlexCAN_1    | 00: PCR[35]<br>01: PCR[43]<br>10: PCR[95] <sup>2</sup> |
|                | PADSEL1       | 0x501               | CAN2RX / FlexCAN_2    | 00: PCR[73]<br>01: PCR[89] <sup>2</sup>                |
|                | PADSEL2       | 0x502               | CAN3RX / FlexCAN_3    | 00: PCR[36]<br>01: PCR[73]<br>10: PCR[89] <sup>2</sup> |
|                | PADSEL3       | 0x503               | CAN4RX / FlexCAN_4    | 00: PCR[35]<br>01: PCR[43]<br>10: PCR[95] <sup>2</sup> |

Table 20-13. Peripheral input pin selection (continued)

| PSMI registers | PADSEL fields | SIUL address offset | Function / Peripheral | Mapping <sup>1</sup>   |
|----------------|---------------|---------------------|-----------------------|--|
| PSMI4_7        | PADSEL4       | 0x504               | CAN5RX / FlexCAN_5    | 00: PCR[64]<br>01: PCR[97] <sup>2</sup>  |
|                | PADSEL5       | 0x505               | SCK_0 / DSPI_0        | 00: PCR[14]<br>01: PCR[15]   |
|                | PADSEL6       | 0x506               | CS0_0 / DSPI_0        | 00: PCR[14]<br>01: PCR[15]<br>10: PCR[27] <sup>3</sup>   |
|                | PADSEL7       | 0x507               | SCK_1 / DSPI_1        | 00: PCR[34]<br>01: PCR[68]<br>10: PCR[114] <sup>2</sup>  |
| PSMI8_11       | PADSEL8       | 0x508               | SIN_1 / DSPI_1        | 00: PCR[36]<br>01: PCR[66]<br>10: PCR[112] <sup>2</sup>  |
|                | PADSEL9       | 0x509               | CS0_1 / DSPI_1        | 00: PCR[]<br>01: PCR[61]<br>10: PCR[69]<br>11: PCR[115] <sup>2</sup>                             |
|                | PADSEL10      | 0x50A               | SCK_2 / DSPI_2        | 00: PCR[46]<br>01: PCR[78] <sup>2</sup><br>10: PCR[105] <sup>2</sup>                             |
|                | PADSEL11      | 0x50B               | SIN_2 / DSPI_2        | 00: PCR[44]<br>01: PCR[76]   |
| PSMI12_15      | PADSEL12      | 0x50C               | CS0_2 / DSPI_2        | 00: PCR[47]<br>01: PCR[79] <sup>2</sup><br>10: PCR[82] <sup>2</sup><br>11: PCR[104] <sup>2</sup> |
|                | PADSEL13      | 0x50D               | E1UC[3] / eMIOS_0     | 00: PCR[3]<br>01: PCR[27] <sup>3</sup><br>10: PCR[40]  |
|                | PADSEL14      | 0x50E               | E0UC[4] / eMIOS_0     | 00: PCR[4]<br>01: PCR[28]  |
|                | PADSEL15      | 0x50F               | E0UC[5] / eMIOS_0     | 00: PCR[5]<br>01: PCR[29]  |
| PSMI16_19      | PADSEL16      | 0x510               | E0UC[6] / eMIOS_0     | 00: PCR[6]<br>01: PCR[30]  |
|                | PADSEL17      | 0x511               | E0UC[7] / eMIOS_0     | 00: PCR[7]<br>01: PCR[31]<br>10: PCR[41]   |
|                | PADSEL18      | 0x512               | E0UC[10] / eMIOS_0    | 00: PCR[10]<br>01: PCR[80] <sup>2</sup>  |
|                | PADSEL19      | 0x513               | E0UC[11] / eMIOS_0    | 00: PCR[11]<br>01: PCR[81] <sup>2</sup>  |

Table 20-13. Peripheral input pin selection (continued)

| PSMI registers | PADSEL fields | SIUL address offset | Function / Peripheral | Mapping <sup>1</sup>  |
|----------------|---------------|---------------------|-----------------------|---|
| PSMI20_23      | PADSEL20      | 0x514               | E0UC[12] / eMIOS_0    | 00: PCR[44]<br>01: PCR[82] <sup>2</sup>   |
|                | PADSEL21      | 0x515               | E0UC[13] / eMIOS_0    | 00: PCR[45]<br>01: PCR[83] <sup>2</sup><br>10: PCR[0]                               |
|                | PADSEL22      | 0x516               | E0UC[14] / eMIOS_0    | 00: PCR[46]<br>01: PCR[84] <sup>2</sup><br>10: PCR[8]                               |
|                | PADSEL23      | 0x517               | E0UC[22] / eMIOS_0    | 00: PCR[70]<br>01: PCR[72]<br>10: PCR[85] <sup>2</sup>                              |
| PSMI24_27      | PADSEL24      | 0x518               | E0UC[23] / eMIOS_0    | 00: PCR[71]<br>01: PCR[73]<br>10: PCR[86] <sup>2</sup>                              |
|                | PADSEL25      | 0x519               | E0UC[24] / eMIOS_0    | 00: PCR[60]<br>01: PCR[106] <sup>2</sup><br>10: PCR[75]                             |
|                | PADSEL26      | 0x51A               | E0UC[25] / eMIOS_0    | 00: PCR[61]<br>01: PCR[107] <sup>2</sup>  |
|                | PADSEL27      | 0x51B               | E0UC[26] / eMIOS_0    | 00: PCR[62]<br>01: PCR[108] <sup>2</sup>  |
| PSMI28_31      | PADSEL28      | 0x51C               | E0UC[27] / eMIOS_0    | 00: PCR[63]<br>01: PCR[109] <sup>2</sup>  |
|                | PADSEL29      | 0x51D               | SCL / f_0             | 00: PCR[11]<br>01: PCR[19]  |
|                | PADSEL30      | 0x51E               | SDA / I2C_0           | 00: PCR[10]<br>01: PCR[18]  |
|                | PADSEL31      | 0x51F               | LIN3RX / LINFlex_3    | 00: PCR[8]<br>01: PCR[75]   |
| PSMI32_35      | PADSEL32      | 0x520               | SCK_3 / DSPI_3        | 00: PCR[100] <sup>2</sup><br>01: PCR[124] <sup>3</sup>                              |
|                | PADSEL33      | 0x521               | SIN_3 / DSPI_3        | 00: PCR[101] <sup>2</sup><br>01: PCR[139] <sup>3</sup>                              |
|                | PADSEL34      | 0x522               | CS0_3 / DSPI_3        | 00: PCR[99] <sup>2</sup><br>01: PCR[125] <sup>3</sup><br>10: PCR[140] <sup>3</sup>  |
|                | PADSEL35      | 0x523               | SCK_4 / DSPI_4        | 00: PCR[109] <sup>2</sup><br>01: PCR[126] <sup>3</sup><br>10: PCR[133] <sup>3</sup> |

Table 20-13. Peripheral input pin selection (continued)

| PSMI registers | PADSEL fields | SIUL address offset | Function / Peripheral     | Mapping <sup>1</sup>  |
|----------------|---------------|---------------------|---------------------------|---|
| PSMI36_39      | PADSEL36      | 0x524               | SIN_4 /<br>DSPI_4Reserved | 00: PCR[106] <sup>2</sup><br>01:<br>PCR[142] <sup>3</sup> —   |
|                | PADSEL37      | 0x525               | CS0_4 /<br>DSPI_4Reserved | 00: PCR[107] <sup>2</sup><br>01: PCR[123] <sup>3</sup><br>10: PCR[134] <sup>3</sup><br>11:<br>PCR[143] <sup>3</sup> — |
|                | PADSEL38      | 0x526               | E0UC[0] / eMIOS_0         | 00: PCR[0]<br>01: PCR[14]   |
|                | PADSEL39      | 0x527               | E0UC[1] / eMIOS_0         | 00: PCR[1]<br>01: PCR[15]   |
| PSMI40_43      | PADSEL40      | 0x528               | E0UC[28] / eMIOS_0        | 00: PCR[12]<br>01: PCR[128] <sup>3</sup>  |
|                | PADSEL41      | 0x529               | E0UC[29] / eMIOS_0        | 00: PCR[13]<br>01: PCR[129] <sup>3</sup>  |
|                | PADSEL42      | 0x52A               | E0UC[30] / eMIOS_0        | 00: PCR[16]<br>01: PCR[18]<br>10: PCR[130] <sup>3</sup>   |
|                | PADSEL43      | 0x52B               | E0UC[31] / eMIOS0         | 00: PCR[17]<br>01: PCR[19]<br>10: PCR[131] <sup>3</sup>   |
| PSMI44_47      | PADSEL44      | 0x52C               | E1UC[1] / eMIOS_1         | 00: PCR[111] <sup>2</sup><br>01: PCR[89] <sup>2</sup>   |
|                | PADSEL45      | 0x52D               | E1UC[2] / eMIOS_1         | 00: PCR[112] <sup>2</sup><br>01: PCR[90] <sup>2</sup>   |
|                | PADSEL46      | 0x52E               | E1UC[3] / eMIOS_1         | 00: PCR[113] <sup>2</sup><br>01: PCR[91] <sup>2</sup>   |
|                | PADSEL47      | 0x52F               | E1UC[4] / eMIOS_1         | 00: PCR[114] <sup>2</sup><br>01: PCR[95] <sup>2</sup>   |
| PSMI48_51      | PADSEL48      | 0x530               | E1UC[5] / eMIOS_1         | 00: PCR[115] <sup>2</sup><br>01: PCR[123] <sup>3</sup>  |
|                | PADSEL49      | 0x531               | E1UC[17] / eMIOS_1        | 00: PCR[104] <sup>2</sup><br>01: PCR[127] <sup>3</sup>  |
|                | PADSEL50      | 0x532               | E1UC[18] / eMIOS_1        | 00: PCR[105] <sup>2</sup><br>01: PCR[148]   |
|                | PADSEL51      | 0x533               | E1UC[25] / eMIOS_1        | 00: PCR[92] <sup>2</sup><br>01: PCR[124] <sup>3</sup>   |

Table 20-13. Peripheral input pin selection (continued)

| PSMI registers         | PADSEL fields | SIUL address offset | Function / Peripheral          | Mapping <sup>1</sup>   |
|------------------------|---------------|---------------------|--------------------------------|--|
| PSMI52_55              | PADSEL52      | 0x534               | E1UC[26] / eMIOS_1             | 00: PCR[93] <sup>2</sup><br>01: PCR[125] <sup>3</sup>                      |
|                        | PADSEL53      | 0x535               | E1UC[27] / eMIOS_1             | 00: PCR[94] <sup>2</sup><br>01: PCR[126] <sup>3</sup>                      |
|                        | PADSEL54      | 0x536               | E1UC[28] / eMIOS_1             | 00: PCR[38]<br>01: PCR[132] <sup>3</sup>                                   |
|                        | PADSEL55      | 0x537               | E1UC[29] / eMIOS_1             | 00: PCR[39]<br>01: PCR[133] <sup>3</sup>                                   |
| PSMI56_59              | PADSEL56      | 0x538               | E1UC[30] /<br>eMIOS_1Reserved  | 00: PCR[74]<br>01: PCR[103] <sup>2</sup><br>10:<br>PCR[134] <sup>3</sup> — |
|                        | PADSEL57      | 0x539               | E1UC[31] /<br>eMIOS_1Reserved  | 00: PCR[36]<br>01: PCR[106] <sup>2</sup><br>10:<br>PCR[135] <sup>3</sup> — |
|                        | PADSEL58      | 0x53A               | LIN2RX / LINFlex _2            | 00: PCR[41]<br>01: PCR[11]   |
|                        | PADSEL59      | 0x53B               | LIN4RX / LINFlex<br>_4Reserved | 00: PCR[6]<br>01: PCR[91] <sup>2</sup> —                                   |
| PSMI60_63 <sup>4</sup> | PADSEL60      | 0x53C               | LIN5RX / LINFlex _5            | 00: PCR[4]<br>01: PCR[93] <sup>2</sup>                                     |
|                        | PADSEL61      | 0x53D               | Reserved                       |  |
|                        | PADSEL62      | 0x53E               | LIN0RX/LINFlexD_0              | 00: PCR[19]<br>01: PCR[17]   |

<sup>1</sup> See Chapter 4, [Signal description](#), for correspondence between PCR and pinout

<sup>2</sup> Not available in 100-pin LQFP

<sup>3</sup> Available only in 176-pin LQFP and 208 BGA packages

<sup>4</sup> PADSEL63 not implemented

### 20.5.3.10 GPIO Pad Data Output Registers (GPDO0\_3–GPDO148\_151)

These registers are used to set or clear GPIO pads. Each pad data out bit can be controlled separately with a byte access.

Offsets: 0x0600–0x06A0 (38 registers)

Access: User read/write

|       |   |   |   |   |   |   |   |        |   |   |    |    |    |    |    |        |
|-------|---|---|---|---|---|---|---|--------|---|---|----|----|----|----|----|--------|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7      | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15     |
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | PDO[0] | 0 | 0 | 0  | 0  | 0  | 0  | 0  | PDO[1] |
| W     |   |   |   |   |   |   |   |        |   |   |    |    |    |    |    |        |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0      | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0      |

|       |    |    |    |    |    |    |    |        |    |    |    |    |    |    |    |        |
|-------|----|----|----|----|----|----|----|--------|----|----|----|----|----|----|----|--------|
|       | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23     | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31     |
| R     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | PDO[2] | 0  | 0  | 0  | 0  | 0  | 0  | 0  | PDO[3] |
| W     |    |    |    |    |    |    |    |        |    |    |    |    |    |    |    |        |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0      | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0      |

Figure 20-11. Port GPIO Pad Data Output Register 0–3 (GPDO0\_3)

Table 20-14. GPDO0\_3 field descriptions

| Field  | Description   |
|--------|---|
| PDO[x] | <p>Pad Data Out</p> <p>This bit stores the data to be driven out on the external GPIO pad controlled by this register.</p> <p>0 Logic low value is driven on the corresponding GPIO pad when the pad is configured as an output</p> <p>1 Logic high value is driven on the corresponding GPIO pad when the pad is configured as an output</p> |

**CAUTION**

Toggling several IOs at the same time can significantly increase the current in a pad group. Caution must be taken to avoid exceeding maximum current thresholds. Please see data sheet.

**20.5.3.11 GPIO Pad Data Input Registers (GPDI0\_3–GPDI148\_151)**

These registers are used to read the GPIO pad data with a byte access.

Offsets: 0x0800–0x08A0 (38 registers)

Access: User read

|       |    |    |    |    |    |    |    |        |    |    |    |    |    |    |    |        |
|-------|----|----|----|----|----|----|----|--------|----|----|----|----|----|----|----|--------|
|       | 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7      | 8  | 9  | 10 | 11 | 12 | 13 | 14 | 15     |
| R     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | PDI[0] | 0  | 0  | 0  | 0  | 0  | 0  | 0  | PDI[1] |
| W     |    |    |    |    |    |    |    |        |    |    |    |    |    |    |    |        |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0      | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0      |
|       | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23     | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31     |
| R     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | PDI[2] | 0  | 0  | 0  | 0  | 0  | 0  | 0  | PDI[3] |
| W     |    |    |    |    |    |    |    |        |    |    |    |    |    |    |    |        |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0      | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0      |

Figure 20-12. Port GPIO Pad Data Input Register 0–3 (GPDIO\_3)

Table 20-15. GPDIO\_3 field descriptions

| Field  | Description   |
|--------|---|
| PDI[x] | Pad Data In<br>This bit stores the value of the external GPIO pad associated with this register.<br>0 Value of the data in signal for the corresponding GPIO pad is logic low<br>1 Value of the data in signal for the corresponding GPIO pad is logic high |

### 20.5.3.12 Parallel GPIO Pad Data Out Registers (PGPDO0 – PGPDO4)

MPC5606BK devices ports are constructed such that they contain 16 GPIO pins, for example PortA[0..15]. Parallel port registers for input (PGPDI) and output (PGPDO) are provided to allow a complete port to be written or read in one operation, dependent on the individual pad configuration.

Writing a parallel PGPDO register directly sets the associated GPDO register bits. There is also a masked parallel port output register allowing the user to determine which pins within a port are written.

While very convenient and fast, this approach does have implications regarding current consumption for the device power segment containing the port GPIO pads. Toggling several GPIO pins simultaneously can significantly increase current consumption.

#### CAUTION

Caution must be taken to avoid exceeding maximum current thresholds when toggling multiple GPIO pins simultaneously. Please see data sheet.

Table 20-16 shows the locations and structure of the PGPDOx registers.

**Table 20-16. PGPDO0 – PGPDO4 register map**

| Offset <sup>1</sup> | Register | Field  |   |   |   |   |   |   |   |   |   |    |    |    |    |    |        |    |    |    |    |    |          |    |    |    |    |    |    |    |    |    |    |
|---------------------|----------|--------|---|---|---|---|---|---|---|---|---|----|----|----|----|----|--------|----|----|----|----|----|----------|----|----|----|----|----|----|----|----|----|----|
|                     |          | 0      | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15     | 16 | 17 | 18 | 19 | 20 | 21       | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| 0x0C00              | PGPDO0   | Port A |   |   |   |   |   |   |   |   |   |    |    |    |    |    | Port B |    |    |    |    |    |          |    |    |    |    |    |    |    |    |    |    |
| 0x0C04              | PGPDO1   | Port C |   |   |   |   |   |   |   |   |   |    |    |    |    |    | Port D |    |    |    |    |    |          |    |    |    |    |    |    |    |    |    |    |
| 0x0C08              | PGPDO2   | Port E |   |   |   |   |   |   |   |   |   |    |    |    |    |    | Port F |    |    |    |    |    |          |    |    |    |    |    |    |    |    |    |    |
| 0x0C0C              | PGPDO3   | Port G |   |   |   |   |   |   |   |   |   |    |    |    |    |    | Port H |    |    |    |    |    |          |    |    |    |    |    |    |    |    |    |    |
| 0x0C10              | PGPDO4   | Port I |   |   |   |   |   |   |   |   |   |    |    |    |    |    | Port J |    |    |    |    |    | Reserved |    |    |    |    |    |    |    |    |    |    |

<sup>1</sup> SIU base address is 0xC3F9\_0000. To calculate register address add offset to base address

It is important to note the bit ordering of the ports in the parallel port registers. The most significant bit of the parallel port register corresponds to the least significant pin in the port.

For example in [Table 20-16](#), the PGPDO0 register contains fields for Port A and Port B.

- Bit 0 is mapped to Port A[0], bit 1 is mapped to Port A[1] and so on, through bit 15, which is mapped to Port A[15]
- Bit 16 is mapped to Port B[0], bit 17 is mapped to Port B[1] and so on, through bit 31, which is mapped to Port B[15].

### 20.5.3.13 Parallel GPIO Pad Data In Registers (PGPDI0 – PGPDI4)

The SIU\_PGPDI registers are similar in operation to the PGPDI0 registers, described in the previous section ([Section 20.5.3.12, Parallel GPIO Pad Data Out Registers \(PGPDO0 – PGPDO4\)](#)) but they are used to read port pins simultaneously.

#### NOTE

The port pins to be read need to be configured as inputs but even if a single pin within a port has IBE set, then you can still read that pin using the parallel port register. However, this does mean you need to be very careful.

Reads of PGPDI registers are equivalent to reading the corresponding GPDIO registers but significantly faster since as many as two ports can be read simultaneously with a single 32-bit read operation.

[Table 20-17](#) shows the locations and structure of the PGPDIx registers. Each 32-bit PGPDIx register contains two 16-bit fields, each field containing the values for a separate port.

**Table 20-17. PGPDI0 – PGPDI4 register map**

| Offset <sup>1</sup> | Register | Field  |   |   |   |   |   |   |   |   |   |    |    |    |    |    |        |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|---------------------|----------|--------|---|---|---|---|---|---|---|---|---|----|----|----|----|----|--------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|                     |          | 0      | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15     | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| 0x0C40              | PGPDI0   | Port A |   |   |   |   |   |   |   |   |   |    |    |    |    |    | Port B |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0x0C44              | PGPDI1   | Port C |   |   |   |   |   |   |   |   |   |    |    |    |    |    | Port D |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0x0C48              | PGPDI2   | Port E |   |   |   |   |   |   |   |   |   |    |    |    |    |    | Port F |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0x0C4C              | PGPDI3   | Port G |   |   |   |   |   |   |   |   |   |    |    |    |    |    | Port H |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |



Table 20-17. PGPDI0 – PGPDI4 register map (continued)

| Offset <sup>1</sup> | Register | Field  |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |        |    |    |    |    |    |    |    |          |    |    |    |    |    |    |    |
|---------------------|----------|--------|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|--------|----|----|----|----|----|----|----|----------|----|----|----|----|----|----|----|
|                     |          | 0      | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16     | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24       | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| 0x0C50              | PGPDI4   | Port I |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    | Port J |    |    |    |    |    |    |    | Reserved |    |    |    |    |    |    |    |

<sup>1</sup> SIU base address is 0xC3F9\_0000. To calculate register address add offset to base address

It is important to note the bit ordering of the ports in the parallel port registers. The most significant bit of the parallel port register corresponds to the least significant pin in the port.

For example in [Table 20-17](#), the PGPDI0 register contains fields for Port A and Port B.

- Bit 0 is mapped to Port A[0], bit 1 is mapped to Port A[1] and so on, through bit 15, which is mapped to Port A[15]
- Bit 16 is mapped to Port B[0], bit 17 is mapped to Port B[1] and so on, through bit 31, which is mapped to Port B[15].

#### 20.5.3.14 Masked Parallel GPIO Pad Data Out Register (MPGPDO0–MPGPDO9)

The MPGPDO<sub>x</sub> registers are similar in operation to the PGPDO<sub>x</sub> ports described in [Section 20.5.3.12, Parallel GPIO Pad Data Out Registers \(PGPDO0 – PGPDO4\)](#), but with two significant differences:

- The MPGPDO<sub>x</sub> registers support *masked* port-wide changes to the data out on the pads of the respective port. Masking effectively allows selective bitwise writes to the full 16-bit port.
- Each 32-bit MPGPDO<sub>x</sub> register is associated to only one port.

#### NOTE

The MPGPDO<sub>x</sub> registers may only be accessed with 32-bit writes. 8-bit or 16-bit writes will not modify any bits in the register and will cause a transfer error response by the module. Read accesses return 0.

[Table 20-18](#) shows the locations and structure of the MPGPDO<sub>x</sub> registers. Each 32-bit MPGPDO<sub>x</sub> register contains two 16-bit fields (MASK<sub>x</sub> and MPPDO<sub>x</sub>). The MASK field is a bitwise mask for its associated port. The MPPDO0 field contains the data to be written to the port.

Table 20-18. MPGPDO0 – MPGPDO9 register map

| Offset <sup>1</sup> | Register | Field          |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |                 |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|---------------------|----------|----------------|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|-----------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|                     |          | 0              | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16              | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| 0x0C80              | MPGPDO0  | MASK0 (Port A) |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    | MPPDO0 (Port A) |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0x0C84              | MPGPDO1  | MASK1 (Port B) |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    | MPPDO1 (Port B) |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0x0C88              | MPGPDO2  | MASK2 (Port C) |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    | MPPDO2 (Port C) |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0x0C8C              | MPGPDO3  | MASK3 (Port D) |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    | MPPDO3 (Port D) |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0x0C90              | MPGPDO4  | MASK4 (Port E) |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    | MPPDO4 (Port E) |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0x0C94              | MPGPDO5  | MASK5 (Port F) |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    | MPPDO5 (Port F) |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0x0C98              | MPGPDO6  | MASK6 (Port G) |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    | MPPDO6 (Port G) |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**Table 20-18. MPGPDO0 – MPGPDO9 register map (continued)**

| Offset <sup>1</sup> | Register | Field          |   |   |          |   |   |   |   |   |   |    |    |    |    |    |                 |    |    |          |    |    |    |    |    |    |    |    |    |    |    |    |    |
|---------------------|----------|----------------|---|---|----------|---|---|---|---|---|---|----|----|----|----|----|-----------------|----|----|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|
|                     |          | 0              | 1 | 2 | 3        | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15              | 16 | 17 | 18       | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| 0x0C9C              | MPGPDO7  | MASK7 (Port H) |   |   |          |   |   |   |   |   |   |    |    |    |    |    | MPPDO7 (Port H) |    |    |          |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0x0CA0              | MPGPDO8  | MASK8 (Port I) |   |   |          |   |   |   |   |   |   |    |    |    |    |    | MPPDO8 (Port I) |    |    |          |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 0x0CAF              | MPGPDO9  | MASK9 (Port J) |   |   | Reserved |   |   |   |   |   |   |    |    |    |    |    | MPPDO9 (Port J) |    |    | Reserved |    |    |    |    |    |    |    |    |    |    |    |    |    |

<sup>1</sup> SIU base address is 0xC3F9\_0000. To calculate register address add offset to base address

It is important to note the bit ordering of the ports in the parallel port registers. The most significant bit of the parallel port register corresponds to the least significant pin in the port.

For example in [Table 20-18](#), the MPGPDO0 register contains field MASK0, which is the bitwise mask for Port A and field MPPDO0, which contains data to be written to Port A.

- MPGPDO0[0] is the mask bit for Port A[0], MPGPDO0[1] is the mask bit for Port A[1] and so on, through MPGPDO0[15], which is the mask bit for Port A[15]
- MPGPDO0[16] is the data bit mapped to Port A[0], MPGPDO0[17] is mapped to Port A[1] and so on, through MPGPDO0[31], which is mapped to Port A[15].

**Table 20-19. MPGPDO0..MPGPDO9 field descriptions**

| Field                        | Description  |
|------------------------------|--|
| MASK <sub>x</sub><br>[15:0]  | Mask Field<br>Each bit corresponds to one data bit in the MPPDO <sub>x</sub> register at the same bit location.<br>0 Associated bit value in the MPPDO <sub>x</sub> field is ignored<br>1 Associated bit value in the MPPDO <sub>x</sub> field is written  |
| MPPDO <sub>x</sub><br>[15:0] | Masked Parallel Pad Data Out<br>Write the data register that stores the value to be driven on the pad in output mode.<br>Accesses to this register location are coherent with accesses to the bitwise GPIO Pad Data Output Registers (GPDO0_3–GPDO148_151).<br>The x and bit index define which MPPDO register bit is equivalent to which PDO register bit according to the following equation:<br>$MPPDO[x][y] = PDO[(x * 16) + y]$ |

**CAUTION**

Toggling several IOs at the same time can significantly increase the current in a pad group. Caution must be taken to avoid exceeding maximum current thresholds. Please see data sheet.

**20.5.3.15 Interrupt Filter Maximum Counter Registers (IFMC0–IFMC23)**

These registers are used to configure the filter counter associated with each digital glitch filter.

**NOTE**

For the pad transition to trigger an interrupt it must be steady for at least the filter period.

Offset: 0x1000–0x105C) (24 registers)

Access: User read/write

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       |    |    |    |    |    |    |    |    |    |    |    |    |         |    |    |    |
|-------|----|----|----|----|----|----|----|----|----|----|----|----|---------|----|----|----|
|       | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28      | 29 | 30 | 31 |
| R     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | MAXCNTx |    |    |    |
| W     |    |    |    |    |    |    |    |    |    |    |    |    |         |    |    |    |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0       | 0  | 0  | 0  |

Figure 20-13. Interrupt Filter Maximum Counter Registers (IFMC0–IFMC23)

Table 20-20. IFMC field descriptions

| Field   | Description   |
|---------|---|
| MAXCNTx | Maximum Interrupt Filter Counter setting<br>$\text{Filter Period} = T(\text{CK}) \times \text{MAXCNTx} + n \times T(\text{CK})$<br>Where ( <i>n</i> can be 1 to 3)<br>MAXCNTx can be 0 to 15<br>T(CK): Prescaled Filter Clock Period, which is the FIRC clock prescaled to IFCP value<br>T(FIRC): Basic Filter Clock Period: 62.5 ns ( $f_{\text{FIRC}} = 16 \text{ MHz}$ ) |

### 20.5.3.16 Interrupt Filter Clock Prescaler Register (IFCPR)

This register is used to configure a clock prescaler that selects the clock for all digital filter counters in the SIUL.

Offsets: 0x1080

Access: User read/write

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       |    |    |    |    |    |    |    |    |    |    |    |    |      |    |    |    |
|-------|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|
|       | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28   | 29 | 30 | 31 |
| R     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | IFCP |    |    |    |
| W     |    |    |    |    |    |    |    |    |    |    |    |    |      |    |    |    |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0    | 0  | 0  | 0  |

Figure 20-14. Interrupt Filter Clock Prescaler Register (IFCPR)

Table 20-21. IFCPR field descriptions

| Field | Description  |
|-------|--|
| IFCP  | Interrupt Filter Clock Prescaler setting<br>Prescaled Filter Clock Period = $T(\text{FIRC}) \times (\text{IFCP} + 1)$<br>$T(\text{FIRC})$ is the fast internal RC oscillator period.<br>IFCP can be 0 to 15. |

## 20.6 Functional description

### 20.6.1 Pad control

The SIUL controls the configuration and electrical characteristic of the device pads. It provides a consistent interface for all pads, both on a by-port and a by-bit basis. The pad configuration registers ( $\text{PCR}_n$ , see [Section 20.5.3.8, Pad Configuration Registers \(PCR0–PCR148\)](#)) allow software control of the static electrical characteristics of external pins with a single write. These are used to configure the following pad features:

- Open drain output enable
- Slew rate control
- Pull control
- Pad assignment
- Control of analog path switches
- Safe mode behavior configuration

### 20.6.2 General purpose input and output pads (GPIO)

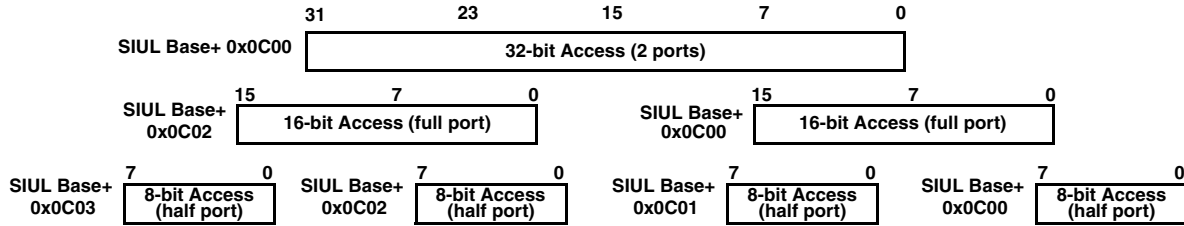
The SIUL manages as many as 149 GPIO pads organized as ports that can be accessed for data reads and writes as 32-, 16-, or 8-bit<sup>1</sup>.

#### NOTE

Ports are organized as groups of 16 GPIO pads, with the exception of Port J, which has 5. A 32-bit R/W operation accesses two ports simultaneously. A 16-bit operation accesses a full port and an 8-bit access either the upper or lower byte of a port.

As shown in [Figure 20-15](#), all port accesses are identical with each read or write being performed only at a different location to access a different port width.

<sup>1</sup>. There are exceptions. Some pads, e.g., precision analog pads, are input only.



**Figure 20-15. Data Port example arrangement showing configuration for different port width accesses**

The SIUL has separate data input (GPDI $n_n$ , see [Section 20.5.3.11, GPIO Pad Data Input Registers \(GPDI0\\_3–GPDI148\\_151\)](#)) and data output (GPDO $n_n$ , see [Section 20.5.3.10, GPIO Pad Data Output Registers \(GPDO0\\_3–GPDO148\\_151\)](#)) registers for all pads, allowing the possibility of reading back an input or output value of a pad directly. This supports the ability to validate what is present on the pad rather than simply confirming the value that was written to the data register by accessing the data input registers.

Data output registers allow an output pad to be driven high or low (with the option of push-pull or open drain drive). Input registers are read-only and reflect the respective pad value.

When the pad is configured to use one of its alternate functions, the data input value reflects the respective value of the pad. If a write operation is performed to the data output register for a pad configured as an alternate function (non-GPIO), this write will not be reflected by the pad value until reconfigured to GPIO.

The allocation of what input function is connected to the pin is defined by the PSMI registers (PCR $n$ , see [Section 20.5.3.9, Pad Selection for Multiplexed Inputs Registers \(PSMI0\\_3–PSMI60\\_63\)](#)).

### 20.6.3 External interrupts

The SIUL supports 24 external interrupts, EIRQ0–EIRQ23. Mapping is shown for external interrupts to pads in [Chapter 4, Signal description](#).

The SIUL supports three interrupt vectors to the interrupt controller. Each vector interrupt has eight external interrupts combined together with the presence of flag generating an interrupt for that vector if enabled. All of the external interrupt pads within a single group have equal priority.

See [Figure 20-16](#) for an overview of the external interrupt implementation.

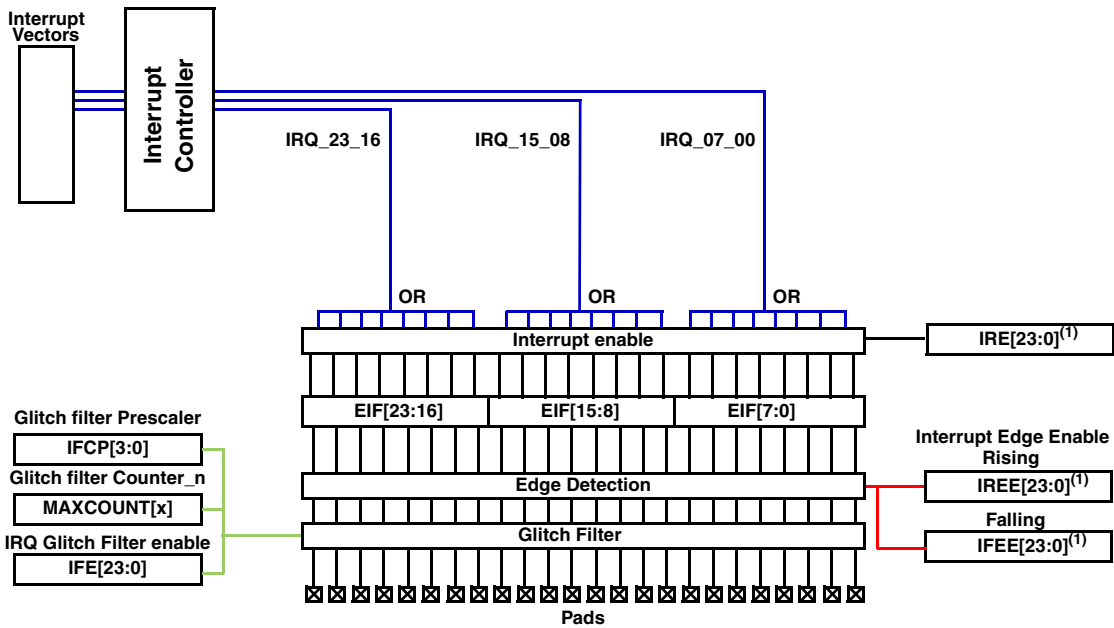


Figure 20-16. External interrupt pad diagram

<sup>3</sup> 20 interrupts in 100-pin LQFP.

Each interrupt can be enabled or disabled independently. This can be performed using the IREER. A pad defined as an external interrupt can be configured to recognize interrupts with an active rising edge, an active falling edge or both edges being active. A setting of having both edge events disabled is reserved and should not be configured.

The active EIRQ edge is controlled through the configuration of the registers IREER and IFEEER.

Each external interrupt supports an individual flag in the Interrupt Status Flag Register (ISR). The bits in the ISR[EIF] field are cleared by writing a 1 to them; this prevents inadvertent overwriting of other flags in the register.

## 20.7 Pin muxing

For pin muxing, please see [Chapter 4, Signal description](#).

# Chapter 21

## Memory Protection Unit (MPU)

### 21.1 Introduction

The Memory Protection Unit (MPU) provides hardware access control for all memory references generated in the device. Using preprogrammed region descriptors that define memory spaces and their associated access rights, the MPU concurrently monitors all system bus transactions and evaluates the appropriateness of each transfer. Memory references that have sufficient access control rights are allowed to complete, while references that are not mapped to any region descriptor or have insufficient rights are terminated with a protection error response.

The MPU module provides the following capabilities:

- Support for 8 program-visible 128-bit (4-word) region descriptors
  - Each region descriptor defines a modulo-32 byte space, aligned anywhere in memory
    - Region sizes can vary from a minimum of 32 bytes to a maximum of 4 Gbytes
  - Two types of access control permissions defined in single descriptor word
    - Processors have separate {read, write, execute} attributes for supervisor and user accesses
    - Non-processor masters have {read, write} attributes
  - Hardware-assisted maintenance of the descriptor valid bit minimizes coherency issues
  - Alternate programming model view of the access control permissions word
- Memory-mapped platform device
  - Interface to three slave XBAR ports: flash controller, system SRAM controller and peripherals bus
    - Connections to the address phase address and attributes
    - Typical location is immediately downstream of the platform's crossbar switch

A simplified block diagram of the MPU module is shown in [Figure 21-1](#).

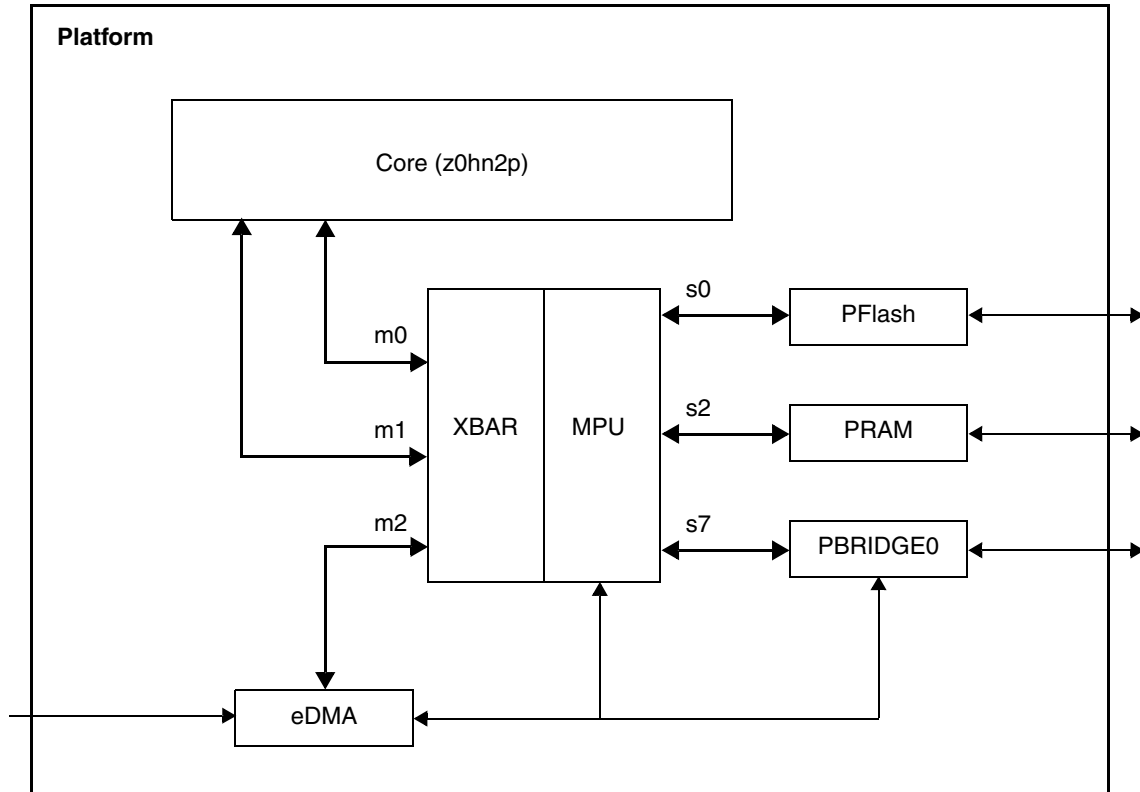


Figure 21-1. MPU block diagram

## 21.2 Features

The Memory Protection Unit implements a two-dimensional hardware array of memory region descriptors and the crossbar slave XBAR ports to continuously monitor the legality of every memory reference generated by each bus master in the system. The feature set includes:

- Support for eight memory region descriptors, each 128 bits in size
  - Specification of start and end addresses provide granularity for region sizes from 32 bytes to 4 GB
  - Access control definitions: two bus masters (processor cores) support the traditional {read, write, execute} permissions with independent definitions for supervisor and user mode accesses
  - Automatic hardware maintenance of the region descriptor valid bit removes issues associated with maintaining a coherent image of the descriptor
  - Alternate memory view of the access control word for each descriptor provides an efficient mechanism to dynamically alter only the access rights of a descriptor
  - For overlapping region descriptors, priority is given to permission granting over access denying as this approach provides more flexibility to system software. See [Section 21.6.2, Putting it all together and AHB error terminations](#), for details and [Section 21.8, Application information](#), for an example.



- Support for three XBAR slave port connections: flash controller, system SRAM controller and peripherals bus:
  - MPU hardware continuously monitors every XBAR slave port access using the preprogrammed memory region descriptors.
  - An access protection error is detected if a memory reference does not hit in any memory region or the reference is flagged as illegal in all memory regions where it does hit. In the event of an access error, the XBAR reference is terminated with an error response and the MPU inhibits the bus cycle being sent to the targeted slave device.
  - 64-bit error registers, one for each XBAR slave port, capture the last faulting address, attributes, and detail information.
- Global MPU enable/disable control bit provides a mechanism to easily load region descriptors during system startup or allow complete access rights during debug with the module disabled.

### 21.3 Modes of operation

The MPU module does not support any special modes of operation. As a memory-mapped device located on the platform's high-speed system bus, it responds based strictly on the memory addresses of the connected system buses. The peripheral bus is used to access the MPU's programming model and the memory protection functions are evaluated on a reference-by-reference basis using the addresses from the XBAR system bus port(s).

Power dissipation is minimized when the MPU's global enable/disable bit is cleared (MPU\_CESR[VLD] = 0).

### 21.4 External signal description

The MPU module does not include any external interface. The MPU's internal interfaces include a peripheral bus connection for accessing the programming model and multiple connections to the address phase signals of the platform crossbar's slave AHB ports. From a platform topology viewpoint, the MPU module appears to be directly connected downstream from the crossbar switch with interfaces to the XBAR slave ports.

### 21.5 Memory map and register description

The MPU module provides an IPS programming model mapped to an SPP-standard on-platform 16 KB space. The programming model is partitioned into three groups: control/status registers, the data structure containing the region descriptors and the alternate view of the region descriptor access control values.

The programming model can only be referenced using 32-bit (word) accesses. Attempted references using different access sizes, to undefined (reserved) addresses, or with a non-supported access type (for example, a write to a read-only register or a read of a write-only register) generate an IPS error termination.

Finally, the programming model allocates space for an MPU definition with 8 region descriptors and as many as three XBAR slave ports, like flash controller, system SRAM controller and peripheral bus.

## 21.5.1 Memory map

The MPU programming model map is shown in [Table 21-1](#).

**Table 21-1. MPU memory map**

| Base address: 0xFFF1_1000 |   |                             |
|---------------------------|---|-----------------------------|
| Address offset            | Register  | Location                    |
| 0x000                     | MPU Control/Error Status Register (MPU_CESR)        | <a href="#">on page 379</a> |
| 0x004–0x00F               | Reserved  |                             |
| 0x010                     | MPU Error Address Register, Slave Port 0 (MPU_EAR0) | <a href="#">on page 380</a> |
| 0x014                     | MPU Error Detail Register, Slave Port 0 (MPU_EDR0)  | <a href="#">on page 381</a> |
| 0x018                     | MPU Error Address Register, Slave Port 1 (MPU_EAR1) | <a href="#">on page 380</a> |
| 0x01C                     | MPU Error Detail Register, Slave Port 1 (MPU_EDR1)  | <a href="#">on page 381</a> |
| 0x020                     | MPU Error Address Register, Slave Port 2 (MPU_EAR2) | <a href="#">on page 380</a> |
| 0x024                     | MPU Error Detail Register, Slave Port 2 (MPU_EDR2)  | <a href="#">on page 381</a> |
| 0x028–0x3FF               | Reserved  |                             |
| 0x400                     | MPU Region Descriptor 0 (MPU_RGD0)                  | <a href="#">on page 382</a> |
| 0x410                     | MPU Region Descriptor 1 (MPU_RGD1)                  | <a href="#">on page 382</a> |
| 0x420                     | MPU Region Descriptor 2 (MPU_RGD2)                  | <a href="#">on page 382</a> |
| 0x430                     | MPU Region Descriptor 3 (MPU_RGD3)                  | <a href="#">on page 382</a> |
| 0x440                     | MPU Region Descriptor 4 (MPU_RGD4)                  | <a href="#">on page 382</a> |
| 0x450                     | MPU Region Descriptor 5 (MPU_RGD5)                  | <a href="#">on page 382</a> |
| 0x460                     | MPU Region Descriptor 6 (MPU_RGD6)                  | <a href="#">on page 382</a> |
| 0x470                     | MPU Region Descriptor 7 (MPU_RGD7)                  | <a href="#">on page 382</a> |
| 0x480–0x7FF               | Reserved  |                             |
| 0x800                     | MPU RGD Alternate Access Control 0 (MPU_RGDAAC0)    | <a href="#">on page 387</a> |
| 0x804                     | MPU RGD Alternate Access Control 1 (MPU_RGDAAC1)    | <a href="#">on page 387</a> |
| 0x808                     | MPU RGD Alternate Access Control 2 (MPU_RGDAAC2)    | <a href="#">on page 387</a> |
| 0x80C                     | MPU RGD Alternate Access Control 3 (MPU_RGDAAC3)    | <a href="#">on page 387</a> |
| 0x810                     | MPU RGD Alternate Access Control 4 (MPU_RGDAAC4)    | <a href="#">on page 387</a> |
| 0x814                     | MPU RGD Alternate Access Control 5 (MPU_RGDAAC5)    | <a href="#">on page 387</a> |
| 0x818                     | MPU RGD Alternate Access Control 6 (MPU_RGDAAC6)    | <a href="#">on page 387</a> |
| 0x81C                     | MPU RGD Alternate Access Control 7 (MPU_RGDAAC7)    | <a href="#">on page 387</a> |

## 21.5.2 Register description

### 21.5.2.1 MPU Control/Error Status Register (MPU\_CESR)

The MPU\_CESR provides one byte of error status plus three bytes of configuration information. A global MPU enable/disable bit is also included in this register.

Offset: 0x000 Access: Read/Partial Write

|       |            |     |     |   |   |   |   |   |   |   |    |    |     |    |    |    |
|-------|------------|-----|-----|---|---|---|---|---|---|---|----|----|-----|----|----|----|
|       | 0          | 1   | 2   | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12  | 13 | 14 | 15 |
| R     | SPERR[0:2] |     |     | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0  | 0  | HRL |    |    |    |
| W     | w1c        | w1c | w1c |   |   |   |   |   |   |   |    |    |     |    |    |    |
| Reset | 0          | 0   | 0   | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0  | 0  | 0   | 0  | 0  | 0  |

|       |     |    |    |    |      |    |    |    |    |    |    |    |    |    |    |     |
|-------|-----|----|----|----|------|----|----|----|----|----|----|----|----|----|----|-----|
|       | 16  | 17 | 18 | 19 | 20   | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31  |
| R     | NSP |    |    |    | NRGD |    |    |    | 0  | 0  | 0  | 0  | 0  | 0  | 0  | VLD |
| W     |     |    |    |    |      |    |    |    |    |    |    |    |    |    |    |     |
| Reset | 0   | 0  | 1  | 1  | 0    | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   |

Figure 21-2. MPU Control/Error Status Register (MPU\_CESR)

Table 21-2. MPU\_CESR field descriptions

| Field     | Description  |
|-----------|--|
| SPERR $n$ | Slave Port $n$ Error, where the slave port number matches the bit number. See <a href="#">Table 21-3</a> . Each bit in this field represents a flag maintained by the MPU for signaling the presence of a captured error contained in the MPU_EAR $n$ and MPU_EDR $n$ registers. The individual bit is set when the hardware detects an error and records the faulting address and attributes. It is cleared when the corresponding bit is written as a logical one. If another error is captured at the exact same cycle as a write of a logical one, this flag remains set. A “find first one” instruction (or equivalent) can be used to detect the presence of a captured error.<br>0 The corresponding MPU_EAR $n$ /MPU_EDR $n$ registers do not contain a captured error.<br>1 The corresponding MPU_EAR $n$ /MPU_EDR $n$ registers do contain a captured error. |
| HRL       | Hardware Revision Level<br>This field specifies the MPU’s hardware and definition revision level. It can be read by software to determine the functional definition of the module.   |
| NSP       | Number of Slave Ports<br>This field specifies the number of slave ports [1–8] connected to the MPU.  |
| NRGD      | Number of Region Descriptors<br>This field specifies the number of region descriptors implemented in the MPU. The defined encodings include:<br>0000 8 region descriptors<br>0001 12 region descriptors<br>0010 16 region descriptors  |

**Table 21-2. MPU\_CESR field descriptions (continued)**

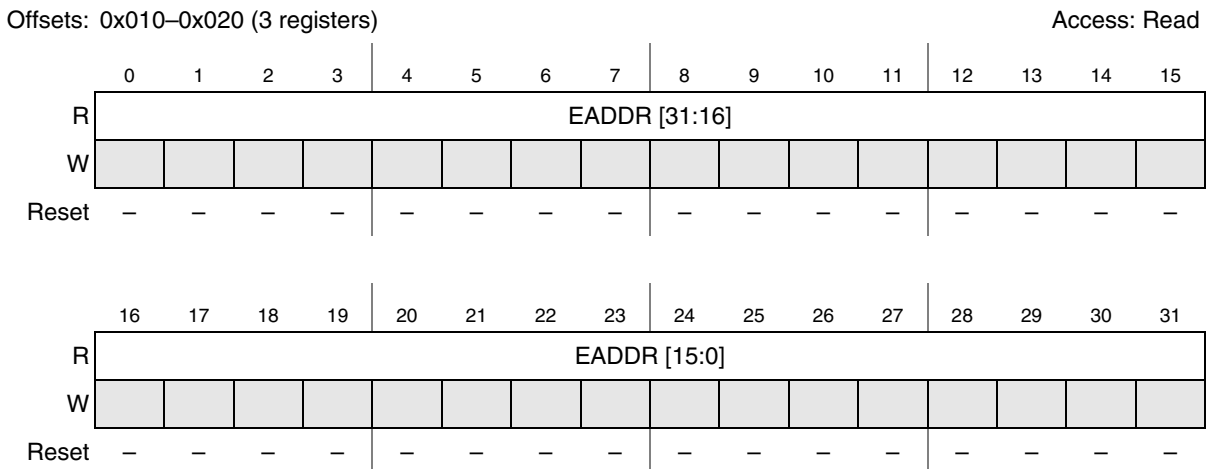
| Field | Description   |
|-------|---|
| VLD   | Valid<br>This bit provides a global enable/disable for the MPU.<br>0 The MPU is disabled.<br>1 The MPU is enabled.<br>While the MPU is disabled, all accesses from all bus masters are allowed. |

**Table 21-3. SPERR implementation**

| SPERR bit | Corresponding port                 |
|-----------|------------------------------------|
| SPERR[0]  | Flash memory controller slave port |
| SPERR[1]  | System RAM controller slave port   |
| SPERR[2]  | IPS peripheral bus slave port      |

### 21.5.2.2 MPU Error Address Register, Slave Port *n* (MPU\_EAR<sub>*n*</sub>)

When the MPU detects an access error on slave port *n*, the 32-bit reference address is captured in this read-only register and the corresponding bit in the MPU\_CESR[SPERR] field set. Additional information about the faulting access is captured in the corresponding MPU\_EDR<sub>*n*</sub> register at the same time. Note this register and the corresponding MPU\_EDR<sub>*n*</sub> register contain the most recent access error; there are no hardware interlocks with the MPU\_CESR[SPERR] field as the error registers are always loaded upon the occurrence of each protection violation.



**Figure 21-3. MPU Error Address Register, Slave Port *n* (MPU\_EAR<sub>*n*</sub>)**

**Table 21-4. MPU\_EAR<sub>*n*</sub> field descriptions**

| Field | Description  |
|-------|--|
| EADDR | Error Address<br>This field is the reference address from slave port <i>n</i> that generated the access error. |

### 21.5.2.3 MPU Error Detail Register, Slave Port $n$ (MPU\_EDR $n$ )

When the MPU detects an access error on slave port  $n$ , 32 bits of error detail are captured in this read-only register and the corresponding bit in the MPU\_CESR[SPERR] field set. Information on the faulting address is captured in the corresponding MPU\_EAR $n$  register at the same time. Note that this register and the corresponding MPU\_EAR $n$  register contain the most recent access error; there are no hardware interlocks with the MPU\_CESR[SPERR] field as the error registers are always loaded upon the occurrence of each protection violation.

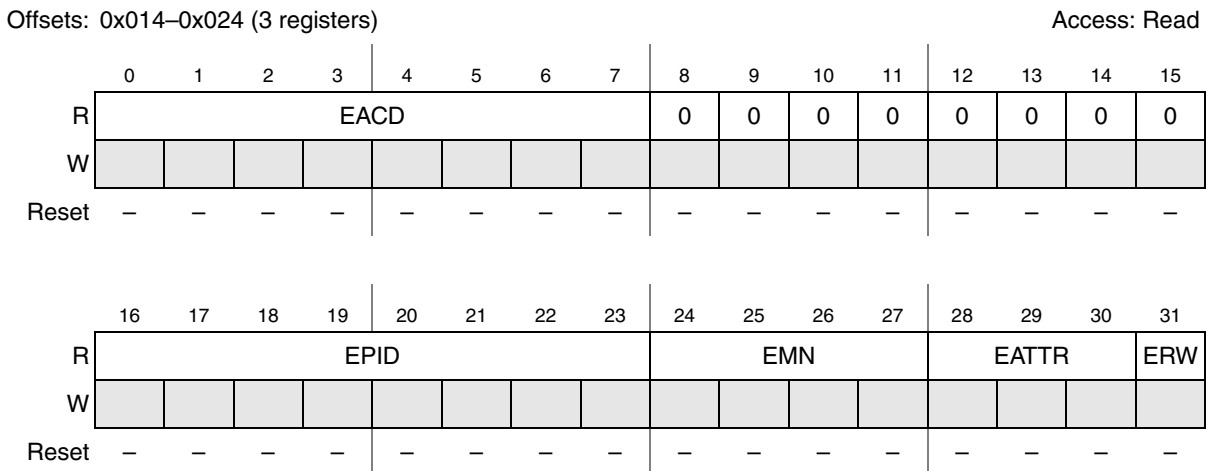


Figure 21-4. MPU Error Detail Register, Slave Port  $n$  (MPU\_EDR $n$ )

Table 21-5. MPU\_EDR $n$  field descriptions

| Field | Description  |
|-------|--|
| EACD  | <p>Error Access Control Detail</p> <p>This field implements one bit per region descriptor and is an indication of the region descriptor hit logically ANDed with the access error indication. The MPU performs a reference-by-reference evaluation to determine the presence/absence of an access error. When an error is detected, the hit-qualified access control vector is captured in this field.</p> <p>If the MPU_EDR<math>n</math> register contains a captured error and the EACD field is all zeroes, this signals an access that did not hit in any region descriptor. All non-zero EACD values signal references that hit in a region descriptor(s), but failed due to a protection error as defined by the specific set bits. If only a single EACD bit is set, then the protection error was caused by a single non-overlapping region descriptor. If two or more EACD bits are set, then the protection error was caused in an overlapping set of region descriptors.</p> |
| EPID  | <p>Error Process Identification</p> <p>This field records the process identifier of the faulting reference. The process identifier is typically driven only by processor cores; for other bus masters, this field is cleared.</p>  |
| EMN   | <p>Error Master Number</p> <p>This field records the logical master number of the faulting reference. This field is used to determine the bus master that generated the access error.</p>  |

**Table 21-5. MPU\_EDRn field descriptions (continued)**

| Field | Description  |
|-------|--|
| EATTR | Error Attributes<br>This field records attribute information about the faulting reference. The supported encodings are defined as:<br>000 User mode, instruction access<br>001 User mode, data access<br>010 Supervisor mode, instruction access<br>011 Supervisor mode, data access<br>All other encodings are reserved. For non-core bus masters, the access attribute information is typically wired to supervisor, data (0b011). |
| ERW   | Error Read/Write<br>This field signals the access type (read, write) of the faulting reference.<br>0 Read<br>1 Write   |

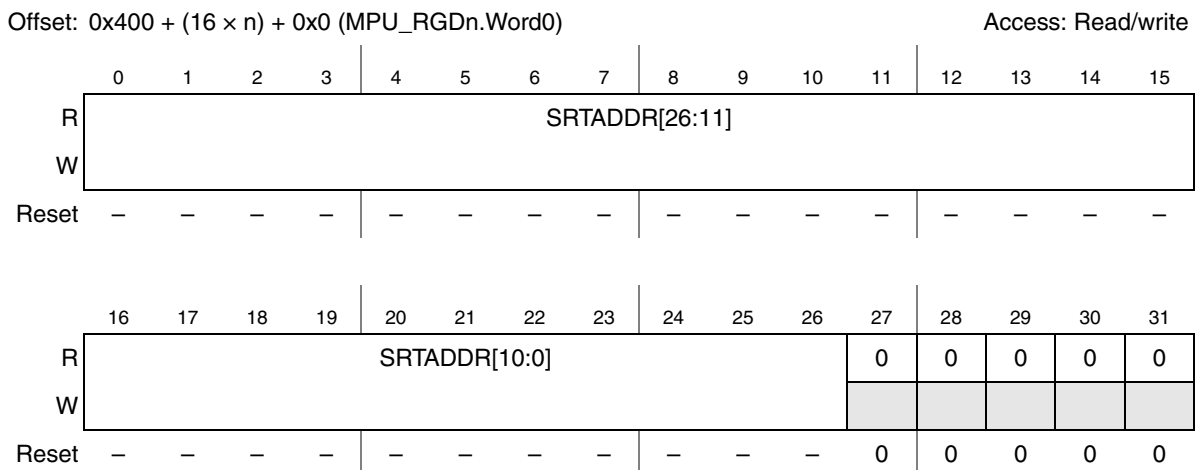
### 21.5.2.4 MPU Region Descriptor *n* (MPU\_RGD*n*)

Each 128-bit (16-byte) region descriptor specifies a given memory space and the access attributes associated with that space. The descriptor definition is the very essence of the operation of the Memory Protection Unit.

The region descriptors are organized sequentially in the MPU’s programming model and each of the four 32-bit words are detailed in the subsequent sections.

#### 21.5.2.4.1 MPU Region Descriptor $\bar{n}$ , Word 0 (MPU\_RGD $\bar{n}$ .Word0)

The first word of the MPU region descriptor defines the 0-modulo-32 byte start address of the memory region. Writes to this word clear the region descriptor’s valid bit (see [Section 21.5.2.4.4, MPU Region Descriptor \*n\*, Word 3 \(MPU\\_RGD\*n\*.Word3\)](#) for more information).



**Figure 21-5. MPU Region Descriptor, Word 0 Register (MPU\_RGD $\bar{n}$ .Word0)**

Table 21-6. MPU\_RGDn.Word0 field descriptions

| Field   | Description   |
|---------|---|
| SRTADDR | Start Address<br>This field defines the most significant bits of the 0-modulo-32 byte start address of the memory region. |

### 21.5.2.4.2 MPU Region Descriptor *n*, Word 1 (MPU\_RGDn.Word1)

The second word of the MPU region descriptor defines the 31-modulo-32 byte end address of the memory region. Writes to this word clear the region descriptor's valid bit (see [Section 21.5.2.4.4, MPU Region Descriptor \*n\*, Word 3 \(MPU\\_RGDn.Word3\)](#) for more information).

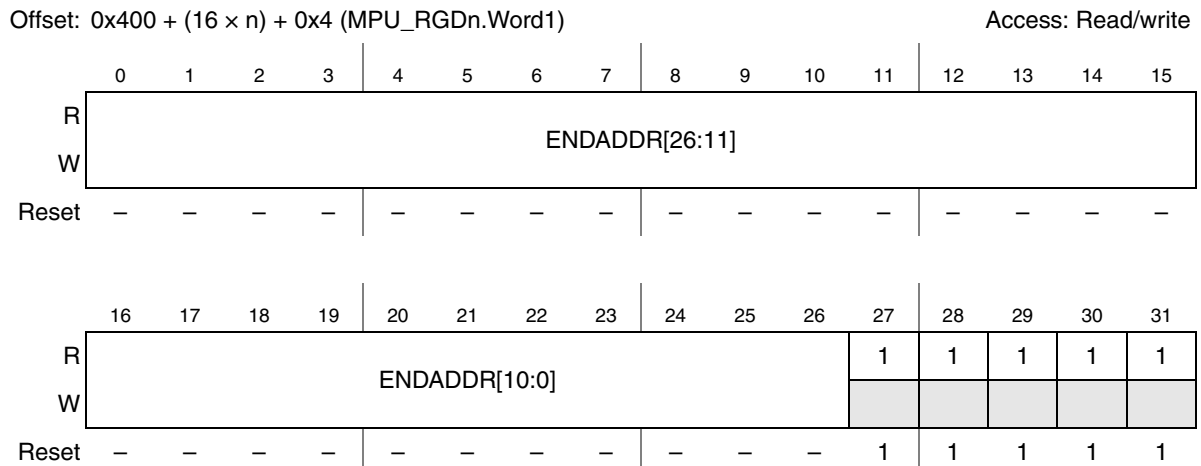


Figure 21-6. MPU Region Descriptor, Word 1 Register (MPU\_RGDn.Word1)

Table 21-7. MPU\_RGDn.Word1 field descriptions

| Field   | Description  |
|---------|--|
| ENDADDR | End Address<br>This field defines the most significant bits of the 31-modulo-32 byte end address of the memory region. There are no hardware checks to verify that $ENDADDR \geq SRTADDR$ ; it is software's responsibility to properly load these region descriptor fields. |

### 21.5.2.4.3 MPU Region Descriptor *n*, Word 2 (MPU\_RGDn.Word2)

The third word of the MPU region descriptor defines the access control rights of the memory region. The access control privileges are dependent on two broad classifications of bus masters. Bus masters 0–3 are typically reserved for processor cores and the corresponding access control is a 6-bit field defining separate privilege rights for user and supervisor mode accesses as well as the optional inclusion of a process identification field within the definition. Bus masters 4–7 are typically reserved for data movement engines, and their capabilities are limited to separate read and write permissions. For these fields, the bus master number refers to the logical master number defined as the XBAR hmaster[3:0] signal.

For the processor privilege rights, there are three flags associated with this function: {read, write, execute}. In this context, these flags follow the traditional definition:

- Read (r) permission refers to the ability to access the referenced memory address using an operand (data) fetch.
- Write (w) permission refers to the ability to update the referenced memory address using a store (data) instruction.
- Execute (x) permission refers to the ability to read the referenced memory address using an instruction fetch.

The evaluation logic defines the processor access type based on multiple AHB signals, as hwrite and hprot[1:0].

For non-processor data movement engines (bus masters 4–7), the evaluation logic simply uses hwrite to determine if the access is a read or write.

Writes to this word clear the region descriptor’s valid bit (see Section 21.5.2.4.4, MPU Region Descriptor n, Word 3 (MPU\_RGDn.Word3) for more information). Since it is also expected that system software may adjust only the access controls within a region descriptor (MPU\_RGDn.Word2) as different tasks execute, an alternate programming view of this 32-bit entity is provided. If only the access controls are being updated, this operation should be performed by writing to MPU\_RGDAACn (Alternate Access Control n) as stores to these locations do *not* affect the descriptor’s valid bit.

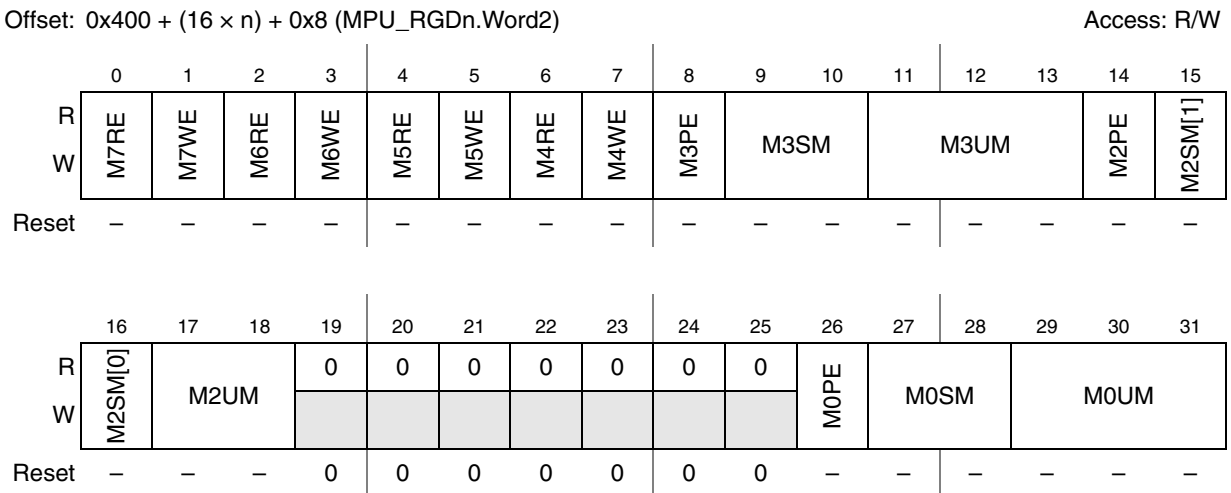


Figure 21-7. MPU Region Descriptor, Word 2 Register (MPU\_RGDn.Word2)

Table 21-8. MPU\_RGDn.Word2 field descriptions

| Field | Description   |
|-------|---|
| M7RE  | Bus master 7 read enable<br>If set, this flag allows bus master 7 to perform read operations. If cleared, any attempted read by bus master 7 terminates with an access error and the read is not performed.     |
| M7WE  | Bus master 7 write enable<br>If set, this flag allows bus master 7 to perform write operations. If cleared, any attempted write by bus master 7 terminates with an access error and the write is not performed. |
| M6RE  | Bus master 6 read enable<br>If set, this flag allows bus master 6 to perform read operations. If cleared, any attempted read by bus master 6 terminates with an access error and the read is not performed.     |



Table 21-8. MPU\_RGDn.Word2 field descriptions (continued)

| Field | Description   |
|-------|---|
| M6WE  | Bus master 6 write enable<br>If set, this flag allows bus master 6 to perform write operations. If cleared, any attempted write by bus master 6 terminates with an access error and the write is not performed.   |
| M5RE  | Bus master 5 read enable<br>If set, this flag allows bus master 5 to perform read operations. If cleared, any attempted read by bus master 5 terminates with an access error and the read is not performed.   |
| M5WE  | Bus master 5 write enable<br>If set, this flag allows bus master 5 to perform write operations. If cleared, any attempted write by bus master 5 terminates with an access error and the write is not performed.   |
| M4RE  | Bus master 4 read enable<br>If set, this flag allows bus master 4 to perform read operations. If cleared, any attempted read by bus master 4 terminates with an access error and the read is not performed.   |
| M4WE  | Bus master 4 write enable<br>If set, this flag allows bus master 4 to perform write operations. If cleared, any attempted write by bus master 4 terminates with an access error and the write is not performed.   |
| M3PE  | Bus master 3 process identifier enable<br>If set, this flag specifies that the process identifier and mask (defined in MPU_RGDn.Word3) are to be included in the region hit evaluation. If cleared, then the region hit evaluation does not include the process identifier.   |
| M3SM  | Bus master 3 supervisor mode access control<br>This field defines the access controls for bus master 3 when operating in supervisor mode. The M3SM field is defined as:<br>00 r, w, x = read, write and execute allowed<br>01 r, -, x = read and execute allowed, but no write<br>10 r, w, - = read and write allowed, but no execute<br>11 Same access controls as that defined by M3UM for user mode  |
| M3UM  | Bus master 3 user mode access control<br>This field defines the access controls for bus master 3 when operating in user mode. The M3UM field consists of three independent bits, enabling read, write and execute permissions: {r,w,x}. If set, the bit allows the given access type to occur; if cleared, an attempted access of that mode may be terminated with an access error (if not allowed by any other descriptor) and the access not performed. |
| M2PE  | Bus master 2 process identifier enable<br>If set, this flag specifies that the process identifier and mask (defined in MPU_RGDn.Word3) are to be included in the region hit evaluation. If cleared, then the region hit evaluation does not include the process identifier.   |
| M2SM  | Bus master 2 supervisor mode access control<br>This field defines the access controls for bus master 2 when operating in supervisor mode. The M2SM field is defined as:<br>00 r, w, = read and write allowed<br>01 r = read allowed, but no write<br>10 r, w = read and write allowed<br>11 Same access controls as that defined by M2UM for user mode  |
| M2UM  | Bus master 2 user mode access control<br>This field defines the access controls for bus master 2 when operating in user mode. The M2UM field consists of two independent bits, enabling read and write permissions: {r,w}. If set, the bit allows the given access type to occur; if cleared, an attempted access of that mode may be terminated with an access error (if not allowed by any other descriptor) and the access not performed.              |

Table 21-8. MPU\_RGDn.Word2 field descriptions (continued)

| Field | Description   |
|-------|---|
| M0PE  | Bus master 0 process identifier enable<br>If set, this flag specifies that the process identifier and mask (defined in MPU_RGDn.Word3) are to be included in the region hit evaluation. If cleared, then the region hit evaluation does not include the process identifier.   |
| M0SM  | Bus master 0 supervisor mode access control<br>This field defines the access controls for bus master 0 when operating in supervisor mode. The M0SM field is defined as:<br>00 r, w, x = read, write and execute allowed<br>01 r, -, x = read and execute allowed, but no write<br>10 r, w, - = read and write allowed, but no execute<br>11 Same access controls as that defined by M0UM for user mode  |
| M0UM  | Bus master 0 user mode access control<br>This field defines the access controls for bus master 0 when operating in user mode. The M0UM field consists of three independent bits, enabling read, write and execute permissions: {r,w,x}. If set, the bit allows the given access type to occur; if cleared, an attempted access of that mode may be terminated with an access error (if not allowed by any other descriptor) and the access not performed. |

#### 21.5.2.4.4 MPU Region Descriptor *n*, Word 3 (MPU\_RGDn.Word3)

The fourth word of the MPU region descriptor contains the optional process identifier and mask, plus the region descriptor's valid bit.

Since the region descriptor is a 128-bit entity, there are potential coherency issues as this structure is being updated since multiple writes are required to update the entire descriptor. Accordingly, the MPU hardware assists in the operation of the descriptor valid bit to prevent incoherent region descriptors from generating spurious access errors. In particular, it is expected that a complete update of a region descriptor is typically done with sequential writes to MPU\_RGDn.Word0, then MPU\_RGDn.Word1,... and finally MPU\_RGDn.Word3. The MPU hardware automatically clears the valid bit on any writes to words {0,1,2} of the descriptor. Writes to this word set/clear the valid bit in a normal manner.

Since it is also expected that system software may adjust only the access controls within a region descriptor (MPU\_RGDn.Word2) as different tasks execute, an alternate programming view of this 32-bit entity is provided. If only the access controls are being updated, this operation should be performed by writing to MPU\_RGDAACn (Alternate Access Control n) as stores to these locations do *not* affect the descriptor's valid bit.

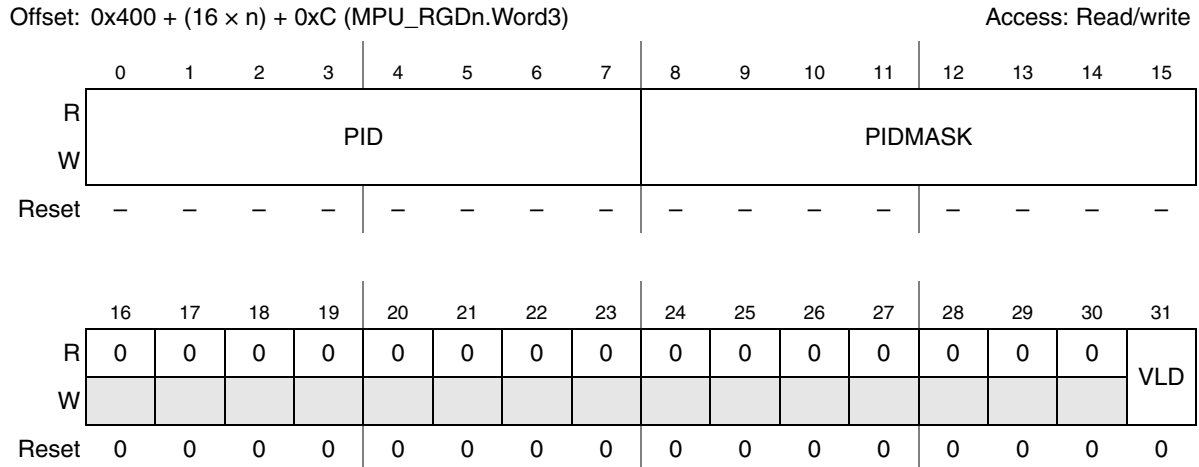


Figure 21-8. MPU Region Descriptor, Word 3 Register (MPU\_RGDn.Word3)

Table 21-9. MPU\_RGDn.Word3 field descriptions

| Field   | Description   |
|---------|---|
| PID     | Process Identifier<br>This field specifies that the optional process identifier is to be included in the determination of whether the current access hits in the region descriptor. This field is combined with the PIDMASK and included in the region hit determination if MPU_RGDn.Word2[MxPE] is set.  |
| PIDMASK | Process Identifier Mask<br>This field provides a masking capability so that multiple process identifiers can be included as part of the region hit determination. If a bit in the PIDMASK is set, then the corresponding bit of the PID is ignored in the comparison. This field is combined with the PID and included in the region hit determination if MPU_RGDn.Word2[MxPE] is set. For more information on the handling of the PID and PIDMASK, see <a href="#">Section 21.6.1.1, Access evaluation – Hit determination</a> . |
| VLD     | Valid<br>This bit signals the region descriptor is valid. Any write to MPU_RGDn.Word{0,1,2} clears this bit, while a write to MPU_RGDn.Word3 sets or clears this bit depending on bit 31 of the write operand.<br>0 Region descriptor is invalid<br>1 Region descriptor is valid  |

### 21.5.2.5 MPU Region Descriptor Alternate Access Control $n$ (MPU\_RGDAAC $n$ )

As noted in [Section 21.5.2.4.3, MPU Region Descriptor  \$n\$ , Word 2 \(MPU\\_RGDn.Word2\)](#), it is expected that since system software may adjust only the access controls within a region descriptor (MPU\_RGDn.Word2) as different tasks execute, an alternate programming view of this 32-bit entity is desired. If only the access controls are being updated, this operation should be performed by writing to MPU\_RGDAAC $n$  (Alternate Access Control  $n$ ) as stores to these locations do not affect the descriptor's valid bit.

The memory address therefore provides an alternate location for updating MPU\_RGDn.Word2.

Offset: 0x800 + (4 × n) (MPU\_RGDAACn)

Access: Read/write

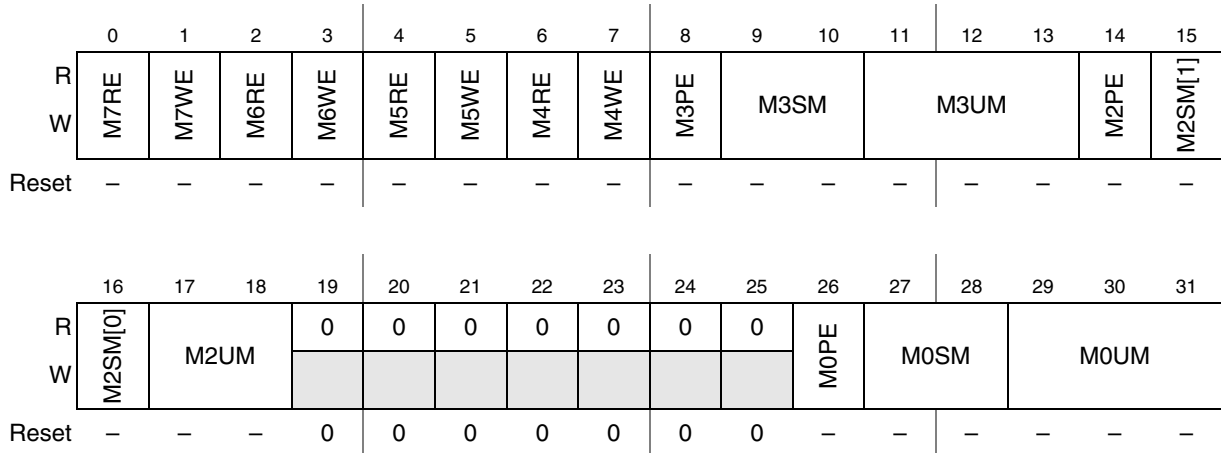


Figure 21-9. MPU RGD Alternate Access Control n (MPU\_RGDAACn)

Since the MPU\_RGDAACn register is simply another memory mapping for MPU\_RGDn.Word2, the field definitions shown in Table 21-10 are identical to those presented in Table 21-8.

Table 21-10. MPU\_RGDAACn field descriptions

| Field | Description   |
|-------|---|
| M7RE  | Bus master 7 read enable.<br>If set, this flag allows bus master 7 to perform read operations. If cleared, any attempted read by bus master 7 terminates with an access error and the read is not performed.    |
| M7WE  | Bus master 7 write enable<br>If set, this flag allows bus master 7 to perform write operations. If cleared, any attempted write by bus master 7 terminates with an access error and the write is not performed. |
| M6RE  | Bus master 6 read enable<br>If set, this flag allows bus master 6 to perform read operations. If cleared, any attempted read by bus master 6 terminates with an access error and the read is not performed.     |
| M6WE  | Bus master 6 write enable<br>If set, this flag allows bus master 6 to perform write operations. If cleared, any attempted write by bus master 6 terminates with an access error and the write is not performed. |
| M5RE  | Bus master 5 read enable<br>If set, this flag allows bus master 5 to perform read operations. If cleared, any attempted read by bus master 5 terminates with an access error and the read is not performed.     |
| M5WE  | Bus master 5 write enable<br>If set, this flag allows bus master 5 to perform write operations. If cleared, any attempted write by bus master 5 terminates with an access error and the write is not performed. |
| M4RE  | Bus master 4 read enable<br>If set, this flag allows bus master 4 to perform read operations. If cleared, any attempted read by bus master 4 terminates with an access error and the read is not performed.     |
| M4WE  | Bus master 4 write enable<br>If set, this flag allows bus master 4 to perform write operations. If cleared, any attempted write by bus master 4 terminates with an access error and the write is not performed. |

Table 21-10. MPU\_RGDAACn field descriptions (continued)

| Field | Description   |
|-------|---|
| M3PE  | Bus master 3 process identifier enable<br>If set, this flag specifies that the process identifier and mask (defined in MPU_RGDN.Word3) are to be included in the region hit evaluation. If cleared, then the region hit evaluation does not include the process identifier.   |
| M3SM  | Bus master 3 supervisor mode access control<br>This field defines the access controls for bus master 3 when operating in supervisor mode. The M3SM field is defined as:<br>00 r, w, x = read, write and execute allowed<br>01 r, -, x = read and execute allowed, but no write<br>10 r, w, - = read and write allowed, but no execute<br>11 Same access controls as that defined by M3UM for user mode  |
| M3UM  | Bus master 3 user mode access control<br>This field defines the access controls for bus master 3 when operating in user mode. The M3UM field consists of three independent bits, enabling read, write and execute permissions: {r,w,x}. If set, the bit allows the given access type to occur; if cleared, an attempted access of that mode may be terminated with an access error (if not allowed by any other descriptor) and the access not performed. |
| M2PE  | Bus master 2 process identifier enable<br>If set, this flag specifies that the process identifier and mask (defined in MPU_RGDN.Word3) are to be included in the region hit evaluation. If cleared, then the region hit evaluation does not include the process identifier.   |
| M2SM  | Bus master 2 supervisor mode access control<br>This field defines the access controls for bus master 2 when operating in supervisor mode. The M2SM field is defined as:<br>00 r, w, = read and write allowed<br>01 r = read allowed, but no write<br>10 r, w = read and write allowed<br>11 Same access controls as that defined by M2UM for user mode  |
| M2UM  | Bus master 2 user mode access control<br>This field defines the access controls for bus master 2 when operating in user mode. The M2UM field consists of three independent bits, enabling read, write and execute permissions: {r,w,x}. If set, the bit allows the given access type to occur; if cleared, an attempted access of that mode may be terminated with an access error (if not allowed by any other descriptor) and the access not performed. |
| M0PE  | Bus master 0 process identifier enable<br>If set, this flag specifies that the process identifier and mask (defined in MPU_RGDN.Word3) are to be included in the region hit evaluation. If cleared, then the region hit evaluation does not include the process identifier.   |
| M0SM  | Bus master 0 supervisor mode access control<br>This field defines the access controls for bus master 0 when operating in supervisor mode. The M0SM field is defined as:<br>00 r, w, x = read, write and execute allowed<br>01 r, -, x = read and execute allowed, but no write<br>10 r, w, - = read and write allowed, but no execute<br>11 Same access controls as that defined by M0UM for user mode  |
| M0UM  | Bus master 0 user mode access control<br>This field defines the access controls for bus master 0 when operating in user mode. The M0UM field consists of three independent bits, enabling read, write and execute permissions: {r,w,x}. If set, the bit allows the given access type to occur; if cleared, an attempted access of that mode may be terminated with an access error (if not allowed by any other descriptor) and the access not performed. |

## 21.6 Functional description

In this section, the functional operation of the MPU is detailed. In particular, subsequent sections discuss the operation of the access evaluation macro as well as the handling of error-terminated bus cycles.

### 21.6.1 Access evaluation macro

As previously discussed, the basic operation of the MPU is performed in the access evaluation macro, a hardware structure replicated in the two-dimensional connection matrix. As shown in Figure 21-10, the access evaluation macro inputs the system bus address phase signals and the contents of a region descriptor (RGDn) and performs two major functions: region hit determination (hit\_b) and detection of an access protection violation (error).

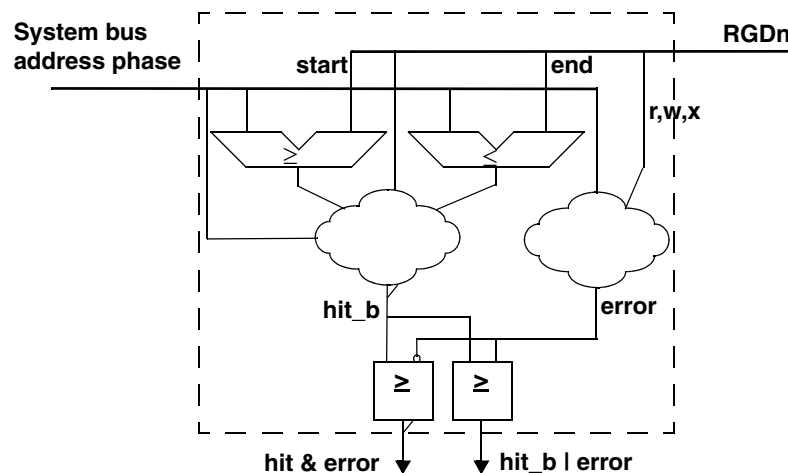


Figure 21-10. MPU access evaluation macro

Figure 21-10 is not intended to be a schematic of the actual access evaluation macro, but rather a generalized block diagram showing the major functions included in this logic block.

#### 21.6.1.1 Access evaluation – Hit determination

To evaluate the region hit determination, the MPU uses two magnitude comparators in conjunction with the contents of a region descriptor: the current access must be included between the region's start and end addresses and simultaneously the region's valid bit must be active.

Recall there are no hardware checks to verify that region's end address is greater than region's start address, and it is software's responsibility to properly load appropriate values into these fields of the region descriptor.

In addition to this, the optional process identifier is examined against the region descriptor's PID and PIDMASK fields. In order to generate the pid\_hit indication: the current PID with its PIDMASK must be equal to the region's PID with its PIDMASK. Also the process identifier enable is taken into account in this comparison so that the MPU forces the pid\_hit term to be asserted in the case of AHB bus master doesn't provide its process identifier.

### 21.6.1.2 Access evaluation – Privilege violation determination

While the access evaluation macro is making the region hit determination, the logic is also evaluating if the current access is allowed by the permissions defined in the region descriptor. The protection violation logic then evaluates the access against the effective permissions using the specification shown in [Table 21-11](#).

**Table 21-11. Protection violation definition**

| Description     | Inputs     |            |            | Output                |
|-----------------|------------|------------|------------|-----------------------|
|                 | eff_rgd[r] | eff_rgd[w] | eff_rgd[x] | Protection violation? |
| inst fetch read | —          | —          | 0          | yes, no x permission  |
| inst fetch read | —          | —          | 1          | no, access is allowed |
| data read       | 0          | —          | —          | yes, no r permission  |
| data read       | 1          | —          | —          | no, access is allowed |
| data write      | —          | 0          | —          | yes, no w permission  |
| data write      | —          | 1          | —          | no, access is allowed |

As shown in [Figure 21-10](#), the output of the protection violation logic is the error signal.

The access evaluation macro then uses the hit\_b and error signals to form two outputs. The combined (hit\_b | error) signal is used to signal the current access is not allowed and (~hit\_b & error) is used as the input to MPU\_EDRn (error detail register) in the event of an error.

### 21.6.2 Putting it all together and AHB error terminations

For each XBAR slave port being monitored, the MPU performs a reduction-AND of all the individual (hit\_b | error) terms from each access evaluation macro. This expression then terminates the bus cycle with an error and reports a protection error for three conditions:

- If the access does not hit in any region descriptor, a protection error is reported.
- If the access hits in a single region descriptor and that region signals a protection violation, then a protection error is reported.
- If the access hits in multiple (overlapping) regions and all regions signal protection violations, then a protection error is reported.

The third condition reflects that priority is given to permission granting over access denying for overlapping regions as this approach provides more flexibility to system software in region descriptor assignments. For an example of the use of overlapping region descriptors, see [Section 21.8, Application information](#).

In event of a protection error, the MPU requires two distinct actions:

- Intercepting the error during the address phase (first cycle out of two) and cancelling the transaction before it is seen by the slave device

- Performing the required logic functions to force the standard 2-cycle AHB error response to properly terminate the bus transaction and then providing the right values to the crossbar switch to commit the transaction to other portions of the platform.

If, instead, the access is allowed, then the MPU simply passes all original signals to the slave device. In this case, from a functionality point of view, the MPU is fully transparent.

## 21.7 Initialization information

The reset state of MPU\_CESR[VLD] disables the entire module. Recall that while the MPU is disabled, all accesses from all bus masters are allowed. This state also minimizes the power dissipation of the MPU. The power dissipation of each access evaluation macro is minimized when the associated region descriptor is marked as invalid or when MPU\_CESR[VLD] = 0.

Typically the appropriate number of region descriptors (MPU\_RGD $n$ ) is loaded at system startup, including the setting of the MPU\_RGD $n$ .Word3[VLD] bits, before MPU\_CESR[VLD] is set, enabling the module. This approach allows all the loaded region descriptors to be enabled simultaneously. Recall if a memory reference does not hit in any region descriptor, the attempted access is terminated with an error.

## 21.8 Application information

In an operational system, interfacing with the MPU can generally be classified into the following activities:

- Creation of a new memory region requires loading the appropriate region descriptor into an available register location. When a new descriptor is loaded into a RGD $n$ , it would typically be performed using four 32-bit word writes. As discussed in [Section 21.5.2.4.4, MPU Region Descriptor  \$n\$ , Word 3 \(MPU\\_RGD \$n\$ .Word3\)](#), the hardware assists in the maintenance of the valid bit, so if this approach is followed, there are no coherency issues associated with the multi-cycle descriptor writes. Deletion/removal of an existing memory region is performed simply by clearing MPU\_RGD $n$ .Word3[VLD].
- If only the access rights for an existing region descriptor need to change, a 32-bit write to the alternate version of the access control word (MPU\_RGDAAC $n$ ) would typically be performed. Recall writes to the region descriptor using this alternate access control location do not affect the valid bit, so there are, by definition, no coherency issues involved with the update. The access rights associated with the memory region switch instantaneously to the new value as the IPS write completes.
- If the region's start and end addresses are to be changed, this would typically be performed by writing a minimum of three words of the region descriptor: MPU\_RGD $n$ .Word{0,1,3}, where the writes to Word0 and Word1 redefine the start and end addresses, respectively, and the write to Word3 re-enables the region descriptor valid bit. In many situations, all four words of the region descriptor would be rewritten.
- Typically, references to the MPU's programming model would be restricted to supervisor mode accesses from a specific processor(s), so a region descriptor would be specifically allocated for this purpose with attempted accesses from other masters or while in user mode terminated with an error.

When the MPU detects an access error, the current bus cycle is terminated with an error response, and information on the faulting reference is captured in the MPU\_EAR $n$  and MPU\_EDR $n$  registers. The




error-terminated bus cycle typically initiates some type of error response in the originating bus master. For example, the CPU errors generate a core exception, whereas the DMA errors generate a MPU (external) interrupt. It is important to highlight that in case of DMA access violations, the core continues to run, but if a core violation occurs, the system stops. In any event, the processor can retrieve the captured error address and detail information by reading the MPU\_E{A,D}Rn registers. Information on that error registers contain captured fault data is signaled by MPU\_CESR[SPERR].

This page is intentionally left blank.

---

## —— Communication modules ——



This page is intentionally left blank.

# Chapter 22

## Inter-Integrated Circuit Bus Controller Module (I<sup>2</sup>C)

### 22.1 Introduction

#### 22.1.1 Overview

The Inter-Integrated Circuit (I<sup>2</sup>C or IIC) bus is a two wire bidirectional serial bus that provides a simple and efficient method of data exchange between devices. It minimizes the number of external connections to devices and does not require an external address decoder.

This bus is suitable for applications requiring occasional communications over a short distance between a number of devices. It also provides flexibility, allowing additional devices to be connected to the bus for further expansion and system development.

The interface is designed to operate as fast as 100 kbit/s in Standard Mode and 400 kbit/s in Fast Mode. The device is capable of operating at higher baud rates, up to a maximum of module clock/20 with reduced bus loading. Actual baud rate can be less than the programmed baud rate and is dependent on the SCL rise time. SCL rise time is dependent on the external pullup resistor value and bus loading. The maximum communication length and the number of devices that can be connected are limited by a maximum bus capacitance of 400 pF.

#### 22.1.2 Features

The I<sup>2</sup>C module has the following key features:

- Compatible with I<sup>2</sup>C Bus standard
- Multi-master operation
- Software programmable for one of 256 different serial clock frequencies
- Software selectable acknowledge bit
- Interrupt driven byte-by-byte data transfer
- Arbitration lost interrupt with automatic mode switching from master to slave
- Calling address identification interrupt
- Start and stop signal generation/detection
- Repeated start signal generation
- Acknowledge bit generation/detection
- Bus busy detection
- Direct memory access

Features currently not supported:

- No support for general call address
- Not compliant to 10-bit addressing

### 22.1.3 Block diagram

The block diagram of the I<sup>2</sup>C module is shown in [Figure 22-1](#).

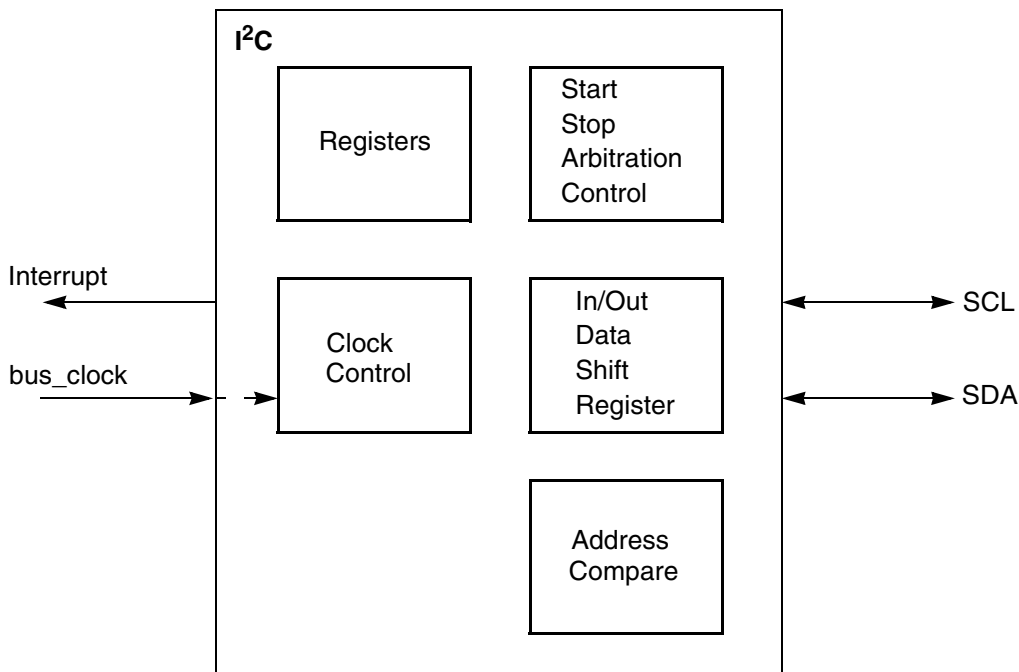


Figure 22-1. I<sup>2</sup>C block diagram

## 22.2 External signal description

The Inter-Integrated Circuit (I<sup>2</sup>C) module has two external pins, SCL and SDA.

### 22.2.1 SCL

This is the bidirectional Serial Clock Line (SCL) of the module, compatible with the I<sup>2</sup>C-Bus specification.

### 22.2.2 SDA

This is the bidirectional Serial Data line (SDA) of the module, compatible with the I<sup>2</sup>C-Bus specification.

## 22.3 Memory map and register description

### 22.3.1 Module memory map

The memory map for the I<sup>2</sup>C module is given below in [Table 22-1](#). The total address for each register is the sum of the base address for the I<sup>2</sup>C module and the address offset for each register.

Table 22-1. I<sup>2</sup>C memory map

| Base address: 0xFFE3_0000 |  |                             |
|---------------------------|--|-----------------------------|
| Address offset            | Register   | Location                    |
| 0x0                       | I <sup>2</sup> C Bus Address Register (IBAD)           | <a href="#">on page 399</a> |
| 0x1                       | I <sup>2</sup> C Bus Frequency Divider Register (IBFD) | <a href="#">on page 400</a> |
| 0x2                       | I <sup>2</sup> C Bus Control Register (IBCR)           | <a href="#">on page 406</a> |
| 0x3                       | I <sup>2</sup> C Bus Status Register (IBSR)            | <a href="#">on page 407</a> |
| 0x4                       | I <sup>2</sup> C Bus Data I/O Register (IBDR)          | <a href="#">on page 408</a> |
| 0x5                       | I <sup>2</sup> C Bus Interrupt Config Register (IBIC)  | <a href="#">on page 409</a> |

All registers are accessible via 8-, 16-, or 32-bit accesses. However, 16-bit accesses must be aligned to 16-bit boundaries, and 32-bit accesses must be aligned to 32-bit boundaries. As an example, the IBDF register for the frequency divider is accessible by a 16-bit read/write to address Base + 0x000, but performing a 16-bit access to Base + 0x001 is illegal.

### 22.3.2 I<sup>2</sup>C Bus Address Register (IBAD)

This register contains the address the I<sup>2</sup>C bus will respond to when addressed as a slave; note that it is not the address sent on the bus during the address transfer.

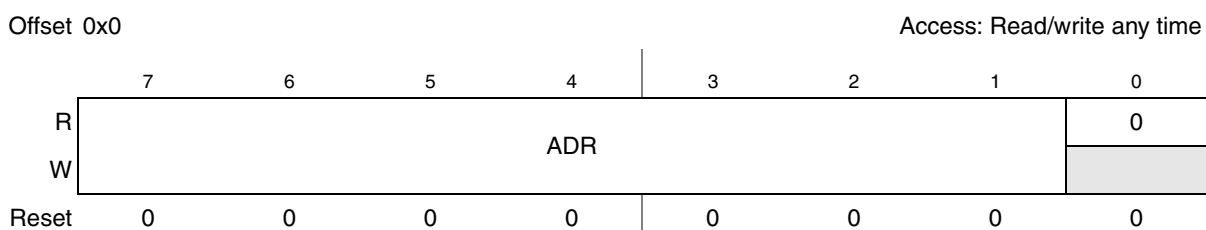
Figure 22-2. I<sup>2</sup>C Bus Address Register (IBAD)

Table 22-2. IBAD field descriptions

| Field | Description  |
|-------|--|
| ADR   | Slave Address. Specific slave address to be used by the I <sup>2</sup> C Bus module.<br><b>Note:</b> The default mode of I <sup>2</sup> C Bus is slave mode for an address match on the bus. |

### 22.3.3 I<sup>2</sup>C Bus Frequency Divider Register (IBFD)

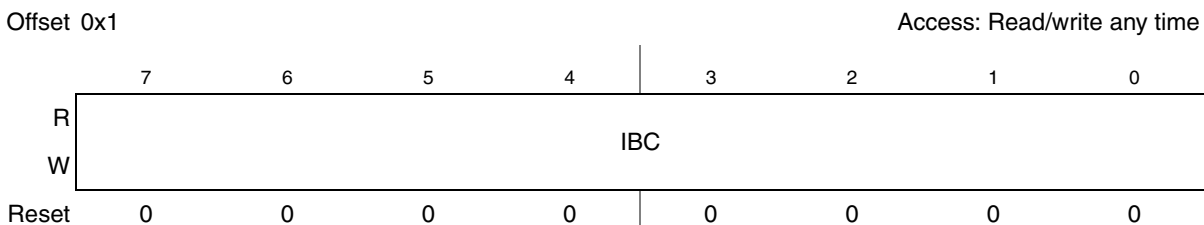


Figure 22-3. I<sup>2</sup>C Bus Frequency Divider Register (IBFD)

Table 22-3. IBFD field descriptions

| Field | Description  |
|-------|--|
| IBC   | I-Bus Clock Rate. This field is used to prescale the clock for bit rate selection. The bit clock generator is implemented as a prescale divider. The IBC bits are decoded to give the Tap and Prescale values as follows:<br>7–6 select the prescaled shift register (see Table 22-4)<br>5–3 select the prescaler divider (see Table 22-5)<br>2–0 select the shift register tap point (see Table 22-6) |

Table 22-4. I-Bus multiplier factor

| IBC7–6 | MUL      |
|--------|----------|
| 00     | 01       |
| 01     | 02       |
| 10     | 04       |
| 11     | RESERVED |

Table 22-5. I-Bus prescaler divider values

| IBC5–3 | scl2start (clocks) | scl2stop (clocks) | scl2tap (clocks) | tap2tap (clocks) |
|--------|--------------------|-------------------|------------------|------------------|
| 000    | 2                  | 7                 | 4                | 1                |
| 001    | 2                  | 7                 | 4                | 2                |
| 010    | 2                  | 9                 | 6                | 4                |
| 011    | 6                  | 9                 | 6                | 8                |
| 100    | 14                 | 17                | 14               | 16               |
| 101    | 30                 | 33                | 30               | 32               |
| 110    | 62                 | 65                | 62               | 64               |
| 111    | 126                | 129               | 126              | 128              |



Table 22-6. I-Bus tap and prescale values

| IBC2-0 | SCL Tap (clocks) | SDA Tap (clocks) |
|--------|------------------|------------------|
| 000    | 5                | 1                |
| 001    | 6                | 1                |
| 010    | 7                | 2                |
| 011    | 8                | 2                |
| 100    | 9                | 3                |
| 101    | 10               | 3                |
| 110    | 12               | 4                |
| 111    | 15               | 4                |

The number of clocks from the falling edge of SCL to the first tap (Tap[1]) is defined by the values shown in the scl2tap column of Table 22-5. All subsequent tap points are separated by  $2^{IBC5-3}$  as shown in the tap2tap column in Table 22-5. The SCL Tap is used to generate the SCL period and the SDA Tap is used to determine the delay from the falling edge of SCL to the change of state of SDA i.e. the SDA hold time.

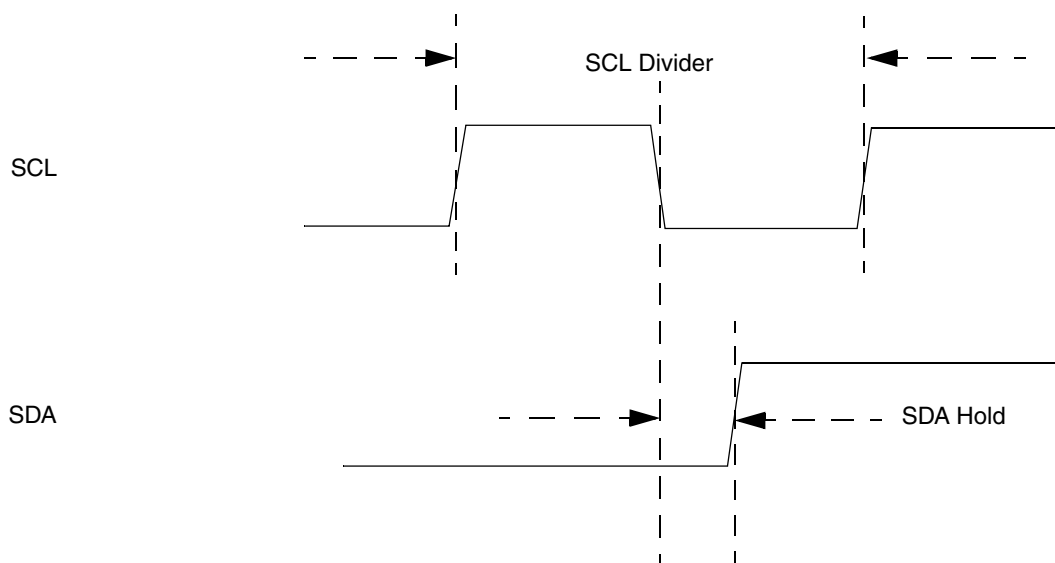


Figure 22-4. SDA hold time

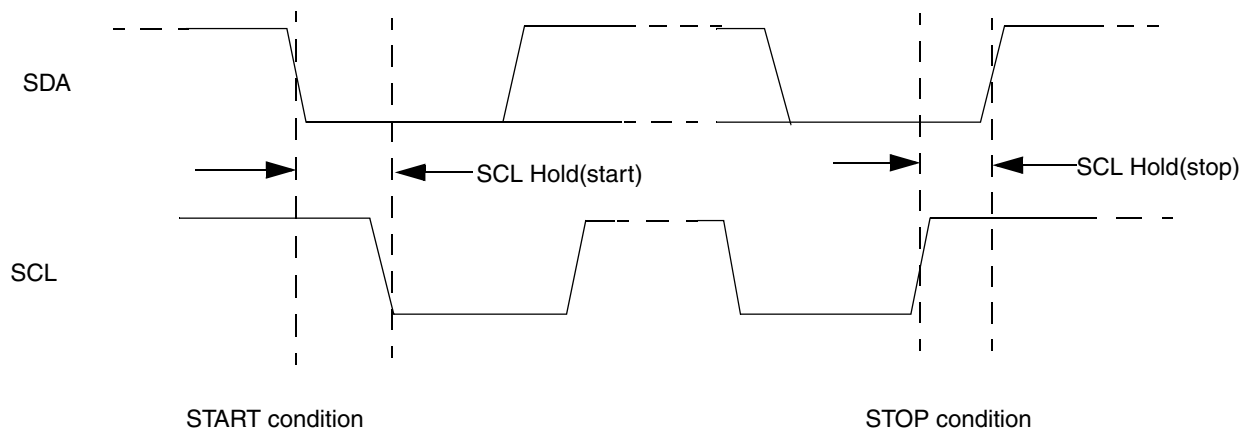


Figure 22-5. SCL divider and SDA hold

The equation used to generate the divider values from the IBFD bits is:

$$\text{SCL Divider} = \text{MUL} \times \{2 \times (\text{scl2tap} + [(\text{SCL\_Tap} - 1) \times \text{tap2tap}] + 2)\} \quad \text{Eqn. 22-1}$$

The SDA hold delay is equal to the CPU clock period multiplied by the SDA Hold value shown in Table 22-7. The equation used to generate the SDA Hold value from the IBFD bits is:

$$\text{SDA Hold} = \text{MUL} \times \{\text{scl2tap} + [(\text{SDA\_Tap} - 1) \times \text{tap2tap}] + 3\} \quad \text{Eqn. 22-2}$$

The equation for SCL Hold values to generate the start and stop conditions from the IBFD bits is:

$$\text{SCL Hold(start)} = \text{MUL} \times [\text{scl2start} + (\text{SCL\_Tap} - 1) \times \text{tap2tap}] \quad \text{Eqn. 22-3}$$

$$\text{SCL Hold(stop)} = \text{MUL} \times [\text{scl2stop} + (\text{SCL\_Tap} - 1) \times \text{tap2tap}] \quad \text{Eqn. 22-4}$$

Table 22-7. I<sup>2</sup>C divider and hold values

|         | IBC7-0<br>(hex) | SCL divider<br>(clocks) | SDA hold<br>(clocks) | SCL hold<br>(start) | SCL hold<br>(stop) |
|---------|-----------------|-------------------------|----------------------|---------------------|--------------------|
| MUL = 1 | 00              | 20                      | 7                    | 6                   | 11                 |
|         | 01              | 22                      | 7                    | 7                   | 12                 |
|         | 02              | 24                      | 8                    | 8                   | 13                 |
|         | 03              | 26                      | 8                    | 9                   | 14                 |
|         | 04              | 28                      | 9                    | 10                  | 15                 |
|         | 05              | 30                      | 9                    | 11                  | 16                 |
|         | 06              | 34                      | 10                   | 13                  | 18                 |
|         | 07              | 40                      | 10                   | 16                  | 21                 |
|         | 08              | 28                      | 7                    | 10                  | 15                 |
|         | 09              | 32                      | 7                    | 12                  | 17                 |
|         | 0A              | 36                      | 9                    | 14                  | 19                 |
|         | 0B              | 40                      | 9                    | 16                  | 21                 |
|         | 0C              | 44                      | 11                   | 18                  | 23                 |
|         | 0D              | 48                      | 11                   | 20                  | 25                 |
|         | 0E              | 56                      | 13                   | 24                  | 29                 |
|         | 0F              | 68                      | 13                   | 30                  | 35                 |
|         | 10              | 48                      | 9                    | 18                  | 25                 |
|         | 11              | 56                      | 9                    | 22                  | 29                 |
|         | 12              | 64                      | 13                   | 26                  | 33                 |
|         | 13              | 72                      | 13                   | 30                  | 37                 |
|         | 14              | 80                      | 17                   | 34                  | 41                 |
|         | 15              | 88                      | 17                   | 38                  | 45                 |
|         | 16              | 104                     | 21                   | 46                  | 53                 |
|         | 17              | 128                     | 21                   | 58                  | 65                 |
|         | 18              | 80                      | 9                    | 38                  | 41                 |
|         | 19              | 96                      | 9                    | 46                  | 49                 |
|         | 1A              | 112                     | 17                   | 54                  | 57                 |
|         | 1B              | 128                     | 17                   | 62                  | 65                 |
|         | 1C              | 144                     | 25                   | 70                  | 73                 |
|         | 1D              | 160                     | 25                   | 78                  | 81                 |
|         | 1E              | 192                     | 33                   | 94                  | 97                 |
|         | 1F              | 240                     | 33                   | 118                 | 121                |
|         | 20              | 160                     | 17                   | 78                  | 81                 |
|         | 21              | 192                     | 17                   | 94                  | 97                 |
|         | 22              | 224                     | 33                   | 110                 | 113                |
|         | 23              | 256                     | 33                   | 126                 | 129                |
|         | 24              | 288                     | 49                   | 142                 | 145                |
|         | 25              | 320                     | 49                   | 158                 | 161                |
|         | 26              | 384                     | 65                   | 190                 | 193                |
|         | 27              | 480                     | 65                   | 238                 | 241                |
|         | 28              | 320                     | 33                   | 158                 | 161                |
|         | 29              | 384                     | 33                   | 190                 | 193                |
|         | 2A              | 448                     | 65                   | 222                 | 225                |
|         | 2B              | 512                     | 65                   | 254                 | 257                |
|         | 2C              | 576                     | 97                   | 286                 | 289                |
|         | 2D              | 640                     | 97                   | 318                 | 321                |
|         | 2E              | 768                     | 129                  | 382                 | 385                |
|         | 2F              | 960                     | 129                  | 478                 | 481                |
| 30      | 640             | 65                      | 318                  | 321                 |                    |
| 31      | 768             | 65                      | 382                  | 385                 |                    |
| 32      | 896             | 129                     | 446                  | 449                 |                    |
| 33      | 1024            | 129                     | 510                  | 513                 |                    |
| 34      | 1152            | 193                     | 574                  | 577                 |                    |
| 35      | 1280            | 193                     | 638                  | 641                 |                    |
| 36      | 1536            | 257                     | 766                  | 769                 |                    |
| 37      | 1920            | 257                     | 958                  | 961                 |                    |
| 38      | 1280            | 129                     | 638                  | 641                 |                    |
| 39      | 1536            | 129                     | 766                  | 769                 |                    |
| 3A      | 1792            | 257                     | 894                  | 897                 |                    |
| 3B      | 2048            | 257                     | 1022                 | 1025                |                    |
| 3C      | 2304            | 385                     | 1150                 | 1153                |                    |
| 3D      | 2560            | 385                     | 1278                 | 1281                |                    |
| 3E      | 3072            | 513                     | 1534                 | 1537                |                    |
| 3F      | 3840            | 513                     | 1918                 | 1921                |                    |

Table 22-7. I<sup>2</sup>C divider and hold values (continued)

|         | IBC7-0<br>(hex) | SCL divider<br>(clocks) | SDA hold<br>(clocks) | SCL hold<br>(start) | SCL hold<br>(stop) |
|---------|-----------------|-------------------------|----------------------|---------------------|--------------------|
| MUL = 2 | 40              | 40                      | 14                   | 12                  | 22                 |
|         | 41              | 44                      | 14                   | 14                  | 24                 |
|         | 42              | 48                      | 16                   | 16                  | 26                 |
|         | 43              | 52                      | 16                   | 18                  | 28                 |
|         | 44              | 56                      | 18                   | 20                  | 30                 |
|         | 45              | 60                      | 18                   | 22                  | 32                 |
|         | 46              | 68                      | 20                   | 26                  | 36                 |
|         | 47              | 80                      | 20                   | 32                  | 42                 |
|         | 48              | 56                      | 14                   | 20                  | 30                 |
|         | 49              | 64                      | 14                   | 24                  | 34                 |
|         | 4A              | 72                      | 18                   | 28                  | 38                 |
|         | 4B              | 80                      | 18                   | 32                  | 42                 |
|         | 4C              | 88                      | 22                   | 36                  | 46                 |
|         | 4D              | 96                      | 22                   | 40                  | 50                 |
|         | 4E              | 112                     | 26                   | 48                  | 58                 |
|         | 4F              | 136                     | 26                   | 60                  | 70                 |
|         | 50              | 96                      | 18                   | 36                  | 50                 |
|         | 51              | 112                     | 18                   | 44                  | 58                 |
|         | 52              | 128                     | 26                   | 52                  | 66                 |
|         | 53              | 144                     | 26                   | 60                  | 74                 |
|         | 54              | 160                     | 34                   | 68                  | 82                 |
|         | 55              | 176                     | 34                   | 76                  | 90                 |
|         | 56              | 208                     | 42                   | 92                  | 106                |
|         | 57              | 256                     | 42                   | 116                 | 130                |
|         | 58              | 160                     | 18                   | 76                  | 82                 |
|         | 59              | 192                     | 18                   | 92                  | 98                 |
|         | 5A              | 224                     | 34                   | 108                 | 114                |
|         | 5B              | 256                     | 34                   | 124                 | 130                |
|         | 5C              | 288                     | 50                   | 140                 | 146                |
|         | 5D              | 320                     | 50                   | 156                 | 162                |
|         | 5E              | 384                     | 66                   | 188                 | 194                |
|         | 5F              | 480                     | 66                   | 236                 | 242                |
|         | 60              | 320                     | 28                   | 156                 | 162                |
|         | 61              | 384                     | 28                   | 188                 | 194                |
|         | 62              | 448                     | 32                   | 220                 | 226                |
|         | 63              | 512                     | 32                   | 252                 | 258                |
|         | 64              | 576                     | 36                   | 284                 | 290                |
|         | 65              | 640                     | 36                   | 316                 | 322                |
|         | 66              | 768                     | 40                   | 380                 | 386                |
|         | 67              | 960                     | 40                   | 476                 | 482                |
|         | 68              | 640                     | 28                   | 316                 | 322                |
|         | 69              | 768                     | 28                   | 380                 | 386                |
|         | 6A              | 896                     | 36                   | 444                 | 450                |
|         | 6B              | 1024                    | 36                   | 508                 | 514                |
|         | 6C              | 1152                    | 44                   | 572                 | 578                |
|         | 6D              | 1280                    | 44                   | 636                 | 642                |
|         | 6E              | 1536                    | 52                   | 764                 | 770                |
|         | 6F              | 1920                    | 52                   | 956                 | 962                |
| 70      | 1280            | 36                      | 636                  | 642                 |                    |
| 71      | 1536            | 36                      | 764                  | 770                 |                    |
| 72      | 1792            | 52                      | 892                  | 898                 |                    |
| 73      | 2048            | 52                      | 1020                 | 1026                |                    |
| 74      | 2304            | 68                      | 1148                 | 1154                |                    |
| 75      | 2560            | 68                      | 1276                 | 1282                |                    |
| 76      | 3072            | 84                      | 1532                 | 1538                |                    |
| 77      | 3840            | 84                      | 1916                 | 1922                |                    |
| 78      | 2560            | 36                      | 1276                 | 1282                |                    |
| 79      | 3072            | 36                      | 1532                 | 1538                |                    |
| 7A      | 3584            | 68                      | 1788                 | 1794                |                    |
| 7B      | 4096            | 68                      | 2044                 | 2050                |                    |
| 7C      | 4608            | 100                     | 2300                 | 2306                |                    |
| 7D      | 5120            | 100                     | 2556                 | 2562                |                    |
| 7E      | 6144            | 132                     | 3068                 | 3074                |                    |
| 7F      | 7680            | 132                     | 3836                 | 3842                |                    |

Table 22-7. I<sup>2</sup>C divider and hold values (continued)

|         | IBC7-0<br>(hex) | SCL divider<br>(clocks) | SDA hold<br>(clocks) | SCL hold<br>(start) | SCL hold<br>(stop) |
|---------|-----------------|-------------------------|----------------------|---------------------|--------------------|
| MUL = 4 | 80              | 80                      | 28                   | 24                  | 44                 |
|         | 81              | 88                      | 28                   | 28                  | 48                 |
|         | 82              | 96                      | 32                   | 32                  | 52                 |
|         | 83              | 104                     | 32                   | 36                  | 56                 |
|         | 84              | 112                     | 36                   | 40                  | 60                 |
|         | 85              | 120                     | 36                   | 44                  | 64                 |
|         | 86              | 136                     | 40                   | 52                  | 72                 |
|         | 87              | 160                     | 40                   | 64                  | 84                 |
|         | 88              | 112                     | 28                   | 40                  | 60                 |
|         | 89              | 128                     | 28                   | 48                  | 68                 |
|         | 8A              | 144                     | 36                   | 56                  | 76                 |
|         | 8B              | 160                     | 36                   | 64                  | 84                 |
|         | 8C              | 176                     | 44                   | 72                  | 92                 |
|         | 8D              | 192                     | 44                   | 80                  | 100                |
|         | 8E              | 224                     | 52                   | 96                  | 116                |
|         | 8F              | 272                     | 52                   | 120                 | 140                |
|         | 90              | 192                     | 36                   | 72                  | 100                |
|         | 91              | 224                     | 36                   | 88                  | 116                |
|         | 92              | 256                     | 52                   | 104                 | 132                |
|         | 93              | 288                     | 52                   | 120                 | 148                |
|         | 94              | 320                     | 68                   | 136                 | 164                |
|         | 95              | 352                     | 68                   | 152                 | 180                |
|         | 96              | 416                     | 84                   | 184                 | 212                |
|         | 97              | 512                     | 84                   | 232                 | 260                |
|         | 98              | 320                     | 36                   | 152                 | 164                |
|         | 99              | 384                     | 36                   | 184                 | 196                |
|         | 9A              | 448                     | 68                   | 216                 | 228                |
|         | 9B              | 512                     | 68                   | 248                 | 260                |
|         | 9C              | 576                     | 100                  | 280                 | 292                |
|         | 9D              | 640                     | 100                  | 312                 | 324                |
|         | 9E              | 768                     | 132                  | 376                 | 388                |
|         | 9F              | 960                     | 132                  | 472                 | 484                |
|         | A0              | 640                     | 68                   | 312                 | 324                |
|         | A1              | 768                     | 68                   | 376                 | 388                |
|         | A2              | 896                     | 132                  | 440                 | 452                |
|         | A3              | 1024                    | 132                  | 504                 | 516                |
|         | A4              | 1152                    | 196                  | 568                 | 580                |
|         | A5              | 1280                    | 196                  | 632                 | 644                |
|         | A6              | 1536                    | 260                  | 760                 | 772                |
|         | A7              | 1920                    | 260                  | 952                 | 964                |
|         | A8              | 1280                    | 132                  | 632                 | 644                |
|         | A9              | 1536                    | 132                  | 760                 | 772                |
|         | AA              | 1792                    | 260                  | 888                 | 900                |
|         | AB              | 2048                    | 260                  | 1016                | 1028               |
|         | AC              | 2304                    | 388                  | 1144                | 1156               |
|         | AD              | 2560                    | 388                  | 1272                | 1284               |
|         | AE              | 3072                    | 516                  | 1528                | 1540               |
|         | AF              | 3840                    | 516                  | 1912                | 1924               |
| 30      | 2560            | 260                     | 1272                 | 1284                |                    |
| B1      | 3072            | 260                     | 1528                 | 1540                |                    |
| B2      | 3584            | 516                     | 1784                 | 1796                |                    |
| B3      | 4096            | 516                     | 2040                 | 2052                |                    |
| B4      | 4608            | 772                     | 2296                 | 2308                |                    |
| B5      | 5120            | 772                     | 2552                 | 2564                |                    |
| B6      | 6144            | 1028                    | 3064                 | 3076                |                    |
| B7      | 7680            | 1028                    | 3832                 | 3844                |                    |
| B8      | 5120            | 516                     | 2552                 | 2564                |                    |
| B9      | 6144            | 516                     | 3064                 | 3076                |                    |
| BA      | 7168            | 1028                    | 3576                 | 3588                |                    |
| BB      | 8192            | 1028                    | 4088                 | 4100                |                    |
| BC      | 9216            | 1540                    | 4600                 | 4612                |                    |
| BD      | 10240           | 1540                    | 5112                 | 5124                |                    |
| BE      | 12288           | 2052                    | 6136                 | 6148                |                    |
| BF      | 15360           | 2052                    | 7672                 | 7684                |                    |

### 22.3.4 I<sup>2</sup>C Bus Control Register (IBCR)

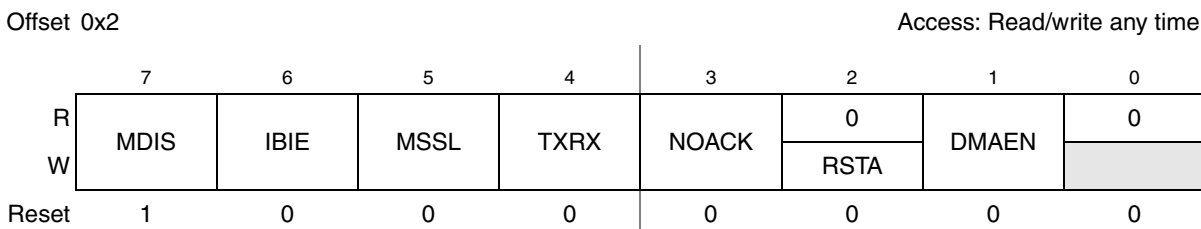


Figure 22-6. I<sup>2</sup>C Bus Control Register (IBCR)

Table 22-8. IBCR field descriptions

| Field | Description   |
|-------|---|
| MDIS  | <p>Module disable. This bit controls the software reset of the entire I<sup>2</sup>C Bus module.</p> <p>1 The module is reset and disabled. This is the power-on reset situation. When high, the interface is held in reset, but registers can still be accessed. Status register bits (IBSR) are not valid when module is disabled.</p> <p>0 The I<sup>2</sup>C Bus module is enabled. This bit must be cleared before any other IBCR bits have any effect</p> <p><b>Note:</b> If the I<sup>2</sup>C Bus module is enabled in the middle of a byte transfer, the interface behaves as follows: slave mode ignores the current transfer on the bus and starts operating whenever a subsequent start condition is detected. Master mode will not be aware that the bus is busy, hence if a start cycle is initiated then the current bus cycle may become corrupt. This would ultimately result in either the current bus master or the I<sup>2</sup>C Bus module losing arbitration, after which, bus operation would return to normal.</p> |
| IBIE  | <p>I-Bus Interrupt Enable.</p> <p>1 Interrupts from the I<sup>2</sup>C Bus module are enabled. An I<sup>2</sup>C Bus interrupt occurs provided the IBIF bit in the status register is also set.</p> <p>0 Interrupts from the I<sup>2</sup>C Bus module are disabled. Note that this does not clear any currently pending interrupt condition</p>  |
| MSSL  | <p>Master/Slave mode select. Upon reset, this bit is cleared. When this bit is changed from 0 to 1, a START signal is generated on the bus and the master mode is selected. When this bit is changed from 1 to 0, a STOP signal is generated and the operation mode changes from master to slave. A STOP signal should be generated only if the IBIF flag is set. MSSL is cleared without generating a STOP signal when the master loses arbitration.</p> <p>1 Master Mode<br/>0 Slave Mode</p>   |
| TXRX  | <p>Transmit/Receive mode select. This bit selects the direction of master and slave transfers. When addressed as a slave this bit should be set by software according to the SRW bit in the status register. In master mode this bit should be set according to the type of transfer required. Therefore, for address cycles, this bit will always be high.</p> <p>1 Transmit<br/>0 Receive</p>   |
| NOACK | <p>Data Acknowledge disable. This bit specifies the value driven onto SDA during data acknowledge cycles for both master and slave receivers. The I<sup>2</sup>C module will always acknowledge address matches, provided it is enabled, regardless of the value of NOACK. Note that values written to this bit are only used when the I<sup>2</sup>C Bus is a receiver, not a transmitter.</p> <p>1 No acknowledge signal response is sent (i.e., acknowledge bit = 1)<br/>0 An acknowledge signal will be sent out to the bus at the 9th clock bit after receiving one byte of data</p>   |

Table 22-8. IBCR field descriptions (continued)

| Field | Description   |
|-------|---|
| RSTA  | Repeat Start. Writing a 1 to this bit will generate a repeated START condition on the bus, provided it is the current bus master. This bit will always be read as a low. Attempting a repeated start at the wrong time, if the bus is owned by another master, will result in loss of arbitration.<br>1 Generate repeat start cycle<br>0 No effect  |
| DMAEN | DMA Enable. When this bit is set, the DMA TX and RX lines will be asserted when the I <sup>2</sup> C module requires data to be read or written to the data register. No Transfer Done interrupts will be generated when this bit is set, however an interrupt will be generated if the loss of arbitration or addressed as slave conditions occur. The DMA mode is only valid when the I <sup>2</sup> C module is configured as a Master and the DMA transfer still requires CPU intervention at the start and the end of each frame of data. See the DMA Application Information section for more details.<br>1 Enable the DMA TX/RX request signals<br>0 Disable the DMA TX/RX request signals |

### 22.3.5 I<sup>2</sup>C Bus Status Register (IBSR)

| Offset 0x3 |     |      |     | Access: Read-write |   |     |      |      |
|------------|-----|------|-----|--------------------|---|-----|------|------|
|            | 7   | 6    | 5   | 4                  | 3 | 2   | 1    | 0    |
| R          | TCF | IAAS | IBB | IBAL               | 0 | SRW | IBIF | RXAK |
| W          |     |      |     | w1c                |   |     | w1c  |      |
| Reset      | 1   | 0    | 0   | 0                  | 0 | 0   | 0    | 0    |

Figure 22-7. I<sup>2</sup>C Bus Status Register (IBSR)

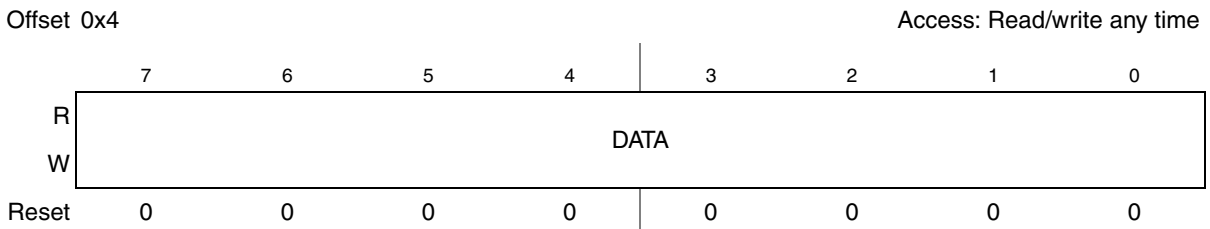
Table 22-9. IBSR Field Descriptions

| Field | Description   |
|-------|---|
| TCF   | Transfer complete. While one byte of data is being transferred, this bit is cleared. It is set by the falling edge of the 9th clock of a byte transfer. Note that this bit is only valid during or immediately following a transfer to the I <sup>2</sup> C module or from the I <sup>2</sup> C module.<br>1 Transfer complete<br>0 Transfer in progress            |
| IAAS  | Addressed as a slave. When its own specific address (I-Bus Address Register) is matched with the calling address, this bit is set. The CPU is interrupted provided the IBIE is set. Then the CPU needs to check the SRW bit and set its Tx/Rx mode accordingly. Writing to the I-Bus Control Register clears this bit.<br>1 Addressed as a slave<br>0 Not addressed |
| IBB   | Bus busy. This bit indicates the status of the bus. When a START signal is detected, the IBB is set. If a STOP signal is detected, IBB is cleared and the bus enters idle state.<br>1 Bus is busy<br>0 Bus is Idle  |

**Table 22-9. IBSR Field Descriptions (continued)**

| Field | Description   |
|-------|---|
| IBAL  | Arbitration Lost. The arbitration lost bit (IBAL) is set by hardware when the arbitration procedure is lost. Arbitration is lost in the following circumstances: <ul style="list-style-type: none"> <li>• SDA is sampled low when the master drives a high during an address or data transmit cycle.</li> <li>• SDA is sampled low when the master drives a high during the acknowledge bit of a data receive cycle.</li> <li>• A start cycle is attempted when the bus is busy.</li> <li>• A repeated start cycle is requested in slave mode.</li> <li>• A stop condition is detected when the master did not request it.</li> </ul> |
| SRW   | Slave Read/Write. When IAAS is set, this bit indicates the value of the R/W command bit of the calling address sent from the master. This bit is only valid when the I-Bus is in slave mode, a complete address transfer has occurred with an address match and no other transfers have been initiated. By programming this bit, the CPU can select slave transmit/receive mode according to the command of the master.<br>1 Slave transmit, master reading from slave<br>0 Slave receive, master writing to slave  |
| IBIF  | I-Bus Interrupt Flag. The IBIF bit is set when one of the following conditions occurs: <ul style="list-style-type: none"> <li>• Arbitration lost (IBAL bit set)</li> <li>• Byte transfer complete (TCF bit set)</li> <li>• Addressed as slave (IAAS bit set)</li> <li>• NoAck from Slave (MS &amp; Tx bits set)</li> <li>• I<sup>2</sup>C Bus going idle (IBB high-low transition and enabled by BIIE)</li> </ul> A processor interrupt request will be caused if the IBIE bit is set.  |
| RXAK  | Received Acknowledge. This is the value of SDA during the acknowledge bit of a bus cycle. If the received acknowledge bit (RXAK) is low, it indicates an acknowledge signal has been received after the completion of 8 bits data transmission on the bus. If RXAK is high, it means no acknowledge signal is detected at the 9th clock. This bit is valid only after transfer is complete.<br>1 No acknowledge received<br>0 Acknowledge received  |

### 22.3.6 I<sup>2</sup>C Bus Data I/O Register (IBDR)



**Figure 22-8. I<sup>2</sup>C Bus Data I/O Register (IBDR)**

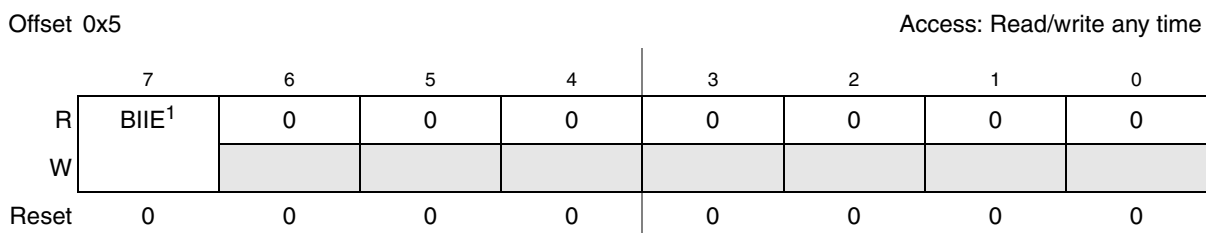
In master transmit mode, when data is written to IBDR, a data transfer is initiated. The most significant bit is sent first. In master receive mode, reading this register initiates next byte data receiving. In slave mode, the same functions are available after an address match has occurred. Note that the IBCR[TXRX] field must correctly reflect the desired direction of transfer in master and slave modes for the transmission to begin. For instance, if the I<sup>2</sup>C is configured for master transmit but a master receive is desired, then reading the IBDR will not initiate the receive.



Reading the IBDR will return the last byte received while the I<sup>2</sup>C is configured in either master receive or slave receive modes. The IBDR does not reflect every byte that is transmitted on the I<sup>2</sup>C bus, nor can software verify that a byte has been written to the IBDR correctly by reading it back.

In master transmit mode, the first byte of data written to IBDR following assertion of  $\overline{MS/\overline{SL}}$  is used for the address transfer and should comprise the calling address (in position D7–D1) concatenated with the required  $\overline{R/\overline{W}}$  bit (in position D0).

### 22.3.7 I<sup>2</sup>C Bus Interrupt Configuration Register (IBIC)



**Figure 22-9. I<sup>2</sup>C Bus Interrupt Configuration Register (IBIC)**

- <sup>1</sup> This bit cannot be set in reset state, when I<sup>2</sup>C is in slave mode. It can be set to 1 only when I<sup>2</sup>C is in Master mode. This information is missing from the spec.

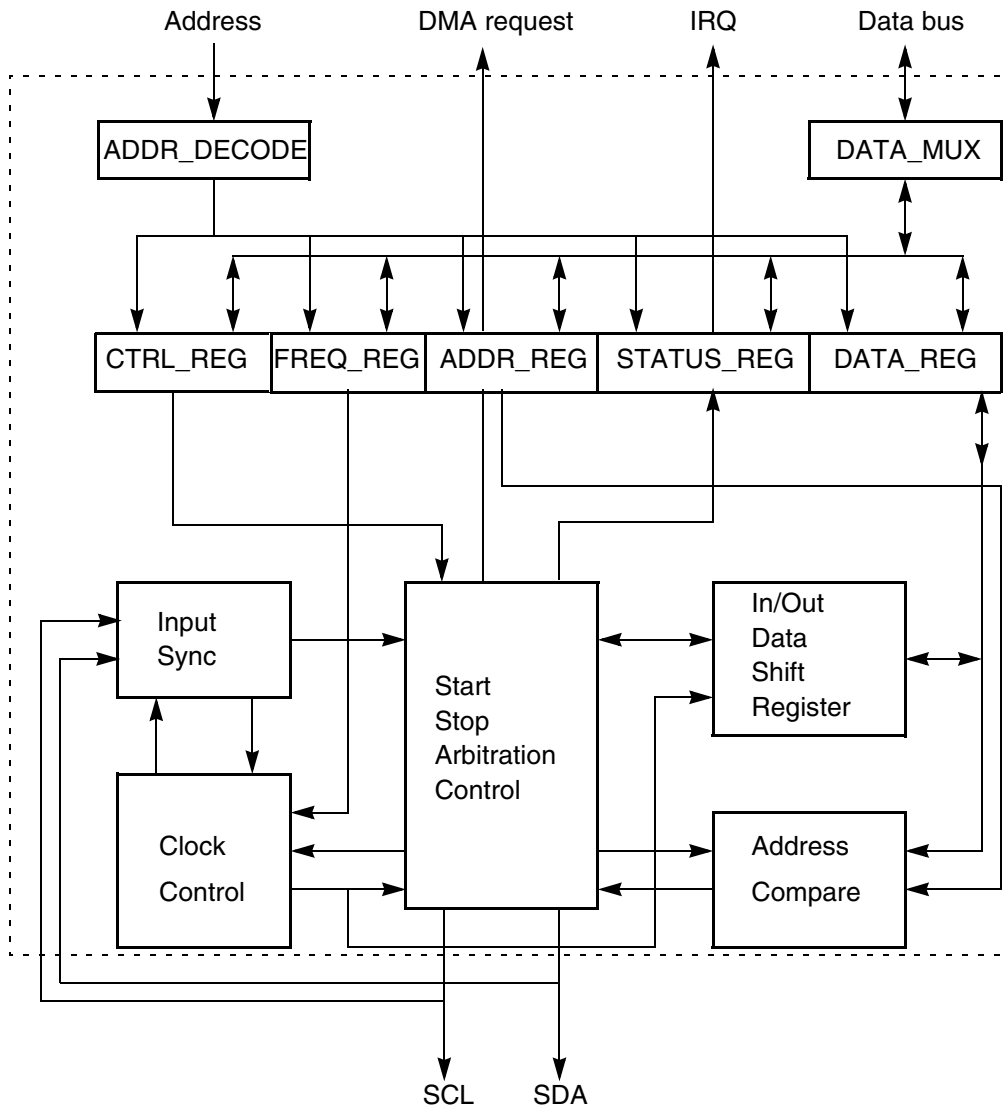
**Table 22-10. IBIC field descriptions**

| Field | Description   |
|-------|---|
| BIIE  | Bus Idle Interrupt Enable bit. This configuration bit can be used to enable the generation of an interrupt once the I <sup>2</sup> C bus becomes idle. Once this bit is set, an IBB high-low transition will set the IBIF bit. This feature can be used to signal to the CPU the completion of a STOP on the I <sup>2</sup> C bus.<br>1 Bus Idle Interrupts enabled<br>0 Bus Idle Interrupts disabled<br><b>Note:</b> |

## 22.4 DMA Interface

A simple DMA interface is implemented so that the I<sup>2</sup>C can request data transfers with minimal support from the CPU. DMA mode is enabled by setting bit 1 in the Control Register.

The DMA interface is only valid when the I<sup>2</sup>C module is configured for Master Mode.



**Figure 22-10. I<sup>2</sup>C module DMA interface block diagram**

At least 3 bytes of data per frame must be transferred from/to the slave when using DMA mode, although in practice it will only be worthwhile using the DMA mode when there is a large number of data bytes to transfer per frame.

Two internal signals, TX request and RX request, are used to signal to a DMA controller when the I<sup>2</sup>C module requires data to be written or read from the data register.

## 22.5 Functional description

### 22.5.1 I-Bus protocol

The I<sup>2</sup>C Bus system uses a Serial Data line (SDA) and a Serial Clock Line (SCL) for data transfer. All devices connected to it must have open drain or open collector outputs. A logical AND function is exercised on both lines with external pull-up resistors. The value of these resistors is system dependent.

Normally, a standard communication is composed of four parts: START signal, slave address transmission, data transfer and STOP signal. They are described briefly in the following sections and illustrated in Figure 22-11.

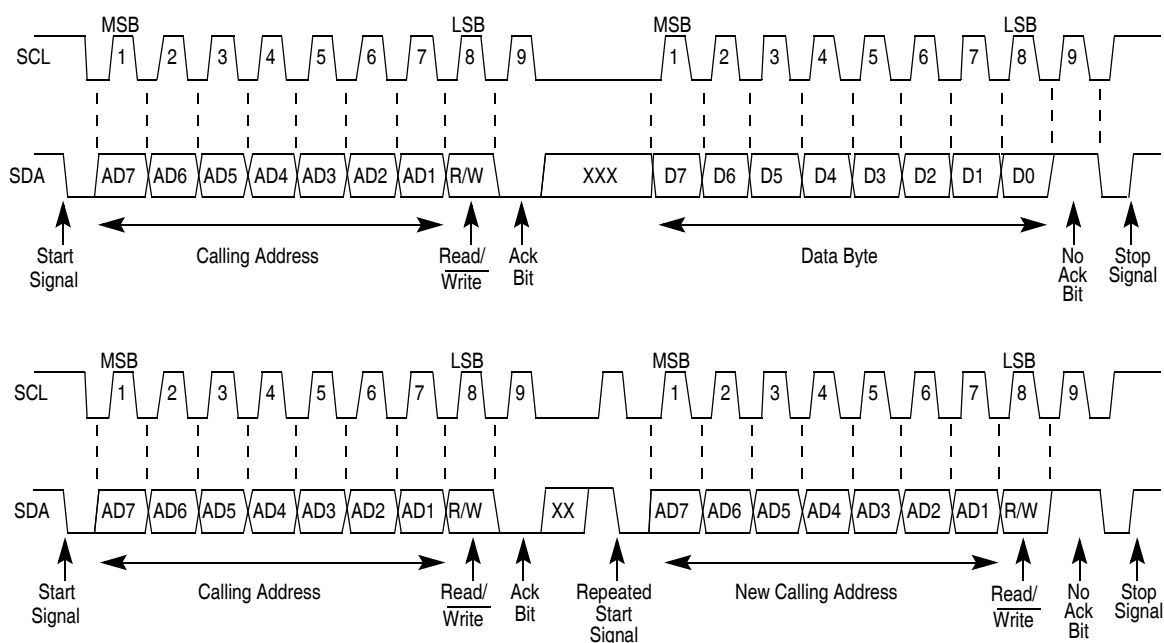


Figure 22-11. I<sup>2</sup>C bus transmission signals

#### 22.5.1.1 START signal

When the bus is free, that is, no master device is engaging the bus (both SCL and SDA lines are at logical high), a master may initiate communication by sending a START signal. As shown in Figure 22-11, a START signal is defined as a high-to-low transition of SDA while SCL is high. This signal denotes the beginning of a new data transfer (each data transfer may contain several bytes of data) and brings all slaves out of their idle states.

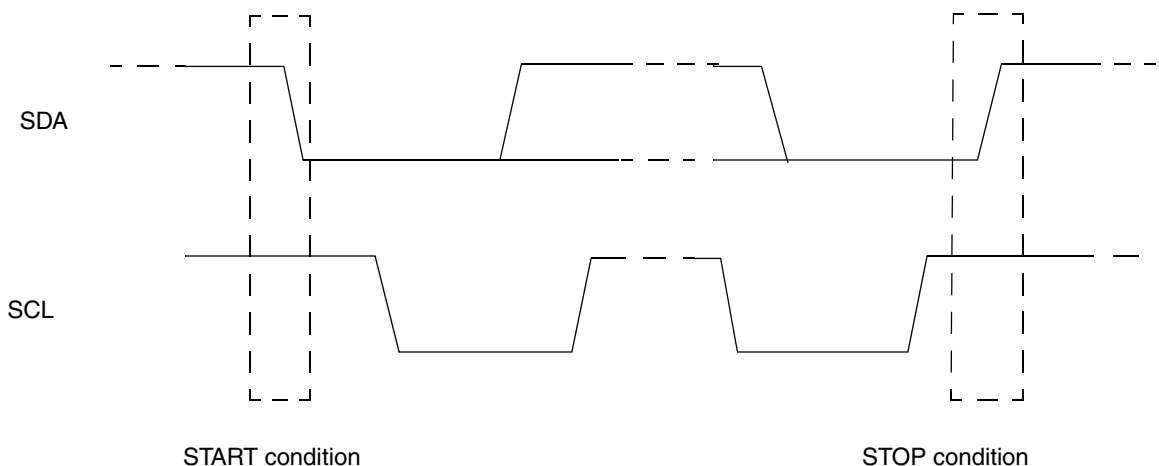


Figure 22-12. Start and stop conditions

### 22.5.1.2 Slave address transmission

The first byte of data transfer immediately after the START signal is the slave address transmitted by the master. This is a 7-bit calling address followed by a R/W bit. The R/W bit tells the slave the desired direction of data transfer.

1 = Read transfer—the slave transmits data to the master

0 = Write transfer—the master transmits data to the slave

Only the slave with a calling address that matches the one transmitted by the master will respond by sending back an acknowledge bit. This is done by pulling the SDA low at the 9th clock (see [Figure 22-11](#)).

No two slaves in the system may have the same address. If the I<sup>2</sup>C bus is master, it must not transmit an address that is equal to its own slave address. The I<sup>2</sup>C bus cannot be master and slave at the same time. However, if arbitration is lost during an address cycle the I<sup>2</sup>C bus will revert to slave mode and operate correctly, even if it is being addressed by another master.

### 22.5.1.3 Data transfer

Once successful slave addressing is achieved, the data transfer can proceed byte-by-byte in a direction specified by the R/W bit sent by the calling master.

All transfers that come after an address cycle are referred to as data transfers, even if they carry sub-address information for the slave device.

Each data byte is 8 bits long. Data may be changed only while SCL is low and must be held stable while SCL is high as shown in [Figure 22-11](#). There is one clock pulse on SCL for each data bit, the MSB being transferred first. Each data byte must be followed by an acknowledge bit, which is signaled from the receiving device by pulling the SDA low at the ninth clock. Therefore, one complete data byte transfer needs 9 clock pulses.

If the slave receiver does not acknowledge the master, the SDA line must be left high by the slave. The master can then generate a stop signal to abort the data transfer or a start signal (repeated start) to commence a new calling.

If the master receiver does not acknowledge the slave transmitter after a byte transmission, it means 'end of data' to the slave, so the slave releases the SDA line for the master to generate a STOP or START signal.

#### 22.5.1.4 STOP signal

The master can terminate the communication by generating a STOP signal to free the bus. However, the master may generate a START signal followed by a calling command without generating a STOP signal first. This is called repeated START. A STOP signal is defined as a low-to-high transition of SDA while SCL is at logical 1 (see [Figure 22-11](#)).

The master can generate a STOP even if the slave has generated an acknowledge, at which point the slave must release the bus.

#### 22.5.1.5 Repeated START signal

As shown in [Figure 22-11](#), a repeated START signal is a START signal generated without first generating a STOP signal to terminate the communication. This is used by the master to communicate with another slave or with the same slave in different mode (transmit/receive mode) without releasing the bus.

#### 22.5.1.6 Arbitration procedure

The Inter-IC bus is a true multi-master bus that allows more than one master to be connected on it. If two or more masters try to control the bus at the same time, a clock synchronization procedure determines the bus clock, for which the low period is equal to the longest clock low period and the high is equal to the shortest one among the masters. The relative priority of the contending masters is determined by a data arbitration procedure. A bus master loses arbitration if it transmits logic 1 while another master transmits logic 0. The losing masters immediately switch over to slave receive mode and stop driving the SDA output. In this case, the transition from master to slave mode does not generate a STOP condition. Meanwhile, a status bit is set by hardware to indicate loss of arbitration.

#### 22.5.1.7 Clock synchronization

Since wire-AND logic is performed on the SCL line, a high-to-low transition on the SCL line affects all the devices connected on the bus. The devices start counting their low period and once a device's clock has gone low, it holds the SCL line low until the clock high state is reached. However, the change of low to high in this device clock may not change the state of the SCL line if another device clock is still within its low period. Therefore, synchronized clock SCL is held low by the device with the longest low period. Devices with shorter low periods enter a high wait state during this time (see [Figure 22-13](#)). When all devices concerned have counted off their low period, the synchronized clock SCL line is released and pulled high. There is then no difference between the device clocks and the state of the SCL line and all the devices start counting their high periods. The first device to complete its high period pulls the SCL line low again.

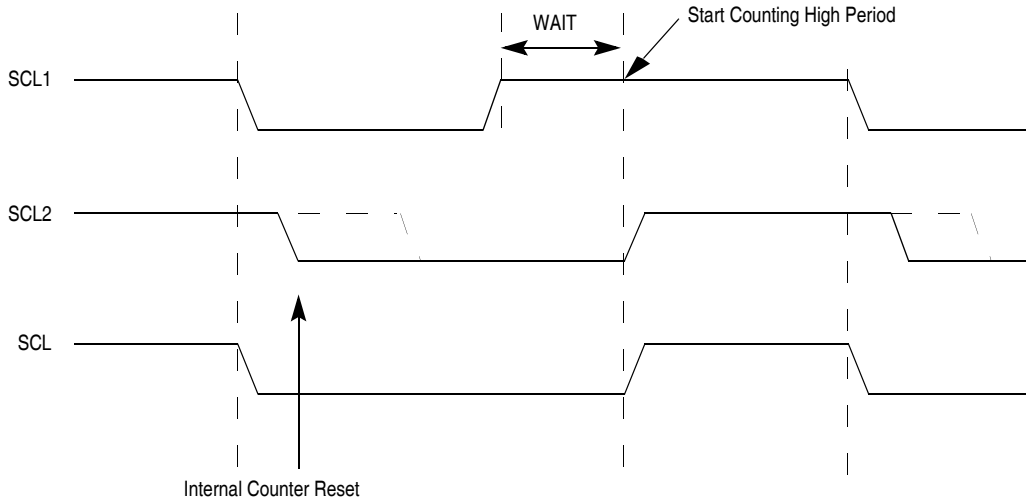


Figure 22-13. I<sup>2</sup>C bus clock synchronization

### 22.5.1.8 Handshaking

The clock synchronization mechanism can be used as a handshake in data transfer. Slave devices may hold the SCL low after completion of one byte transfer (9 bits). In such cases, it halts the bus clock and forces the master clock into wait state until the slave releases the SCL line.

### 22.5.1.9 Clock stretching

The clock synchronization mechanism can be used by slaves to slow down the bit rate of a transfer. After the master has driven SCL low, the slave can drive SCL low for the required period and then release it. If the slave SCL low period is greater than the master SCL low period then the resulting SCL bus signal low period is stretched.

## 22.5.2 Interrupts

### 22.5.2.1 General

The I<sup>2</sup>C uses only one interrupt vector.

Table 22-11. Interrupt summary

| Interrupt                  | Offset | Vector | Priority | Source                                     | Description  |
|----------------------------|--------|--------|----------|--|--|
| I <sup>2</sup> C Interrupt | —      | —      | —        | IBAL, TCF, IAAS, IBB bits in IBSR register | When any of IBAL, TCF or IAAS bits is set an interrupt may be caused based on Arbitration lost, Transfer Complete or Address Detect conditions. If enabled by BIIE, the deassertion of IBB can also cause an interrupt, indicating that the bus is idle. |

### 22.5.2.2 Interrupt description

There are five types of internal interrupts in the I<sup>2</sup>C. The interrupt service routine can determine the interrupt type by reading the Status Register.

The I<sup>2</sup>C interrupt can be generated on

- Arbitration Lost condition (IBAL bit set)
- Byte Transfer condition (TCF bit set and DMAEN bit not set)
- Address Detect condition (IAAS bit set)
- No Acknowledge from slave received when expected
- Bus Going Idle (IBB bit not set)

The I<sup>2</sup>C interrupt is enabled by the IBIE bit in the I<sup>2</sup>C Control Register. It must be cleared by writing 1 to the IBIF bit in the interrupt service routine. The Bus Going Idle interrupt needs to be additionally enabled by the BIIE bit in the IBIC register.

## 22.6 Initialization/application information

### 22.6.1 I<sup>2</sup>C programming examples

#### 22.6.1.1 Initialization sequence

Reset will put the I<sup>2</sup>C Bus Control Register to its default state. Before the interface can be used to transfer serial data, an initialization procedure must be carried out, as follows:

1. Update the Frequency Divider Register (IBFD) and select the required division ratio to obtain SCL frequency from system clock.
2. Update the I<sup>2</sup>C Bus Address Register (IBAD) to define its slave address.
3. Clear the IBCR[MDIS] field to enable the I<sup>2</sup>C interface system.
4. Modify the bits of the I<sup>2</sup>C Bus Control Register (IBCR) to select Master/Slave mode, Transmit/Receive mode and interrupt enable or not. Optionally also modify the bits of the I<sup>2</sup>C Bus Interrupt Configuration Register (IBIC) to further refine the interrupt behavior.

#### 22.6.1.2 Generation of START

After completion of the initialization procedure, serial data can be transmitted by selecting the 'master transmitter' mode. If the device is connected to a multi-master bus system, the state of the I<sup>2</sup>C Bus Busy bit (IBB) must be tested to check whether the serial bus is free.

If the bus is free (IBB=0), the start condition and the first byte (the slave address) can be sent. The data written to the data register comprises the slave calling address and the LSB, which is set to indicate the direction of transfer required from the slave.

The bus free time (i.e., the time between a STOP condition and the following START condition) is built into the hardware that generates the START cycle. Depending on the relative frequencies of the system

clock and the SCL period, it may be necessary to wait until the I<sup>2</sup>C is busy after writing the calling address to the IBDR before proceeding with the following instructions. This is illustrated in the following example.

An example of the sequence of events that generates the START signal and transmits the first byte of data (slave address) is shown below:

```
while (bit 5, IBSR ==1)// wait in loop for IBB flag to clear
bit4 and bit 5, IBCR = 1// set transmit and master mode, i.e. generate start condition
IBDR = calling_address// send the calling address to the data register
while (bit 5, IBSR ==0)// wait in loop for IBB flag to be set
```

### 22.6.1.3 Post-transfer software response

Transmission or reception of a byte will set the data transferring bit (TCF) to 1, which indicates one byte communication is finished. The I<sup>2</sup>C Bus interrupt bit (IBIF) is set also; an interrupt will be generated if the interrupt function is enabled during initialization by setting the IBIE bit. The IBIF (interrupt flag) can be cleared by writing 1 (in the interrupt service routine, if interrupts are used).

The TCF bit will be cleared to indicate data transfer in progress whenever data register is written to in transmit mode, or during reading out from data register in receive mode. The TCF bit should not be used as a data transfer complete flag as the flag timing is dependent on a number of factors including the I<sup>2</sup>C bus frequency. This bit may not conclusively provide an indication of a transfer complete situation. It is recommended that transfer complete situations are detected using the IBIF flag

Software may service the I<sup>2</sup>C I/O in the main program by monitoring the IBIF bit if the interrupt function is disabled. Note that polling should monitor the IBIF bit rather than the TCF bit since their operation is different when arbitration is lost.

Note that when a Transfer Complete interrupt occurs at the end of the address cycle, the master will always be in transmit mode, i.e. the address is transmitted. If master receive mode is required, indicated by R/W bit sent with slave calling address, then the Tx/Rx bit at Master side should be toggled at this stage. If Master does not receive an ACK from Slave, then transmission must be re-initiated or terminated.

In slave mode, IAAS bit will get set in IBSR if Slave address (IBAD) matches the Master calling address. This is an indication that Master-Slave data communication can now start. During address cycles (IAAS=1), the SRW bit in the status register is read to determine the direction of the subsequent transfer and the Tx/Rx bit is programmed accordingly. For slave mode data cycles (IAAS=0), the SRW bit is not valid. The Tx/Rx bit in the control register should be read to determine the direction of the current transfer.

### 22.6.1.4 Transmit/receive sequence

Follow this sequence in case of Master Transmit (Address/Data):

1. Clear IBSR[IBIF].
2. Write data in Data Register (IBDR).
3. IBSR[TCF] bit will get cleared when transfer is in progress.
4. IBSR[TCF] bit will get set when transfer is complete.
5. Wait for IBSR[IBIF] to get set, then read IBSR register to determine its source:
  - TCF = 1, transfer is complete.



- No Acknowledge condition (RXAK = 1) is found.
  - IBB = 0, Bus has transitioned from Busy to Idle state.
  - If IBB = 1, ignore check of Arbitration Loss (IBAL = 1).
  - Ignore Address Detect (IAAS = 1) for Master mode (valid only for Slave mode).
6. f) Check RXAK in IBSR for an acknowledge from slave.

Follow this sequence in case of Slave Receive (Address/Data):

1. Clear IBSR[IBIF].
2. IBSR[TCF] will get cleared when transfer is in progress for address transfer.
3. IBSR[TCF] will get set when transfer is complete.
4. Wait for IBSR[IBIF] to get set. Then read IBSR register to determine its source:
  - Address Detect has occurred (IAAS = 1)—determination of Slave mode.
5. Clear IBIF.
6. Wait until IBSR[TCF] bit gets cleared (that is, “Transfer under Progress” condition is reached for data transfer).
7. Wait until IBSR[TCF] bit gets cleared (proof that transfer completes from “Transfer under Progress” state).
8. Wait until IBSR[IBIF] bit gets set. To find its source, check if:
  - TCF = 1 i.e. reception is complete
  - IBSR[IBB] = 0, that is, bus has transitioned from Busy to Idle state
  - Ignore Arbitration Loss (IBAL = 1) for IBB = 1
  - Ignore No Acknowledge condition (RXAK = 1) for receiver
9. Read the Data Register (IBDR) to determine data received from Master.

Sequence followed in case of Slave Transmit (Steps 1–4 of Slave Receive for Address Detect, followed by 1–6 of Master Transmit for Data Transmit).

Sequence followed in case of Master Receive (Steps 1–6 of Master Transmit for Address dispatch, followed by 5–8 of Slave Receive for Data Receive).

### 22.6.1.5 Generation of STOP

A data transfer ends with a STOP signal generated by the 'master' device. A master transmitter can simply generate a STOP signal after all the data has been transmitted. The following is an example showing how a stop condition is generated by a master transmitter.

```

if (tx_count == 0) or// check to see if all data bytes have been transmitted
    (bit 0, IBSR == 1) {// or if no ACK generated
        clear bit 5, IBCR// generate stop condition
    }
else {
    IBDR = data_to_transmit// write byte of data to DATA register
    tx_count --// decrement counter
} // return from interrupt

```

If a master receiver wants to terminate a data transfer, it must inform the slave transmitter by not acknowledging the last byte of data, which can be done by setting the transmit acknowledge bit (TXAK) before reading the 2nd last byte of data. Before reading the last byte of data, a STOP signal must first be generated. The following is an example showing how a STOP signal is generated by a master receiver.

```
rx_count --// decrease the rx counter
if (rx_count ==1)// 2nd last byte to be read ?
    bit 3, IBCR = 1// disable ACK
if (rx_count == 0)// last byte to be read ?
    bit 5, IBCR = 0// generate stop signal
else
    data_received = IBDR// read RX data and store
```

### 22.6.1.6 Generation of repeated START

At the end of data transfer, if the master still wants to communicate on the bus, it can generate another START signal followed by another slave address without first generating a STOP signal. A program example is as shown.

```
bit 2, IBCR = 1// generate another start ( restart)
IBDR == calling_address// transmit the calling address
```

### 22.6.1.7 Slave mode

In the slave interrupt service routine, the module addressed as slave bit (IAAS) should be tested to check if a calling of its own address has just been received. If IAAS is set, software should set the transmit/receive mode select bit (Tx/Rx bit of IBCR) according to the R/W command bit (SRW). Writing to the IBCR clears IAAS automatically. Note that the only time IAAS is read as set is from the interrupt at the end of the address cycle where an address match occurred. Interrupts resulting from subsequent data transfers will have IAAS cleared. A data transfer may now be initiated by writing information to IBDR for slave transmits or dummy reading from IBDR in slave receive mode. The slave will drive SCL low in-between byte transfers SCL is released when the IBDR is accessed in the required mode.

In slave transmitter routine, the received acknowledge bit (RXAK) must be tested before transmitting the next byte of data. Setting RXAK means an 'end of data' signal from the master receiver, after which it must be switched from transmitter mode to receiver mode by software. A dummy read then releases the SCL line so that the master can generate a STOP signal.

### 22.6.1.8 Arbitration lost

If several masters try to engage the bus simultaneously, only one master wins and the others lose arbitration. The devices that lost arbitration are immediately switched to slave receive mode by the hardware. Their data output to the SDA line is stopped, but SCL is still generated until the end of the byte during which arbitration was lost. An interrupt occurs at the falling edge of the ninth clock of this transfer with IBAL=1 and MS/SL=0. If one master attempts to start transmission, while the bus is being engaged by another master, the hardware will inhibit the transmission, switch the MS/SL bit from 1 to 0 without generating a STOP condition, generate an interrupt to CPU and set the IBAL to indicate that the attempt to engage the bus is failed. When considering these cases, the slave service routine should test the IBAL first and the software should clear the IBAL bit if it is set.

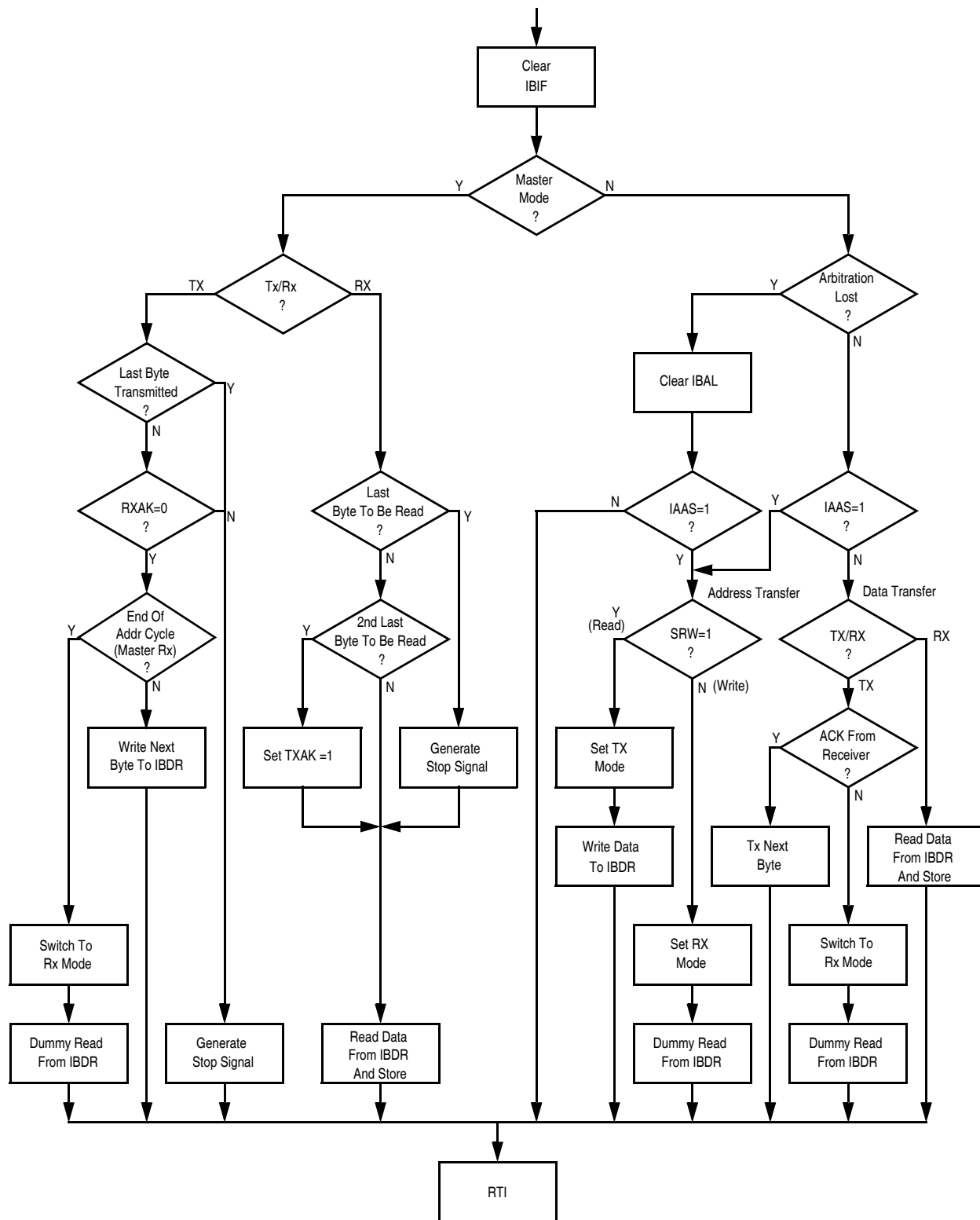


Figure 22-14. Flow chart of typical I<sup>2</sup>C interrupt routine

This page is intentionally left blank.

# Chapter 23

## LIN Controller (LINFlex)

### 23.1 Introduction

The LINFlex (Local Interconnect Network Flexible) controller interfaces the LIN network and supports the LIN protocol versions 1.3; 2.0 and 2.1; and J2602 in both Master and Slave modes. LINFlex includes a LIN mode that provides additional features (compared to standard UART) to ease LIN implementation, improve system robustness, minimize CPU load and allow slave node resynchronization.

### 23.2 Main features

#### 23.2.1 LIN mode features

- Supports LIN protocol versions 1.3, 2.0, 2.1, and J2602
- Master mode with autonomous message handling
- Classic and enhanced checksum calculation and check
- Single 8-byte buffer for transmission/reception
- Extended frame mode for In-Application Programming (IAP) purposes
- Wake-up event on dominant bit detection
- True LIN field state machine
- Advanced LIN error detection
- Header, response, and frame timeout
- Slave mode
  - Autonomous header handling
  - Autonomous transmit/receive data handling
- LIN automatic resynchronization, allowing operation with 16 MHz fast internal RC oscillator as clock source
- 16 identifier filters for autonomous message handling in Slave mode
- Peripheral DMA request sources possible from LINFlex

#### 23.2.2 UART mode features

- Full duplex communication
- 8- or 9-bit with parity
- 4-byte buffer for reception, 4-byte buffer for transmission
- 8-bit counter for timeout management

#### 23.2.3 Features common to LIN and UART

- Fractional baud rate generator

- Three operating modes for power saving and configuration registers lock:
  - Initialization
  - Normal
  - Sleep
- Two test modes:
  - Loop Back
  - Self Test
- Maskable interrupts

### 23.3 General description

The increasing number of communication peripherals embedded on microcontrollers, for example CAN, LIN, and SPI, requires more and more CPU resources for communication management. Even a 32-bit microcontroller is overloaded if its peripherals do not provide high-level features to autonomously handle the communication.

Even though the LIN protocol with a maximum baud rate of 20 kbit/s is relatively slow, it still generates a non-negligible load on the CPU if the LIN is implemented on a standard UART, as usually the case.

To minimize the CPU load in Master mode, LINFlex handles the LIN messages autonomously.

In Master mode, once the software has triggered the header transmission, LINFlex does not request any software intervention until the next header transmission request in transmission mode or until the checksum reception in reception mode.

To minimize the CPU load in Slave mode, LINFlex requires software intervention only to:

- Trigger transmission or reception or data discard depending on the identifier
- Write data into the buffer (transmission mode) or read data from the buffer (reception mode) after checksum reception

If filter mode is activated for Slave mode, LINFlex requires software intervention only to write data into the buffer (transmission mode) or read data from the buffer (reception mode)

The software uses the control, status, and configuration registers to:

- Configure LIN parameters (for example, baud rate or mode)
- Request transmissions
- Handle receptions
- Manage interrupts
- Configure LIN error and timeout detection
- Process diagnostic information

The message buffer stores transmitted or received LIN frames.

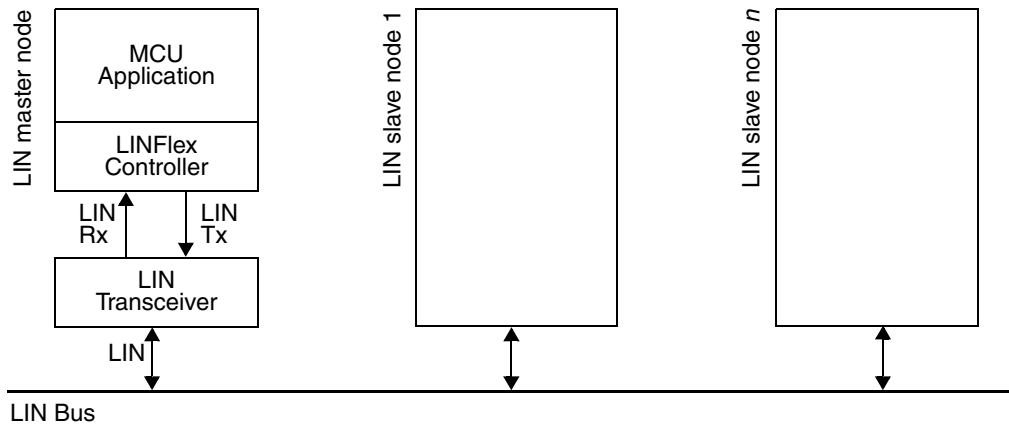
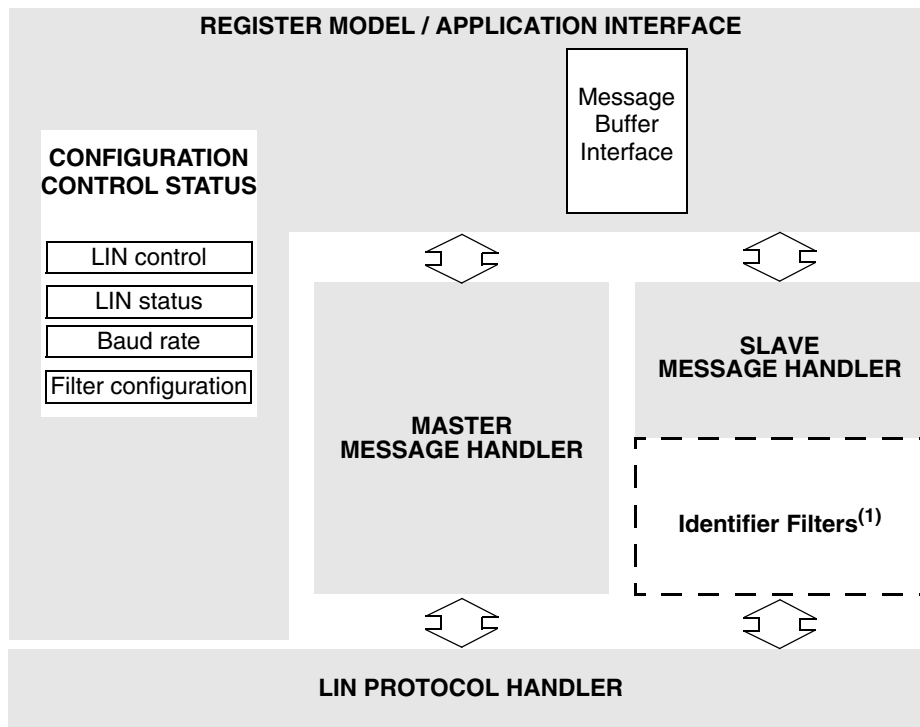


Figure 23-1. LIN topology network



1. Filter activation optional

Figure 23-2. LINFlex block diagram

## 23.4 Fractional baud rate generation

The baud rates for the receiver and transmitter are both set to the same value as programmed in the Mantissa (LINIBRR) and Fraction (LINFBR) registers.

$$\text{Tx/ Rx baud} = \frac{f_{\text{periph\_set\_1\_clk}}}{(16 \times \text{LFDIV})}$$

LFDIV is an unsigned fixed point number. The 12-bit mantissa is coded in the LINIBRR and the fraction is coded in the LINFBR.

The following examples show how to derive LFDIV from LINIBRR and LINFBR register values:

**Example 23-1. Deriving LFDIV from LINIBRR and LINFBR register values**

If LINIBRR = 27d and LINFBR = 12d, then

Mantissa (LFDIV) = 27d

Fraction (LFDIV) = 12/16 = 0.75d

Therefore LFDIV = 27.75d

**Example 23-2. Programming LFDIV from LINIBRR and LINFBR register values**

To program LFDIV = 25.62d,

LINFBR = 16 × 0.62 = 9.92, nearest real number 10d = 0xA

LINIBRR = mantissa (25.620d) = 25d = 0x19

**NOTE**

The baud counters are updated with the new value of the baud registers after a write to LINIBRR. Hence the baud register value must not be changed during a transaction. The LINFBR (containing the Fraction bits) must be programmed before the LINIBRR.

**NOTE**

LFDIV must be greater than or equal to 1.5d, i.e. LINIBRR = 1 and LINFBR = 8. Therefore, the maximum possible baudrate is  $f_{\text{periph\_set\_1\_clk}} / 24$ .

**Table 23-1. Error calculation for programmed baud rates**

| Baud rate | $f_{\text{periph\_set\_1\_clk}} = 64 \text{ MHz}$ |  |        |  | $f_{\text{periph\_set\_1\_clk}} = 16 \text{ MHz}$ |  |        |  |
|-----------|---|--|--------|--|---|--|--------|--|
|           | Actual  | Value programmed in the baud rate register |        | % Error = (Calculated – Desired) baud rate / Desired baud rate | Actual  | Value programmed in the baud rate register |        | % Error = (Calculated – Desired) baud rate / Desired baud rate |
|           |   | LINIBRR                                    | LINFBR |  |   | LINIBRR                                    | LINFBR |  |
| 2400      | 2399.97   | 1666                                       | 11     | –0.001   | 2399.88   | 416  | 11     | –0.005   |
| 9600      | 9599.52   | 416  | 11     | –0.005   | 9598.08   | 104  | 3      | –0.02  |
| 10417     | 10416.7   | 384  | 0      | –0.003   | 10416.7   | 96   | 0      | –0.003   |



Table 23-1. Error calculation for programmed baud rates (continued)

| Baud rate | $f_{\text{periph\_set\_1\_clk}} = 64 \text{ MHz}$ |  |        |  | $f_{\text{periph\_set\_1\_clk}} = 16 \text{ MHz}$ |  |        |  |
|-----------|---|--|--------|--|---|--|--------|--|
|           | Actual  | Value programmed in the baud rate register |        | % Error = (Calculated – Desired) baud rate / Desired baud rate | Actual  | Value programmed in the baud rate register |        | % Error = (Calculated – Desired) baud rate / Desired baud rate |
|           |   | LINIBRR                                    | LINFBR |  |   | LINIBRR                                    | LINFBR |  |
| 19200     | 19201.9   | 208  | 5      | 0.01   | 19207.7   | 52   | 1      | 0.04   |
| 57600     | 57605.8   | 69   | 7      | 0.01   | 57554   | 17   | 6      | -0.08  |
| 115200    | 115108  | 34   | 12     | -0.08  | 115108  | 8  | 11     | -0.08  |
| 230400    | 230216  | 17   | 6      | -0.08  | 231884  | 4  | 5      | 0.644  |
| 460800    | 460432  | 8  | 11     | -0.08  | 457143  | 2  | 3      | -0.794   |
| 921600    | 927536  | 4  | 5      | 0.644  | 941176  | 1  | 1      | 2.124  |

## 23.5 Operating modes

LINFlex has three main operating modes: Initialization, Normal, and Sleep. After a hardware reset, LINFlex is in Sleep mode to reduce power consumption. The software instructs LINFlex to enter Initialization mode or Sleep mode by setting the INIT bit or SLEEP bit in the LINCRR1.

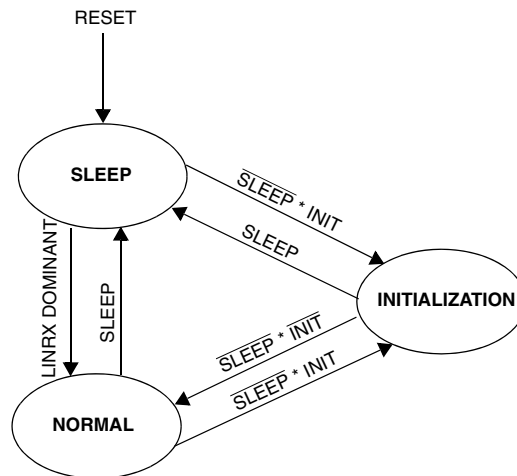


Figure 23-3. LINFlex operating modes

### 23.5.1 Initialization mode

The software can be initialized while the hardware is in Initialization mode. To enter this mode the software sets the INIT bit in the LINCRI.

To exit Initialization mode, the software clears the INIT bit.

While in Initialization mode, all message transfers to and from the LIN bus are stopped and the status of the LIN bus output LINTX is recessive (high).

Entering Initialization mode does not change any of the configuration registers.

To initialize the LINFlex controller, the software selects the mode (LIN Master, LIN Slave or UART), sets up the baud rate register and, if LIN Slave mode with filter activation is selected, initializes the identifier list.

### 23.5.2 Normal mode

Once initialization is complete, software clears the INIT bit in the LINCRI to put the hardware into Normal mode.

### 23.5.3 Low power mode (Sleep)

To reduce power consumption, LINFlex has a low power mode called Sleep mode. To enter Sleep mode, software sets the SLEEP bit in the LINCRI. In this mode, the LINFlex clock is stopped. Consequently, the LINFlex will not update the status bits but software can still access the LINFlex registers.

LINFlex can be awakened (exit Sleep mode) either by software clearing the SLEEP bit or on detection of LIN bus activity if automatic wake-up mode is enabled (AWUM bit is set).

On LIN bus activity detection, hardware automatically performs the wake-up sequence by clearing the SLEEP bit if the AWUM bit in the LINCRI is set. To exit from Sleep mode if the AWUM bit is cleared, software clears the SLEEP bit when a wake-up event occurs.

## 23.6 Test modes

Two test modes are available to the user: Loop Back mode and Self Test mode. They can be selected by the LBKM and SFTM bits in the LINCRI. These bits must be configured while LINFlex is in Initialization mode. Once one of the two test modes has been selected, LINFlex must be started in Normal mode.

### 23.6.1 Loop Back mode

LINFlex can be put in Loop Back mode by setting the LBKM bit in the LINCRI. In Loop Back mode, the LINFlex treats its own transmitted messages as received messages.

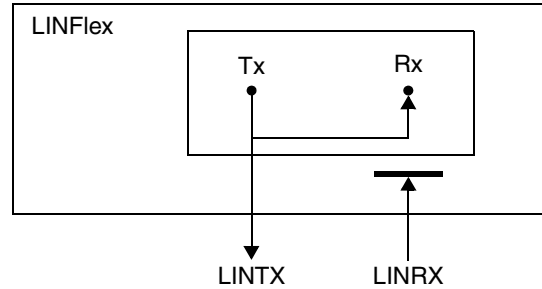


Figure 23-4. LINFlex in loop back mode

This mode is provided for self test functions. To be independent of external events, the LIN core ignores the LINRX signal. In this mode, the LINFlex performs an internal feedback from its Tx output to its Rx input. The actual value of the LINRX input pin is disregarded by the LINFlex. The transmitted messages can be monitored on the LINTX pin.

### 23.6.2 Self Test mode

LINFlex can be put in Self Test mode by setting the LBKM and SFTM bits in the LINCR. This mode can be used for a Hot Self Test, meaning the LINFlex can be tested as in Loop Back mode but without affecting a running LIN system connected to the LINTX and LINRX pins. In this mode, the LINRX pin is disconnected from the LINFlex and the LINTX pin is held recessive.

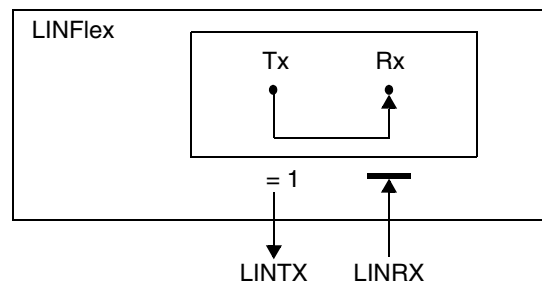


Figure 23-5. LINFlex in self test mode

## 23.7 Memory map and registers description

### 23.7.1 Memory map

See [Chapter 3, Memory Map](#), of this reference manual for the base addresses for the LINFlex modules. [Table 23-2](#) shows the LINFlex memory map.

Table 23-2. LINFlex memory map

| Address offset | Register                                       | Location    |
|----------------|--|-------------|
| 0x0000         | LIN control register 1 (LINC1R1)               | on page 429 |
| 0x0004         | LIN interrupt enable register (LINIER)         | on page 432 |
| 0x0008         | LIN status register (LINSR)                    | on page 433 |
| 0x000C         | LIN error status register (LINESR)             | on page 436 |
| 0x0010         | UART mode control register (UARTCR)            | on page 437 |
| 0x0014         | UART mode status register (UARTSR)             | on page 438 |
| 0x0018         | LIN timeout control status register (LINTCSR)  | on page 440 |
| 0x001C         | LIN output compare register (LINOOCR)          | on page 441 |
| 0x0020         | LIN timeout control register (LINTOCR)         | on page 442 |
| 0x0024         | LIN fractional baud rate register (LINFBR)     | on page 442 |
| 0x0028         | LIN integer baud rate register (LINIBRR)       | on page 443 |
| 0x002C         | LIN checksum field register (LINCFR)           | on page 444 |
| 0x0030         | LIN control register 2 (LINC1R2)               | on page 444 |
| 0x0034         | Buffer identifier register (BIDR)              | on page 445 |
| 0x0038         | Buffer data register LSB (BDRL) <sup>1</sup>   | on page 446 |
| 0x003C         | Buffer data register MSB (BDRM) <sup>2</sup>   | on page 447 |
| 0x0040         | Identifier filter enable register (IFER)       | on page 448 |
| 0x0044         | Identifier filter match index (IFMI)           | on page 449 |
| 0x0048         | Identifier filter mode register (IFMR)         | on page 450 |
| 0x004C         | Identifier filter control register 0 (IFCR0)   | on page 451 |
| 0x0050         | Identifier filter control register 1 (IFCR1)   | on page 452 |
| 0x0054         | Identifier filter control register 2 (IFCR2)   | on page 452 |
| 0x0058         | Identifier filter control register 3 (IFCR3)   | on page 452 |
| 0x005C         | Identifier filter control register 4 (IFCR4)   | on page 452 |
| 0x0060         | Identifier filter control register 5 (IFCR5)   | on page 452 |
| 0x0064         | Identifier filter control register 6 (IFCR6)   | on page 452 |
| 0x0068         | Identifier filter control register 7 (IFCR7)   | on page 452 |
| 0x006C         | Identifier filter control register 8 (IFCR8)   | on page 452 |
| 0x0070         | Identifier filter control register 9 (IFCR9)   | on page 452 |
| 0x0074         | Identifier filter control register 10 (IFCR10) | on page 452 |
| 0x0078         | Identifier filter control register 11 (IFCR11) | on page 452 |
| 0x007C         | Identifier filter control register 12 (IFCR12) | on page 452 |
| 0x0080         | Identifier filter control register 13 (IFCR13) | on page 452 |

Table 23-2. LINFlex memory map (continued)

| Address offset | Register                                       | Location                    |
|----------------|--|-----------------------------|
| 0x0084         | Identifier filter control register 14 (IFCR14) | <a href="#">on page 452</a> |
| 0x0088         | Identifier filter control register 15 (IFCR15) | <a href="#">on page 452</a> |
| 0x008C–0x000F  | Reserved                                       |                             |

<sup>1</sup> LSB: Least significant byte

<sup>2</sup> MSB: Most significant byte

### 23.7.1.1 LIN control register 1 (LINCRI)

Offset: 0x0000 Access: User read/write

|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       | 16  | 17  | 18   | 19   | 20  | 21 | 22 | 23 | 24 | 25   | 26   | 27  | 28   | 29   | 30    | 31   |
|-------|-----|-----|------|------|-----|----|----|----|----|------|------|-----|------|------|-------|------|
| R     | CCD | CFD | LASE | AWUM | MBL |    |    |    | BF | SFTM | LBKM | MME | SBDT | RBLM | SLEEP | INIT |
| W     |     |     |      |      |     |    |    |    |    |      |      |     |      |      |       |      |
| Reset | 0   | 0   | 0    | 0    | 0   | 0  | 0  | 0  | 1  | 0    | 0    | 0   | 0    | 0    | 1     | 0    |

Figure 23-6. LIN control register 1 (LINCRI)

Table 23-3. LINCRI field descriptions

| Field | Description   |
|-------|---|
| CCD   | Checksum calculation disable<br>This bit disables the checksum calculation (see <a href="#">Table 23-4</a> ).<br>0 Checksum calculation is done by hardware. When this bit is 0, the LINCRI is read-only.<br>1 Checksum calculation is disabled. When this bit is set the LINCRI is read/write. User can program this register to send a software-calculated CRC (provided CFD is 0).<br><b>Note:</b> This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode. |
| CFD   | Checksum field disable<br>This bit disables the checksum field transmission (see <a href="#">Table 23-4</a> ).<br>0 Checksum field is sent after the required number of data bytes is sent.<br>1 No checksum field is sent.<br><b>Note:</b> This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode.   |
| LASE  | LIN Slave Automatic Resynchronization Enable<br>0 Automatic resynchronization disable.<br>1 Automatic resynchronization enable.<br><b>Note:</b> This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode.   |

Table 23-3. LINCR1 field descriptions (continued)

| Field | Description  |
|-------|--|
| AWUM  | Automatic Wake-Up Mode<br>This bit controls the behavior of the LINFlex hardware during Sleep mode.<br>0 The Sleep mode is exited on software request by clearing the SLEEP bit of the LINCR.<br>1 The Sleep mode is exited automatically by hardware on LINRX dominant state detection. The SLEEP bit of the LINCR is cleared by hardware whenever WUF bit in the LINSR is set.<br><b>Note:</b> This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode. |
| MBL   | LIN Master Break Length<br>This field indicates the Break length in Master mode (see <a href="#">Table 23-5</a> ).<br><b>Note:</b> This field can be written in Initialization mode only. It is read-only in Normal or Sleep mode.   |
| BF    | Bypass filter<br>0 No interrupt if identifier does not match any filter.<br>1 An RX interrupt is generated on identifier not matching any filter.<br><b>Note:</b> <ul style="list-style-type: none"> <li>If no filter is activated, this bit is reserved and always reads 1.</li> <li>This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode.</li> </ul>   |
| SFTM  | Self Test Mode<br>This bit controls the Self Test mode. For more details, see <a href="#">Section 23.6.2, Self Test mode</a> .<br>0 Self Test mode disable.<br>1 Self Test mode enable.<br><b>Note:</b> This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode.  |
| LBKM  | Loop Back Mode<br>This bit controls the Loop Back mode. For more details see <a href="#">Section 23.6.1, Loop Back mode</a> .<br>0 Loop Back mode disable.<br>1 Loop Back mode enable.<br><b>Note:</b> This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode.   |
| MME   | Master Mode Enable<br>0 Slave mode enable.<br>1 Master mode enable.<br><b>Note:</b> This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode.  |
| SBDT  | Slave Mode Break Detection Threshold<br>0 11-bit break.<br>1 10-bit break.<br><b>Note:</b> This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode.   |
| RBLM  | Receive Buffer Locked Mode<br>0 Receive Buffer not locked on overrun. Once the Slave Receive Buffer is full the next incoming message overwrites the previous one.<br>1 Receive Buffer locked against overrun. Once the Receive Buffer is full the next incoming message is discarded.<br><b>Note:</b> This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode.   |
| SLEEP | Sleep Mode Request<br>This bit is set by software to request LINFlex to enter Sleep mode.<br>This bit is cleared by software to exit Sleep mode or by hardware if the AWUM bit in LINCR1 and the WUF bit in LINSR are set (see <a href="#">Table 23-6</a> ).   |
| INIT  | Initialization Request<br>The software sets this bit to switch hardware into Initialization mode. If the SLEEP bit is reset, LINFlex enters Normal mode when clearing the INIT bit (see <a href="#">Table 23-6</a> ).  |

Table 23-4. Checksum bits configuration

| CFD | CCD | LINCFR     | Checksum sent                        |
|-----|-----|------------|--------------------------------------|
| 1   | 1   | Read/Write | None                                 |
| 1   | 0   | Read-only  | None                                 |
| 0   | 1   | Read/Write | Programmed in LINCFR by bits CF[0:7] |
| 0   | 0   | Read-only  | Hardware calculated                  |

Table 23-5. LIN master break length selection

| MBL  | Length |
|------|--------|
| 0000 | 10-bit |
| 0001 | 11-bit |
| 0010 | 12-bit |
| 0011 | 13-bit |
| 0100 | 14-bit |
| 0101 | 15-bit |
| 0110 | 16-bit |
| 0111 | 17-bit |
| 1000 | 18-bit |
| 1001 | 19-bit |
| 1010 | 20-bit |
| 1011 | 21-bit |
| 1100 | 22-bit |
| 1101 | 23-bit |
| 1110 | 36-bit |
| 1111 | 50-bit |

Table 23-6. Operating mode selection

| SLEEP | INIT | Operating mode      |
|-------|------|---------------------|
| 1     | 0    | Sleep (reset value) |
| x     | 1    | Initialization      |
| 0     | 0    | Normal              |

### 23.7.1.2 LIN interrupt enable register (LINIER)

Offset: 0x0004

Access: User read/write

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       |      |      |      |      |      |    |    |      |      |      |      |       |       |      |      |      |
|-------|------|------|------|------|------|----|----|------|------|------|------|-------|-------|------|------|------|
|       | 16   | 17   | 18   | 19   | 20   | 21 | 22 | 23   | 24   | 25   | 26   | 27    | 28    | 29   | 30   | 31   |
| R     | SZIE | OCIE | BEIE | CEIE | HEIE | 0  | 0  | FEIE | BOIE | LSIE | WUIE | DBFIE | DBEIE | DRIE | DTIE | HRIE |
| W     |      |      |      |      |      |    |    |      |      |      |      |       |       |      |      |      |
| Reset | 0    | 0    | 0    | 0    | 0    | 0  | 0  | 0    | 0    | 0    | 0    | 0     | 0     | 0    | 0    | 0    |

**Figure 23-7. LIN interrupt enable register (LINIER)**

**Table 23-7. LINIER field descriptions**

| Field | Description   |
|-------|---|
| SZIE  | Stuck at Zero Interrupt Enable<br>0 No interrupt when SZF bit in LINESR or UARTSR is set.<br>1 Interrupt generated when SZF bit in LINESR or UARTSR is set.   |
| OCIE  | Output Compare Interrupt Enable<br>0 No interrupt when OCF bit in LINESR or UARTSR is set.<br>1 Interrupt generated when OCF bit in LINESR or UARTSR is set.  |
| BEIE  | Bit Error Interrupt Enable<br>0 No interrupt when BEF bit in LINESR is set.<br>1 Interrupt generated when BEF bit in LINESR is set.   |
| CEIE  | Checksum Error Interrupt Enable<br>0 No interrupt on Checksum error.<br>1 Interrupt generated when checksum error flag (CEF) in LINESR is set.  |
| HEIE  | Header Error Interrupt Enable<br>0 No interrupt on Break Delimiter error, Synch Field error, Identifier field error.<br>1 Interrupt generated on Break Delimiter error, Synch Field error, Identifier field error.                                      |
| FEIE  | Framing Error Interrupt Enable<br>0 No interrupt on Framing error.<br>1 Interrupt generated on Framing error.   |
| BOIE  | Buffer Overrun Interrupt Enable<br>0 No interrupt on Buffer overrun.<br>1 Interrupt generated on Buffer overrun.  |
| LSIE  | LIN State Interrupt Enable<br>0 No interrupt on LIN state change.<br>1 Interrupt generated on LIN state change.<br>This interrupt can be used for debugging purposes. It has no status flag but is reset when writing 1111 into LINS[0:3] in the LINSR. |



Table 23-7. LINIER field descriptions (continued)

| Field | Description   |
|-------|---|
| WUIE  | Wake-up Interrupt Enable<br>0 No interrupt when WUF bit in LINSR or UARTSR is set.<br>1 Interrupt generated when WUF bit in LINSR or UARTSR is set.   |
| DBFIE | Data Buffer Full Interrupt Enable<br>0 No interrupt when buffer data register is full.<br>1 Interrupt generated when data buffer register is full.  |
| DBEIE | Data Buffer Empty Interrupt Enable<br>0 No interrupt when buffer data register is empty.<br>1 Interrupt generated when data buffer register is empty.   |
| DRIE  | Data Reception Complete Interrupt Enable<br>0 No interrupt when data reception is completed.<br>1 Interrupt generated when data received flag (DRF) in LINSR or UARTSR is set.                      |
| DTIE  | Data Transmitted Interrupt Enable<br>0 No interrupt when data transmission is completed.<br>1 Interrupt generated when data transmitted flag (DTF) is set in LINSR or UARTSR.                       |
| HRIE  | Header Received Interrupt Enable<br>0 No interrupt when a valid LIN header has been received.<br>1 Interrupt generated when a valid LIN header has been received, that is, HRF bit in LINSR is set. |

### 23.7.1.3 LIN status register (LINSR)

Offset: 0x0008

Access: User read/write

|       |      |    |    |    |    |    |     |    |      |     |     |      |      |     |     |     |
|-------|------|----|----|----|----|----|-----|----|------|-----|-----|------|------|-----|-----|-----|
|       | 0    | 1  | 2  | 3  | 4  | 5  | 6   | 7  | 8    | 9   | 10  | 11   | 12   | 13  | 14  | 15  |
| R     | 0    | 0  | 0  | 0  | 0  | 0  | 0   | 0  | 0    | 0   | 0   | 0    | 0    | 0   | 0   | 0   |
| W     |      |    |    |    |    |    |     |    |      |     |     |      |      |     |     |     |
| Reset | 0    | 0  | 0  | 0  | 0  | 0  | 0   | 0  | 0    | 0   | 0   | 0    | 0    | 0   | 0   | 0   |
|       | 16   | 17 | 18 | 19 | 20 | 21 | 22  | 23 | 24   | 25  | 26  | 27   | 28   | 29  | 30  | 31  |
| R     | LINS |    |    |    | 0  | 0  | RMB | 0  | RBSY | RPS | WUF | DBFF | DBEF | DRF | DTF | HRF |
| W     | w1c  |    |    |    |    |    | w1c |    | w1c  |     | w1c | w1c  | w1c  | w1c | w1c | w1c |
| Reset | 0    | 0  | 0  | 0  | 0  | 0  | 0   | 0  | 0    | 1   | 0   | 0    | 0    | 0   | 0   | 0   |

Figure 23-8. LIN status register (LINSR)

Table 23-8. LINSR field descriptions

| Field | Description   |
|-------|---|
| LINS  | <p>LIN modes / normal mode states</p> <p><b>0000: Sleep mode</b><br/>LINFlex is in Sleep mode to save power consumption.</p> <p><b>0001: Initialization mode</b><br/>LINFlex is in Initialization mode.</p> <p>Normal mode states</p> <p><b>0010: Idle</b><br/>This state is entered on several events:</p> <ul style="list-style-type: none"> <li>• SLEEP bit and INIT bit in LINCR1 have been cleared by software,</li> <li>• A falling edge has been received on RX pin and AWUM bit is set,</li> <li>• The previous frame reception or transmission has been completed or aborted.</li> </ul> <p><b>0011: Break</b><br/>In Slave mode, a falling edge followed by a dominant state has been detected. Receiving Break.<br/><b>Note:</b> In Slave mode, in case of error new LIN state can be either Idle or Break depending on last bit state. If last bit is dominant new LIN state is Break, otherwise Idle.<br/>In Master mode, Break transmission ongoing.</p> <p><b>0100: Break Delimiter</b><br/>In Slave mode, a valid Break has been detected. See <a href="#">Section 23.7.1.1, LIN control register 1 (LINCR1)</a> for break length configuration (10-bit or 11-bit). Waiting for a rising edge.<br/>In Master mode, Break transmission has been completed. Break Delimiter transmission is ongoing.</p> <p><b>0101: Synch Field</b><br/>In Slave mode, a valid Break Delimiter has been detected (recessive state for at least one bit time). Receiving Synch Field.<br/>In Master mode, Synch Field transmission is ongoing.</p> <p><b>0110: Identifier Field</b><br/>In Slave mode, a valid Synch Field has been received. Receiving Identifier Field.<br/>In Master mode, identifier transmission is ongoing.</p> <p><b>0111: Header reception/transmission completed</b><br/>In Slave mode, a valid header has been received and identifier field is available in the BIDR.<br/>In Master mode, header transmission is completed.</p> <p><b>1000: Data reception/transmission</b><br/>Response reception/transmission is ongoing.</p> <p><b>1001: Checksum</b><br/>Data reception/transmission completed. Checksum reception/transmission ongoing.<br/>In UART mode, only the following states are flagged by the LIN state bits:</p> <ul style="list-style-type: none"> <li>• Init</li> <li>• Sleep</li> <li>• Idle</li> <li>• Data transmission/reception</li> </ul> |
| RMB   | <p>Release Message Buffer</p> <p>0 Buffer is free.<br/>1 Buffer ready to be read by software. This bit must be cleared by software after reading data received in the buffer.<br/>This bit is cleared by hardware in Initialization mode.</p>   |
| RBSY  | <p>Receiver Busy Flag</p> <p>0 Receiver is idle<br/>1 Reception ongoing</p> <p><b>Note:</b> In Slave mode, after header reception, if BIDR[DIR] = 0 and reception starts then this bit is set. In this case, user cannot program LINCR2[DTRQ] = 1.</p>  |

Table 23-8. LINSR field descriptions (continued)

| Field | Description   |
|-------|---|
| RPS   | LIN receive pin state<br>This bit reflects the current status of LINRX pin for diagnostic purposes.   |
| WUF   | Wake-up Flag<br>This bit is set by hardware and indicates to the software that LINFlex has detected a falling edge on the LINRX pin when: <ul style="list-style-type: none"> <li>• Slave is in Sleep mode</li> <li>• Master is in Sleep mode or idle state</li> </ul> This bit must be cleared by software. It is reset by hardware in Initialization mode. An interrupt is generated if WUIE bit in LINIER is set.   |
| DBFF  | Data Buffer Full Flag<br>This bit is set by hardware and indicates the buffer is full. It is set only when receiving extended frames (DFL > 7).<br>This bit must be cleared by software.<br>It is reset by hardware in Initialization mode.   |
| DBEF  | Data Buffer Empty Flag<br>This bit is set by hardware and indicates the buffer is empty. It is set only when transmitting extended frames (DFL > 7).<br>This bit must be cleared by software, once buffer has been filled again, in order to start transmission.<br>This bit is reset by hardware in Initialization mode.   |
| DRF   | Data Reception Completed Flag<br>This bit is set by hardware and indicates the data reception is completed.<br>This bit must be cleared by software.<br>It is reset by hardware in Initialization mode.<br><b>Note:</b> This flag is not set in case of bit error or framing error.   |
| DTF   | Data Transmission Completed Flag<br>This bit is set by hardware and indicates the data transmission is completed.<br>This bit must be cleared by software.<br>It is reset by hardware in Initialization mode.<br><b>Note:</b> This flag is not set in case of bit error if IOBE bit is reset.   |
| HRF   | Header Reception Flag<br>This bit is set by hardware and indicates a valid header reception is completed.<br>This bit must be cleared by software.<br>This bit is reset by hardware in Initialization mode and at end of completed or aborted frame.<br><b>Note:</b> If filters are enabled, this bit is set only when identifier software filtering is required, that is to say: <ul style="list-style-type: none"> <li>• All filters are inactive and BF bit in LINCR1 is set</li> <li>• No match in any filter and BF bit in LINCR1 is set</li> <li>• TX filter match</li> </ul> |

## 23.7.1.4 LIN error status register (LINESR)

Offset: 0x000C

Access: User read/write

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       |     |     |     |     |      |      |       |     |     |    |    |    |    |    |    |     |
|-------|-----|-----|-----|-----|------|------|-------|-----|-----|----|----|----|----|----|----|-----|
|       | 16  | 17  | 18  | 19  | 20   | 21   | 22    | 23  | 24  | 25 | 26 | 27 | 28 | 29 | 30 | 31  |
| R     | SZF | OCF | BEF | CEF | SFEF | BDEF | IDPEF | FEF | BOF | 0  | 0  | 0  | 0  | 0  | 0  | NF  |
| W     | w1c | w1c | w1c | w1c | w1c  | w1c  | w1c   | w1c | w1c |    |    |    |    |    |    | w1c |
| Reset | 0   | 0   | 0   | 0   | 0    | 0    | 0     | 0   | 0   | 0  | 0  | 0  | 0  | 0  | 0  | 0   |

Figure 23-9. LIN error status register (LINESR)

Table 23-9. LINESR field descriptions

| Field | Description  |
|-------|--|
| SZF   | Stuck at Zero Flag<br>This bit is set by hardware when the bus is dominant for more than a 100-bit time. If the dominant state continues, SZF flag is set again after 87-bit time. It is cleared by software.  |
| OCF   | Output Compare Flag<br>0 No output compare event occurred<br>1 The content of the counter has matched the content of OC1[0:7] or OC2[0:7] in LINOCCR. If this bit is set and IOT bit in LINTCSR is set, LINFlex moves to Idle state.<br>If LTOM bit in LINTCSR is set, then OCF is cleared by hardware in Initialization mode. If LTOM bit is cleared, then OCF maintains its status whatever the mode is. |
| BEF   | Bit Error Flag<br>This bit is set by hardware and indicates to the software that LINFlex has detected a bit error. This error can occur during response field transmission (Slave and Master modes) or during header transmission (in Master mode).<br>This bit is cleared by software.  |
| CEF   | Checksum Error Flag<br>This bit is set by hardware and indicates that the received checksum does not match the hardware calculated checksum.<br>This bit is cleared by software.<br><b>Note:</b> This bit is never set if CCD or CFD bit in LINCR1 is set.   |
| SFEF  | Synch Field Error Flag<br>This bit is set by hardware and indicates that a Synch Field error occurred (inconsistent Synch Field).  |
| BDEF  | Break Delimiter Error Flag<br>This bit is set by hardware and indicates that the received Break Delimiter is too short (less than one bit time).   |
| IDPEF | Identifier Parity Error Flag<br>This bit is set by hardware and indicates that a Identifier Parity error occurred.<br><b>Note:</b> Header interrupt is triggered when SFEF or BDEF or IDPEF bit is set and HEIE bit in LINIER is set.  |

Table 23-9. LINESR field descriptions (continued)

| Field | Description  |
|-------|--|
| FEF   | Framing Error Flag<br>This bit is set by hardware and indicates to the software that LINFlex has detected a framing error (invalid stop bit). This error can occur during reception of any data in the response field (Master or Slave mode) or during reception of Synch Field or Identifier Field in Slave mode. |
| BOF   | Buffer Overrun Flag<br>This bit is set by hardware when a new data byte is received and the buffer full flag is not cleared. If RBLM in LINCR1 is set then the new byte received is discarded. If RBLM is reset then the new byte overwrites the buffer. It can be cleared by software.                            |
| NF    | Noise Flag<br>This bit is set by hardware when noise is detected on a received character. This bit is cleared by software.   |

### 23.7.1.5 UART mode control register (UARTCR)

Offset: 0x0010

Access: User read/write

|       |    |    |      |    |    |      |    |    |    |    |      |      |    |     |    |      |
|-------|----|----|------|----|----|------|----|----|----|----|------|------|----|-----|----|------|
|       | 0  | 1  | 2    | 3  | 4  | 5    | 6  | 7  | 8  | 9  | 10   | 11   | 12 | 13  | 14 | 15   |
| R     | 0  | 0  | 0    | 0  | 0  | 0    | 0  | 0  | 0  | 0  | 0    | 0    | 0  | 0   | 0  | 0    |
| W     |    |    |      |    |    |      |    |    |    |    |      |      |    |     |    |      |
| Reset | 0  | 0  | 0    | 0  | 0  | 0    | 0  | 0  | 0  | 0  | 0    | 0    | 0  | 0   | 0  | 0    |
|       | 16 | 17 | 18   | 19 | 20 | 21   | 22 | 23 | 24 | 25 | 26   | 27   | 28 | 29  | 30 | 31   |
| R     | 0  |    |      | 0  |    |      | 0  | 0  | 0  | 0  |      |      |    |     |    |      |
| W     |    |    | TDFL |    |    | RDFL |    |    |    |    | RXEN | TXEN | OP | PCE | WL | UART |
| Reset | 0  | 0  | 0    | 0  | 0  | 0    | 0  | 0  | 0  | 0  | 0    | 0    | 0  | 0   | 0  | 0    |

Figure 23-10. UART mode control register (UARTCR)

Table 23-10. UARTCR field descriptions

| Field | Description  |
|-------|--|
| TDFL  | Transmitter Data Field length<br>This field sets the number of bytes to be transmitted in UART mode. It can be programmed only when the UART bit is set. TDFL[0:1] = Transmit buffer size – 1.<br>00 Transmit buffer size = 1.<br>01 Transmit buffer size = 2.<br>10 Transmit buffer size = 3.<br>11 Transmit buffer size = 4. |
| RDFL  | Receiver Data Field length<br>This field sets the number of bytes to be received in UART mode. It can be programmed only when the UART bit is set. RDFL[0:1] = Receive buffer size – 1.<br>00 Receive buffer size = 1.<br>01 Receive buffer size = 2.<br>10 Receive buffer size = 3.<br>11 Receive buffer size = 4.            |

**Table 23-10. UARTCR field descriptions (continued)**

| Field | Description  |
|-------|--|
| RXEN  | Receiver Enable<br>0 Receiver disable.<br>1 Receiver enable.<br>This bit can be programmed only when the UART bit is set.  |
| TXEN  | Transmitter Enable<br>0 Transmitter disable.<br>1 Transmitter enable.<br>This bit can be programmed only when the UART bit is set.<br><b>Note:</b> Transmission starts when this bit is set and when writing DATA0 in the BDRL register. |
| OP    | Odd Parity<br>0 Sent parity is even.<br>1 Sent parity is odd.<br>This bit can be programmed in Initialization mode only when the UART bit is set.  |
| PCE   | Parity Control Enable<br>0 Parity transmit/check disable.<br>1 Parity transmit/check enable.<br>This bit can be programmed in Initialization mode only when the UART bit is set.   |
| WL    | Word Length in UART mode<br>0 7-bit data + parity bit.<br>1 8-bit data (or 9-bit if PCE is set).<br>This bit can be programmed in Initialization mode only when the UART bit is set.   |
| UART  | UART mode enable<br>0 LIN mode.<br>1 UART mode.<br>This bit can be programmed in Initialization mode only.   |

### 23.7.1.6 UART mode status register (UARTSR)

Offset: 0x0014

Access: User read/write

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       |     |     |     |     |     |     |     |     |     |     |     |    |    |     |     |     |
|-------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|----|----|-----|-----|-----|
|       | 16  | 17  | 18  | 19  | 20  | 21  | 22  | 23  | 24  | 25  | 26  | 27 | 28 | 29  | 30  | 31  |
| R     | SZF | OCF | PE3 | PE2 | PE1 | PE0 | RMB | FEF | BOF | RPS | WUF | 0  | 0  | DRF | DTF | NF  |
| W     | w1c | w1c | w1c | w1c | w1c | w1c | w1c | w1c | w1c | w1c | w1c |    |    | w1c | w1c | w1c |
| Reset | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0  | 0  | 0   | 0   | 0   |

**Figure 23-11. UART mode status register (UARTSR)**

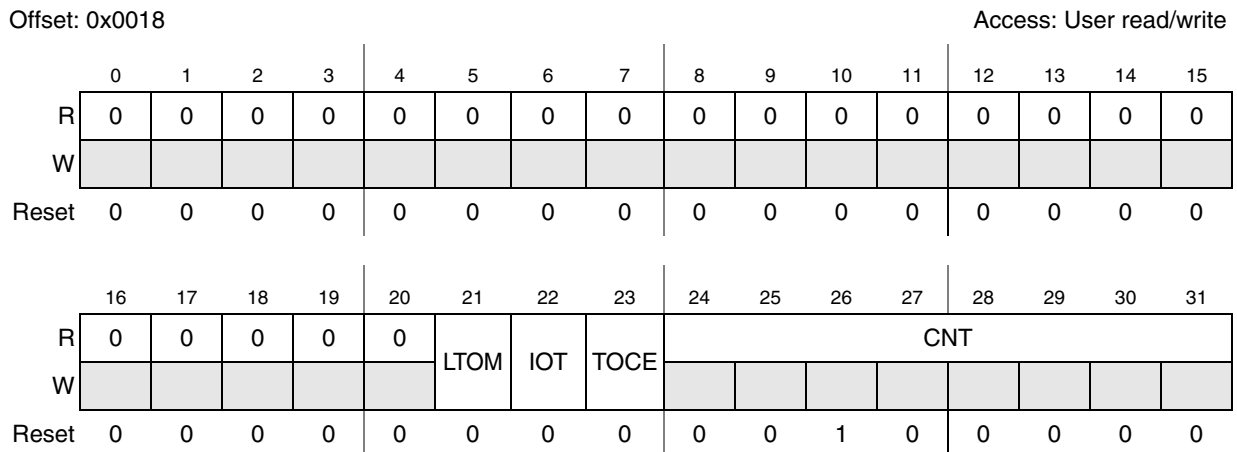
Table 23-11. UARTSR field descriptions

| Field | Description  |
|-------|--|
| SZF   | Stuck at Zero Flag<br>This bit is set by hardware when the bus is dominant for more than a 100-bit time. It is cleared by software.  |
| OCF   | OCF Output Compare Flag<br>0 No output compare event occurred.<br>1 The content of the counter has matched the content of OC1[0:7] or OC2[0:7] in LINOCCR.<br>An interrupt is generated if the OCIE bit in LINIER register is set.   |
| PE3   | Parity Error Flag Rx3<br>This bit indicates if there is a parity error in the corresponding received byte (Rx3). See <a href="#">Section 23.8.1.1, Buffer in UART mode</a> . No interrupt is generated if this error occurs.<br>0 No parity error.<br>1 Parity error.                                  |
| PE2   | Parity Error Flag Rx2<br>This bit indicates if there is a parity error in the corresponding received byte (Rx2). See <a href="#">Section 23.8.1.1, Buffer in UART mode</a> . No interrupt is generated if this error occurs.<br>0 No parity error.<br>1 Parity error.                                  |
| PE1   | Parity Error Flag Rx1<br>This bit indicates if there is a parity error in the corresponding received byte (Rx1). See <a href="#">Section 23.8.1.1, Buffer in UART mode</a> . No interrupt is generated if this error occurs.<br>0 No parity error.<br>1 Parity error.                                  |
| PE0   | Parity Error Flag Rx0<br>This bit indicates if there is a parity error in the corresponding received byte (Rx0). See <a href="#">Section 23.8.1.1, Buffer in UART mode</a> . No interrupt is generated if this error occurs.<br>0 No parity error.<br>1 Parity error.                                  |
| RMB   | Release Message Buffer<br>0 Buffer is free.<br>1 Buffer ready to be read by software. This bit must be cleared by software after reading data received in the buffer.<br>This bit is cleared by hardware in Initialization mode.   |
| FEF   | Framing Error Flag<br>This bit is set by hardware and indicates to the software that LINFlex has detected a framing error (invalid stop bit).  |
| BOF   | Buffer Overrun Flag<br>This bit is set by hardware when a new data byte is received and the buffer full flag is not cleared. If RBLM in LINCR1 is set then the new byte received is discarded. If RBLM is reset then the new byte overwrites buffer. it can be cleared by software.                    |
| RPS   | LIN Receive Pin State<br>This bit reflects the current status of LINRX pin for diagnostic purposes.  |
| WUF   | Wake-up Flag<br>This bit is set by hardware and indicates to the software that LINFlex has detected a falling edge on the LINRX pin in Sleep mode.<br>This bit must be cleared by software. It is reset by hardware in Initialization mode.<br>An interrupt is generated if WUIE bit in LINIER is set. |

**Table 23-11. UARTSR field descriptions (continued)**

| Field | Description   |
|-------|---|
| DRF   | Data Reception Completed Flag<br>This bit is set by hardware and indicates the data reception is completed, that is, the number of bytes programmed in RDFL[0:1] in UARTCR have been received.<br>This bit must be cleared by software.<br>It is reset by hardware in Initialization mode.<br>An interrupt is generated if DRIE bit in LINIER is set.<br><b>Note:</b> In UART mode, this flag is set in case of framing error, parity error or overrun. |
| DTF   | Data Transmission Completed Flag<br>This bit is set by hardware and indicates the data transmission is completed, that is, the number of bytes programmed in TDFL[0:1] have been transmitted.<br>This bit must be cleared by software.<br>It is reset by hardware in Initialization mode.<br>An interrupt is generated if DTIE bit in LINIER is set.  |
| NF    | Noise Flag<br>This bit is set by hardware when noise is detected on a received character. This bit is cleared by software.  |

**23.7.1.7 LIN timeout control status register (LINTCSR)**



**Figure 23-12. LIN timeout control status register (LINTCSR)**

**Table 23-12. LINTCSR field descriptions**

| Field | Description   |
|-------|---|
| LTOM  | LIN timeout mode<br>0 LIN timeout mode (header, response and frame timeout detection).<br>1 Output compare mode.<br>This bit can be set/cleared in Initialization mode only.                  |
| IOT   | Idle on Timeout<br>0 LIN state machine not reset to Idle on timeout event.<br>1 LIN state machine reset to Idle on timeout event.<br>This bit can be set/cleared in Initialization mode only. |



Table 23-12. LINTCSR field descriptions (continued)

| Field | Description   |
|-------|---|
| TOCE  | Timeout counter enable<br>0 Timeout counter disable. OCF bit in LINESR or UARTSR is not set on an output compare event.<br>1 Timeout counter enable. OCF bit is set if an output compare event occurs.<br>TOCE bit is configurable by software in Initialization mode. If LIN state is not Init and if timer is in LIN timeout mode, then hardware takes control of TOCE bit. |
| CNT   | Counter Value<br>This field indicates the LIN timeout counter value.  |

### 23.7.1.8 LIN output compare register (LINOCR)

Offset: 0x001C

Access: User read/write

|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       | 16               | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24               | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|-------|------------------|----|----|----|----|----|----|----|------------------|----|----|----|----|----|----|----|
| R     | OC2 <sup>1</sup> |    |    |    |    |    |    |    | OC1 <sup>1</sup> |    |    |    |    |    |    |    |
| W     |                  |    |    |    |    |    |    |    |                  |    |    |    |    |    |    |    |
| Reset | 1                | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1                | 1  | 1  | 1  | 1  | 1  | 1  | 1  |

<sup>1</sup> If LINTCSR[LTOM] = 0, this field is read-only.

Figure 23-13. LIN output compare register (LINOCR)

Table 23-13. LINOCR field descriptions

| Field | Description   |
|-------|---|
| OC2   | Output compare 2 value<br>These bits contain the value to be compared to the value of bits CNT[0:7] in LINTCSR. |
| OC1   | Output compare 1 value<br>These bits contain the value to be compared to the value of bits CNT[0:7] in LINTCSR. |

### 23.7.1.9 LIN timeout control register (LINTOCR)

Offset: 0x0020 Access: User read/write

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       |    |    |    |    |     |    |    |    |    |     |    |    |    |    |    |    |  |
|-------|----|----|----|----|-----|----|----|----|----|-----|----|----|----|----|----|----|--|
|       | 16 | 17 | 18 | 19 | 20  | 21 | 22 | 23 | 24 | 25  | 26 | 27 | 28 | 29 | 30 | 31 |  |
| R     | 0  | 0  | 0  | 0  | RTO |    |    |    | 0  | HTO |    |    |    |    |    |    |  |
| W     |    |    |    |    |     |    |    |    |    |     |    |    |    |    |    |    |  |
| Reset | 0  | 0  | 0  | 0  | 1   | 1  | 1  | 0  | 0  | 0   | 1  | 0  | 1  | 1  | 0  | 0  |  |

Figure 23-14. LIN timeout control register (LINTOCR)

Table 23-14. LINTOCR field descriptions

| Field | Description   |
|-------|---|
| RTO   | Response timeout value<br>This field contains the response timeout duration (in bit time) for 1 byte.<br>The reset value is 0xE = 14, corresponding to $T_{Response\_Maximum} = 1.4 \times T_{Response\_Nominal}$   |
| HTO   | Header timeout value<br>This field contains the header timeout duration (in bit time). This value does not include the Break and the Break Delimiter. The reset value is the 0x2C = 44, corresponding to $T_{Header\_Maximum}$ .<br>Programming LINSR[MME] = 1 changes the HTO value to 0x1C = 28.<br>This field can be written only in Slave mode. |

### 23.7.1.10 LIN fractional baud rate register (LINFBR)

Offset: 0x0024 Access: User read/write

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       |    |    |    |    |    |    |    |    |    |    |    |    |       |    |    |    |
|-------|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|----|----|
|       | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28    | 29 | 30 | 31 |
| R     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | DIV_F |    |    |    |
| W     |    |    |    |    |    |    |    |    |    |    |    |    |       |    |    |    |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0     | 0  | 0  | 0  |

Figure 23-15. LIN fractional baud rate register (LINFBR)

Table 23-15. LINFBR field descriptions

| Field | Description   |
|-------|---|
| DIV_F | Fraction bits of LFDIV<br>The 4 fraction bits define the value of the fraction of the LINFlex divider (LFDIV).<br>Fraction (LFDIV) = Decimal value of DIV_F / 16.<br>This field can be written in Initialization mode only. |

### 23.7.1.11 LIN integer baud rate register (LINIBRR)

Offset: 0x0028

Access: User read/write

|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       | 16 | 17 | 18 | 19    | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|-------|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|
| R     | 0  | 0  | 0  | DIV_M |    |    |    |    |    |    |    |    |    |    |    |    |
| W     |    |    |    |       |    |    |    |    |    |    |    |    |    |    |    |    |
| Reset | 0  | 0  | 0  | 0     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |

Figure 23-16. LIN integer baud rate register (LINIBRR)

Table 23-16. LINIBRR field descriptions

| Field | Description  |
|-------|--|
| DIV_M | LFDIV mantissa<br>This field defines the LINFlex divider (LFDIV) mantissa value (see <a href="#">Table 23-17</a> ). This field can be written in Initialization mode only. |

Table 23-17. Integer baud rate selection

| DIV_M[0:12] | Mantissa           |
|-------------|--------------------|
| 0x0000      | LIN clock disabled |
| 0x0001      | 1                  |
| ...         | ...                |
| 0x1FFE      | 8190               |
| 0x1FFF      | 8191               |

### 23.7.1.12 LIN checksum field register (LINCFR)

Offset: 0x002C Access: User read/write

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|       | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| R     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | CF |    |    |    |    |    |    |    |
| W     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |

Figure 23-17. LIN checksum field register (LINCFR)

Table 23-18. LINCFR field descriptions

| Field | Description  |
|-------|--|
| CF    | Checksum bits<br>When LINCR1[CCD] = 0, this field is read-only. When LINCR1[CCD] = 1, this field is read/write. See <a href="#">Table 23-4</a> . |

### 23.7.1.13 LIN control register 2 (LINCR2)

Offset: 0x0030 Access: User read/write

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       |    |      |      |    |      |      |      |      |      |    |    |    |    |    |    |    |
|-------|----|------|------|----|------|------|------|------|------|----|----|----|----|----|----|----|
|       | 16 | 17   | 18   | 19 | 20   | 21   | 22   | 23   | 24   | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| R     | 0  |      |      | 0  | 0    | 0    | 0    | 0    | 0    | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |    | IOBE | IOPE |    | WURQ | DDRQ | DTRQ | ABRQ | HTRQ |    |    |    |    |    |    |    |
| Reset | 0  | 1    | 1    | 0  | 0    | 0    | 0    | 0    | 0    | 0  | 0  | 0  | 0  | 0  | 0  | 0  |

Figure 23-18. LIN control register 2 (LINCR2)

Table 23-19. LINCR2 field descriptions

| Field | Description  |
|-------|--|
| IOBE  | Idle on Bit Error<br>0 Bit error does not reset LIN state machine.<br>1 Bit error reset LIN state machine.<br>This bit can be set/cleared in Initialization mode only. |

Table 23-19. LINC2 field descriptions (continued)

| Field | Description  |
|-------|--|
| IOPE  | Idle on Identifier Parity Error<br>0 Identifier Parity error does not reset LIN state machine.<br>1 Identifier Parity error reset LIN state machine.<br>This bit can be set/cleared in Initialization mode only.   |
| WURQ  | Wake-up Generation Request<br>Setting this bit generates a wake-up pulse. It is reset by hardware when the wake-up character has been transmitted. The character sent is copied from DATA0 in BDRL buffer. Note that this bit cannot be set in Sleep mode. Software has to exit Sleep mode before requesting a wake-up. Bit error is not checked when transmitting the wake-up request.                      |
| DDRQ  | Data Discard Request<br>Set by software to stop data reception if the frame does not concern the node. This bit is reset by hardware once LINFlex has moved to idle state. In Slave mode, this bit can be set only when HRF bit in LINSR is set and identifier did not match any filter.   |
| DTRQ  | Data Transmission Request<br>Set by software in Slave mode to request the transmission of the LIN Data field stored in the Buffer data register. This bit can be set only when HRF bit in LINSR is set.<br>Cleared by hardware when the request has been completed or aborted or on an error condition. In Master mode, this bit is set by hardware when BIDR[DIR] = 1 and header transmission is completed. |
| ABRQ  | Abort Request<br>Set by software to abort the current transmission.<br>Cleared by hardware when the transmission has been aborted. LINFlex aborts the transmission at the end of the current bit.<br>This bit can also abort a wake-up request.<br>It can also be used in UART mode.   |
| HTRQ  | Header Transmission Request<br>Set by software to request the transmission of the LIN header.<br>Cleared by hardware when the request has been completed or aborted.<br>This bit has no effect in UART mode.   |

### 23.7.1.14 Buffer identifier register (BIDR)

Offset: 0x0034

Access: User read/write

|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

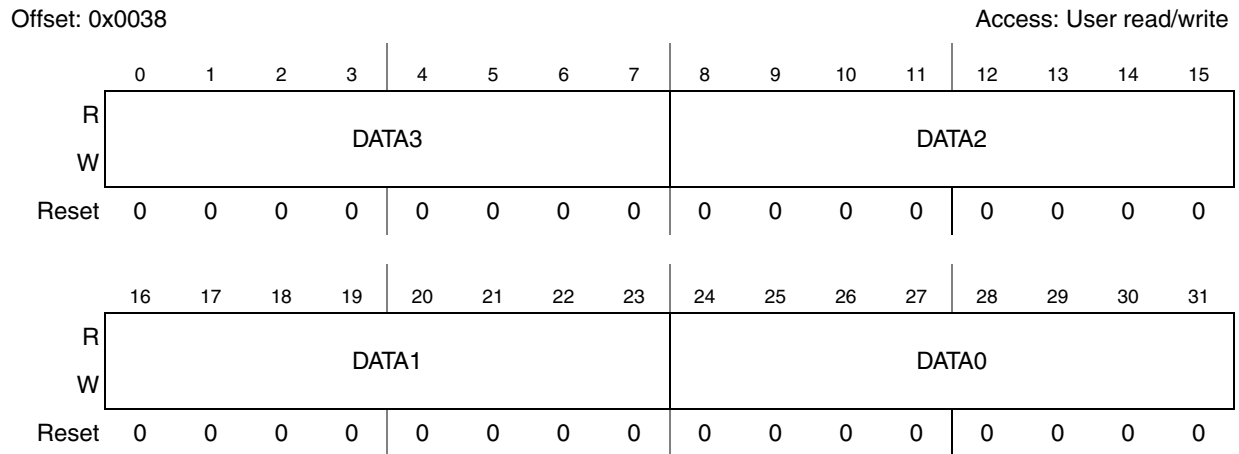
|       | 16  | 17 | 18 | 19 | 20  | 21  | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|-------|-----|----|----|----|-----|-----|----|----|----|----|----|----|----|----|----|----|
| R     | DFL |    |    |    | DIR | CCS | 0  | 0  | ID |    |    |    |    |    |    |    |
| W     |     |    |    |    |     |     |    |    |    |    |    |    |    |    |    |    |
| Reset | 0   | 0  | 0  | 0  | 0   | 0   | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |

Figure 23-19. Buffer identifier register (BIDR)

**Table 23-20. BDR field descriptions**

| Field | Description   |
|-------|---|
| DFL   | Data Field Length<br>This field defines the number of data bytes in the response part of the frame.<br>DFL = Number of data bytes – 1.<br>Normally, LIN uses only DFL[2:0] to manage frames with a maximum of 8 bytes of data. Identifier filters are compatible with DFL[2:0] only. DFL[5:3] are provided to manage extended frames.   |
| DIR   | Direction<br>This bit controls the direction of the data field.<br>0 LINFlex receives the data and copies them in the BDR registers.<br>1 LINFlex transmits the data from the BDR registers.  |
| CCS   | Classic Checksum<br>This bit controls the type of checksum applied on the current message.<br>0 Enhanced Checksum covering Identifier and Data fields. This is compatible with LIN specification 2.0 and higher.<br>1 Classic Checksum covering Data fields only. This is compatible with LIN specification 1.3 and earlier.<br>In LIN slave mode (MME bit cleared in LINCR1), this bit must be configured before the header reception. If the slave has to manage frames with 2 types of checksum, filters must be configured. |
| ID    | Identifier<br>Identifier part of the identifier field without the identifier parity.  |

**23.7.1.15 Buffer data register LSB (BDRL)**



**Figure 23-20. Buffer data register LSB (BDRL)**

**Table 23-21. BDRL field descriptions**

| Field | Description                                   |
|-------|---|
| DATA3 | Data Byte 3<br>Data byte 3 of the data field. |
| DATA2 | Data Byte 2<br>Data byte 2 of the data field. |

Table 23-21. BDRL field descriptions (continued)

| Field | Description                                   |
|-------|---|
| DATA1 | Data Byte 1<br>Data byte 1 of the data field. |
| DATA0 | Data Byte 0<br>Data byte 0 of the data field. |

### 23.7.1.16 Buffer data register MSB (BDRM)

Offset: 0x003C

Access: User read/write

|       |       |    |    |    |    |    |    |    |       |    |    |    |    |    |    |    |
|-------|-------|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|
|       | 0     | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8     | 9  | 10 | 11 | 12 | 13 | 14 | 15 |
| R     | DATA7 |    |    |    |    |    |    |    | DATA6 |    |    |    |    |    |    |    |
| W     | DATA7 |    |    |    |    |    |    |    | DATA6 |    |    |    |    |    |    |    |
| Reset | 0     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0     | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
|       | 16    | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24    | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| R     | DATA5 |    |    |    |    |    |    |    | DATA4 |    |    |    |    |    |    |    |
| W     | DATA5 |    |    |    |    |    |    |    | DATA4 |    |    |    |    |    |    |    |
| Reset | 0     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0     | 0  | 0  | 0  | 0  | 0  | 0  | 0  |

Figure 23-21. Buffer data register MSB (BDRM)

Table 23-22. BDRM field descriptions

| Field | Description                                   |
|-------|---|
| DATA7 | Data Byte 7<br>Data byte 7 of the data field. |
| DATA6 | Data Byte 6<br>Data byte 6 of the data field. |
| DATA5 | Data Byte 5<br>Data byte 5 of the data field. |
| DATA4 | Data Byte 4<br>Data byte 4 of the data field. |

### 23.7.1.17 Identifier filter enable register (IFER)

Offset: 0x0040

Access: User read/write

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       |    |    |    |    |    |    |    |    |      |    |    |    |    |    |    |    |
|-------|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|
|       | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24   | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| R     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | FACT |    |    |    |    |    |    |    |
| W     |    |    |    |    |    |    |    |    |      |    |    |    |    |    |    |    |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0    | 0  | 0  | 0  | 0  | 0  | 0  | 0  |

Figure 23-22. Identifier filter enable register (IFER)

Table 23-23. IFER field descriptions

| Field | Description   |
|-------|---|
| FACT  | Filter activation (see Table 23-24)<br>0 Filters 2n and 2n + 1 are deactivated.<br>1 Filters 2n and 2n + 1 are activated.<br>This field can be set/cleared in Initialization mode only. |

Table 23-24. IFER[FACT] configuration

| Bit     | Value | Result                             |
|---------|-------|------------------------------------|
| FACT[0] | 0     | Filters 0 and 1 are deactivated.   |
|         | 1     | Filters 0 and 1 are activated.     |
| FACT[1] | 0     | Filters 2 and 3 are deactivated.   |
|         | 1     | Filters 2 and 3 are activated.     |
| FACT[2] | 0     | Filters 4 and 5 are deactivated.   |
|         | 1     | Filters 4 and 5 are activated.     |
| FACT[3] | 0     | Filters 6 and 7 are deactivated.   |
|         | 1     | Filters 6 and 7 are activated.     |
| FACT[4] | 0     | Filters 8 and 9 are deactivated.   |
|         | 1     | Filters 8 and 9 are activated.     |
| FACT[5] | 0     | Filters 10 and 11 are deactivated. |
|         | 1     | Filters 10 and 11 are activated.   |
| FACT[6] | 0     | Filters 12 and 13 are deactivated. |
|         | 1     | Filters 12 and 13 are activated.   |



Table 23-24. IFER[FACT] configuration (continued)

| Bit     | Value | Result                             |
|---------|-------|------------------------------------|
| FACT[7] | 0     | Filters 14 and 15 are deactivated. |
|         | 1     | Filters 14 and 15 are activated.   |

### 23.7.1.18 Identifier filter match index (IFMI)

Address: Base + 0x0044

Access: User read-only

|       |    |    |    |    |    |    |    |    |    |    |    |    |           |    |    |    |
|-------|----|----|----|----|----|----|----|----|----|----|----|----|-----------|----|----|----|
|       | 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12        | 13 | 14 | 15 |
| R     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0         | 0  | 0  | 0  |
| W     |    |    |    |    |    |    |    |    |    |    |    |    |           |    |    |    |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0         | 0  | 0  | 0  |
|       | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28        | 29 | 30 | 31 |
| R     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |    | IFMI[0:4] |    |    |    |
| W     |    |    |    |    |    |    |    |    |    |    |    |    |           |    |    |    |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0         | 0  | 0  | 0  |

Figure 23-23. Identifier filter match index (IFMI)

Table 23-25. IFMI field descriptions

| Field              | Description  |
|--------------------|--|
| 0:26               | Reserved   |
| IFMI[0:4]<br>27:31 | Filter match index<br>This register contains the index corresponding to the received identifier. It can be used to directly write or read the data in SRAM (see <a href="#">Section 23.8.2.2, Slave mode</a> for more details).<br>When no filter matches, IFMI[0:4] = 0. When Filter $n$ is matching, IFMI[0:4] = $n + 1$ . |

### 23.7.1.19 Identifier filter mode register (IFMR)

Offset: 0x0048

Access: User read/write

|       |    |    |    |    |    |    |    |    |     |    |    |    |    |    |    |    |
|-------|----|----|----|----|----|----|----|----|-----|----|----|----|----|----|----|----|
|       | 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8   | 9  | 10 | 11 | 12 | 13 | 14 | 15 |
| R     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |    |    |    |    |    |    |    |    |     |    |    |    |    |    |    |    |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
|       | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24  | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| R     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | IFM |    |    |    |    |    |    |    |
| W     |    |    |    |    |    |    |    |    |     |    |    |    |    |    |    |    |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 0  | 0  | 0  | 0  | 0  | 0  | 0  |

Figure 23-24. Identifier filter mode register (IFMR)

Table 23-26. IFMR field descriptions

| Field | Description   |
|-------|---|
| IFM   | Filter mode (see <a href="#">Table 23-27</a> ).<br>0 Filters $2n$ and $2n + 1$ are in identifier list mode.<br>1 Filters $2n$ and $2n + 1$ are in mask mode (filter $2n + 1$ is the mask for the filter $2n$ ). |

Table 23-27. IFMR[IFM] configuration

| Bit    | Value | Result  |
|--------|-------|---|
| IFM[0] | 0     | Filters 0 and 1 are in identifier list mode.                                  |
|        | 1     | Filters 0 and 1 are in mask mode (filter 1 is the mask for the filter 0).     |
| IFM[1] | 0     | Filters 2 and 3 are in identifier list mode.                                  |
|        | 1     | Filters 2 and 3 are in mask mode (filter 3 is the mask for the filter 2).     |
| IFM[2] | 0     | Filters 4 and 5 are in identifier list mode.                                  |
|        | 1     | Filters 4 and 5 are in mask mode (filter 5 is the mask for the filter 4).     |
| IFM[3] | 0     | Filters 6 and 7 are in identifier list mode.                                  |
|        | 1     | Filters 6 and 7 are in mask mode (filter 7 is the mask for the filter 6).     |
| IFM[4] | 0     | Filters 8 and 9 are in identifier list mode.                                  |
|        | 1     | Filters 8 and 9 are in mask mode (filter 9 is the mask for the filter 8).     |
| IFM[5] | 0     | Filters 10 and 11 are in identifier list mode.                                |
|        | 1     | Filters 10 and 11 are in mask mode (filter 11 is the mask for the filter 10). |
| IFM[6] | 0     | Filters 12 and 13 are in identifier list mode.                                |
|        | 1     | Filters 12 and 13 are in mask mode (filter 13 is the mask for the filter 12). |

Table 23-27. IFMR[IFM] configuration (continued)

| Bit    | Value | Result  |
|--------|-------|---|
| IFM[7] | 0     | Filters 14 and 15 are in identifier list mode.                                |
|        | 1     | Filters 14 and 15 are in mask mode (filter 15 is the mask for the filter 14). |

### 23.7.1.20 Identifier filter control register (IFCR2 $n$ )

Offsets: 0x004C–0x0084 (8 registers)

Access: User read/write

|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       | 16 | 17 | 18 | 19  | 20 | 21 | 22  | 23  | 24 | 25 | 26 | 27 | 28  | 29 | 30 | 31 |
|-------|----|----|----|-----|----|----|-----|-----|----|----|----|----|-----|----|----|----|
| R     | 0  | 0  | 0  |     |    |    |     |     | 0  | 0  |    |    | ID  |    |    |    |
| W     |    |    |    | DFL |    |    | DIR | CCS |    |    |    |    | w1c |    |    |    |
| Reset | 0  | 0  | 0  | 0   | 0  | 0  | 0   | 0   | 0  | 0  | 0  | 0  | 0   | 0  | 0  | 0  |

Figure 23-25. Identifier filter control register (IFCR2 $n$ )

#### NOTE

This register can be written in Initialization mode only.

Table 23-28. IFCR2 $n$  field descriptions

| Field | Description  |
|-------|--|
| DFL   | Data Field Length<br>This field defines the number of data bytes in the response part of the frame.  |
| DIR   | Direction<br>This bit controls the direction of the data field.<br>0 LINFlex receives the data and copies them in the BDRL and BDRM registers.<br>1 LINFlex transmits the data from the BDRL and BDRM registers.   |
| CCS   | Classic Checksum<br>This bit controls the type of checksum applied on the current message.<br>0 Enhanced Checksum covering Identifier and Data fields. This is compatible with LIN specification 2.0 and higher.<br>1 Classic Checksum covering Data fields only. This is compatible with LIN specification 1.3 and earlier. |
| ID    | Identifier<br>Identifier part of the identifier field without the identifier parity.   |

### 23.7.1.21 Identifier filter control register (IFCR2n + 1)

Offsets: 0x0050–0x0088 (8 registers) Access: User read/write

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       |    |    |    |    |     |    |    |     |     |    |    |    |    |    |    |    |
|-------|----|----|----|----|-----|----|----|-----|-----|----|----|----|----|----|----|----|
|       | 16 | 17 | 18 | 19 | 20  | 21 | 22 | 23  | 24  | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| R     | 0  | 0  | 0  |    | DFL |    |    | DIR | CCS | 0  | 0  | ID |    |    |    |    |
| W     |    |    |    |    |     |    |    |     |     |    |    |    |    |    |    |    |
| Reset | 0  | 0  | 0  | 0  | 0   | 0  | 0  | 0   | 0   | 0  | 0  | 0  | 0  | 0  | 0  | 0  |

**Figure 23-26. Identifier filter control register (IFCR2n + 1)**

**NOTE**

This register can be written in Initialization mode only.

**Table 23-29. IFCR2n + 1 field descriptions**

| Field | Description   |
|-------|---|
| DFL   | Data Field Length<br>This field defines the number of data bytes in the response part of the frame.<br>DFL = Number of data bytes – 1.  |
| DIR   | Direction<br>This bit controls the direction of the data field.<br>0 LINFlex receives the data and copies them in the BDRL and BDRM registers.<br>1 LINFlex transmits the data from the BDRL and BDRM registers.  |
| CCS   | Classic Checksum<br>This bit controls the type of checksum applied on the current message.<br>0 Enhanced Checksum covering Identifier and Data fields. This is compatible with LIN specification 2.0 and higher.<br>1 Classic Checksum covering Data field only. This is compatible with LIN specification 1.3 and earlier. |
| ID    | Identifier<br>Identifier part of the identifier field without the identifier parity   |

## 23.8 Functional description

### 23.8.1 UART mode

The main features in the UART mode are

- Full duplex communication
- 8- or 9-bit data with parity
- 4-byte buffer for reception, 4-byte buffer for transmission
- 8-bit counter for timeout management

**8-bit data frames:** The 8th bit can be a data or a parity bit. Even/Odd Parity can be selected by the Odd Parity bit in the UARTCR. An even parity is set if the modulo-2 sum of the 7 data bits is 1. An odd parity is cleared in this case.

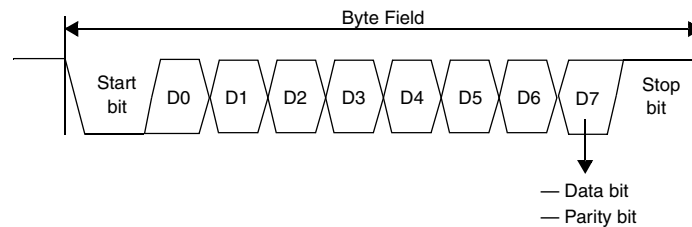


Figure 23-27. UART mode 8-bit data frame

**9-bit frames:** The 9th bit is a parity bit. Even/Odd Parity can be selected by the Odd Parity bit in the UARTCR. An even parity is set if the modulo-2 sum of the 8 data bits is 1. An odd parity is cleared in this case.

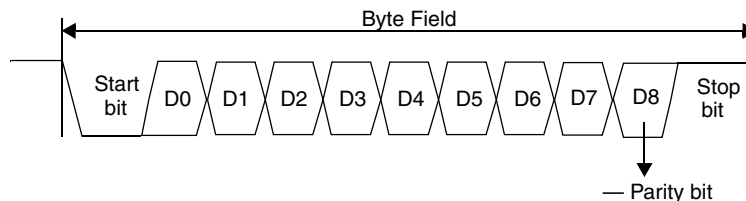


Figure 23-28. UART mode 9-bit data frame

#### 23.8.1.1 Buffer in UART mode

The 8-byte buffer is divided into two parts: one for receiver and one for transmitter, as shown in [Table 23-30](#).

Table 23-30. Message buffer

| Buffer data register | LIN mode                |            | UART mode       |     |
|----------------------|-------------------------|------------|-----------------|-----|
| BDRL[0:31]           | Transmit/Receive buffer | DATA0[0:7] | Transmit buffer | Tx0 |
|                      |                         | DATA1[0:7] |                 | Tx1 |
|                      |                         | DATA2[0:7] |                 | Tx2 |
|                      |                         | DATA3[0:7] |                 | Tx3 |
| BDRM[0:31]           |                         | DATA4[0:7] | Receive buffer  | Rx0 |
|                      |                         | DATA5[0:7] |                 | Rx1 |
|                      |                         | DATA6[0:7] |                 | Rx2 |
|                      |                         | DATA7[0:7] |                 | Rx3 |

### 23.8.1.2 UART transmitter

In order to start transmission in UART mode, you must program the UART bit and the transmitter enable (TXEN) bit in the UARTCR to 1. Transmission starts when DATA0 (least significant data byte) is programmed. The number of bytes transmitted is equal to the value configured by UARTCR[TDFL] (see Table 23-10).

The Transmit buffer is 4 bytes, hence a 4-byte maximum transmission can be triggered. Once the programmed number of bytes has been transmitted, the UARTSR[DTF] bit is set. If UARTCR[TXEN] is reset during a transmission then the current transmission is completed and no further transmission can be invoked.

### 23.8.1.3 UART receiver

The UART receiver is active as soon as the user exits Initialization mode and programs UARTCR[RXEN] = 1. There is a dedicated 4-byte data buffer for received data bytes. Once the programmed number (RDFL bits) of bytes has been received, the UARTSR[DRF] bit is set. If the RXEN bit is reset during a reception then the current reception is completed and no further reception can be invoked until RXEN is set.

If a parity error occurs during reception of any byte, then the corresponding PEx bit in the UARTSR is set. No interrupt is generated in this case. If a framing error occurs in any byte (UARTSR[FE] = 1) then an interrupt is generated if the LINIER[FEIE] bit is set.

If the last received frame has not been read from the buffer (that is, RMB bit is not reset by the user) then upon reception of the next byte an overrun error occurs (UARTSR[BOF] = 1) and one message will be lost. Which message is lost depends on the configuration of LINCR1[RBLM].

- If the buffer lock function is disabled (LINCR1[RBLM] = 0), the last message stored in the buffer is overwritten by the new incoming message. In this case the latest message is always available to the application.
- If the buffer lock function is enabled (LINCR1[RBLM] = 1), the most recent message is discarded and the previous message is available in the buffer.

An interrupt is generated if the LINIER[BOIE] bit is set.

#### 23.8.1.4 Clock gating

The LINFlex clock can be gated from the Mode Entry module (MC\_ME). In UART mode, the LINFlex controller acknowledges a clock gating request once the data transmission and data reception are completed, that is, once the Transmit buffer is empty and the Receive buffer is full.

### 23.8.2 LIN mode

LIN mode comprises four submodes:

- Master mode
- Slave mode
- Slave mode with identifier filtering
- Slave mode with automatic resynchronization

These submodes are described in the following pages.

#### 23.8.2.1 Master mode

In Master mode the application uses the message buffer to handle the LIN messages. Master mode is selected when the LINCR1[MME] bit is set.

##### 23.8.2.1.1 LIN header transmission

According to the LIN protocol any communication on the LIN bus is triggered by the Master sending a header. The header is transmitted by the Master task while the data is transmitted by the Slave task of a node.

To transmit a header with LINFlex, the application must set up the identifier and the data field length, and configure the message (direction and checksum type) in the BIDR before requesting the header transmission by setting LINCR2[HTRQ].

##### 23.8.2.1.2 Data transmission (transceiver as publisher)

When the master node is publisher of the data corresponding to the identifier sent in the header, then the slave task of the master has to send the data in the Response part of the LIN frame. Therefore, the application must provide the data to LINFlex before requesting the header transmission. The application stores the data in the message buffer BDR. According to the data field length, LINFlex transmits the data and the checksum. The application uses the BDR[CCS] bit to configure the checksum type (classic or enhanced) for each message.

If the response has been sent successfully, the LINSR[DTF] bit is set. In case of error, the DTF flag is not set and the corresponding error flag is set in the LINESR (see [Section 23.8.2.1.6, Error handling](#)).

It is possible to handle frames with a Response size larger than 8 bytes of data (extended frames). If the data field length in the BIDR is configured with a value higher than 8 data bytes, the LINSR[DBEF] bit is

set after the first 8 bytes have been transmitted. The application has to update the buffer BDR before resetting the DBEF bit. The transmission of the next bytes starts when the DBEF bit is reset.

After the last data byte (or the checksum byte) has been sent, the DTF flag is set.

The direction of the message buffer is controlled by the BIDR[DIR] bit. When the application sets this bit the response is sent by LINFlex (publisher). Resetting this bit configures the message buffer as subscriber.

### 23.8.2.1.3 Data reception (transceiver as subscriber)

To receive data from a slave node, the master sends a header with the corresponding identifier. LINFlex stores the data received from the slave in the message buffer and stores the message status in the LINSR.

If the response has been received successfully, the LINSR[DRF] is set. In case of error, the DRF flag is not set and the corresponding error flag is set in the LINESR (see [Section 23.8.2.1.6, Error handling](#)).

It is possible to handle frames with a Response size larger than 8 bytes of data (extended frames). If the data field length in the BIDR is configured with a value higher than 8 data bytes, the LINSR[DBFF] bit is set once the first 8 bytes have been received. The application has to read the buffer BDR before resetting the DBFF bit. Once the last data byte (or the checksum byte) has been received, the DRF flag is set.

### 23.8.2.1.4 Data discard

To discard data from a slave, the BIDR[DIR] bit must be reset and the LINC2R[DDRQ] bit must be set before starting the header transmission.

### 23.8.2.1.5 Error detection

LINFlex is able to detect and handle LIN communication errors. A code stored in the LIN error status register (LINESR) signals the errors to the software.

In Master mode, the following errors are detected:

- Bit error: During transmission, the value read back from the bus differs from the transmitted value.
- Framing error: A dominant state has been sampled on the stop bit of the currently received character (synch field, identifier field or data field).
- Checksum error: The computed checksum does not match the received one.
- Response and Frame timeout: See [Section 23.8.3, 8-bit timeout counter](#), for more details.

### 23.8.2.1.6 Error handling

In case of Bit Error detection during transmission, LINFlex stops the transmission of the frame after the corrupted bit. LINFlex returns to idle state and an interrupt is generated if LINIER[BEIE] = 1.

During reception, a Framing Error leads LINFlex to discard the current frame. LINFlex returns immediately to idle state. An interrupt is generated if LINIER[FEIE] = 1.

During reception, a Checksum Error leads LINFlex to discard the received frame. LINFlex returns to idle state. An interrupt is generated if LINIER[CEIE] = 1.



### 23.8.2.1.7 Overrun

After the message buffer is full, the next valid message reception causes an overrun and a message is lost. The LINFlexD controller sets LINSR[BOF] to signal the overrun condition. Which message is lost depends on the configuration of the RX message buffer:

- If the buffer lock function is disabled (LINCR1[RBLM] cleared), the last message stored in the buffer is overwritten by the new incoming message. In this case, the latest message is always available to the software.
- If the buffer lock function is enabled (LINCR1[RBLM] set), the most recent message is discarded and the previous message is available in the buffer.

### 23.8.2.2 Slave mode

In Slave mode the application uses the message buffer to handle the LIN messages. Slave mode is selected when LINCR1[MME] = 0.

#### 23.8.2.2.1 Data transmission (transceiver as publisher)

When LINFlex receives the identifier, the LINSR[HRF] is set and, if LINIER[HRIE] = 1, an RX interrupt is generated. The software must read the received identifier in the BIDR, fill the BDR registers, specify the data field length using the BIDR[DFL], and trigger the data transmission by setting the LINCR2[DTRQ] bit.

One or several identifier filters can be configured for transmission by setting the IFCRx[DIR] bit and activated by setting one or several bits in the IFER.

When at least one identifier filter is configured in transmission and activated, and if the received identifier matches the filter, a specific TX interrupt (instead of an RX interrupt) is generated.

Typically, the application has to copy the data from SRAM locations to the BDAR. To copy the data to the right location, the application has to identify the data by means of the identifier. To avoid this and to ease the access to the SRAM locations, the LINFlex controller provides a Filter Match Index. This index value is the number of the filter that matched the received identifier.

The software can use the index in the IFMI register to directly access the pointer that points to the right data array in the SRAM area and copy this data to the BDAR (see [Figure 23-30](#)).

Using a filter avoids the software having to configure the direction, the data field length and the checksum type in the BIDR. The software fills the BDAR and triggers the data transmission by programming LINCR2[DTRQ] = 1.

If LINFlex cannot provide enough TX identifier filters to handle all identifiers the software has to transmit data for, then a filter can be configured in mask mode (see [Section 23.8.2.3, Slave mode with identifier filtering](#)) in order to manage several identifiers with one filter only.

### 23.8.2.2.2 Data reception (transceiver as subscriber)

When LINFlex receives the identifier, the LINSR[HRF] bit is set and, if LINIER[HRIE] = 1, an RX interrupt is generated. The software must read the received identifier in the BIDR and specify the data field length using the BIDR[DFL] field before receiving the stop bit of the first byte of data field.

When the checksum reception is completed, an RX interrupt is generated to allow the software to read the received data in the BDR registers.

One or several identifier filters can be configured for reception by programming IFCR<sub>x</sub>[DIR] = 0 and activated by setting one or several bits in the IFER.

When at least one identifier filter is configured in reception and activated, and if the received identifier matches the filter, an RX interrupt is generated after the checksum reception only.

Typically, the application has to copy the data from the BDAR to SRAM locations. To copy the data to the right location, the application has to identify the data by means of the identifier. To avoid this and to ease the access to the SRAM locations, the LINFlex controller provides a Filter Match Index. This index value is the number of the filter that matched the received identifier.

The software can use the index in the IFMI register to directly access the pointer that points to the right data array in the SRAM area and copy this data from the BDAR to the SRAM (see [Figure 23-30](#)).

Using a filter avoids the software reading the ID value in the BIDR and configuring the direction, the data field length, and the checksum type in the BIDR.

If LINFlex cannot provide enough RX identifier filters to handle all identifiers the software has to receive the data for, then a filter can be configured in mask mode (see [Section 23.8.2.3, Slave mode with identifier filtering](#)) in order to manage several identifiers with one filter only.

### 23.8.2.2.3 Data discard

When LINFlex receives the identifier, the LINSR[HRF] bit is set and, if LINIER[HRIE] = 1, an RX interrupt is generated. If the received identifier does not concern the node, you must program LINCR2[DDRQ] = 1. LINFlex returns to idle state after bit DDRQ is set.

### 23.8.2.2.4 Error detection

In Slave mode, the following errors are detected:

- **Header error:** An error occurred during header reception (Break Delimiter error, Inconsistent Synch Field, Header Timeout).
- **Bit error:** During transmission, the value read back from the bus differs from the transmitted value.
- **Framing error:** A dominant state has been sampled on the stop bit of the currently received character (synch field, identifier field or data field).
- **Checksum error:** The computed checksum does not match the received one.

### 23.8.2.2.5 Error handling

In case of Bit Error detection during transmission, LINFlex stops the transmission of the frame after the corrupted bit. LINFlex returns to idle state and an interrupt is generated if the BEIE bit in the LINIER is set.

During reception, a Framing Error leads LINFlex to discard the current frame. LINFlex returns immediately to idle state. An interrupt is generated if  $LINIER[FEIE] = 1$ .

During reception, a Checksum Error leads LINFlex to discard the received frame. LINFlex returns to idle state. An interrupt is generated if  $LINIER[CEIE] = 1$ .

During header reception, a Break Delimiter error, an Inconsistent Synch Field or a Timeout error leads LINFlex to discard the header. An interrupt is generated if  $LINIER[HEIE] = 1$ . LINFlex returns to idle state.

#### 23.8.2.2.6 Valid header

A received header is considered as valid when it has been received correctly according to the LIN protocol.

If a valid Break Field and Break Delimiter come before the end of the current header or at any time during a data field, the current header or data is discarded and the state machine synchronizes on this new break.

#### 23.8.2.2.7 Valid message

A received or transmitted message is considered as valid when the data has been received or transmitted without error according to the LIN protocol.

#### 23.8.2.2.8 Overrun

Once the message buffer is full, the next valid message reception leads to an overrun and a message is lost. The hardware sets the BOF bit in the LINSR to signal the overrun condition. Which message is lost depends on the configuration of the RX message buffer:

- If the buffer lock function is disabled ( $LINCR1[RBLM] = 0$ ), the last message stored in the buffer is overwritten by the new incoming message. In this case the latest message is always available to the application.
- If the buffer lock function is enabled ( $LINCR1[RBLM] = 1$ ), the most recent message is discarded and the previous message is available in the buffer.

### 23.8.2.3 Slave mode with identifier filtering

In the LIN protocol the identifier of a message is not associated with the address of a node but related to the content of the message. Consequently a transmitter broadcasts its message to all receivers. On header reception a slave node decides—depending on the identifier value—whether the software needs to receive or send a response. If the message does not target the node, it must be discarded without software intervention.

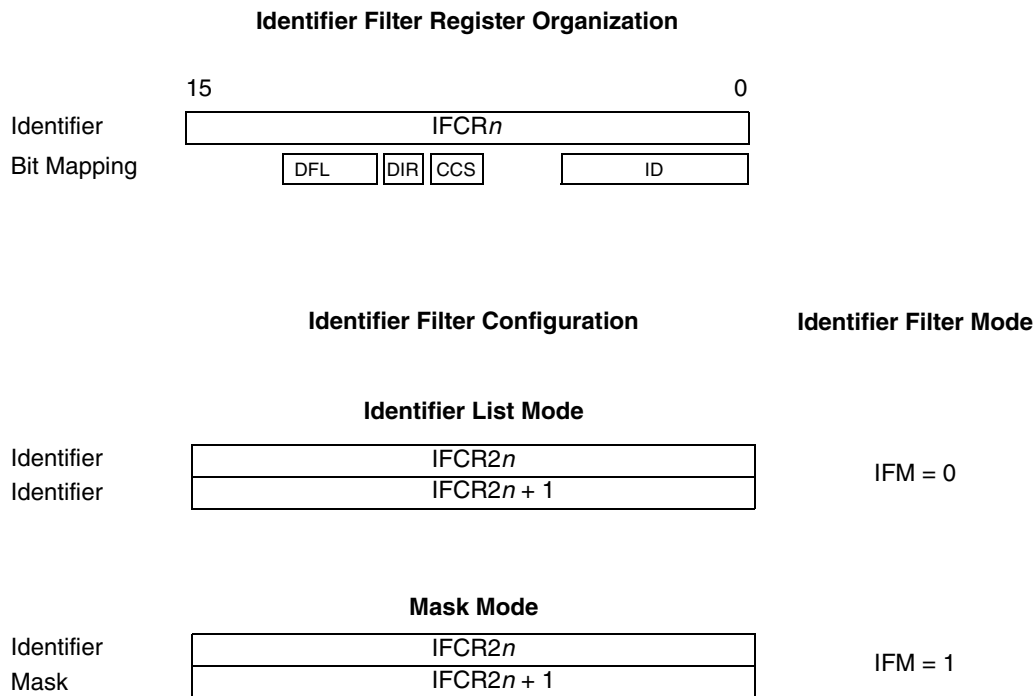
To fulfill this requirement, the LINFlex controller provides configurable filters in order to request software intervention only if needed. This hardware filtering saves CPU resources that would otherwise be needed by software for filtering.

### 23.8.2.3.1 Filter mode

Usually each of the eight IFCR registers filters one dedicated identifier, but this limits the number of identifiers LINFlex can handle to the number of IFCR registers implemented in the device. Therefore, in order to be able to handle more identifiers, the filters can be configured in mask mode.

In identifier list mode (the default mode), both filter registers are used as identifier registers. All bits of the incoming identifier must match the bits specified in the filter register.

In mask mode, the identifier registers are associated with mask registers specifying which bits of the identifier are handled as “must match” or as “don’t care”. For the bit mapping and registers organization, please see [Figure 23-29](#).



**Figure 23-29. Filter configuration—register organization**

### 23.8.2.3.2 Identifier filter mode configuration

The identifier filters are configured in the  $IFCR_x$  registers. To configure an identifier filter the filter must first be deactivated by programming  $IFER[FACT] = 0$ . The identifier list or identifier mask mode for the corresponding  $IFCR_x$  registers is configured by the  $IFMR[IFM]$  bit. For each filter, the  $IFCR_x$  register configures the ID (or the mask), the direction (TX or RX), the data field length, and the checksum type.

If no filter is active, an RX interrupt is generated on any received identifier event.

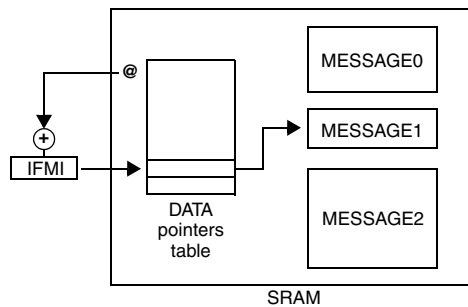
If at least one active filter is configured as TX, all received identifiers matching this filter generate a TX interrupt.

If at least one active filter is configured as RX, all received identifiers matching this filter generate an RX interrupt.

If no active filter is configured as RX, all received identifiers not matching TX filter(s) generate an RX interrupt.

**Table 23-31. Filter to interrupt vector correlation**

| Number of active filters | Number of active filters configured as TX | Number of active filters configured as RX | Interrupt vector   |
|--------------------------|---|---|--|
| 0                        | 0   | 0   | RX interrupt on all identifiers  |
| a<br>(a > 0)             | a   | 0   | — TX interrupt on identifiers matching the filters,<br>— RX interrupt on all other identifiers if BF bit is set, no RX interrupt if BF bit is reset                  |
| n<br>(n = a + b)         | a<br>(a > 0)                              | b<br>(b > 0)                              | — TX interrupt on identifiers matching the TX filters,<br>— RX interrupt on identifiers matching the RX filters,<br>— all other identifiers discarded (no interrupt) |
| b<br>(b > 0)             | 0   | b   | — RX interrupt on identifiers matching the filters,<br>— TX interrupt on all other identifiers if BF bit is set, no TX interrupt if BF bit is reset                  |



**Figure 23-30. Identifier match index**

#### 23.8.2.4 Slave mode with automatic resynchronization

Automatic resynchronization must be enabled in Slave mode if  $f_{\text{periph\_set\_1\_clk}}$  tolerance is greater than 1.5%. This feature compensates a  $f_{\text{periph\_set\_1\_clk}}$  deviation up to 14%, as specified in LIN standard.

This mode is similar to Slave mode as described in [Section 23.8.2.2, Slave mode](#), with the addition of automatic resynchronization enabled by the LASE bit. In this mode LINFlex adjusts the fractional baud rate generator after each Synch Field reception.

### 23.8.2.4.1 Automatic resynchronization method

When automatic resynchronization is enabled, after each LIN Break, the time duration between five falling edges on RDI is sampled on  $f_{\text{periph\_set\_1\_clk}}$  and the result of this measurement is stored in an internal 19-bit register called SM (not user accessible) (see Figure 23-31). Then the LFDIV value (and its associated registers LINIBRR and LINFBR) is automatically updated at the end of the fifth falling edge. During LIN Synch Field measurement, the LINFlex state machine is stopped and no data is transferred to the data register.

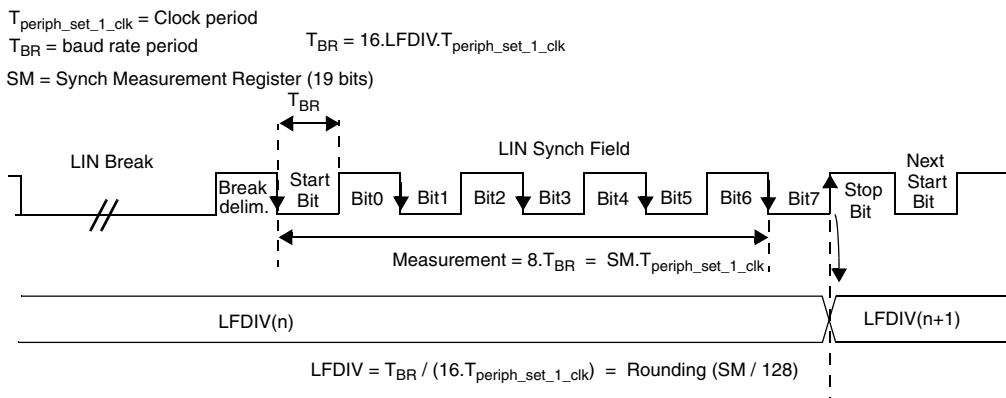


Figure 23-31. LIN synch field measurement

LFDIV is an unsigned fixed point number. The mantissa is coded on 12 bits in the LINIBRR and the fraction is coded on 4 bits in the LINFBR.

If LASE bit = 1 then LFDIV is automatically updated at the end of each LIN Synch Field.

Three internal registers (not user-accessible) manage the auto-update of the LINFlex divider (LFDIV):

- LFDIV\_NOM (nominal value written by software at LINIBRR and LINFBR addresses)
- LFDIV\_MEAS (results of the Field Synch measurement)
- LFDIV (used to generate the local baud rate)

On transition to idle, break or break delimiter state due to any error or on reception of a complete frame, hardware reloads LFDIV with LFDIV\_NOM.

#### 23.8.2.4.2 Deviation error on the Synch Field

The deviation error is checked by comparing the current baud rate (relative to the slave oscillator) with the received LIN Synch Field (relative to the master oscillator). Two checks are performed in parallel.

The first check is based on a measurement between the first falling edge and the last falling edge of the Synch Field:

- If  $D1 > 14.84\%$ , LHE is set.
- If  $D1 < 14.06\%$ , LHE is not set.
- If  $14.06\% < D1 < 14.84\%$ , LHE can be either set or reset depending on the dephasing between the signal on LINFlex\_RX pin the  $f_{\text{periph\_set\_1\_clk}}$  clock.

The second check is based on a measurement of time between each falling edge of the Synch Field:

- If  $D2 > 18.75\%$ , LHE is set.
- If  $D2 < 15.62\%$ , LHE is not set.
- If  $15.62\% < D2 < 18.75\%$ , LHE can be either set or reset depending on the dephasing between the signal on LINFlex\_RX pin the  $f_{\text{periph\_set\_1\_clk}}$  clock.

Note that the LINFlex does not need to check if the next edge occurs slower than expected. This is covered by the check for deviation error on the full synch byte.

### 23.8.2.5 Clock gating

The LINFlex clock can be gated from the Mode Entry module (MC\_ME). In LIN mode, the LINFlex controller acknowledges a clock gating request once the frame transmission or reception is completed.

## 23.8.3 8-bit timeout counter

### 23.8.3.1 LIN timeout mode

Clearing the LTOM bit (setting its value to 0) in the LINTCSR enables the LIN timeout mode. The LINOCCR becomes read-only, and OC1 and OC2 output compare values in the LINOCCR are automatically updated by hardware.

This configuration detects header timeout, response timeout, and frame timeout.

Depending on the LIN mode (selected by the LINC1R1[MME] bit), the 8-bit timeout counter will behave differently.

LIN timeout mode must not be enabled during LIN extended frames transmission or reception (that is, if the data field length in the BDR is configured with a value higher than 8 data bytes).

#### 23.8.3.1.1 LIN Master mode

The LINTOCR[RT0] field can be used to tune response timeout and frame timeout values. Header timeout value is fixed to  $HTO = 28$ -bit time.

Field OC1 checks  $T_{\text{Header}}$  and  $T_{\text{Response}}$  and field OC2 checks  $T_{\text{Frame}}$  (see [Figure 23-32](#)).

When LINFlex moves from Break delimiter state to Synch Field state (see [Section 23.7.1.3, LIN status register \(LINSR\)](#)):

- OC1 is updated with the value of  $OC_{\text{Header}}$  ( $OC_{\text{Header}} = CNT + 28$ ),
- OC2 is updated with the value of  $OC_{\text{Frame}}$  ( $OC_{\text{Frame}} = CNT + 28 + RTO \times 9$  (frame timeout value for an 8-byte frame)),
- The TOCE bit is set.

On the start bit of the first response data byte (and if no error occurred during the header reception), OC1 is updated with the value of  $OC_{\text{Response}}$  ( $OC_{\text{Response}} = CNT + RTO \times 9$  (response timeout value for an 8-byte frame)).

On the first response byte is received, OC1 and OC2 are automatically updated to check  $T_{\text{Response}}$  and  $T_{\text{Frame}}$  according to RTO (tolerance) and DFL.

On the checksum reception or in case of error in the header or response, the TOCE bit is reset.

If there is no response, frame timeout value does not take into account the DFL value, and an 8-byte response (DFL = 7) is always assumed.

### 23.8.3.1.2 LIN Slave mode

The LINTOCR[RTO] field can be used to tune response timeout and frame timeout values. Header timeout value is fixed to HTO.

OC1 checks  $T_{\text{Header}}$  and  $T_{\text{Response}}$  and OC2 checks  $T_{\text{Frame}}$  (see Figure 23-32).

When LINFlex moves from Break state to Break Delimiter state (see Section 23.7.1.3, LIN status register (LINSR)):

- OC1 is updated with the value of  $OC_{\text{Header}}$  ( $OC_{\text{Header}} = \text{CNT} + \text{HTO}$ ),
- OC2 is updated with the value of  $OC_{\text{Frame}}$  ( $OC_{\text{Frame}} = \text{CNT} + \text{HTO} + \text{RTO} \times 9$  (frame timeout value for an 8-byte frame)),
- The TOCE bit is set.

On the start bit of the first response data byte (and if no error occurred during the header reception), OC1 is updated with the value of  $OC_{\text{Response}}$  ( $OC_{\text{Response}} = \text{CNT} + \text{RTO} \times 9$  (response timeout value for an 8-byte frame)).

Once the first response byte is received, OC1 and OC2 are automatically updated to check  $T_{\text{Response}}$  and  $T_{\text{Frame}}$  according to RTO (tolerance) and DFL.

On the checksum reception or in case of error in the header or data field, the TOCE bit is reset.

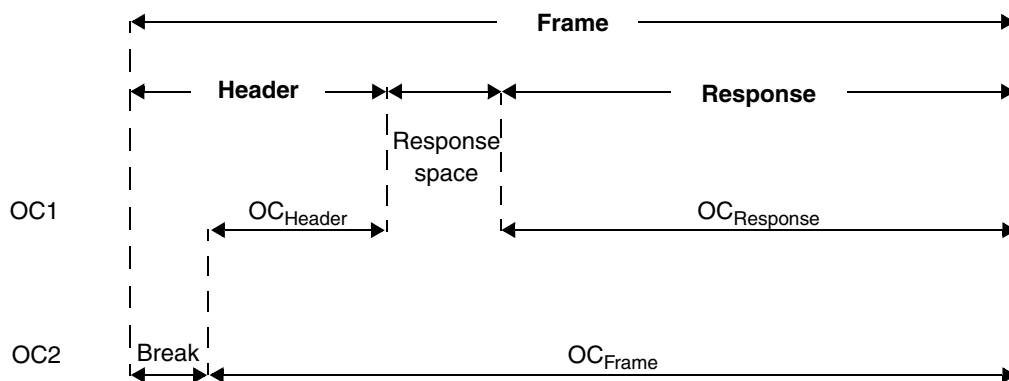


Figure 23-32. Header and response timeout

### 23.8.3.2 Output compare mode

Setting LINTCSR[LTOM] = 1 enables the output compare mode. This mode allows the user to fully customize the use of the counter.

OC1 and OC2 output compare values can be updated in the LINTOCR by software.



## 23.8.4 Interrupts

**Table 23-32. LINFlex interrupt control**

| Interrupt event                  | Event flag bit | Enable control bit | Interrupt vector |
|----------------------------------|----------------|--------------------|------------------|
| Header Received interrupt        | HRF            | HRIE               | RXI <sup>1</sup> |
| Data Transmitted interrupt       | DTF            | DTIE               | TXI              |
| Data Received interrupt          | DRF            | DRIE               | RXI              |
| Data Buffer Empty interrupt      | DBEF           | DBEIE              | TXI              |
| Data Buffer Full interrupt       | DBFF           | DBFIE              | RXI              |
| Wake-up interrupt                | WUPF           | WUPIE              | RXI              |
| LIN State interrupt <sup>2</sup> | LSF            | LSIE               | RXI              |
| Buffer Overrun interrupt         | BOF            | BOIE               | ERR              |
| Framing Error interrupt          | FEF            | FEIE               | ERR              |
| Header Error interrupt           | HEF            | HEIE               | ERR              |
| Checksum Error interrupt         | CEF            | CEIE               | ERR              |
| Bit Error interrupt              | BEF            | BEIE               | ERR              |
| Output Compare interrupt         | OCF            | OCIE               | ERR              |
| Stuck at Zero interrupt          | SZF            | SZIE               | ERR              |

<sup>1</sup> In Slave mode, if at least one filter is configured as TX and enabled, header received interrupt vector is RXI or TXI depending on the value of identifier received.

<sup>2</sup> For debug and validation purposes

This page is intentionally left blank.

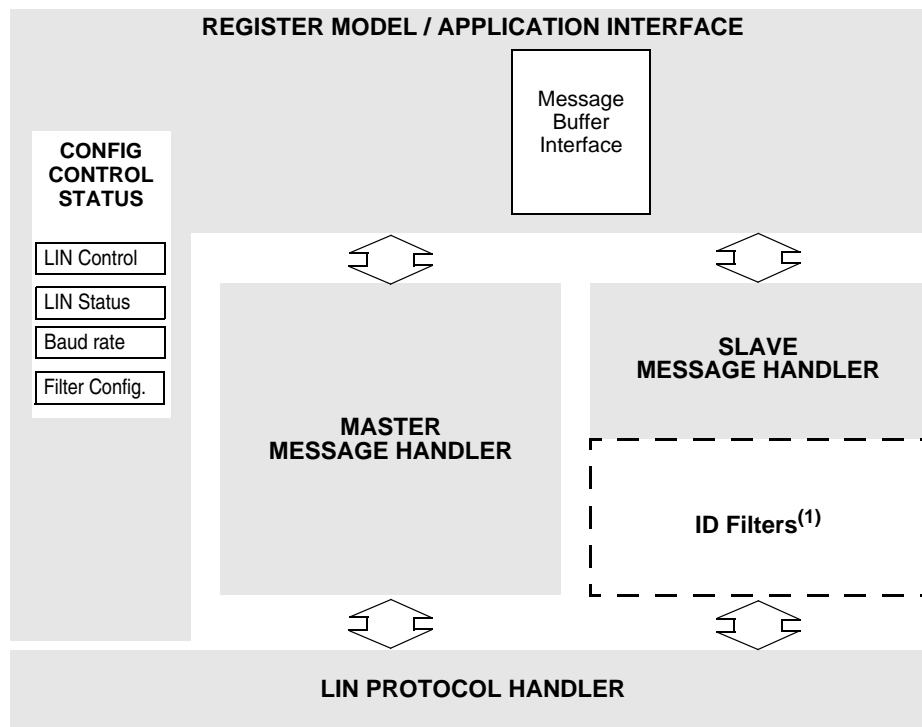
# Chapter 24

## LIN Controller (LINFlexD)

### 24.1 Introduction

The LINFlexD (Local Interconnect Network Flexible with DMA support) controller interfaces the LIN network and supports the LIN protocol versions 1.3, 2.0, 2.1 and J2602 in both Master and Slave modes. LINFlexD includes a LIN mode that provides additional features (compared to standard UART) to ease LIN implementation, improve system robustness, minimize CPU load, and allow slave node resynchronization.

Figure 24-1 shows the LINFlexD block diagram.



<sup>1</sup> Filter activation optional

Figure 24-1. LINFlexD block diagram

### 24.2 Main features

The LINFlexD controller can operate in several modes, each of which has a distinct set of features. These distinct features are described in the following sections.

In addition, the LINFlexD controller has several features common to all modes:

- Fractional baud rate generator

- Three operating modes for power saving and configuration registers lock
  - Initialization
  - Normal
  - Sleep
- Two test modes
  - Loop Back
  - Self Test
- Maskable interrupts

### 24.2.1 LIN mode features

- Supports LIN protocol versions 1.3, 2.0, 2.1, and J2602
- Master mode with autonomous message handling
- Classic and enhanced checksum calculation and check
- Single 8-byte buffer for transmission/reception
- Extended frame mode for In-application Programming purposes
- Wake-up event on dominant bit detection
- True LIN field state machine
- Advanced LIN error detection
- Header, response, and frame timeout
- Slave mode
  - Autonomous header handling
  - Autonomous transmit/receive data handling
- LIN automatic resynchronization, allowing operation with as clock source
- Identifier filters for autonomous message handling in Slave mode

### 24.2.2 UART mode features

- Full-duplex communication
- Selectable frame size:
  - 8-bit frame
  - 9-bit frame
  - 16-bit frame
  - 17-bit frame
- Selectable parity:
  - Even
  - Odd
  - 0
  - 1

- 4-byte buffer for reception, 4-byte buffer for transmission
- 12-bit counter for timeout management

## 24.3 The LIN protocol

The LIN (Local Interconnect Network) is a serial communication protocol. The topology of a LIN network is shown in [Figure 24-2](#). A LIN network consists of:

- One master
- Several slave
- The LIN bus

A master node contains the master task as well as a slave task, all other nodes contain a slave task only. The master node decides when and which frame shall be transferred on the bus. The slave task provides the data to be transported by the frame.

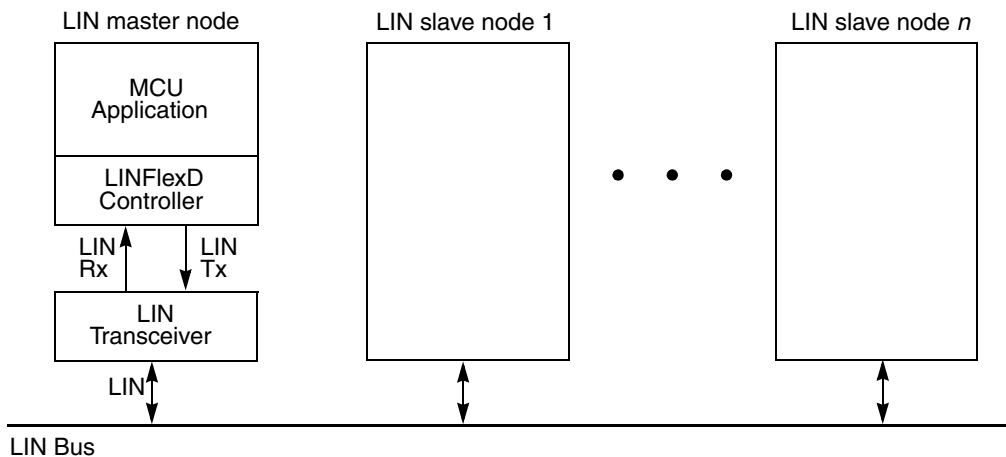


Figure 24-2. LIN network topology

### 24.3.1 Dominant and recessive logic levels

The LIN bus defines two logic levels, dominant and recessive, as follows:

- Dominant: logical low level (0)
- Recessive: logical high level (1)

### 24.3.2 LIN frames

A frame consists of a header provided by the master task and a response provided by the slave task, as shown in [Figure 24-3](#).

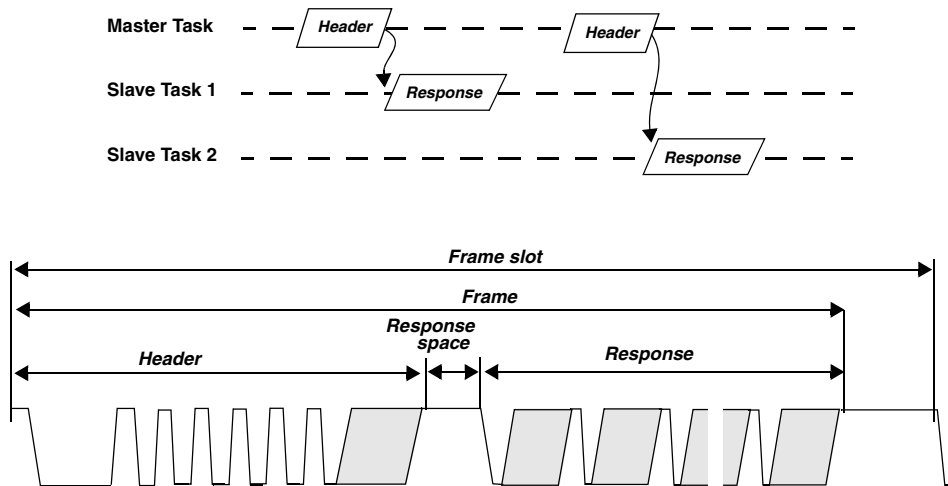


Figure 24-3. LIN frame structure

### 24.3.3 LIN header

The header consists of:

- A break field (described in [Section 24.3.3.1, Break field](#))
- A sync (described in [Section 24.3.3.2, Sync](#))
- An identifier (described in [Section 24.3.4.2, Identifier](#))

The slave task associated with the identifier provides the response.

#### 24.3.3.1 Break field

The break field, shown in [Figure 24-4](#), is used to signal the beginning of a new frame. It is always generated by the master and consists of:

- At least 13 dominant bits including the start bit
- At least one recessive bit that functions as break delimiter



Figure 24-4. Break field

#### 24.3.3.2 Sync

The sync pattern is a byte consisting of alternating dominant and recessive bits as shown in [Figure 24-5](#). It forms a data value of 0x55.

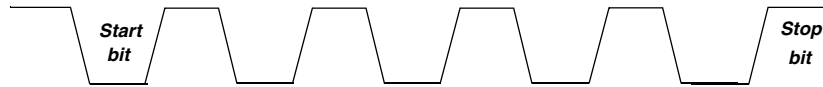


Figure 24-5. Sync pattern

### 24.3.4 Response

The response consists of:

- A data field (described in [Section 24.3.4.1, Data field](#))
- A checksum (described in [Section 24.3.4.3, Checksum](#))

The slave task interested in the data associated with the identifier receives the response and verifies the checksum.

#### 24.3.4.1 Data field

The structure of the data field transmitted on the LIN bus is shown in [Figure 24-6](#). The LSB of the data is sent first and the MSB last. The start bit is encoded as a dominant bit and the stop bit is encoded as a recessive bit.

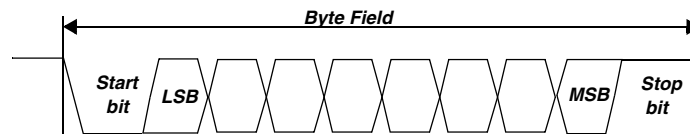


Figure 24-6. Structure of the data field

#### 24.3.4.2 Identifier

The identifier, shown in [Figure 24-7](#), consists of two subfields:

- The identifier value (in bits 0–5)
- The identifier parity (in bits 6–7)

The parity bits P0 and P1 are defined as follows:

- $P0 = ID0 \text{ xor } ID1 \text{ xor } ID2 \text{ xor } ID4$
- $P1 = \text{not}(ID1 \text{ xor } ID3 \text{ xor } ID4 \text{ xor } ID5)$

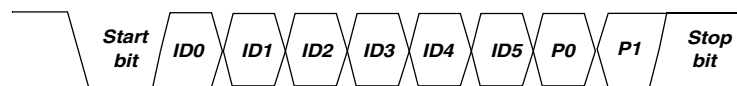


Figure 24-7. Identifier

#### 24.3.4.3 Checksum

The checksum contains the inverted 8-bit sum (with carry) over one of two possible groups:

- The classic checksum sums all data bytes, and is used for communication with LIN 1.3 slaves.
- The enhanced checksum sums all data bytes and the identifier, and is used for communication with LIN 2.0 (or later) slaves.

## 24.4 LINFlexD and software intervention

The increasing number of communication peripherals embedded on microcontrollers (for example, CAN, LIN, SPI) requires more and more CPU resources for the communication management. Even a 32-bit microcontroller is overloaded if its peripherals do not provide high level features to autonomously handle the communication.

Even though the LIN protocol with a maximum baud rate of 20 Kbit/s is relatively slow, it still generates a non-negligible load on the CPU if the LIN is implemented on a standard UART, as is usually the case.

To minimize the CPU load in Master mode, LINFlexD handles the LIN messages autonomously.

In Master mode, once the software has triggered the header transmission, LINFlexD does not request any software (that is, application) intervention until the next header transmission request in transmission mode or until the checksum reception in reception mode.

To minimize the CPU load in Slave mode, LINFlexD requires software intervention only to:

- Trigger transmission or reception or data discard depending on the identifier
- Write data into the buffer (transmission mode) or read data from the buffer (reception mode) after checksum reception

If filter mode is activated for Slave mode, LINFlexD requires software intervention only to write data into the buffer (transmission mode) or read data from the buffer (reception mode)

The software uses the control, status, and configuration registers to:

- Configure LIN parameters (for example, baud rate or mode)
- Request transmissions
- Handle receptions
- Manage interrupts
- Configure LIN error and timeout detection
- Process diagnostic information

The message buffer stores transmitted or received LIN frames.

## 24.5 Summary of operating modes

The LINFlexD controller has three operating modes:

- Normal
- Initialization
- Sleep

After a hardware reset, the LINFlexD controller is in Sleep mode to reduce power consumption.



The transitions between these modes are shown in Figure 24-8. The software instructs LINFlexD to enter Initialization mode or Sleep mode by setting LINCRC1[INIT] or LINCRC1[SLEEP], respectively.

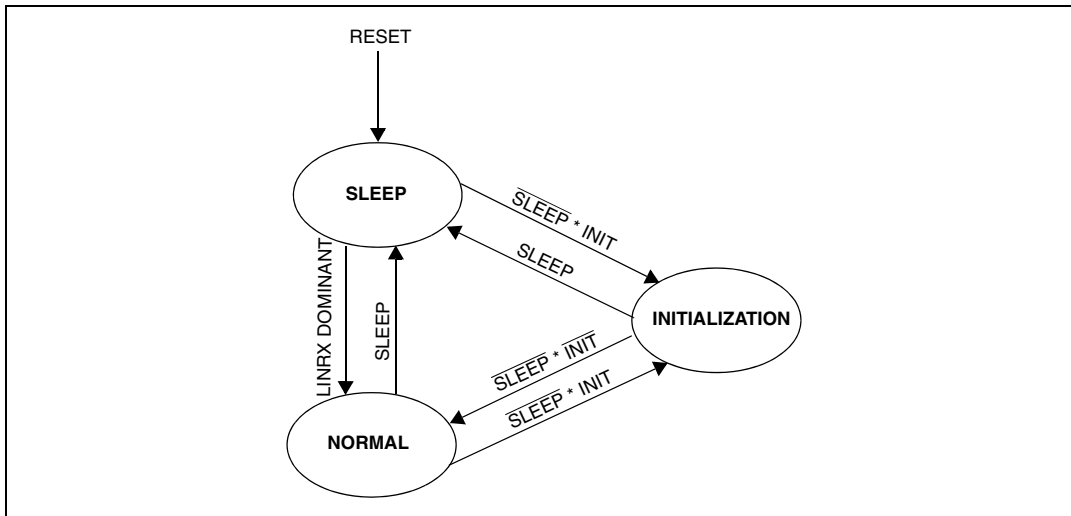


Figure 24-8. LINFlexD controller operating modes

In addition to these controller-level operating modes, the LINFlexD controller also supports several protocol-level modes:

- LIN mode:
  - Master mode
  - Slave mode
  - Slave mode with identifier filtering
  - Slave mode with automatic resynchronization
- UART mode
- Test modes:
  - Loop Back mode
  - Self Test mode

These modes are discussed in detail in subsequent sections.

## 24.6 Controller-level operating modes

### 24.6.1 Initialization mode

The software initialization can be done while the hardware is in Initialization mode. To enter or exit this mode, the software sets or clears LINCRC1[INIT], respectively.

In Initialization mode, all message transfers to and from the LIN bus are stopped and the LIN bus output (LINTX) is recessive.

Entering Initialization mode does not change any of the configuration registers.

To initialize the LINFlexD controller, the software must:

- Select the desired mode (Master, Slave or UART)
- Set up the baud rate register
- If LIN Slave mode with filter activation is selected, initialize the identifier list

## 24.6.2 Normal mode

After initialization is complete, the software must clear LINC1R1[INIT] to put the LINFLEXD controller into Normal mode.

## 24.6.3 Sleep (low-power) mode

To reduce power consumption, LINFLEXD has a low-power mode called Sleep mode. In this mode, the LINFLEXD clock is stopped. Consequently, the LINFLEXD will not update the status bits, but software can still access the LINFLEXD registers.

To enter this mode, the software must set LINC1R1[SLEEP].

LINFLEXD can be awakened (exit Sleep mode) in one of two ways:

- The software clears LINC1R1[SLEEP]
- Automatic wake-up is enabled (LINC1R1[AWUM] is set) and LINFLEXD detects LIN bus activity (that is, if a wakeup pulse of 150  $\mu$ s is detected on the LIN bus)

On LIN bus activity detection, hardware automatically performs the wake-up sequence by clearing LINC1R1[SLEEP] if LINC1R1[AWUM] is set. To exit from Sleep mode if LINC1R1[AWUM] is cleared, the software must clear LINC1R1[SLEEP] when a wake-up event occurs.

## 24.7 LIN modes

### 24.7.1 Master mode

In Master mode, the software uses the message buffer to handle the LIN messages.

Master mode is selected when LINC1R1[MME] is set.

#### 24.7.1.1 LIN header transmission

According to the LIN protocol, any communication on the LIN bus is triggered by the master sending a header. The header is transmitted by the master task while the data is transmitted by the slave task of a node.

To transmit a header with LINFLEXD the application must set up the identifier and the data field length, and configure the message (direction and checksum type) in the BIDR register before requesting the header transmission by setting LINC2R2[HTRQ].

### 24.7.1.2 Data transmission (transceiver as publisher)

When the master node is publisher of the data corresponding to the identifier sent in the header, then the slave task of the master has to send the data in the Response part of the LIN frame. Therefore, the software must provide the data to LINFlexD before requesting the header transmission. The software stores the data in the message buffer BDR. According to the data field length LINFlexD transmits the data and the checksum. The software uses the BIDR[CCS] bit to configure the checksum type (classic or enhanced) for each message.

The direction of the message buffer is controlled by the BIDR[DIR] bit. When the software sets this bit the response is sent by LINFlexD (publisher). Clearing this bit configures the message buffer as subscriber.

### 24.7.1.3 Data reception (transceiver as subscriber)

To receive data from a slave node, the master sends a header with the corresponding identifier. LINFlexD stores the data received from the slave in the message buffer and stores the message status in the LINSR.

### 24.7.1.4 Error detection and handling

LINFlexD is able to detect and handle LIN communication errors. A code stored in the LIN error status register (LINESR) signals the errors to the software.

Table 24-1 lists the errors detected in Master mode and the LINFlexD controller's response to these errors.

**Table 24-1. Errors in Master mode**

| Error                      | Description   | LINFlexD response to error  |
|----------------------------|---|---|
| Bit error                  | During transmission, the value read back from the bus differs from the transmitted value                                      | <ul style="list-style-type: none"> <li>Stops the transmission of the frame after the corrupted bit</li> <li>Generates an interrupt if LINIER[BEIE] is set</li> <li>Returns to idle state</li> </ul>             |
| Framing error              | A dominant state has been sampled on the stop bit of the currently received character (sync field, identifier, or data field) | If encountered during reception: <ul style="list-style-type: none"> <li>Discards the current frame</li> <li>Generates an interrupt if LINIER[FEIE] is set</li> <li>Returns immediately to idle state</li> </ul> |
| Checksum error             | The computed checksum does not match the received checksum  | If encountered during reception: <ul style="list-style-type: none"> <li>Discards the current frame</li> <li>Generates an interrupt if LINIER[CEIE] is set</li> <li>Returns to idle state</li> </ul>             |
| Response and frame timeout | Refer to <a href="#">Section 24.12.1, 8-bit timeout counter</a> , for more details  |   |

### 24.7.1.5 Overrun

After the message buffer is full, the next valid message reception causes an overrun and a message is lost. The LINFlexD controller sets LINSR[BOF] to signal the overrun condition. Which message is lost depends on the configuration of the RX message buffer:

- If the buffer lock function is disabled (LINC1[RBLM] cleared), the last message stored in the buffer is overwritten by the new incoming message. In this case, the latest message is always available to the software.
- If the buffer lock function is enabled (LINC1[RBLM] set), the most recent message is discarded and the previous message is available in the buffer.

## 24.7.2 Slave mode

In Slave mode the software uses the message buffer to handle the LIN messages.

Slave mode is selected when the LINC1[MME] is cleared.

### 24.7.2.1 Data transmission (transceiver as publisher)

When LINFlexD receives the identifier, an RX interrupt is generated. The software must:

- Read the received ID in the BDIR register
- Fill the BDR registers
- Specify the data field length using the BDIR[DFL] field
- Trigger the data transmission by setting LINC2[DTRQ]

One or several identifier filters can be configured for transmission by setting the DIR bits in the corresponding IFCR registers and activated by setting one or several bits in the IFER register.

When at least one identifier filter is configured in transmission and activated. If the received ID matches the filter, a specific TX interrupt is generated.

Typically, the software has to copy the data from RAM locations to the BDRL and BDRM registers. To copy the data to the right location, the software has to identify the data by means of the identifier. To avoid this and to ease the access to the RAM locations, the LINFlexD controller provides a Filter Match Index. This index value is the number of the filter that matched the received identifier.

The software can use the index in the IFMI register to directly access the pointer that points to the right data array in the RAM area and copy this data to the BDRL and BDRM registers (see [Figure 24-10](#)).

Using a filter avoids the software having to configure the direction, the data field length and the checksum type in the BDIR register. The software fills the BDRL and BDRM registers and triggers the data transmission by setting LINC2[DTRQ].

If LINFlexD cannot provide enough TX identifier filters to handle all identifiers the software has to transmit data for, then a filter can be configured in mask mode (refer to [Section 24.7.3, Slave mode with identifier filtering](#)) in order to manage several identifiers with one filter only.

### 24.7.2.2 Data reception (transceiver as subscriber)

When LINFlexD receives the identifier, an RX interrupt is generated. The software must:

- Read the received ID in the BDIR register
- Specify the data field length using the BDIR[DFL] field before the reception of the stop bit of the first byte of data field

When the checksum reception is completed, an RX interrupt is generated to allow the software to read the received data in the BDR registers.

One or several identifier filters can be configured for reception by clearing the DIR bit in the corresponding IFCR registers and activated by clearing one or several bits in the IFER register.

When at least one identifier filter is configured in reception and activated. If the received ID matches the filter, an RX interrupt is generated after the checksum reception only.

Typically, the software has to copy the data from the BDRL and BDRM registers to RAM locations. To copy the data to the right location, the software has to identify the data by means of the identifier. To avoid this and to ease the access to the RAM locations, the LINFlexD controller provides a Filter Match Index. This index value is the number of the filter that matched the received identifier.

The software can use the index in the IFMI register to directly access the pointer that points to the right data array in the RAM area and copy this data from the BDRL and BDRM registers to the RAM (see [Figure 24-10](#)).

Using a filter avoids the software reading the ID value in the BIDR register and configuring the direction, the data field length, and the checksum type in the BIDR register.

If LINFlexD cannot provide enough RX identifier filters to handle all identifiers the software has to receive the data for, then a filter can be configured in mask mode (refer to [Section 24.7.3, Slave mode with identifier filtering](#)) in order to manage several identifiers with one filter only.

### 24.7.2.3 Data discard

When LINFlexD receives the identifier, an RX interrupt is generated. If the received identifier does not concern the node, the software must set LINCR2[DDRQ]. LINFlexD returns to idle state.

### 24.7.2.4 Error detection and handling

[Table 24-2](#) lists the errors detected in Slave mode and the LINFlexD controller's response to these errors.

**Table 24-2. Errors in Slave mode**

| Error         | Description   | LINFlexD response to error  |
|---------------|---|---|
| Bit error     | During transmission, the value read back from the bus differs from the transmitted value                                      | <ul style="list-style-type: none"> <li>Stops the transmission of the frame after the corrupted bit</li> <li>Generates an interrupt if LINIER[BEIE] is set</li> <li>Returns to idle state</li> </ul>             |
| Framing error | A dominant state has been sampled on the stop bit of the currently received character (sync field, identifier, or data field) | If encountered during reception: <ul style="list-style-type: none"> <li>Discards the current frame</li> <li>Generates an interrupt if LINIER[FEIE] is set</li> <li>Returns immediately to idle state</li> </ul> |

Table 24-2. Errors in Slave mode (continued)

| Error          | Description  | LINFlexD response to error   |
|----------------|--|--|
| Checksum error | The computed checksum does not match the received checksum   | If encountered during reception: <ul style="list-style-type: none"> <li>Discards the received frame</li> <li>Generates an interrupt if LINIER[CEIE] is set</li> <li>Returns to idle state</li> </ul>   |
| Header error   | An error occurred during header reception (break delimiter error, inconsistent sync field, header timeout) | If encountered during header reception, a break field error, an inconsistent sync field, or a timeout: <ul style="list-style-type: none"> <li>Discards the header</li> <li>Generates an interrupt if LINIER[HEIE] is set</li> <li>Returns to idle state</li> </ul> |

### 24.7.2.5 Valid header

A received header is considered as valid when it has been received correctly according to the LIN protocol.

If a valid break field and break delimiter come before the end of the current header, or at any time during a data field, the current header or data is discarded and the state machine synchronizes on this new break.

### 24.7.2.6 Valid message

A received or transmitted message is considered as valid when the data has been received or transmitted without error according to the LIN protocol.

### 24.7.2.7 Overrun

After the message buffer is full, the next valid message reception causes an overrun and a message is lost. The LINFlexD controller sets LINSR[BOF] to signal the overrun condition. Which message is lost depends on the configuration of the RX message buffer:

- If the buffer lock function is disabled (LINCRI[RBLM] cleared), the last message stored in the buffer is overwritten by the new incoming message. In this case, the latest message is always available to the software.
- If the buffer lock function is enabled (LINCRI[RBLM] set), the most recent message is discarded and the previous message is available in the buffer.

## 24.7.3 Slave mode with identifier filtering

In the LIN protocol, the identifier of a message is not associated with the address of a node but related to the content of the message. Consequently a transmitter broadcasts its message to all receivers. When a slave node receives a header, it decides—depending on the identifier value—whether the software needs to receive or send a response. If the message does not target the node, it must be discarded without software intervention.

To fulfill this requirement, the LINFlexD controller provides configurable filters in order to request software intervention only if needed. This hardware filtering saves CPU resources, which would otherwise be needed by software for filtering.

The filtering is accomplished through the use of IFCR registers. These registers have the names IFCR0 through IFCR. This section also uses the nomenclature  $IFCR_{2n}$  and  $IFCR_{2n+1}$ ; in this nomenclature,  $n$  is an integer, and the corresponding IFCR register is calculated using the formula in the subscript.

### 24.7.3.1 Filter submodes

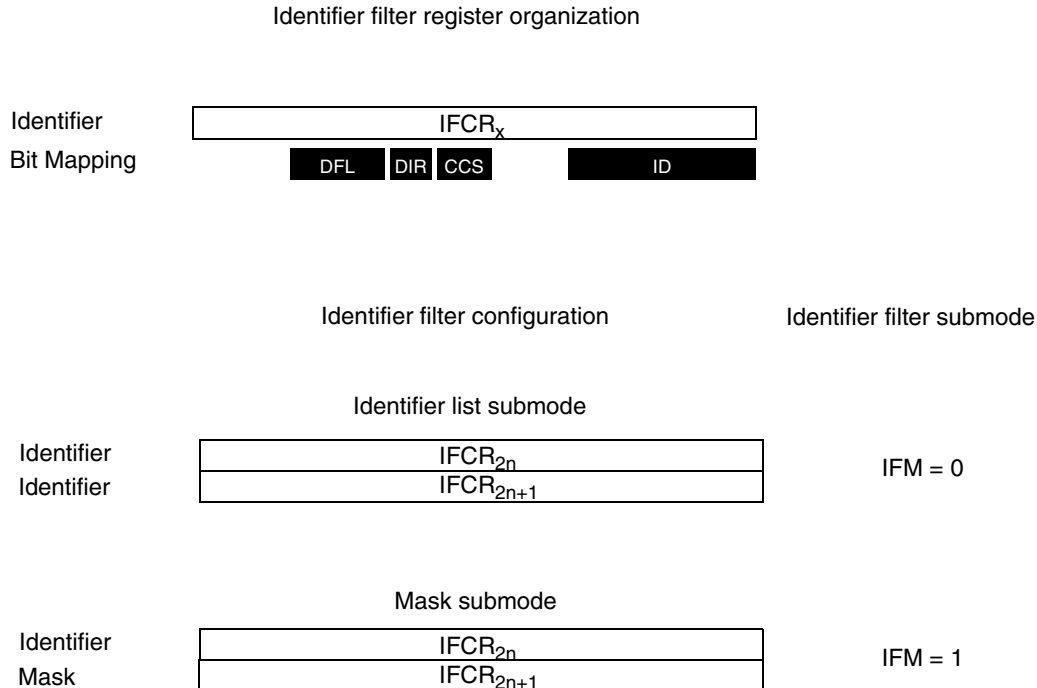
Usually each of the 16 IFCRs is used to filter one dedicated identifier, but this means that the LINFlexD controller could filter a maximum of 16 identifiers. In order to be able to handle more identifiers, the filters can be configured to operate as masks.

Table 24-3 describes the two available filter submodes.

**Table 24-3. Filter submodes**

| Submode         | Description  |
|-----------------|--|
| Identifier list | Both filter registers are used as identifier registers. All bits of the incoming identifier must match the bits specified in the filter register. This is the default submode for the LINFlexD controller. |
| Mask            | The identifier registers are associated with mask registers specifying which bits of the identifier are handled as “must match” or as “don’t care”.  |

The bit mapping and register organization in these two submodes is shown in Figure 24-9.



**Figure 24-9. Filter configuration—register organization**

### 24.7.3.2 Identifier filter submode configuration

The identifier filters are configured in the IFCR registers. To configure an identifier filter the filter must first be activated by setting the corresponding bit in the IFER[FACT] field. The submode (identifier list or mask) for the corresponding IFCR register is configured by the IFMR[IFM] field. For each filter, the IFCR register is used to configure:

- The ID or mask
- The direction (TX or RX)
- The data field length
- The checksum type

If no filter is active, an RX interrupt is generated on any received identifier event.

If at least one active filter is configured as TX, all received identifiers matching this filter generate a TX interrupt.

If at least one active filter is configured as RX, all received identifiers matching this filter generate an RX interrupt.

If no active filter is configured as RX, all received identifiers not matching TX filter(s) generate an RX interrupt.

Further details are provided in [Table 24-4](#) and [Figure 24-10](#).

**Table 24-4. Filter to interrupt vector correlation**

| Number of active filters | Number of active filters configured as TX | Number of active filters configured as RX | Interrupt vector   |
|--------------------------|---|---|--|
| 0                        | 0   | 0   | RX interrupt on all IDs  |
| a<br>(a > 0)             | a   | 0   | <ul style="list-style-type: none"> <li>• TX interrupt on IDs matching the filters</li> <li>• RX interrupt on all other IDs if BF bit is set, no RX interrupt if BF bit is reset</li> </ul>               |
| n<br>(n = a + b)         | a<br>(a > 0)                              | b<br>(b > 0)                              | <ul style="list-style-type: none"> <li>• TX interrupt on IDs matching the TX filters</li> <li>• RX interrupt on IDs matching the RX filters</li> <li>• All other IDs discarded (no interrupt)</li> </ul> |
| b<br>(b > 0)             | 0   | b   | <ul style="list-style-type: none"> <li>• RX interrupt on IDs matching the filters</li> <li>• TX interrupt on all other IDs if BF bit is set, no TX interrupt if BF bit is reset</li> </ul>               |



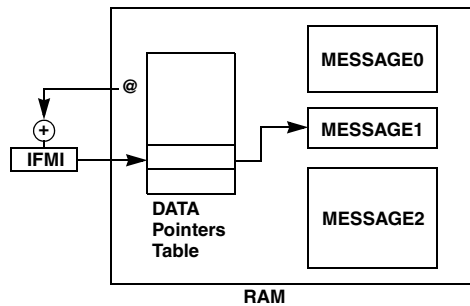


Figure 24-10. Identifier match index

## 24.7.4 Slave mode with automatic resynchronization

Automatic resynchronization must be enabled in Slave mode if  $f_{\text{ipg\_clock\_lin}}$  tolerance is greater than 1.5%. This feature compensates a deviation up to 14%, as specified in the LIN standard.

This mode is similar to Slave mode as described in [Section 24.7.2, Slave mode](#), with the addition of automatic resynchronization enabled by the LINCR1[LASE] bit. In this mode LINFlexD adjusts the fractional baud rate generator after each synch field reception.

### 24.7.4.1 Automatic resynchronization method

When automatic resynchronization is enabled, after each LIN break, the time duration between five falling edges on RDI is sampled as shown in [Figure 24-11](#). Then the LFDIV value (and its associated LINIBRR and LINFBR registers) is automatically updated at the end of the fifth falling edge. During LIN sync field measurement, the LINFlexD state machine is stopped and no data is transferred to the data register.

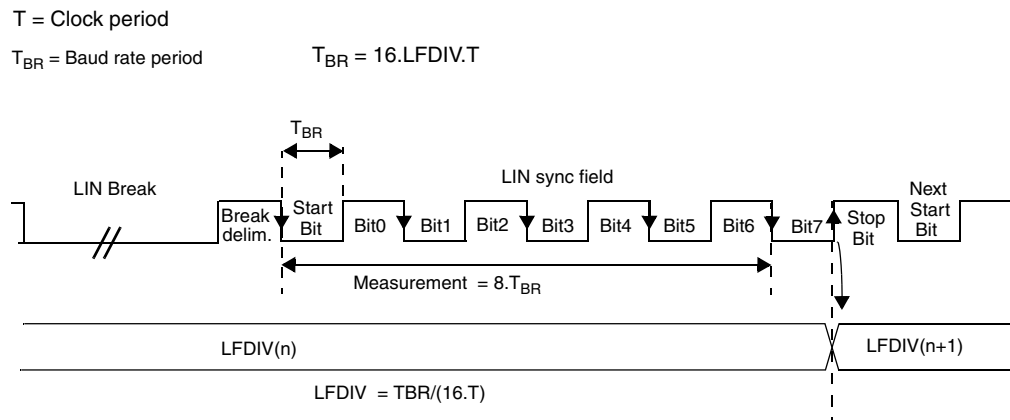


Figure 24-11. LIN sync field measurement

LFDIV is an unsigned fixed point number. The mantissa is coded on 20 bits in the LINIBRR register and the fraction is coded on 4 bits in the LINFBR register.

If LINCR1[LASE] is set, LFDIV is automatically updated at the end of each LIN sync field.

Three registers are used internally to manage the auto-update of the LINFlexD divider (LFDIV):

- LFDIV\_NOM (nominal value written by software at LINIBRR and LINFBR addresses)
- LFDIV\_MEAS (results of the Field Synch measurement)
- LFDIV (used to generate the local baud rate)

On transition to idle, break or break delimiter state due to any error or on reception of a complete frame, hardware reloads LFDIV with LFDIV\_NOM.

#### 24.7.4.2 Deviation error on the sync field

The deviation error is checked by comparing the current baud rate (relative to the slave oscillator) with the received LIN sync field (relative to the master oscillator). Two checks are performed in parallel.

The first check is based on a measurement between the first falling edge and the last falling edge of the sync field:

- If  $D1 > 14.84\%$ , LHE is set.
- If  $D1 < 14.06\%$ , LHE is not set.
- If  $14.06\% < D1 < 14.84\%$ , LHE can be either set or reset depending on the dephasing between the signal on LINFlexD\_RX pin the  $f_{ipg\_clock\_lin}$  clock.

The second check is based on a measurement of time between each falling edge of the sync field:

- If  $D2 > 18.75\%$ , LHE is set.
- If  $D2 < 15.62\%$ , LHE is not set.
- If  $15.62\% < D2 < 18.75\%$ , LHE can be either set or reset depending on the dephasing between the signal on LINFlexD\_RX pin the  $f_{ipg\_clock\_lin}$  clock.

Note that the LINFlexD does not need to check if the next edge occurs slower than expected. This is covered by the check for deviation error on the full synch byte.

## 24.8 Test modes

The LINFlexD controller includes two test modes, Loop Back mode and Self Test mode. They can be selected by the LBKM and SFTM bits in the LINCR1 register. These bits must be configured while LINFlexD is in Initialization mode. After one of the two test modes has been selected, LINFlexD must be started in Normal mode.

### 24.8.1 Loop Back mode

LINFlexD can be put in Loop Back mode by setting LINCR1[LBKM]. In Loop Back mode, the LINFlexD treats its own transmitted messages as received messages. This is illustrated in [Figure 24-12](#).

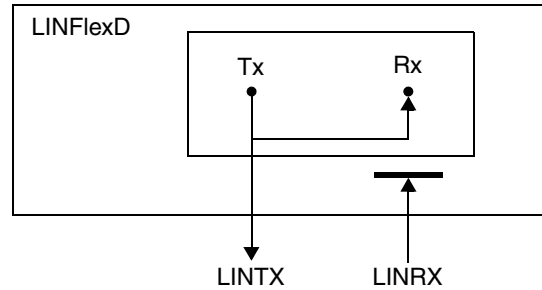


Figure 24-12. LINFlexD in Loop Back mode

This mode is provided for self-test functions. To be independent of external events, the LIN core ignores the LINRX signal. In this mode, the LINFlexD performs an internal feedback from its Tx output to its Rx input. The actual value of the LINRX input pin is disregarded by the LINFlexD. The transmitted messages can be monitored on the LINTX pin.

## 24.8.2 Self Test mode

LINFlexD can be put in Self Test mode by setting LINC1[LBKM] and LINC1[SFTM]. This mode can be used for a Hot Self Test, meaning the LINFlexD can be tested as in Loop Back mode but without affecting a running LIN system connected to the LINTX and LINRX pins. In this mode, the LINRX pin is disconnected from the LINFlexD and the LINTX pin is held recessive. This is illustrated in Figure 24-13.

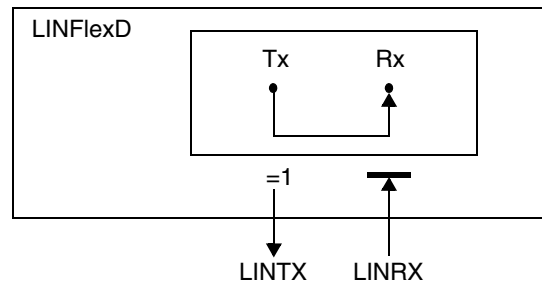


Figure 24-13. LINFlexD in Self Test mode

## 24.9 UART mode

The main features of UART mode are presented in [Section 24.2.2, UART mode features](#).

### 24.9.1 Data frame structure

#### 24.9.1.1 8-bit data frame

The 8-bit UART data frame is shown in [Figure 24-14](#). The 8th bit can be a data or a parity bit. Parity (even, odd, 0, or 1) can be selected by the UARTCR[PC] field. An even parity is set if the modulo-2 sum of the 7 data bits is 1. An odd parity is cleared in this case.

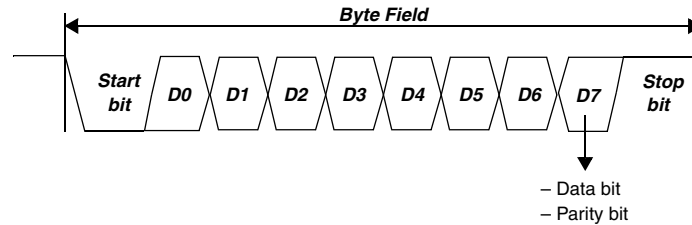


Figure 24-14. UART mode 8-bit data frame

### 24.9.1.2 9-bit data frame

The 9-bit UART data frame is shown in Figure 24-15. The 9th bit is a parity bit. Parity (even, odd, 0, or 1) can be selected by the by the UARTCR[PC] field. An even parity is set if the modulo-2 sum of the 8 data bits is 1. An odd parity is cleared in this case. Parity 0 forces a zero logical value. Parity 1 forces a high logical value.

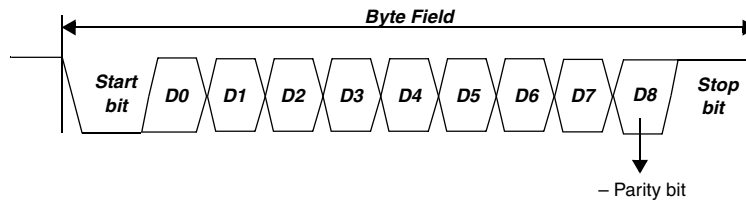


Figure 24-15. UART mode 9-bit data frame

### 24.9.1.3 16-bit data frame

The 16-bit UART data frame is shown in Figure 24-16. The 16th bit can be a data or a parity bit. Parity (even, odd, 0, or 1) can be selected by the UARTCR[PC] field. Parity 0 forces a zero logical value. Parity 1 forces a high logical value.

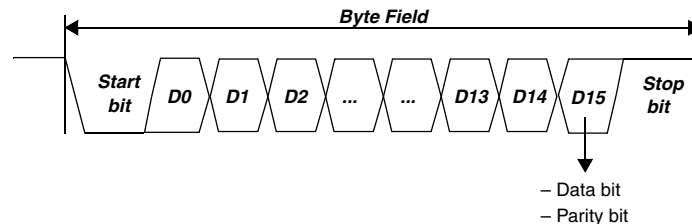


Figure 24-16. UART mode 16-bit data frame

### 24.9.1.4 17-bit data frame

The 17-bit UART data frame is shown in Figure 24-17. The 17th bit is the parity bit. Parity (even, odd, 0, or 1) can be selected by the UARTCR[PC] field. Parity 0 forces a zero logical value. Parity 1 forces a high logical value.

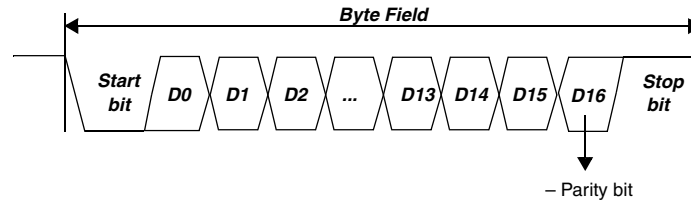


Figure 24-17. UART mode 17-bit data frame

## 24.9.2 Buffer

The 8-byte buffer is divided into two parts—one for receiver and one for transmitter—as shown in [Table 24-5](#).

Table 24-5. UART buffer structure

| BDR | UART mode |
|-----|-----------|
| 0   | Tx0       |
| 1   | Tx1       |
| 2   | Tx2       |
| 3   | Tx3       |
| 4   | Rx0       |
| 5   | Rx1       |
| 6   | Rx2       |
| 7   | Rx3       |

For 16-bit frames, the lower 8 bits will be written in BDR0 and the upper 8 bits will be written in BDR1.

## 24.9.3 UART transmitter

In order to start transmission in UART mode, the UARTCR[UART] and UARTCR[TXEN] bits must be set. Transmission starts when BDR0 (least significant data byte) is programmed. The number of bytes transmitted is equal to the value configured by the UARTCR[TDFLTFC] field (see [Table 24-16](#)).

The Transmit buffer size is as follows:

- 4 bytes when UARTCR[WL1] = 0
- 2 half-words when UARTCR[WL1] = 1

Therefore, the maximum transmission that can be triggered is 4 bytes (2 half-words). After the programmed number of bytes has been transmitted, the UARTSR[DTFTFF] flag is set. If the UARTCR[TXEN] field is cleared during a transmission, the current transmission is completed, but no further transmission can be invoked. The buffer can be configured in FIFO mode (mandatory when DMA Tx is enabled) by setting UARTCR[TFBM].

The access to the BDRL register is shown in [Table 24-6](#).

Table 24-6. BDRL access in UART mode

| Access             | Mode <sup>1</sup> | Word length <sup>2</sup> | IPS operation result |
|--------------------|-------------------|--------------------------|----------------------|
| Write Byte0        | FIFO              | Byte                     | OK                   |
| Write Byte1-2-3    | FIFO              | Byte                     | IPS transfer error   |
| Write Half-word0-1 | FIFO              | Byte                     | IPS transfer error   |
| Write Word         | FIFO              | Byte                     | IPS transfer error   |
| Write Byte0-1-2-3  | FIFO              | Half-word                | IPS transfer error   |
| Write Half-word0   | FIFO              | Half-word                | OK                   |
| Write Half-word1   | FIFO              | Half-word                | IPS transfer error   |
| Write Word         | FIFO              | Half-word                | IPS transfer error   |
| Read Byte0-1-2-3   | FIFO              | Byte/Half-word           | IPS transfer error   |
| Read Half-word0-1  | FIFO              | Byte/Half-word           | IPS transfer error   |
| Read Word          | FIFO              | Byte/Half-word           | IPS transfer error   |
| Write Byte0-1-2-3  | BUFFER            | Byte/Half-word           | OK                   |
| Write Half-word0-1 | BUFFER            | Byte/Half-word           | OK                   |
| Write Word         | BUFFER            | Byte/Half-word           | OK                   |
| Read Byte0-1-2-3   | BUFFER            | Byte/Half-word           | OK                   |
| Read Half-word0-1  | BUFFER            | Byte/Half-word           | OK                   |
| Read Word          | BUFFER            | Byte/Half-word           | OK                   |

<sup>1</sup> As specified by UARTCR[TFBM]

<sup>2</sup> As specified by the WL1 and WL0 bits of the UARTCR register. In UART FIFO mode (UARTCR[TFBM] = 1), any read operation causes an IPS transfer error.

## 24.9.4 UART receiver

Reception of a data byte is started as soon as the software completes the following tasks in order:

1. Exits Initialization mode
2. Sets the UARTCR[RXEN] field
3. Detects the start bit

There is a dedicated data buffer for received data bytes. Its size is as follows:

- 4 bytes when UARTCR[WL1] = 0
- 2 half-words when UARTCR[WL1] = 1

After the programmed number (RDFL bits) of bytes has been received, the UARTSR[DRFRFE] field is set. If the UARTCR[RXEN] field is cleared during a reception, the current reception is completed, but no further reception can be invoked until UARTCR[RXEN] is set again.

The buffer can be configured in FIFO mode (required when DMA Rx is enabled) by setting UARTCR[RFBM].

The access to the BDRM register is shown in [Table 24-7](#).

**Table 24-7. BDRM access in UART mode**

| Access             | Mode <sup>1</sup> | Word length <sup>2</sup> | IPS operation result |
|--------------------|-------------------|--------------------------|----------------------|
| Read Byte4         | FIFO              | Byte                     | OK                   |
| Read Byte5-6-7     | FIFO              | Byte                     | IPS transfer error   |
| Read Half-word2-3  | FIFO              | Byte                     | IPS transfer error   |
| Read Word          | FIFO              | Byte                     | IPS transfer error   |
| Read Byte4-5-6-7   | FIFO              | Half-word                | IPS transfer error   |
| Read Half-word2    | FIFO              | Half-word                | OK                   |
| Read Half-word3    | FIFO              | Half-word                | IPS transfer error   |
| Read Word          | FIFO              | Half-word                | IPS transfer error   |
| Write Byte4-5-6-7  | FIFO              | Byte/Half-word           | IPS transfer error   |
| Write Half-word2-3 | FIFO              | Byte/Half-word           | IPS transfer error   |
| Write Word         | FIFO              | Byte/Half-word           | IPS transfer error   |
| Read Byte4-5-6-7   | BUFFER            | Byte/Half-word           | OK                   |
| Read Half-word2-3  | BUFFER            | Byte/Half-word           | OK                   |
| Read Word          | BUFFER            | Byte/Half-word           | OK                   |
| Write Byte4-5-6-7  | BUFFER            | Byte/Half-word           | IPS transfer error   |
| Write Half-word2-3 | BUFFER            | Byte/Half-word           | IPS transfer error   |
| Write Word         | BUFFER            | Byte/Half-word           | IPS transfer error   |

<sup>1</sup> As specified by UARTCR[RFBM]

<sup>2</sup> As specified by the WL1 and WL0 bits of the UARTCR register

[Table 24-8](#) lists some common scenarios, controller responses, and suggestions when the LINFlexD controller is acting as a UART receiver.

**Table 24-8. UART receiver scenarios**

| Scenario  | Responses and suggestions  |
|---|--|
| The software does not know (in advance) how many bytes will be received.  | Do not program UARTCR[RDFLRFC] in advance. When this field is zero (as it is after reset), reception occurs on a byte-by-byte basis. Therefore, the state machine will move to IDLE state after each byte is received. |
| UARTCR[RDFLRFC] is programmed for a certain number of bytes received, but the actual number of bytes received is smaller. | The reception will hang. In this case, the software must monitor the UARTSR[TO] field, and move to IDLE state by setting LINCR1[SLEEP].  |

Table 24-8. UART receiver scenarios (continued)

| Scenario   | Responses and suggestions   |
|--|---|
| A STOP request arrives before the reception is completed.  | The request is acknowledged only after the programmed number of data bytes are received. In other words, the STOP request is not serviced immediately. In this case, the software must monitor the UARTSR[TO] field and move the state machine to IDLE state as appropriate. The stop request will be serviced only after this is complete. |
| A parity error occurs during the reception of a byte.  | The corresponding UARTSR[PE $n$ ] field is set. No interrupt is generated.  |
| A framing error occurs during the reception of a byte.   | <ul style="list-style-type: none"> <li>• UARTSR[FE] is set.</li> <li>• If LINIER[FEIE] = 1, an interrupt is generated. This interrupt is helpful in identifying which byte has the framing error, since there is only one register bit for framing errors.</li> </ul>   |
| A new byte has been received, but the last received frame has not been read from the buffer (UARTSR[RMB] has not yet been cleared by the software) | <ul style="list-style-type: none"> <li>• An overrun error will occur (UARTSR[BOF] will be set).</li> <li>• One message will be lost (depending on the setting of LINCR[RBLM]).</li> <li>• An interrupt is generated if LINIER[BOIE] is set.</li> </ul>  |

## 24.10 Memory map and register description

### 24.10.1 LIN control register 1 (LINCR1)

Offset:0x00 Access: User read/write

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       |                  |                  |                   |                   |                  |    |    |    |                 |                               |                   |                  |                   |                   |       |      |
|-------|------------------|------------------|-------------------|-------------------|------------------|----|----|----|-----------------|-------------------------------|-------------------|------------------|-------------------|-------------------|-------|------|
|       | 16               | 17               | 18                | 19                | 20               | 21 | 22 | 23 | 24              | 25                            | 26                | 27               | 28                | 29                | 30    | 31   |
| R     | CCD <sub>1</sub> | CFD <sub>1</sub> | LASE <sup>1</sup> | AWUM <sup>1</sup> | MBL <sup>1</sup> |    |    |    | BF <sup>1</sup> | SFT <sub>M</sub> <sup>1</sup> | LBKM <sup>1</sup> | MME <sup>1</sup> | SBDT <sup>1</sup> | RBLM <sup>1</sup> | SLEEP | INIT |
| W     |                  |                  |                   |                   |                  |    |    |    |                 |                               |                   |                  |                   |                   |       |      |
| Reset | 0                | 0                | 0                 | 0                 | 0                | 0  | 0  | 0  | 1               | 0                             | 0                 | 0/1 <sup>2</sup> | 0                 | 0                 | 1     | 0    |

<sup>1</sup> These fields are writable only in Initialization mode (LINCR1[INIT] = 1).

<sup>2</sup> Resets to 0 in Slave mode and to 1 in Master mode

Figure 24-18. LIN control register 1 (LINCR1)



Table 24-9. LINC1 field descriptions

| Field | Description  |
|-------|--|
| CCD   | Checksum Calculation disable<br>This bit is used to disable the checksum calculation (see <a href="#">Table 24-10</a> ).<br>0: Checksum calculation is done by hardware. When this bit is reset the LINC1FR register is read-only.<br>1: Checksum calculation is disabled. When this bit is set the LINC1FR register is read/write. User can program this register to send a software calculated CRC (provided CFD is reset).<br>Note: This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode. |
| CFD   | Checksum field disable<br>This bit is used to disable the checksum field transmission (see <a href="#">Table 24-10</a> ).<br>0: Checksum field is sent after the required number of data bytes is sent.<br>1: No checksum field is sent.<br>Note: This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode.  |
| LASE  | LIN Slave Automatic Resynchronization Enable<br>0: Automatic resynchronization disable<br>1: Automatic resynchronization enable<br>Note: This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode.   |
| AWUM  | Automatic Wake-Up Mode<br>This bit controls the behavior of the LINFlexD hardware during Sleep mode.<br>0: The Sleep mode is exited on software request by clearing the SLEEP bit of the LINC1R register.<br>1: The Sleep mode is exited automatically by hardware on RX dominant state detection. The SLEEP bit of the LINC1R register is cleared by hardware whenever WUF bit in LINSR is set.<br>Note: This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode.                              |
| MBL   | LIN Master Break Length<br>These bits indicate the Break length in Master mode (see <a href="#">Table 24-11</a> ).<br>Note: These bits can be written in Initialization mode only. They are read-only in Normal or Sleep mode.   |
| BF    | Bypass filter<br>0: No interrupt if ID does not match any filter<br>1: An RX interrupt is generated on ID not matching any filter<br>Notes:<br><ul style="list-style-type: none"> <li>If no filter is activated, this bit is reserved.</li> <li>This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode.</li> </ul>   |
| SFTM  | Self Test Mode<br>This bit controls the Self Test mode. For more details please refer to <a href="#">Section 24.8.2, Self Test mode</a> .<br>0: Self Test mode disable<br>1: Self Test mode enable<br>Note: This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode.  |
| LBKM  | Loop Back Mode<br>This bit controls the Loop Back mode. For more details please refer to <a href="#">Section 24.8.1, Loop Back mode</a> .<br>0: Loop Back mode disable<br>1: Loop Back mode enable<br>Note: This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode   |
| MME   | Master Mode Enable<br>0: Slave mode enable<br>1: Master mode enable<br>Note: This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode.   |

**Table 24-9. LINCR1 field descriptions (continued)**

| Field | Description   |
|-------|---|
| SBDT  | Slave Mode Break Detection Threshold<br>0: 11-bit break<br>1: 10-bit break<br>Note: This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode.   |
| RBLM  | Receive Buffer Locked Mode<br>0: Receive Buffer not locked on overrun. Once the Slave Receive Buffer is full the next incoming message overwrites the previous one.<br>1: Receive Buffer locked against overrun. Once the Receive Buffer is full the next incoming message is discarded.<br>Note: This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode. |
| SLEEP | Sleep Mode Request<br>This bit is set by software to request LINFlexD to enter Sleep mode.<br>This bit is cleared by software to exit Sleep mode or by hardware if the AWUM bit in LINCR1 and the WUF bit in LINSR are set (see <a href="#">Table 24-12</a> ).  |
| INIT  | Initialization Request<br>The software sets this bit to switch hardware into Initialization mode. If the SLEEP bit is reset, LINFlexD enters Normal mode when clearing the INIT bit (see <a href="#">Table 24-12</a> ).   |

**Table 24-10. Checksum bits configuration**

| CFD | CCD | LINCFR     | Checksum sent                        |
|-----|-----|------------|--------------------------------------|
| 1   | 1   | Read/Write | None                                 |
| 1   | 0   | Read-only  | None                                 |
| 0   | 1   | Read/Write | Programmed in LINCFR by bits CF[0:7] |
| 0   | 0   | Read-only  | Hardware calculated                  |

**Table 24-11. LIN master break length selection**

| MBL  | Length |
|------|--------|
| 0000 | 10-bit |
| 0001 | 11-bit |
| 0010 | 12-bit |
| 0011 | 13-bit |
| 0100 | 14-bit |
| 0101 | 15-bit |
| 0110 | 16-bit |
| 0111 | 17-bit |
| 1000 | 18-bit |
| 1001 | 19-bit |
| 1010 | 20-bit |

Table 24-11. LIN master break length selection (continued)

| MBL  | Length |
|------|--------|
| 1011 | 21-bit |
| 1100 | 22-bit |
| 1101 | 23-bit |
| 1110 | 36-bit |
| 1111 | 50-bit |

Table 24-12. Operating mode selection

| SLEEP | INIT | Operating mode      |
|-------|------|---------------------|
| 1     | 0    | Sleep (reset value) |
| x     | 1    | Initialization      |
| 0     | 0    | Normal              |

## 24.10.2 LIN interrupt enable register (LINIER)

Offset: 0x04

Access: User read/write

|       |      |      |      |      |      |    |    |      |      |      |       |       |           |      |      |      |
|-------|------|------|------|------|------|----|----|------|------|------|-------|-------|-----------|------|------|------|
|       | 0    | 1    | 2    | 3    | 4    | 5  | 6  | 7    | 8    | 9    | 10    | 11    | 12        | 13   | 14   | 15   |
| R     | 0    | 0    | 0    | 0    | 0    | 0  | 0  | 0    | 0    | 0    | 0     | 0     | 0         | 0    | 0    | 0    |
| W     |      |      |      |      |      |    |    |      |      |      |       |       |           |      |      |      |
| Reset | 0    | 0    | 0    | 0    | 0    | 0  | 0  | 0    | 0    | 0    | 0     | 0     | 0         | 0    | 0    | 0    |
|       | 16   | 17   | 18   | 19   | 20   | 21 | 22 | 23   | 24   | 25   | 26    | 27    | 28        | 29   | 30   | 31   |
| R     | SZIE | OCIE | BEIE | CEIE | HEIE | 0  | 0  | FEIE | BOIE | LSIE | WUJIE | DBFIE | DBEIETOIE | DRIE | DTIE | HRIE |
| W     | w1c  | w1c  | w1c  | w1c  | w1c  |    |    | w1c  | w1c  | w1c  | w1c   | w1c   | w1c       | w1c  | w1c  | w1c  |
| Reset | 0    | 0    | 0    | 0    | 0    | 0  | 0  | 0    | 0    | 0    | 0     | 0     | 0         | 0    | 0    | 0    |

Figure 24-19. LIN interrupt enable register (LINIER)

Table 24-13. LINIER field descriptions

| Field | Description   |
|-------|---|
| SZIE  | Stuck at Zero Interrupt Enable<br>0: No interrupt when SZF bit in LINESR or UARTSR is set<br>1: Interrupt generated when SZF bit in LINESR or UARTSR is set |

Table 24-13. LINIER field descriptions (continued)

| Field     | Description   |
|-----------|---|
| OCIE      | Output Compare Interrupt Enable<br>0: No interrupt when OCF bit in LINESR or UARTSR is set<br>1: Interrupt generated when OCF bit in LINESR or UARTSR is set  |
| BEIE      | Bit Error Interrupt Enable<br>0: No interrupt when BEF bit in LINESR is set<br>1: Interrupt generated when BEF bit in LINESR is set   |
| CEIE      | Checksum Error Interrupt Enable<br>0: No interrupt on Checksum error<br>1: Interrupt generated when checksum error flag (CEF) is set in LINESR  |
| HEIE      | Header Error Interrupt Enable<br>0: No interrupt on Break Delimiter error, Synch Field error, ID field error<br>1: Interrupt generated on Break Delimiter error, Synch Field error, ID field error  |
| FEIE      | Framing Error Interrupt Enable<br>0: No interrupt on Framing error<br>1: Interrupt generated on Framing error   |
| BOIE      | Buffer Overrun Interrupt Enable<br>0: No interrupt on Buffer overrun<br>1: Interrupt generated on Buffer overrun  |
| LSIE      | LIN State Interrupt Enable<br>0: No interrupt on LIN state change<br>1: Interrupt generated on LIN state change<br>This interrupt can be used for debugging purposes. It has no status flag but is reset when writing 1111 into the LIN state bits in the LINSR register.   |
| WUIE      | Wake-up Interrupt Enable<br>0: No interrupt when WUF bit in LINSR or UARTSR is set<br>1: Interrupt generated when WUF bit in LINSR or UARTSR is set   |
| DBFIE     | Data Buffer Full Interrupt Enable<br>0: No interrupt when buffer data register is full<br>1: Interrupt generated when data buffer register is full  |
| DBEIETOIE | Data Buffer Empty Interrupt Enable / Timeout Interrupt Enable<br>0: No interrupt when buffer data register is empty<br>1: Interrupt generated when data buffer register is empty<br><b>Note:</b> An interrupt is generated if this bit is set and one of the following is true:<br>LINFlexD is in LIN mode and LINSR[DBEF] is set<br>LINFlexD is in UART mode and UARTSR[TO] is set |
| DRIE      | Data Reception Complete Interrupt Enable<br>0: No interrupt when data reception is completed<br>1: Interrupt generated when data received flag (DRF) in LINSR or UARTSR is set  |
| DTIE      | Data Transmitted Interrupt Enable<br>0: No interrupt when data transmission is completed<br>1: Interrupt generated when data transmitted flag (DTF) is set in LINSR or UARTSR register  |
| HRIE      | Header Received Interrupt Enable<br>0: No interrupt when a valid LIN header has been received<br>1: Interrupt generated when a valid LIN header has been received, that is, HRF bit in LINSR register is set  |

### 24.10.3 LIN status register (LINSR)

Offset: 0x08

Access: User read/write

|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       | 16   | 17 | 18 | 19 | 20 | 21 | 22  | 23 | 24   | 25  | 26  | 27   | 28   | 29  | 30  | 31  |
|-------|------|----|----|----|----|----|-----|----|------|-----|-----|------|------|-----|-----|-----|
| R     | LINS |    |    |    | 0  | 0  | RMB | 0  | RBSY | RPS | WUF | DBIF | DBEF | DRF | DTF | HRF |
| W     | w1c  |    |    |    |    |    | w1c |    | w1c  | w1c | w1c | w1c  | w1c  | w1c | w1c | w1c |
| Reset | 0    | 0  | 0  | 0  | 0  | 0  | 0   | 0  | 0    | 1   | 0   | 0    | 0    | 0   | 0   | 0   |

Figure 24-20. LIN status register (LINSR)

Table 24-14. LINSR field descriptions

| Field | Description  |
|-------|--|
| LINS  | <p>LIN state<br/>LIN mode states description</p> <p><b>0000: Sleep mode</b><br/>LINFlexD is in Sleep mode to save power consumption.</p> <p><b>0001: Initialization mode</b><br/>LINFlexD is in Initialization mode.</p> <p><b>0010: Idle</b><br/>This state is entered on several events:</p> <ul style="list-style-type: none"> <li>• SLEEP bit and INIT in LINCR1 register have been cleared by software,</li> <li>• A falling edge has been received on RX pin and AWUM bit is set,</li> <li>• The previous frame reception or transmission has been completed or aborted.</li> </ul> <p><b>0011: Break</b><br/>In Slave mode, a falling edge followed by a dominant state has been detected. Receiving Break.<br/>Note: In Slave mode, in case of error new LIN state can be either Idle or Break depending on last bit state. If last bit is dominant new LIN state is Break, otherwise Idle.<br/>In Master mode, Break transmission ongoing.</p> <p><b>0100: Break Delimiter</b><br/>In Slave mode, a valid Break has been detected. Refer to LINCR1 register for break length configuration (10-bit or 11-bit). Waiting for a rising edge.<br/>In Master mode, Break transmission has been completed. Break Delimiter transmission is ongoing.</p> <p><b>0101: Synch Field</b><br/>In Slave mode, a valid Break Delimiter has been detected (recessive state for at least one bit time). Receiving Synch Field.<br/>In Master mode, Synch Field transmission is ongoing.</p> <p><b>0110: Identifier Field</b><br/>In Slave mode, a valid Synch Field has been received. Receiving ID Field.<br/>In Master mode, identifier transmission is ongoing.</p> <p><b>0111: Header reception/transmission completed</b><br/>In Slave mode, a valid header has been received and identifier field is available in the BIDR register.<br/>In Master mode, header transmission is completed.</p> <p><b>1000: Data reception/transmission</b><br/>Response reception/transmission is ongoing.</p> <p><b>1001: Checksum</b><br/>Data reception/transmission completed. Checksum reception/transmission ongoing.<br/>In UART mode, only the following states are flagged by the LIN state bits:</p> <ul style="list-style-type: none"> <li>• Init</li> <li>• Sleep</li> <li>• Idle</li> <li>• Data transmission/reception</li> </ul> |
| RMB   | <p>Release Message Buffer</p> <p>0: Buffer is free<br/>1: Buffer ready to be read by software. This bit must be cleared by software after reading data received in the buffer.<br/>This bit is cleared by hardware in Initialization mode.</p>   |
| RBSY  | <p>Receiver Busy Flag</p> <p>0: Receiver is Idle<br/>1: Reception ongoing</p> <p>Note: In Slave mode, after header reception, if DIR bit in BIDR is reset and reception starts then this bit is set. In this case, user cannot set DTRQ bit in LINCR2.</p>   |
| RPS   | <p>LIN receive pin state<br/>This bit reflects the current status of LINRX pin for diagnostic purposes.</p>  |

Table 24-14. LINSR field descriptions (continued)

| Field | Description   |
|-------|---|
| WUF   | <p>Wake-up Flag</p> <p>This bit is set by hardware and indicates to the software that LINFlexD has detected a falling edge on the LINRX pin when</p> <ul style="list-style-type: none"> <li>• slave is in Sleep mode,</li> <li>• master is in Sleep mode or idle state.</li> </ul> <p>This bit must be cleared by software. It is reset by hardware in Initialization mode. An interrupt is generated if WUIE bit in LINIER is set.</p>   |
| DBFF  | <p>Data Buffer Full Flag</p> <p>This bit is set by hardware and indicates the buffer is full. It is set only when receiving extended frames (DFL &gt; 7).</p> <p>This bit must be cleared by software.</p> <p>It is reset by hardware in Initialization mode.</p>   |
| DBEF  | <p>Data Buffer Empty Flag</p> <p>This bit is set by hardware and indicates the buffer is empty. It is set only when transmitting extended frames (DFL &gt; 7).</p> <p>This bit must be cleared by software, once buffer has been filled again, in order to start transmission.</p> <p>This bit is reset by hardware in Initialization mode.</p>   |
| DRF   | <p>Data Reception Completed Flag</p> <p>This bit is set by hardware and indicates the data reception is completed.</p> <p>This bit must be cleared by software.</p> <p>It is reset by hardware in Initialization mode.</p> <p>Note: This flag is not set in case of bit error or framing error.</p>   |
| DTF   | <p>Data Transmission Completed Flag</p> <p>This bit is set by hardware and indicates the data transmission is completed.</p> <p>This bit must be cleared by software.</p> <p>It is reset by hardware in Initialization mode.</p> <p>Note: This flag is not set in case of bit error if IOBE bit is reset.</p>   |
| HRF   | <p>Header Reception Flag</p> <p>This bit is set by hardware and indicates a valid header reception is completed.</p> <p>This bit must be cleared by software.</p> <p>This bit is reset by hardware in Initialization mode and at end of completed or aborted frame.</p> <p>Note: If filters are enabled, this bit is set only when identifier software filtering is required, that is to say:</p> <ul style="list-style-type: none"> <li>• all filters are inactive and BF bit in LINCR1 is set</li> <li>• no match in any filter and BF bit in LINCR1 is set</li> <li>• TX filter match</li> </ul> |

## 24.10.4 LIN error status register (LINESR)

Offset: 0x0C

Access: User read/write

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       |     |     |     |     |      |      |       |     |     |    |    |    |    |    |    |     |
|-------|-----|-----|-----|-----|------|------|-------|-----|-----|----|----|----|----|----|----|-----|
|       | 16  | 17  | 18  | 19  | 20   | 21   | 22    | 23  | 24  | 25 | 26 | 27 | 28 | 29 | 30 | 31  |
| R     | SZF | OCF | BEF | CEF | SFEF | BDEF | IDPEF | FEF | BOF | 0  | 0  | 0  | 0  | 0  | 0  | NF  |
| W     | w1c | w1c | w1c | w1c | w1c  | w1c  | w1c   | w1c | w1c |    |    |    |    |    |    | w1c |
| Reset | 0   | 0   | 0   | 0   | 0    | 0    | 0     | 0   | 0   | 0  | 0  | 0  | 0  | 0  | 0  | 0   |

Figure 24-21. LIN error status register (LINESR)

Table 24-15. LINESR field descriptions

| Field | Description  |
|-------|--|
| SZF   | Stuck at zero Flag<br>This bit is set by hardware when the bus is dominant for more than a 100-bit time. It is cleared by software.  |
| OCF   | Output Compare Flag<br>0: No output compare event occurred<br>1: The content of the counter has matched the content of OC1[0:7] or OC2[0:7] in LINOCR. If this bit is set and IOT bit in LINTCSR is set, LINFlexD moves to Idle state.<br>If LTOM bit in LINTCSR register is set then OCF is reset by hardware in Initialization mode. If LTOM bit is reset, then OCF maintains its status whatever the mode is. |
| BEF   | Bit Error Flag<br>This bit is set by hardware and indicates to the software that LINFlexD has detected a bit error. This error can occur during response field transmission (Slave and Master modes) or during header transmission (in Master mode).<br>This bit is cleared by software.   |
| CEF   | Checksum error Flag<br>This bit is set by hardware and indicates that the received checksum does not match the hardware calculated checksum.<br>This bit is cleared by software.<br>Note: This bit is never set if CCD or CFD bit in LINCR1 register is set.   |
| SFEF  | Synch Field Error Flag<br>This bit is set by hardware and indicates that a Synch Field error occurred (inconsistent Synch Field).  |
| BDEF  | Break Delimiter Error Flag<br>This bit is set by hardware and indicates that the received Break Delimiter is too short (less than one bit time).   |



Table 24-15. LINESR field descriptions (continued)

| Field | Description   |
|-------|---|
| IDPEF | Identifier Parity Error Flag<br>This bit is set by hardware and indicates that a Identifier Parity error occurred.<br>Note: Header interrupt is triggered when SFEF or BDEF or IDPEF bit is set and HEIE bit in LINIER is set.  |
| FEF   | Framing Error Flag<br>This bit is set by hardware and indicates to the software that LINFlexD has detected a framing error (invalid stop bit). This error can occur during reception of any data in the response field (Master or Slave mode) or during reception of Synch Field or Identifier Field in Slave mode. |
| BOF   | Buffer Overrun Flag<br>This bit is set by hardware when a new data byte is received and the buffer full flag is not cleared. If RBLM in LINCR1 is set then the new byte received is discarded. If RBLM is reset then the new byte overwrites the buffer. It can be cleared by software.                             |
| NF    | Noise Flag<br>This bit is set by hardware when noise is detected on a received character. This bit is cleared by software.  |

### 24.10.5 UART mode control register (UARTCR)

Offset: 0x10

Access: User read/write

|       |                      |    |    |    |                      |    |    |    |      |                   |                    |                  |      |      |                  |                  |                    |                   |   |
|-------|----------------------|----|----|----|----------------------|----|----|----|------|-------------------|--------------------|------------------|------|------|------------------|------------------|--------------------|-------------------|---|
|       | 0                    | 1  | 2  | 3  | 4                    | 5  | 6  | 7  | 8    | 9                 | 10                 | 11               | 12   | 13   | 14               | 15               |                    |                   |   |
| R     | 0                    | 0  | 0  | 0  | 0                    | 0  | 0  | 0  | 0    | 0                 | 0                  | 0                | 0    | 0    | 0                | 0                |                    |                   |   |
| W     |                      |    |    |    |                      |    |    |    |      |                   |                    |                  |      |      |                  |                  |                    |                   |   |
| Reset | 0                    | 0  | 0  | 0  | 0                    | 0  | 0  | 0  | 0    | 0                 | 0                  | 0                | 0    | 0    | 0                | 0                |                    |                   |   |
|       | 16                   | 17 | 18 | 19 | 20                   | 21 | 22 | 23 | 24   | 25                | 26                 | 27               | 28   | 29   | 30               | 31               |                    |                   |   |
| R     | TDFLTFC <sup>1</sup> |    |    |    | RDFLRFC <sup>1</sup> |    |    |    | RFBM | TFBM <sup>2</sup> | WL[1] <sup>2</sup> | PC1 <sup>2</sup> | RXEN | TXEN | PC0 <sup>2</sup> | PCE <sup>2</sup> | WL[0] <sup>2</sup> | UART <sup>2</sup> |   |
| W     |                      |    |    |    |                      |    |    |    |      |                   |                    |                  |      |      |                  |                  |                    |                   |   |
| Reset | 0                    | 0  | 0  | 0  | 0                    | 0  | 0  | 0  | 0    | 0                 | 0                  | 0                | 0    | 0    | 0                | 0                | 0                  | 0                 | 0 |

<sup>1</sup> These fields are read/write in UART buffer mode and read-only in other modes.

<sup>2</sup> These fields are writable only in Initialization mode (LINCR1[INIT] = 1).

Figure 24-22. UART mode control register (UARTCR)

Table 24-16. UARTCR field descriptions

| Field   | Description   |
|---------|---|
| TDFLTC  | <p>Transmitter data field length / Tx FIFO counter</p> <p>This field has one of two functions depending on the mode of operation as follows:</p> <ul style="list-style-type: none"> <li>When LINFlexD is in UART buffer mode (TFBM = 0), TDFLTC defines the number of bytes to be transmitted. The field is read/write in this configuration. The first bit is reserved and not implemented. The permissible values are as follows (with X representing the unimplemented first bit):<br/> 0bX00: 1 byte<br/> 0bX01: 2 bytes<br/> 0bX10: 3 bytes<br/> 0bX11: 4 bytes<br/> When the UART data length is configured as half-word (WL = 0b10 or 0b11), the only valid values for TDFLTC are 0b001 and 0b011.</li> <li>When LINFlexD is in UART FIFO mode (TFBM = 1), TDFLTC contains the number of entries (bytes) of the Tx FIFO. The field is read-only in this configuration. The permissible values are as follows:<br/> 0b000: Empty<br/> 0b001: 1 byte<br/> 0b010: 2 bytes<br/> 0b011: 3 bytes<br/> 0b100: 4 bytes<br/> All other values are reserved.</li> </ul> <p>This field is meaningful and can be programmed only when the UART bit is set.</p> |
| RDFLRFC | <p>Receiver data field length / Rx FIFO counter</p> <p>This field has one of two functions depending on the mode of operation as follows:</p> <ul style="list-style-type: none"> <li>When LINFlexD is in UART buffer mode (RFBM = 0), RDFLRFC defines the number of bytes to be received. The field is read/write in this configuration. The first bit is reserved and not implemented. The permissible values are as follows (with X representing the unimplemented first bit):<br/> 0bX00: 1 byte<br/> 0bX01: 2 bytes<br/> 0bX10: 3 bytes<br/> 0bX11: 4 bytes<br/> When the UART data length is configured as half-word (WL = 0b10 or 0b11), the only valid values for RDFLRFC are 0b001 and 0b011.</li> <li>When LINFlexD is in UART FIFO mode (RFBM = 1), RDFLRFC contains the number of entries (bytes) of the Rx FIFO. The field is read-only in this configuration. The permissible values are as follows:<br/> 0b000: Empty<br/> 0b001: 1 byte<br/> 0b010: 2 bytes<br/> 0b011: 3 bytes<br/> 0b100: 4 bytes<br/> All other values are reserved.</li> </ul> <p>This field is meaningful and can be programmed only when the UART bit is set.</p>    |
| RFBM    | <p>Rx FIFO/buffer mode</p> <p>0 Rx buffer mode enabled<br/> 1 Rx FIFO mode enabled (mandatory in DMA Rx mode)</p> <p>This field can be programmed in initialization mode only when the UART bit is set.</p>   |

Table 24-16. UARTCR field descriptions (continued)

| Field | Description  |
|-------|--|
| TFBM  | <p>Tx FIFO/buffer mode</p> <p>0 Tx buffer mode enabled</p> <p>1 Tx FIFO mode enabled (mandatory in DMA Tx mode)</p> <p>This field can be programmed in initialization mode only when the UART bit is set.</p>  |
| RXEN  | <p>Receiver Enable</p> <p>0: Receiver disabled</p> <p>1: Receiver enabled</p> <p>This field can be programmed only when the UART bit is set.</p>   |
| TXEN  | <p>Transmitter Enable</p> <p>0: Transmitter disabled</p> <p>1: Transmitter enabled</p> <p>This field can be programmed only when the UART bit is set.</p> <p>Note: Transmission starts when this bit is set and when writing DATA0 in the BDRL register.</p>   |
| PC    | <p>Parity control</p> <p>00 Parity sent is even</p> <p>01 Parity sent is odd</p> <p>10 A logical 0 is always transmitted/checked as parity bit</p> <p>11 A logical 1 is always transmitted/checked as parity bit</p> <p>This field can be programmed in initialization mode only when the UART bit is set.</p>                             |
| PCE   | <p>Parity Control Enable</p> <p>0: Parity transmit/check disabled</p> <p>1: Parity transmit/check enabled</p> <p>This field can be programmed in Initialization mode only when the UART bit is set.</p>  |
| WL    | <p>Word length in UART mode</p> <p>00 7 bits data + parity</p> <p>01 8 bits data when PCE = 0 or 8 bits data + parity when PCE = 1</p> <p>10 15 bits data + parity</p> <p>11 16 bits data when PCE = 0 or 16 bits data + parity when PCE = 1</p> <p>This field can be programmed in Initialization mode only when the UART bit is set.</p> |
| UART  | <p>UART mode enable</p> <p>0: LIN mode</p> <p>1: UART mode</p> <p>This field can be programmed in Initialization mode only.</p>  |

### 24.10.6 UART mode status register (UARTSR)

Offset: 0x14

Access: User read/write

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       |     |     |     |     |     |     |     |     |     |     |     |    |     |        |        |     |
|-------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|----|-----|--------|--------|-----|
|       | 16  | 17  | 18  | 19  | 20  | 21  | 22  | 23  | 24  | 25  | 26  | 27 | 28  | 29     | 30     | 31  |
| R     | SZF | OCF | PE3 | PE2 | PE1 | PE0 | RMB | FEF | BOF | RPS | WUF | 0  | TO  | DRFRFE | DTFTFF | NF  |
| W     | w1c | w1c | w1c | w1c | w1c | w1c | w1c | w1c | w1c | w1c | w1c |    | w1c | w1c    | w1c    | w1c |
| Reset | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0  | 0   | 0      | 0      | 0   |

Figure 24-23. UART mode status register (UARTSR)

Table 24-17. UARTSR field descriptions

| Field | Description  |
|-------|--|
| SZF   | Stuck at zero Flag<br>This bit is set by hardware when the bus is dominant for more than a 100-bit time. It is cleared by software.  |
| OCF   | OCF Output Compare Flag<br>0: No output compare event occurred<br>1: The content of the counter has matched the content of OC1[0:7] or OC2[0:7] in LINOCR.<br>An interrupt is generated if the OCIE bit in LINIER register is set. |
| PE3   | Parity Error Flag Rx3<br>This bit indicates if there is a parity error in the corresponding received byte (Rx3). No interrupt is generated if this error occurs.<br>0: No parity error<br>1: Parity error                          |
| PE2   | Parity Error Flag Rx2<br>This bit indicates if there is a parity error in the corresponding received byte (Rx2). No interrupt is generated if this error occurs.<br>0: No parity error<br>1: Parity error                          |
| PE1   | Parity Error Flag Rx1<br>This bit indicates if there is a parity error in the corresponding received byte (Rx1). No interrupt is generated if this error occurs.<br>0: No parity error<br>1: Parity error                          |
| PE0   | Parity Error Flag Rx0<br>This bit indicates if there is a parity error in the corresponding received byte (Rx0). No interrupt is generated if this error occurs.<br>0: No parity error<br>1: Parity error                          |

Table 24-17. UARTSR field descriptions (continued)

| Field  | Description  |
|--------|--|
| RMB    | Release Message Buffer<br>0: Buffer is free<br>1: Buffer ready to be read by software. This bit must be cleared by software after reading data received in the buffer.<br>This bit is cleared by hardware in Initialization mode.  |
| FEF    | Framing Error Flag<br>This bit is set by hardware and indicates to the software that LINFlexD has detected a framing error (invalid stop bit).   |
| BOF    | FIFO/buffer overrun flag<br>This bit is set by hardware when a new data byte is received and the RMB bit is not cleared in UART buffer mode. In UART FIFO mode, this bit is set when there is a new byte and the Rx FIFO is full. In UART FIFO mode, once Rx FIFO is full, the new received message is discarded regardless of the value of LINCR1[RBLM].<br>If LINCR1[RBLM] = 1, the new byte received is discarded.<br>If LINCR1[RBLM] = 0, the new byte overwrites buffer.<br>This field can be cleared by writing a 1 to it. An interrupt is generated if LINIER[BOIE] is set.   |
| RPS    | LIN Receive Pin State<br>This bit reflects the current status of LINRX pin for diagnostic purposes.  |
| WUF    | Wake-up Flag<br>This bit is set by hardware and indicates to the software that LINFlexD has detected a falling edge on the LINRX pin in Sleep mode.<br>This bit must be cleared by software. It is reset by hardware in Initialization mode.<br>An interrupt is generated if WUIE bit in LINIER is set.  |
| TO     | Timeout<br>The LINFlexD controller sets this field when a UART timeout occurs — that is, when the value of UARTCTO becomes equal to the preset value of the timeout (UARTPTO register setting). This field should be cleared by software. The GCR[SR] field should be used to reset the receiver FSM to idle state in case of UART timeout for UART reception depending on the application both in buffer and FIFO mode.<br>An interrupt is generated when LINIER[DBEIETOIE] is set on the Error interrupt line in UART mode.  |
| DRFRFE | Data reception completed flag / Rx FIFO empty flag<br>The LINFlexD controller sets this field as follows: <ul style="list-style-type: none"> <li>In UART buffer mode (RFBM = 0), it indicates that the number of bytes programmed in RDFL has been received. This field should be cleared by software. An interrupt is generated if LINIER[DRIE] is set. This field is set in case of framing error, parity error, or overrun. This field reflects the same value as in LINESR when in Initialization mode and UART bit is set.</li> <li>In UART FIFO mode (RFBM = 1), it indicates that the Rx FIFO is empty. This field is a read-only field used internally by the DMA Rx interface.</li> </ul> |
| DTFTFF | Data transmission completed flag / Tx FIFO full flag<br>The LINFlexD controller sets this field as follows: <ul style="list-style-type: none"> <li>In UART buffer mode (TFBM = 0), it indicates that the data transmission is completed. This field should be cleared by software. An interrupt is generated if LINIER[DTIE] is set. This field reflects the same value as in LINESR when in Initialization mode and UART bit is set.</li> <li>In UART FIFO mode (TFBM = 1), it indicates that the Tx FIFO is full. This field is a read-only field used internally by the DMA Tx interface.</li> </ul>  |
| NF     | Noise Flag<br>This bit is set by hardware when noise is detected on a received character. This bit is cleared by software.   |

## 24.10.7 LIN timeout control status register (LINTCSR)

Offset: 0x18

Access: User read/write

|       |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|       | 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 | 14 | 15 |
| R     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
|       | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 1  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |

<sup>1</sup> These fields are writable only in Initialization mode (LINCR1[INIT] = 1).

**Figure 24-24. LIN timeout control status register (LINTCSR)**

**Table 24-18. LINTCSR field descriptions**

| Name | Description   |
|------|---|
| LTOM | LIN timeout mode<br>0: LIN timeout mode (header, response and frame timeout detection)<br>1: Output compare mode<br>This bit can be set/cleared in Initialization mode only.  |
| IOT  | Idle on Timeout<br>0: LIN state machine not reset to Idle on timeout event<br>1: LIN state machine reset to Idle on timeout event<br>This bit can be set/cleared in Initialization mode only.   |
| TOCE | Timeout counter enable<br>0: Timeout counter disable. OCF bit in LINESR or UARTSR is not set on an output compare event.<br>1: Timeout counter enable. OCF bit is set if an output compare event occurs.<br>TOCE bit is configurable by software in Initialization mode. If LIN state is not Init and if timer is in LIN timeout mode, then hardware takes control of TOCE bit. |
| CNT  | Counter Value<br>These bits indicate the LIN Timeout counter value.   |

## 24.10.8 LIN output compare register (LINOCR)

Offset: 0x1C

Access: User read/write

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       |                  |    |    |    |    |    |    |    |                  |    |    |    |    |    |    |    |
|-------|------------------|----|----|----|----|----|----|----|------------------|----|----|----|----|----|----|----|
|       | 16               | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24               | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| R     | OC2 <sup>1</sup> |    |    |    |    |    |    |    | OC1 <sup>1</sup> |    |    |    |    |    |    |    |
| W     | w1c <sup>1</sup> |    |    |    |    |    |    |    | w1c <sup>1</sup> |    |    |    |    |    |    |    |
| Reset | 1                | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1                | 1  | 1  | 1  | 1  | 1  | 1  | 1  |

<sup>1</sup> If LINTCSR[LTOM] = 0, these fields are read-only.

**Figure 24-25. LIN output compare register (LINOCR)**

**Table 24-19. LINOCR field descriptions**

| Field | Description   |
|-------|---|
| OC2   | Output compare 2 value<br>These bits contain the value to be compared to the value of LINTCSR[CNT]. |
| OC1   | Output compare 1 value<br>These bits contain the value to be compared to the value of LINTCSR[CNT]. |

### 24.10.9 LIN timeout control register (LINTOCR)

Offset: 0x20

Access: User read/write

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       |    |    |    |    |     |    |    |    |    |                  |                  |                  |    |    |    |    |  |
|-------|----|----|----|----|-----|----|----|----|----|------------------|------------------|------------------|----|----|----|----|--|
|       | 16 | 17 | 18 | 19 | 20  | 21 | 22 | 23 | 24 | 25               | 26               | 27               | 28 | 29 | 30 | 31 |  |
| R     | 0  | 0  | 0  | 0  | RTO |    |    |    | 0  | HTO <sup>3</sup> |                  |                  |    |    |    |    |  |
| W     |    |    |    |    |     |    |    |    |    |                  |                  |                  |    |    |    |    |  |
| Reset | 0  | 0  | 0  | 0  | 1   | 1  | 1  | 0  | 0  | 0                | 0/1 <sup>1</sup> | 0/1 <sup>2</sup> | 1  | 1  | 0  | 0  |  |

- <sup>1</sup> Resets to 1 in Slave mode and to 0 in Master mode
- <sup>2</sup> Resets to 0 in Slave mode and to 1 in Master mode
- <sup>3</sup> HTO field can only be written in slave mode, LINC1R[MME] = 0.

**Figure 24-26. LIN timeout control register (LINTOCR)**

**Table 24-20. LINTOCR field descriptions**

| Field | Description   |
|-------|---|
| RTO   | Response timeout value<br>This register contains the response timeout duration (in bit time) for 1 byte.<br>The reset value is 0xE = 14, corresponding to $T_{Response\_Maximum} = 1.4 \times T_{Response\_Nominal}$  |
| HTO   | Header timeout value<br>This register contains the header timeout duration (in bit time). This value does not include the first 11 dominant bits of the Break. The reset value depends on which mode LINFlexD is in.<br>HTO can be written only for Slave mode. |



## 24.10.10 LIN fractional baud rate register (LINFBR)

Offset: 0x24

Access: User read/write

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       |    |    |    |    |    |    |    |    |    |    |    |    |                    |    |    |    |
|-------|----|----|----|----|----|----|----|----|----|----|----|----|--------------------|----|----|----|
|       | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28                 | 29 | 30 | 31 |
| R     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | DIV_F <sup>1</sup> |    |    |    |
| W     |    |    |    |    |    |    |    |    |    |    |    |    |                    |    |    |    |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0                  | 0  | 0  | 0  |

<sup>1</sup> This field is writable only in Initialization mode, LINCR1[INIT] = 1.

**Figure 24-27. LIN timeout control register (LINTOCR)**

**Table 24-21. LINFBR field descriptions**

| Field | Description   |
|-------|---|
| DIV_F | Fraction bits of LFDIV<br>The 4 fraction bits define the value of the fraction of the LINFlexD divider (LFDIV).<br>Fraction (LFDIV) = Decimal value of DIV_F / 16.<br><br>This register can be written in Initialization mode only, LINCR1[INIT] = 1. |

## 24.10.11 LIN integer baud rate register (LINIBRR)

Offset: 0x28

Access: User read/write

|       |   |   |   |   |   |   |   |   |   |   |    |    |                    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|--------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12                 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | DIV_M <sup>1</sup> |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |                    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0                  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |

<sup>1</sup> This field is writable only in Initialization mode (LINCR1[INIT] = 1).

**Figure 24-28. LIN integer baud rate register (LINIBRR)**

**Table 24-22. LINIBRR field descriptions**

| Field | Description  |
|-------|--|
| DIV_M | LFDIV mantissa<br>These bits define the LINFlexD divider (LFDIV) mantissa value (see <a href="#">Table 24-23</a> ).<br>This register can be written in Initialization mode only. |

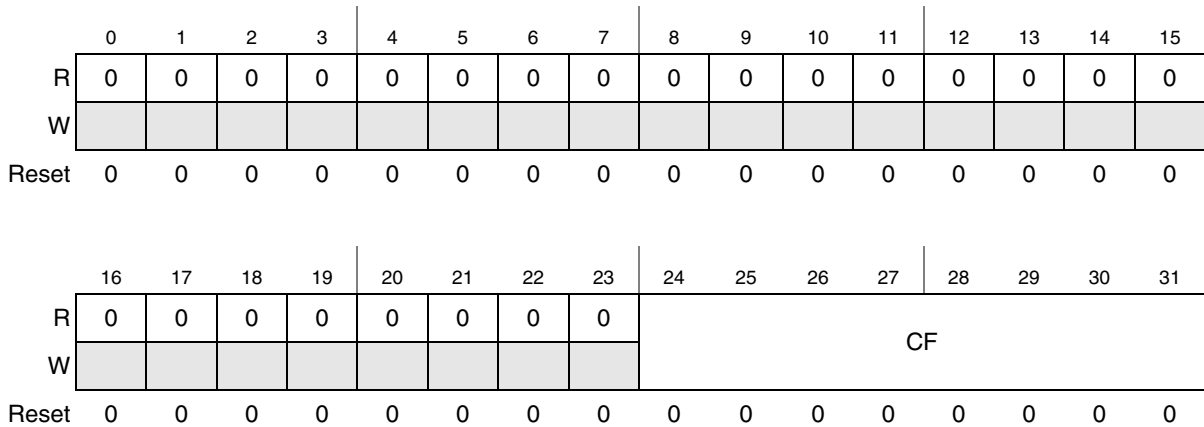
**Table 24-23. Integer baud rate selection**

| DIV_M   | Mantissa           |
|---------|--------------------|
| 0x0     | LIN clock disabled |
| 0x1     | 1                  |
| ...     | ...                |
| 0xFFFFE | 1048574            |
| 0xFFFFF | 1048575            |

### 24.10.12 LIN checksum field register (LINCFR)

Offset: 0x2C

Access: User read/write



**Figure 24-29. LIN checksum field register (LINCFR)**

**Table 24-24. LINCFR field descriptions**

| Field | Description   |
|-------|---|
| CF    | Checksum bits<br>When LINCR1[CCD] is cleared, these bits are read-only. When LINCR1[CCD] is set, these bits are read/write. See <a href="#">Table 24-10</a> . |

### 24.10.13 LIN control register 2 (LINCR2)

Offset: 0x30

Access: User read/write

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       |    |                   |                   |      |      |      |      |      |    |    |    |    |    |    |    |    |
|-------|----|-------------------|-------------------|------|------|------|------|------|----|----|----|----|----|----|----|----|
|       | 16 | 17                | 18                | 19   | 20   | 21   | 22   | 23   | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| R     | 0  | IOBE <sup>1</sup> | IOPE <sup>1</sup> | WURQ | DDRQ | DTRQ | ABRQ | HTRQ | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |    |                   |                   | w1c  | w1c  | w1c  | w1c  | w1c  |    |    |    |    |    |    |    |    |
| Reset | 0  | 1                 | 0/1 <sup>2</sup>  | 0    | 0    | 0    | 0    | 0    | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |

<sup>1</sup> These fields are writable only in Initialization mode (LINCR1[INIT] = 1).

<sup>2</sup> Resets to 1 in Slave mode and to 0 in Master mode

**Figure 24-30. LIN control register 2 (LINCR2)**

**Table 24-25. LINCR2 field descriptions**

| Field | Description  |
|-------|--|
| IOBE  | Idle on Bit Error<br>0: Bit error does not reset LIN state machine<br>1: Bit error reset LIN state machine<br>This bit can be set/cleared in Initialization mode only (LINCR1[INIT]) = 1.  |
| IOPE  | Idle on Identifier Parity Error<br>0: Identifier Parity error does not reset LIN state machine.<br>1: Identifier Parity error reset LIN state machine.<br>This bit can be set/cleared in Initialization mode only (LINCR1[INIT]) = 1.  |
| WURQ  | Wake-up Generation Request<br>Setting this bit generates a wake-up pulse. It is reset by hardware when the wake-up character has been transmitted. The character sent is copied from DATA0 in BDRL buffer. Note that this bit cannot be set in Sleep mode. Software has to exit Sleep mode before requesting a wake-up. Bit error is not checked when transmitting the wake-up request.                                  |
| DDRQ  | Data Discard Request<br>Set by software to stop data reception if the frame does not concern the node. This bit is reset by hardware once LINFLEXD has moved to idle state. In Slave mode, this bit can be set only when HRF bit in LINSR is set and identifier did not match any filter.  |
| DTRQ  | Data Transmission Request<br>Set by software in Slave mode to request the transmission of the LIN Data field stored in the Buffer data register. This bit can be set only when HRF bit in LINSR is set.<br>Cleared by hardware when the request has been completed or aborted or on an error condition.<br>In Master mode, this bit is set by hardware when DIR bit in BIDR is set and header transmission is completed. |

**Table 24-25. LINC2 field descriptions (continued)**

| Field | Description   |
|-------|---|
| ABRQ  | Abort Request<br>Set by software to abort the current transmission.<br>Cleared by hardware when the transmission has been aborted. LINFlexD aborts the transmission at the end of the current bit.<br>This bit can also abort a wake-up request.<br>It can also be used in UART mode. |
| HTRQ  | Header Transmission Request<br>Set by software to request the transmission of the LIN header.<br>Cleared by hardware when the request has been completed or aborted.<br>This bit has no effect in UART mode.  |

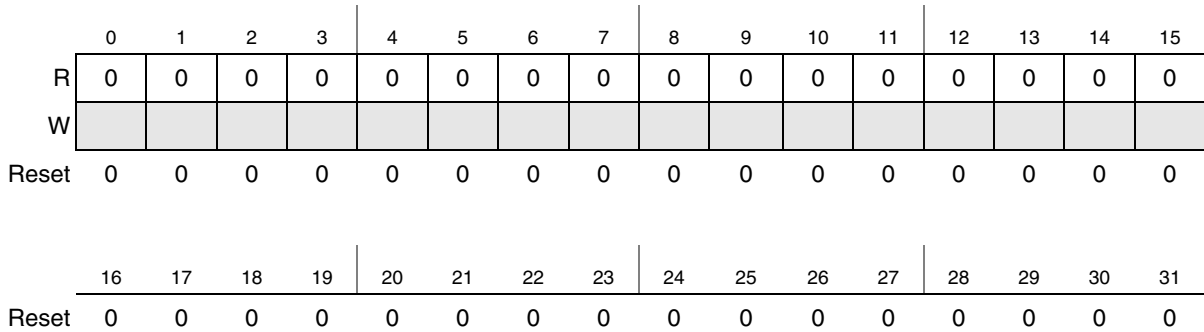
### 24.10.14 Buffer identifier register (BIDR)

This register contains the fields that identify a transaction and provide other information related to it.

All the fields in this register must be updated when an ID filter (enabled) in slave mode (Tx or Rx) matches the ID received.

Offset: 0x34

Access: User read/write



**Figure 24-31. Buffer identifier register (BIDR)**

**Table 24-26. BIDR field descriptions**

| Field | Description  |
|-------|--|
| DFL   | Data Field Length<br>These bits define the number of data bytes in the response part of the frame.<br>DFL = Number of data bytes – 1.<br>Normally, LIN uses only DFL[0:2] to manage frames with a maximum of 8 bytes of data. Identifier filters are compatible with DFL[0:2] and DFL[0:5]. DFL[3:5] are provided to manage extended frames. |
| DIR   | Direction<br>This bit controls the direction of the data field.<br>0: LINFlexD receives the data and copy them in the BDR registers.<br>1: LINFlexD transmits the data from the BDR registers.   |

Table 24-26. BDR field descriptions (continued)

| Field | Description  |
|-------|--|
| CCS   | Classic Checksum<br>This bit controls the type of checksum applied on the current message.<br>0: Enhanced Checksum covering Identifier and Data fields. This is compatible with LIN specification 2.0 and higher.<br>1: Classic Checksum covering Data fields only. This is compatible with LIN specification 1.3 and below. |
| ID    | Identifier<br>Identifier part of the identifier field without the identifier parity.   |

### 24.10.15 Buffer data register least significant (BDRL)

Offset: 0x38

Access: User read/write

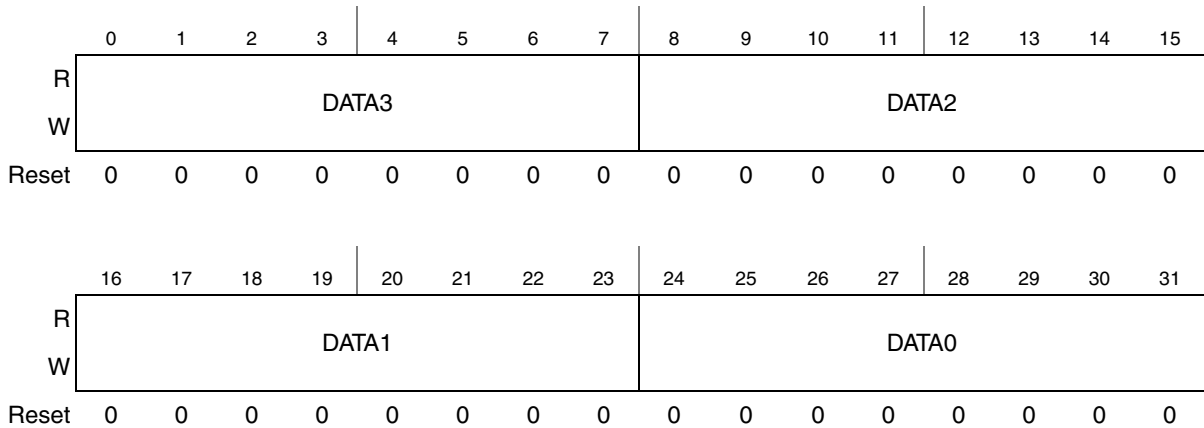
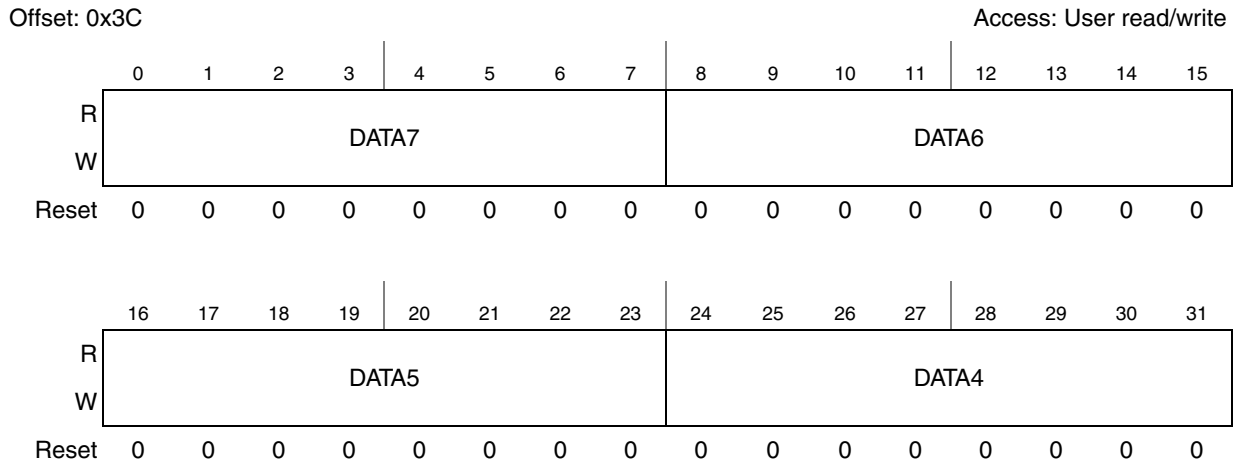


Figure 24-32. Buffer data register least significant (BDRL)

Table 24-27. BDRL field descriptions

| Field | Description                                  |
|-------|--|
| DATA3 | Data Byte 3<br>Data byte 3 of the data field |
| DATA2 | Data Byte 2<br>Data byte 2 of the data field |
| DATA1 | Data Byte 1<br>Data byte 1 of the data field |
| DATA0 | Data Byte 0<br>Data byte 0 of the data field |

### 24.10.16 Buffer data register most significant (BDRM)



**Figure 24-33. Buffer data register most significant (BDRM)**

**Table 24-28. BDRM field descriptions**

| Field | Description                                  |
|-------|--|
| DATA7 | Data Byte 7<br>Data byte 7 of the data field |
| DATA6 | Data Byte 6<br>Data byte 6 of the data field |
| DATA5 | Data Byte 5<br>Data byte 5 of the data field |
| DATA4 | Data Byte 4<br>Data byte 4 of the data field |

## 24.10.17 Identifier filter enable register (IFER)

Offset: 0x40

Access: User read/write

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       |    |    |    |    |    |    |    |    |                   |    |    |    |    |    |    |    |
|-------|----|----|----|----|----|----|----|----|-------------------|----|----|----|----|----|----|----|
|       | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24                | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| R     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | FACT <sup>1</sup> |    |    |    |    |    |    |    |
| W     |    |    |    |    |    |    |    |    | FACT <sup>1</sup> |    |    |    |    |    |    |    |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0                 | 0  | 0  | 0  | 0  | 0  | 0  | 0  |

<sup>1</sup> This field is writable only in Initialization mode (LINCR1[INIT] = 1).

**Figure 24-34. Identifier filter enable register (IFER)**

**Table 24-29. IFER field descriptions**

| Field | Description  |
|-------|--|
| FACT  | Filter activation (see <a href="#">Table 24-30</a> )<br>The software sets the bit FACT[x] to activate the filters x in identifier list mode.<br>In identifier mask mode bits FACT(2n + 1) have no effect on the corresponding filters as they act as masks for the Identifiers 2n.<br>0 Filters 2n and 2n + 1 are deactivated.<br>1 Filters 2n and 2n + 1 are activated. |

**Table 24-30. IFER[FACT] configuration**

| Bit     | Value | Result                           |
|---------|-------|----------------------------------|
| FACT[0] | 0     | Filters 0 and 1 are deactivated. |
|         | 1     | Filters 0 and 1 are activated.   |
| FACT[1] | 0     | Filters 2 and 3 are deactivated. |
|         | 1     | Filters 2 and 3 are activated.   |
| FACT[2] | 0     | Filters 4 and 5 are deactivated. |
|         | 1     | Filters 4 and 5 are activated.   |
| FACT[3] | 0     | Filters 6 and 7 are deactivated. |
|         | 1     | Filters 6 and 7 are activated.   |
| FACT[4] | 0     | Filters 8 and 9 are deactivated. |
|         | 1     | Filters 8 and 9 are activated.   |

**Table 24-30. IFER[FACT] configuration (continued)**

| Bit     | Value | Result                             |
|---------|-------|------------------------------------|
| FACT[5] | 0     | Filters 10 and 11 are deactivated. |
|         | 1     | Filters 10 and 11 are activated.   |
| FACT[6] | 0     | Filters 12 and 13 are deactivated. |
|         | 1     | Filters 12 and 13 are activated.   |
| FACT[7] | 0     | Filters 14 and 15 are deactivated. |
|         | 1     | Filters 14 and 15 are activated.   |

### 24.10.18 Identifier filter match index (IFMI)

Offset: 0x44 Access: User read/write

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|       | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |

**Figure 24-35. Identifier filter match index (IFMI)**

**Table 24-31. IFMI field descriptions**

| Field | Description   |
|-------|---|
| IFMI  | Filter match index<br>This register contains the index corresponding to the received ID. It can be used to directly write or read the data in RAM (refer to <a href="#">Section 24.7.2, Slave mode</a> , for more details).<br>When no filter matches, IFMI = 0. When Filter n is matching, IFMI = n + 1. |



## 24.10.19 Identifier filter mode register (IFMR)

Offset:0x48 Access: User read/write

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|       | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |

Figure 24-36. Identifier filter mode register (IFMR)

Table 24-32. IFMR field descriptions

| Field | Description   |
|-------|---|
| IFM   | Filter mode<br>0 Filters $2n$ and $2n + 1$ are in identifier list mode.<br>1 Filters $2n$ and $2n + 1$ are in mask mode (filter $2n + 1$ is the mask for the filter $2n$ ). |

## 24.10.20 Identifier filter control registers (IFCR0–IFCR15)

The function of these registers is different depending on which mode the LINFlexD controller is in, as described in [Table 24-33](#).

Table 24-33. IFCR functionality based on mode

| Mode            | IFCR functionality  |
|-----------------|---|
| Identifier list | Each IFCR register acts as a filter.  |
| Identifier mask | If $a = (\text{number of filters}) / 2$ , and $n = 0$ to $(a - 1)$ , then IFCR[ $2n$ ] acts as a filter and IFCR[ $2n + 1$ ] acts as the mask for IFCR[ $2n$ ]. |

Offsets: 0x4C–0x88 (16 registers)

Access: User read/write

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       |                  |    |    |    |    |    |                  |                  |    |    |                 |    |    |    |    |    |
|-------|------------------|----|----|----|----|----|------------------|------------------|----|----|-----------------|----|----|----|----|----|
|       | 16               | 17 | 18 | 19 | 20 | 21 | 22               | 23               | 24 | 25 | 26              | 27 | 28 | 29 | 30 | 31 |
| R     | DFL <sup>1</sup> |    |    |    |    |    | DIR <sup>1</sup> | CCS <sup>1</sup> | 0  | 0  | ID <sup>1</sup> |    |    |    |    |    |
| W     |                  |    |    |    |    |    |                  |                  |    |    |                 |    |    |    |    |    |
| Reset | 0                | 0  | 0  | 0  | 0  | 0  | 0                | 0                | 0  | 0  | 0               | 0  | 0  | 0  | 0  | 0  |

<sup>1</sup> These fields are writable only in Initialization mode (LINCR1[INIT] = 1).

**Figure 24-37. Identifier filter control registers (IFCR0–IFCR15)**

**Table 24-34. IFCR field descriptions**

| Field | Description  |
|-------|--|
| DFL   | Data Field Length<br>This field defines the number of data bytes in the response part of the frame.  |
| DIR   | Direction<br>This bit controls the direction of the data field.<br>0: LINFlexD receives the data and copy them in the BDRL and BDRM registers.<br>1: LINFlexD transmits the data from the BDRL and BDRM registers.   |
| CCS   | Classic Checksum<br>This bit controls the type of checksum applied on the current message.<br>0: Enhanced Checksum covering Identifier and Data fields. This is compatible with LIN specification 2.0 and higher.<br>1: Classic Checksum covering Data fields only. This is compatible with LIN specification 1.3 and below. |
| ID    | Identifier<br>Identifier part of the identifier field without the identifier parity.   |

### 24.10.21 Global control register (GCR)

This register can be programmed only in Initialization mode. The configuration specified in this register applies in both LIN and UART modes.

Offset: 0x8C

Access: User read/write

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       |    |    |    |    |    |    |    |    |    |    |                    |                    |                    |                    |                   |                 |
|-------|----|----|----|----|----|----|----|----|----|----|--------------------|--------------------|--------------------|--------------------|-------------------|-----------------|
|       | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26                 | 27                 | 28                 | 29                 | 30                | 31              |
| R     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | TDFBM <sup>1</sup> | RDFBM <sup>1</sup> | TDLIS <sup>1</sup> | RDLIS <sup>1</sup> | STOP <sup>1</sup> | 0               |
| W     |    |    |    |    |    |    |    |    |    |    |                    |                    |                    |                    |                   | SR <sup>1</sup> |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0                  | 0                  | 0                  | 0                  | 0                 | 0               |

<sup>1</sup> This field is writable only in Initialization mode (LINCR1[INIT] = 1).

**Figure 24-38. Global control register (GCR)**

**Table 24-35. GCR field descriptions**

| Field | Description   |
|-------|---|
| TDFBM | Transmit data first bit MSB<br>This field controls the first bit of transmitted data (payload only) as MSB/LSB in both UART and LIN modes.<br>0 The first bit of transmitted data is LSB – that is, the first bit transmitted is mapped on the LSB bit (BDR(0), BDR(8), BDR(16), BDR(24)).<br>1 The first bit of transmitted data is MSB – that is, the first bit transmitted is mapped on the MSB bit (BDR(7), BDR(15), BDR(23), BDR(31)). |
| RDFBM | Received data first bit MSB<br>This field controls the first bit of received data (payload only) as MSB/LSB in both UART and LIN modes.<br>0 The first bit of received data is LSB – that is, the first bit received is mapped on the LSB bit (BDR(0), BDR(8), BDR(16), BDR(24)).<br>1 The first bit of received data is MSB – that is, the first bit received is mapped on the MSB bit (BDR(7), BDR(15), BDR(23), BDR(31)).                |
| TDLIS | Transmit data level inversion selection<br>This field controls the data inversion of transmitted data (payload only) in both UART and LIN modes.<br>0 Transmitted data is not inverted.<br>1 Transmitted data is inverted.  |
| RDLIS | Received data level inversion selection<br>This field controls the data inversion of received data (payload only) in both UART and LIN modes.<br>0 Received data is not inverted.<br>1 Received data is inverted.   |
| STOP  | Stop bit configuration<br>This field controls the number of stop bits in transmitted data in both UART and LIN modes. The stop bit is configured for all the fields (delimiter, sync, ID, checksum, and payload).<br>0 One stop bit<br>1 Two stop bits  |

Table 24-35. GCR field descriptions (continued)

| Field | Description   |
|-------|---|
| SR    | Soft reset<br>If the software writes a 1 to this field, the LINFlexD controller executes a soft reset in which the FSMs, FIFO pointers, counters, timers, status registers, and error registers are reset but the configuration registers are unaffected.<br>This field always reads 0. |

### 24.10.22 UART preset timeout register (UARTPTO)

This register contains the preset timeout value in UART mode, and is used to monitor the IDLE state of the reception line. The timeout detection uses this register and the UARTCTO register described in [Section 24.10.23, UART current timeout register \(UARTCTO\)](#).

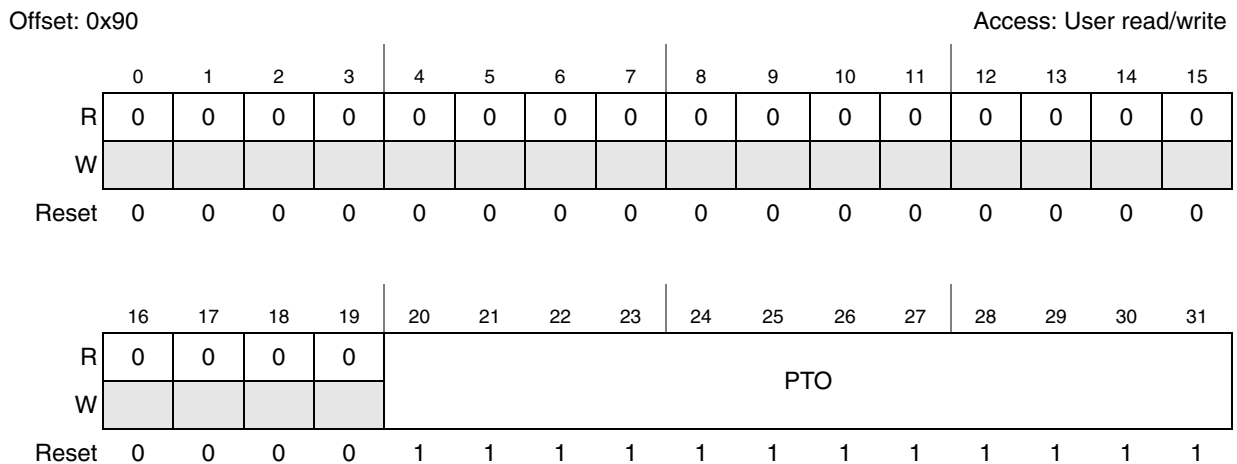


Figure 24-39. UART preset timeout register (UARTPTO)

Table 24-36. UARTPTO field descriptions

| Field | Description   |
|-------|---|
| PTO   | Preset value of the timeout counter<br>Do not set PTO = 0 (otherwise, UARTSR[TO] would immediately be set). |

### 24.10.23 UART current timeout register (UARTCTO)

This register contains the current timeout value in UART mode, and is used in conjunction with the UARTPTO register (see [Section 24.10.22, UART preset timeout register \(UARTPTO\)](#)) to monitor the IDLE state of the reception line. UART timeout works in both CPU and DMA modes.

The timeout counter:

- Starts at 0 and counts upward
- Is clocked with the baud rate clock prescaled by a hard-wired scaling factor of 16
- Is automatically enabled when UARTCR[RXEN] = 1

Offset: 0x94

Access: User read/write

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       |    |    |    |    |     |    |    |    |    |    |    |    |    |    |    |    |
|-------|----|----|----|----|-----|----|----|----|----|----|----|----|----|----|----|----|
|       | 16 | 17 | 18 | 19 | 20  | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| R     | 0  | 0  | 0  | 0  | CTO |    |    |    |    |    |    |    |    |    |    |    |
| W     |    |    |    |    |     |    |    |    |    |    |    |    |    |    |    |    |
| Reset | 0  | 0  | 0  | 0  | 0   | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |

Figure 24-40. UART current timeout register (UARTCTO)

Table 24-37. UARTCTO field descriptions

| Field | Description  |
|-------|--|
| CTO   | <p>Current value of the timeout counter</p> <p>This field is reset whenever one of the following occurs:</p> <ul style="list-style-type: none"> <li>• A new value is written to the UARTPTO register</li> <li>• The value of this field matches the value of UARTPTO[PTO]</li> <li>• A hard or soft reset occurs</li> <li>• New incoming data is received</li> </ul> <p>When CTO matches the value of UARTPTO[PTO], UARTSR[TO] is set.</p> |

### 24.10.24 DMA Tx enable register (DMATXE)

This register enables the DMA Tx interface.

Offset: 0x98

Access: User read/write

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|       | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |

Figure 24-41. DMA Tx enable register (DMATXE)

Table 24-38. DMATXE field descriptions

| Field   | Description  |
|---------|--|
| DTE $n$ | DMA Tx channel $n$ enable<br>0 DMA Tx channel $n$ disabled<br>1 DMA Tx channel $n$ enabled<br><b>Note:</b> When DMATXE = 0x0, the DMA Tx interface FSM is forced (soft reset) into the IDLE state. |

### 24.10.25 DMA Rx enable register (DMARXE)

This register enables the DMA Rx interface.

Offset: 0x9C Access: User read/write

|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |

Figure 24-42. DMA Rx enable register (DMARXE)

Table 24-39. DMARXE field descriptions

| Field   | Description  |
|---------|--|
| DRE $n$ | DMA Rx channel $n$ enable<br>0 DMA Rx channel $n$ disabled<br>1 DMA Rx channel $n$ enabled<br><b>Note:</b> When DMARXE = 0x0, the DMA Rx interface FSM is forced (soft reset) into the IDLE state. |

## 24.11 DMA interface

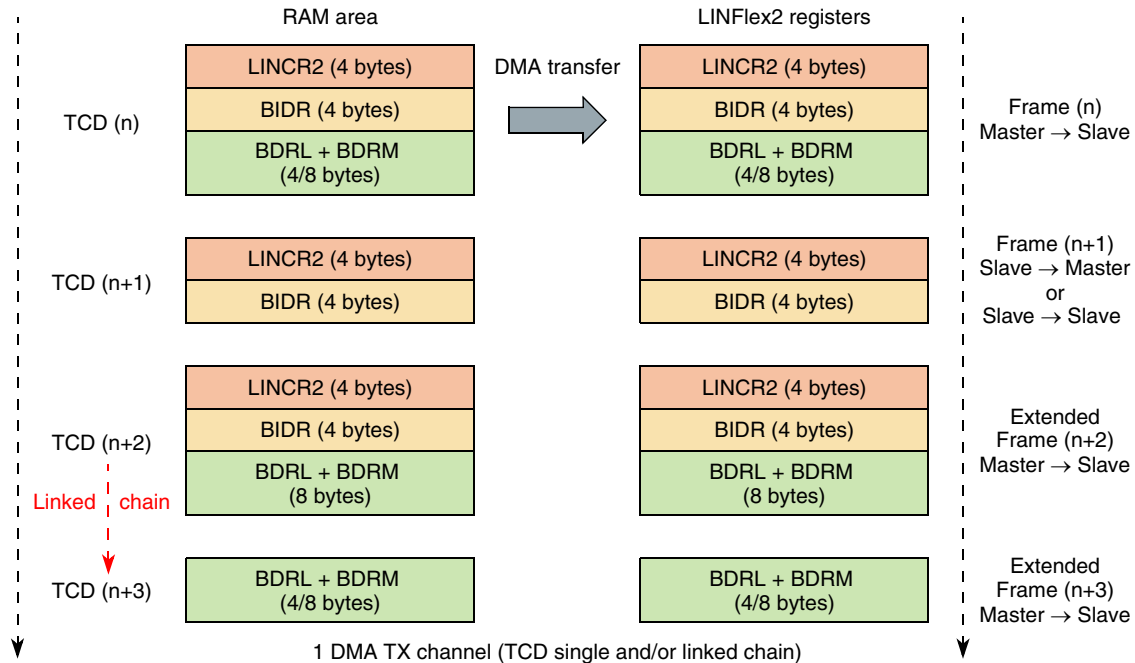
The LINFlexD DMA interface offers a parametric and programmable solution with the following features:

- LIN Master node, TX mode: single DMA channel
- LIN Master node, RX mode: single DMA channel
- LIN Slave node, TX mode: 1 to N DMA channels where N = max number of ID filters
- LIN Slave node, RX mode: 1 to N DMA channels where N = max number of ID filters
- UART node, TX mode: single DMA channel
- UART node, RX mode: single DMA channel + timeout

The LINFlexD controller interacts with an enhanced direct memory access (eDMA) controller; see the description of that controller for details on its operation and the transfer control descriptors (TCDs) referenced in this section.

### 24.11.1 Master node, TX mode

On a master node in TX mode, the DMA interface requires a single TX channel. Each TCD controls a single frame, except for the extended frames (multiple TCDs). The memory map associated with the TCD chain (RAM area and LINFlexD registers) is shown in Figure 24-43.



**Figure 24-43. TCD chain memory map (master node, TX mode)**

The TCD chain of the DMA Tx channel on a master node supports:

- Master to Slave: transmission of the entire frame (header + data)
- Slave to Master: transmission of the header. The data reception is controlled by the Rx channel on the master node.
- Slave to Slave: transmission of the header.

The register settings for the LINC2 and BIDR registers for each class of LIN frame are shown in Table 24-40.

**Table 24-40. Register settings (master node, TX mode)**

| LIN frame       | LINC2                      | BIDR   |
|-----------------|----------------------------|--|
| Master to Slave | DDRQ=1<br>DTRQ=0<br>HTRQ=0 | DFL = payload size<br>ID = address<br>CCS = checksum<br>DIR = 1 (TX) |

**Table 24-40. Register settings (master node, TX mode) (continued)**

| LIN frame       | LINC2                      | BIDR   |
|-----------------|----------------------------|--|
| Slave to Master | DDRQ=0<br>DTRQ=0<br>HTRQ=0 | DFL = payload size<br>ID = address<br>CCS = checksum<br>DIR = 0 (RX) |
| Slave to Slave  | DDRQ=1<br>DTRQ=0<br>HTRQ=0 | DFL = payload size<br>ID = address<br>CCS = checksum<br>DIR = 0 (RX) |

The concept FSM to control the DMA TX interface is shown in [Figure 24-44](#). The DMA TX FSM will move to IDLE state immediately at next clock edge if DMATXE[0] = 0.



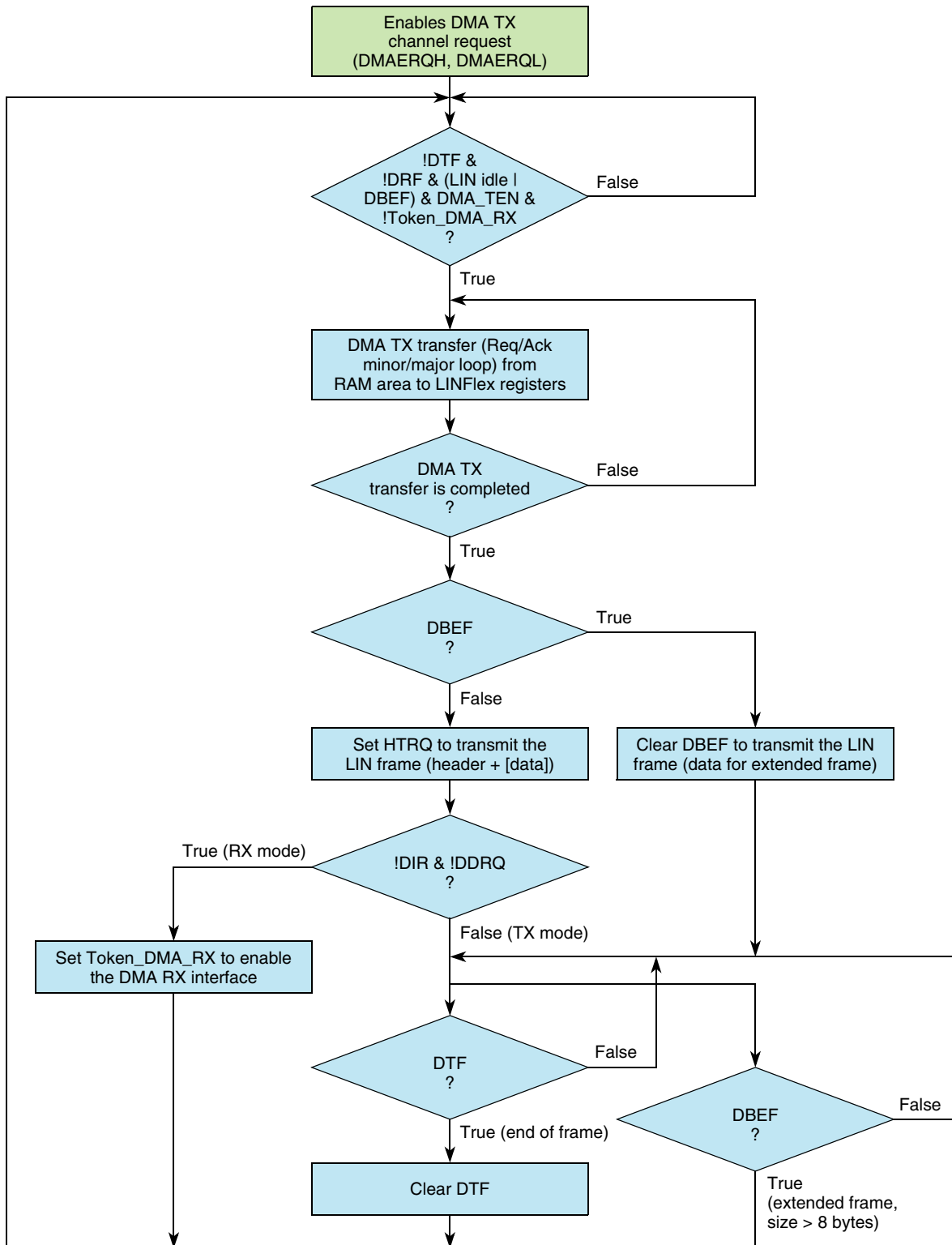


Figure 24-44. FSM to control the DMA TX interface (master node)

The TCD settings (word transfer) are shown in [Table 24-41](#). All other TCD fields are equal to 0. TCD settings based on half-word or byte transfers are allowed.

Table 24-41. TCD settings (master node, TX mode)

| TCD field       | Value                 | Description   |
|-----------------|-----------------------|---|
| CITER[14:0]     | 1                     | Single iteration for the major loop   |
| BITER[14:0]     | 1                     | Single iteration for the major loop   |
| NBYTES[31:0]    | $[4 + 4] + 0/4/8 = N$ | Data buffer is stuffed with dummy bytes if the length is not word aligned.<br>LINCR2 + BIDR + BDRL + BDRM |
| SADDR[31:0]     | RAM address           |   |
| SOFF[15:0]      | 4                     | Word increment  |
| SSIZE[2:0]      | 2                     | Word transfer   |
| SLAST[31:0]     | -N                    |   |
| DADDR[31:0]     | LINCR2 address        |   |
| DOFF[15:0]      | 4                     | Word increment  |
| DSIZE[2:0]      | 2                     | Word transfer   |
| DLAST_SGA[31:0] | -N                    | No scatter/gather processing  |
| INT_MAJ         | 0/1                   | Interrupt disabled/enabled  |
| D_REQ           | 1                     | Only on the last TCD of the chain.  |
| START           | 0                     | No software request   |

### 24.11.2 Master node, RX mode

On a master node in RX mode, the DMA interface requires a single RX channel. Each TCD controls a single frame, except for the extended frames (multiple TCDs). The memory map associated to the TCD chain (RAM area and LINFlexD registers) is shown in Figure 24-45.

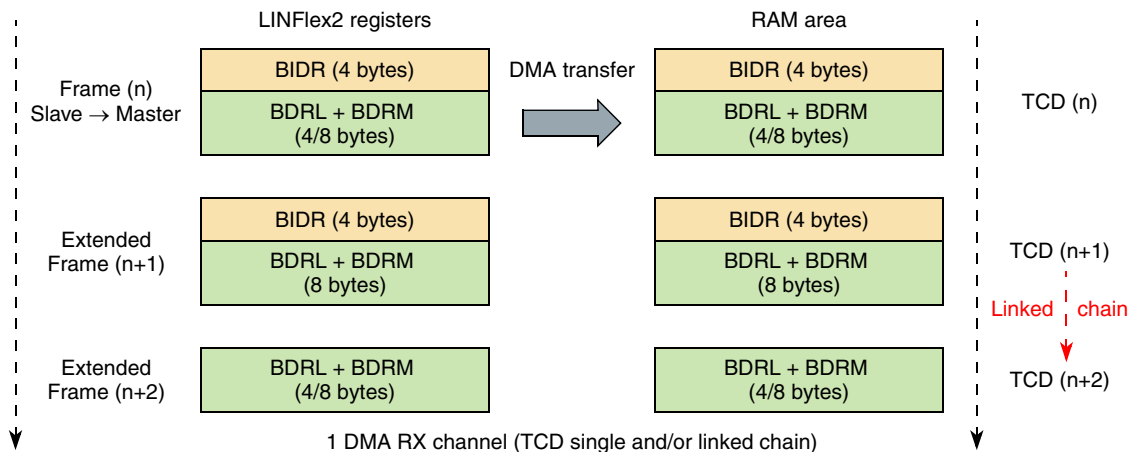


Figure 24-45. TCD chain memory map (master node, RX mode)

The TCD chain of the DMA Rx channel on a master node supports Slave-to-Master reception of the data field.

The BIDR register is optionally copied into the RAM area. This BIDR field (part of FIFO data) contains the ID of each message to allow the CPU to figure out which ID was received by the LINFlexD DMA if only the “one DMA channel” setup is used.

The concept FSM to control the DMA RX interface is shown in Figure 24-46. The DMA RX FSM will move to IDLE state immediately at next clock edge if DMARXE[0]=0.

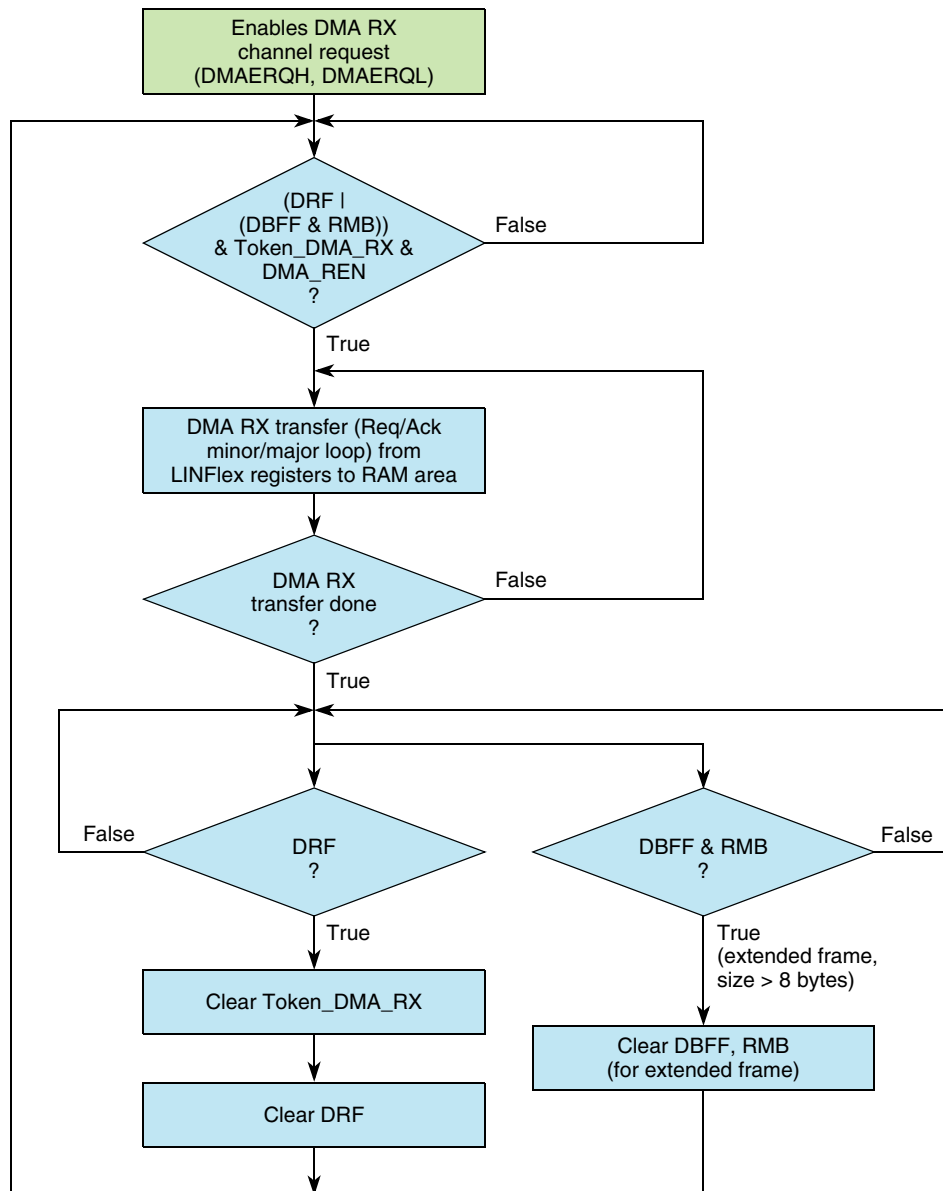


Figure 24-46. FSM to control the DMA RX interface (master node)

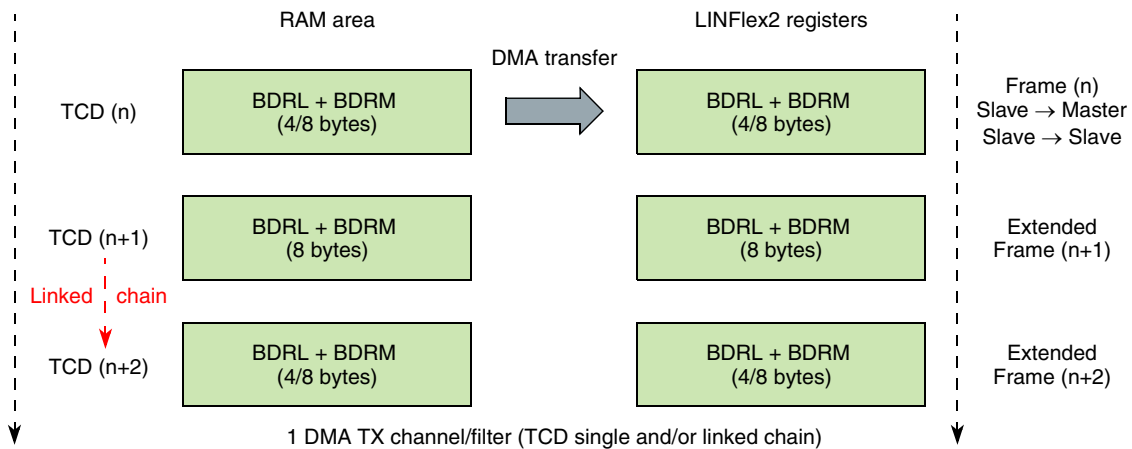
The TCD settings (word transfer) are shown in Table 24-42. All other TCD fields are equal to 0. TCD settings based on half-word or byte transfer are allowed.

**Table 24-42. TCD settings (master node, RX mode)**

| TCD field       | Value         | Description  |
|-----------------|---------------|--|
| CITER[14:0]     | 1             | Single iteration for the major loop  |
| BITER[14:0]     | 1             | Single iteration for the major loop  |
| NBYTES[31:0]    | [4] + 4/8 = N | Data buffer is stuffed with dummy bytes if the length is not word aligned.<br>BIDR + BDRL + BDRM |
| SADDR[31:0]     | BIDR address  |  |
| SOFF[15:0]      | 4             | Word increment   |
| SSIZE[2:0]      | 2             | Word transfer  |
| SLAST[31:0]     | -N            |  |
| DADDR[31:0]     | RAM address   |  |
| DOFF[15:0]      | 4             | Word increment   |
| DSIZE[2:0]      | 2             | Word transfer  |
| DLAST_SGA[31:0] | -N            | No scatter/gather processing   |
| INT_MAJ         | 0/1           | Interrupt disabled/enabled   |
| D_REQ           | 1             | Only on the last TCD of the chain.   |
| START           | 0             | No software request  |

### 24.11.3 Slave node, TX mode

On a slave node in TX mode, the DMA interface requires a DMA TX channel for each ID filter programmed in TX mode. In case a single DMA TX channel is available, a single ID field filter must be programmed in TX mode. Each TCD controls a single frame, except for the extended frames (multiple TCDs). The memory map associated to the TCD chain (RAM area and LINFlexD registers) is shown in Figure 24-47.



**Figure 24-47. TCD chain memory map (slave node, TX mode)**

The TCD chain of the DMA Tx channel on a slave node supports:

- Slave to Master: transmission of the data field
- Slave to Slave: transmission of the data field

The register settings of the LINCR2, IFER, IFMR, and IFCR registers are shown in [Table 24-43](#).

**Table 24-43. Register settings (slave node, TX mode)**

| LIN frame                            | LINCR2                           | IFER   | IFMR   | IFCR  |
|--------------------------------------|----------------------------------|--|--|---|
| Slave to Master<br>or Slave to Slave | DDRQ = 0<br>DTRQ = 0<br>HTRQ = 0 | To enable an ID filter<br>(Tx mode) for each<br>DMA TX channel | Identifier list mode<br>Identifier mask mode | DFL = payload size<br>ID = address<br>CCS = checksum<br>DIR = 1(TX) |

The concept FSM to control the DMA Tx interface is shown in [Figure 24-48](#). DMA TX FSM will move to idle state if  $DMATXE[x] = 0$ , where  $x = IFMI - 1$ .

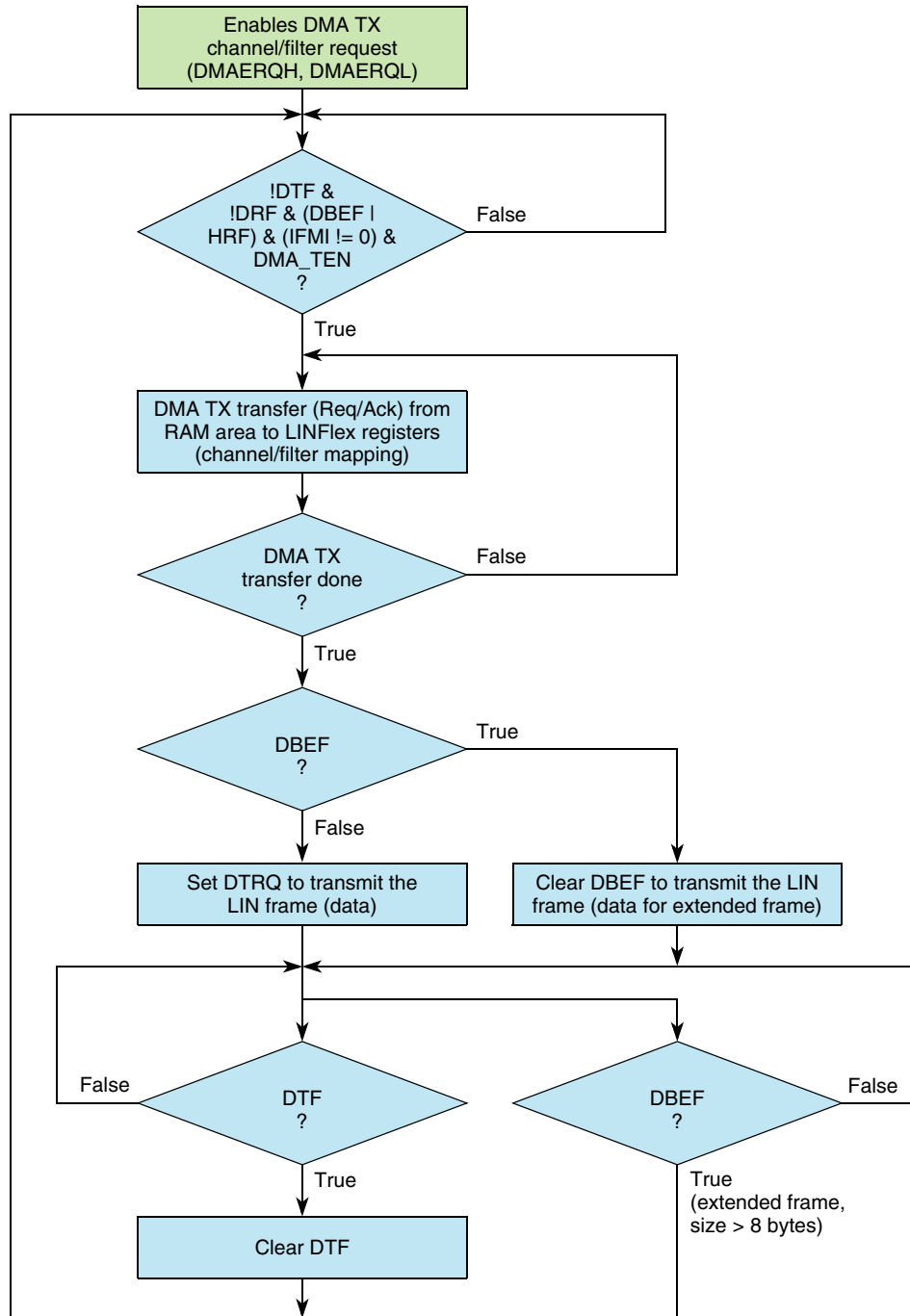


Figure 24-48. FSM to control the DMA TX interface (slave node)

The TCD settings (word transfer) are shown in [Table 24-44](#). All other TCD fields are equal to 0. TCD settings based on half-word or byte transfer are allowed.

Table 24-44. TCD settings (slave node, TX mode)

| TCD field       | Value        | Description   |
|-----------------|--------------|---|
| CITER[14:0]     | 1            | Single iteration for the major loop   |
| BITER[14:0]     | 1            | Single iteration for the major loop   |
| NBYTES[31:0]    | 4/8 = N      | Data buffer is stuffed with dummy bytes if the length is not word aligned.<br>BDRL + BDRM |
| SADDR[31:0]     | RAM address  |   |
| SOFF[15:0]      | 4            | Word increment  |
| SSIZE[2:0]      | 2            | Word transfer   |
| SLAST[31:0]     | -N           |   |
| DADDR[31:0]     | BDRL address |   |
| DOFF[15:0]      | 4            | Word increment  |
| DSIZE[2:0]      | 2            | Word transfer   |
| DLAST_SGA[31:0] | -N           | No scatter/gather processing  |
| INT_MAJ         | 0/1          | Interrupt disabled/enabled  |
| D_REQ           | 1            | Only on the last TCD of the chain.  |
| START           | 0            | No software request   |

### 24.11.4 Slave node, RX mode

On a slave node in RX mode, the DMA interface requires a DMA RX channel for each ID filter programmed in RX mode. In case a single DMA RX channel is available, a single ID field filter must be programmed in RX mode. Each TCD controls a single frame, except for the extended frames (multiple TCDs). The memory map associated to the TCD chain (RAM area and LINFlexD registers) is shown in Figure 24-49.

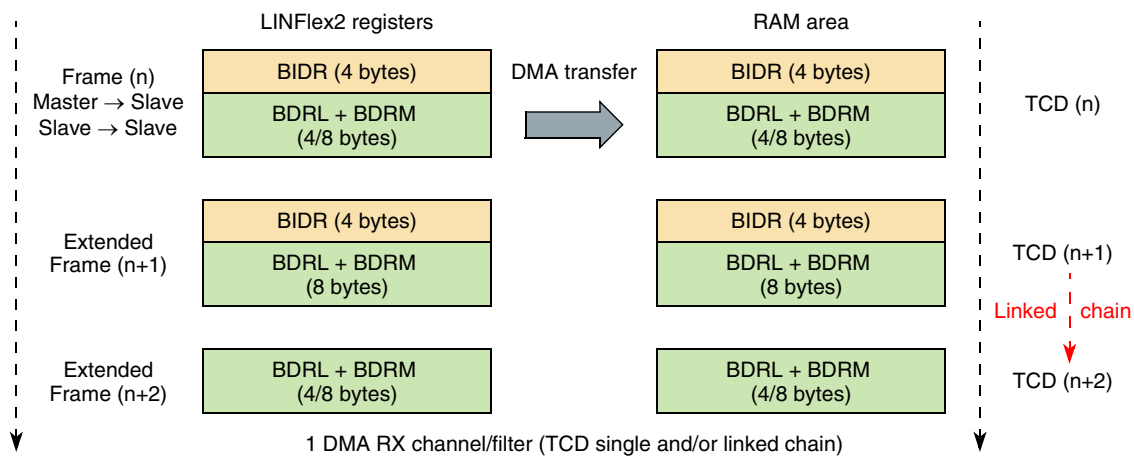


Figure 24-49. TCD chain memory map (slave node, RX mode)

The TCD chain of the DMA RX channel on a slave node supports:

- Master to Slave: reception of the data field.
- Slave to Slave: reception of the data field.

The register setting of the LINCR2, IFER, IFMR, and IFCR registers are given in [Table 24-45](#).

**Table 24-45. Register settings (slave node, RX mode)**

| LIN frame                            | LINCR2                           | IFER   | IFMR   | IFCR   |
|--------------------------------------|----------------------------------|--|--|--|
| Master to Slave<br>or Slave to Slave | DDRQ = 0<br>DTRQ = 0<br>HTRQ = 0 | To enable an ID filter<br>(Rx mode) for each<br>DMA RX channel | Identifier list mode<br>Identifier mask mode | DFL = payload size<br>ID = address<br>CCS = checksum<br>DIR = 0 (RX) |

The concept FSM to control the DMA Rx interface is shown in [Figure 24-50](#). DMA RX FSM will move to idle state if  $DMARXE[x] = 0$  where  $x = IFMI - 1$ .



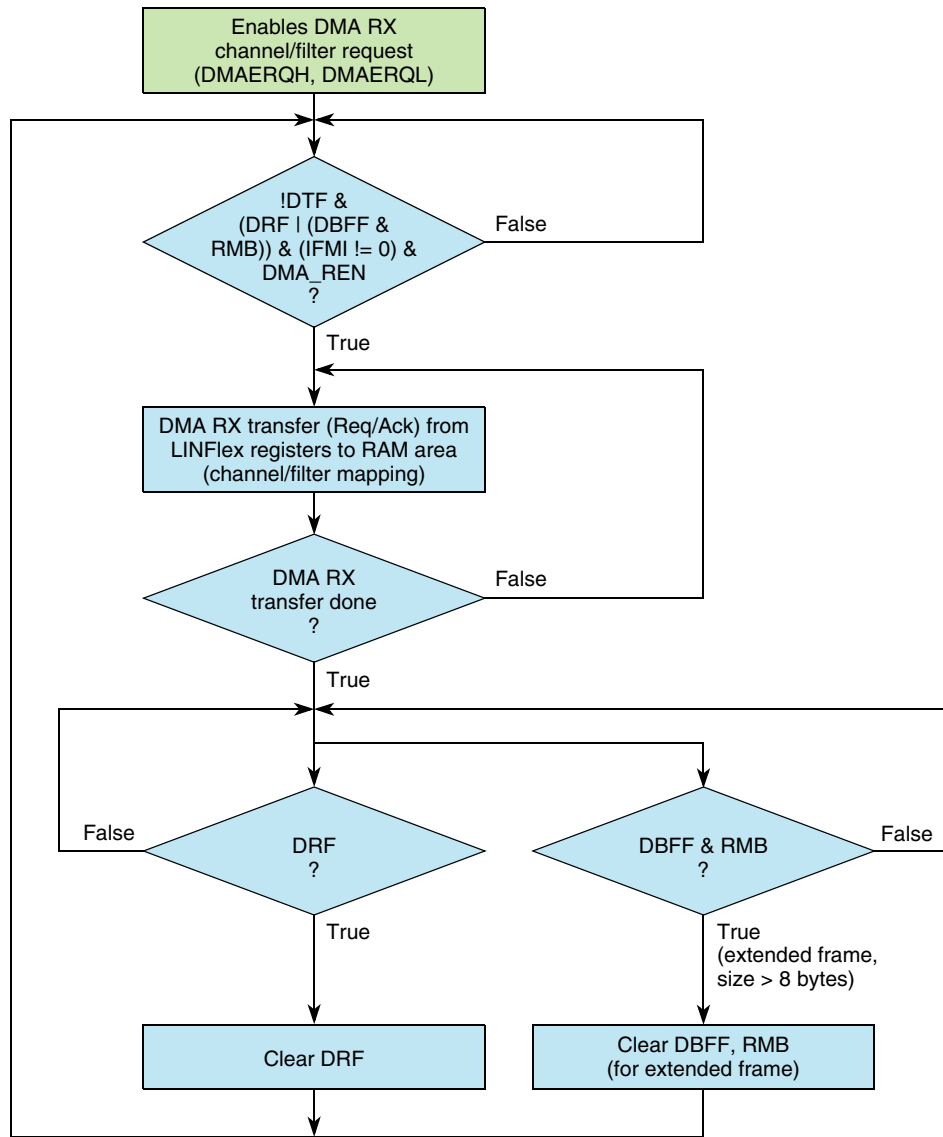


Figure 24-50. FSM to control the DMA RX interface (slave node)

The TCD settings (word transfer) are shown in Table 24-46. All other TCD fields = 0. TCD settings based on half-word or byte transfer are allowed.

Table 24-46. TCD settings (slave node, RX mode)

| TCD Field    | Value           | Description  |
|--------------|-----------------|--|
| CITER[14:0]  | 1               | Single iteration for the major loop  |
| BITER[14:0]  | 1               | Single iteration for the major loop  |
| NBYTES[31:0] | $[4] + 4/8 = N$ | Data buffer is stuffed with dummy bytes if the length is not word aligned.<br>BIDR + BDRL + BDRM |
| SADDR[31:0]  | BDRL address    |  |

Table 24-46. TCD settings (slave node, RX mode) (continued)

| TCD Field       | Value       | Description                        |
|-----------------|-------------|------------------------------------|
| SOFF[15:0]      | 4           | Word increment                     |
| SSIZE[2:0]      | 2           | Word transfer                      |
| SLAST[31:0]     | -N          |                                    |
| DADDR[31:0]     | RAM address |                                    |
| DOFF[15:0]      | 4           | Word increment                     |
| DSIZE[2:0]      | 2           | Word transfer                      |
| DLAST_SGA[31:0] | -N          | No scatter/gather processing       |
| INT_MAJ         | 0/1         | Interrupt disabled/enabled         |
| D_REQ           | 1           | Only on the last TCD of the chain. |
| START           | 0           | No software request                |

### 24.11.5 UART node, TX mode

In UART TX mode, the DMA interface requires a DMA TX channel. A single TCD can control the transmission of an entire Tx buffer. The memory map associated with the TCD chain (RAM area and LINFlexD registers) is shown in Figure 24-51.

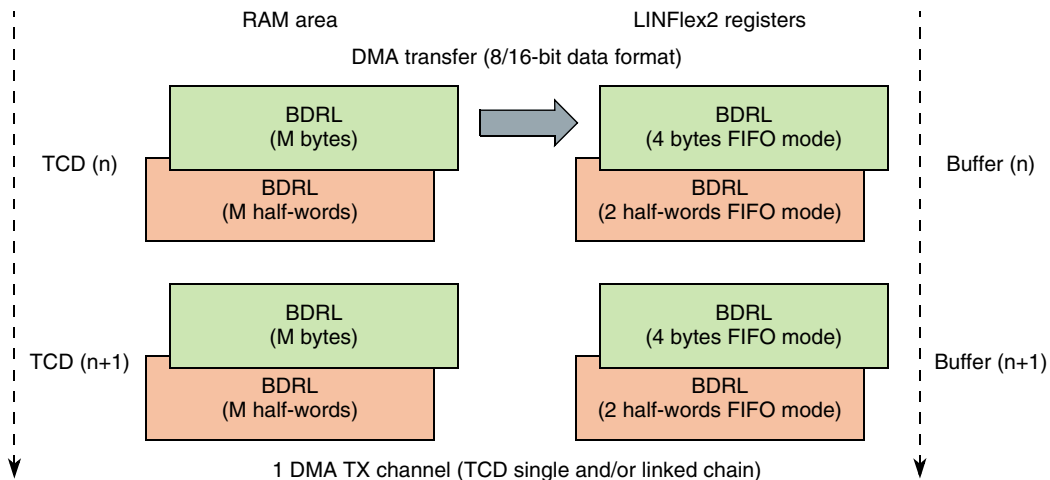


Figure 24-51. TCD chain memory map (UART node, TX mode)

The UART TX buffer must be configured in FIFO mode in order to:

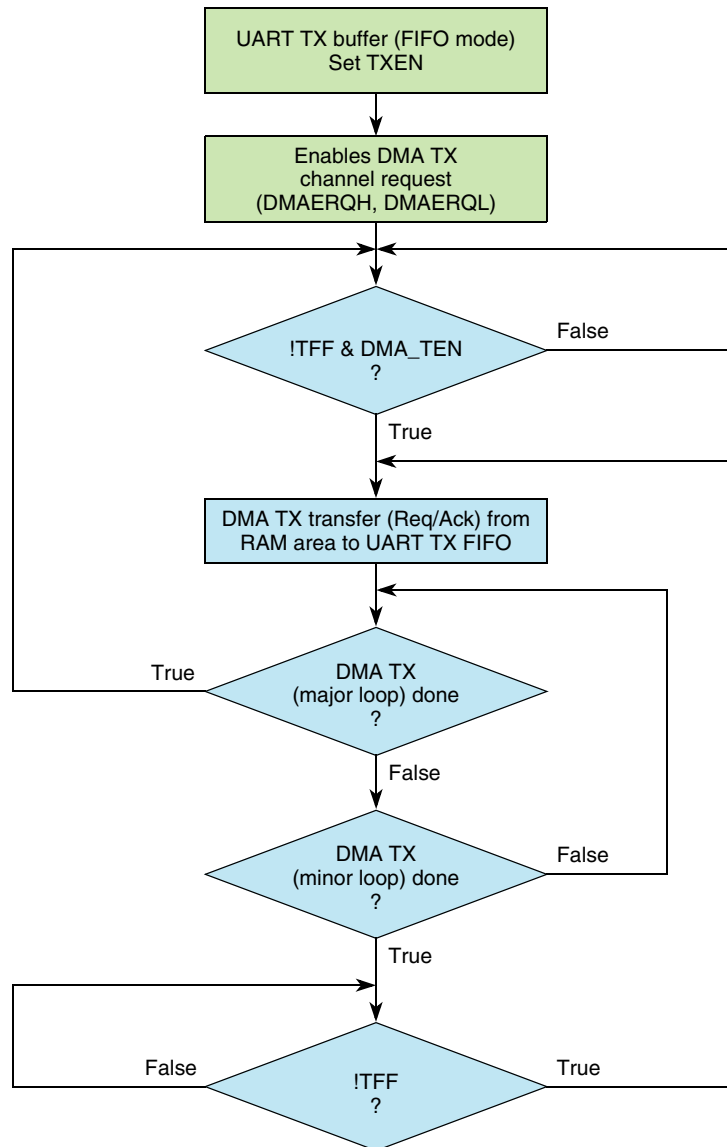
- Allow the transfer of large data buffer by a single TCD
- Adsorb the latency, following a DMA request (due to the DMA arbitration), to move data from the RAM to the FIFO
- Use low priority DMA channels
- Support the UART baud rate (2 Mb/s) without underrun events

The Tx FIFO size is:

- 4 bytes in 8-bit data format
- 2 half-words in 16-bit data format

A DMA request is triggered by FIFO not full (TX) status signals.

The concept FSM to control the DMA TX interface is shown in [Figure 24-52](#). DMA TX FSM will move to idle state if `DMATXE[0] = 0`.



**Figure 24-52. FSM to control the DMA TX interface (UART node)**

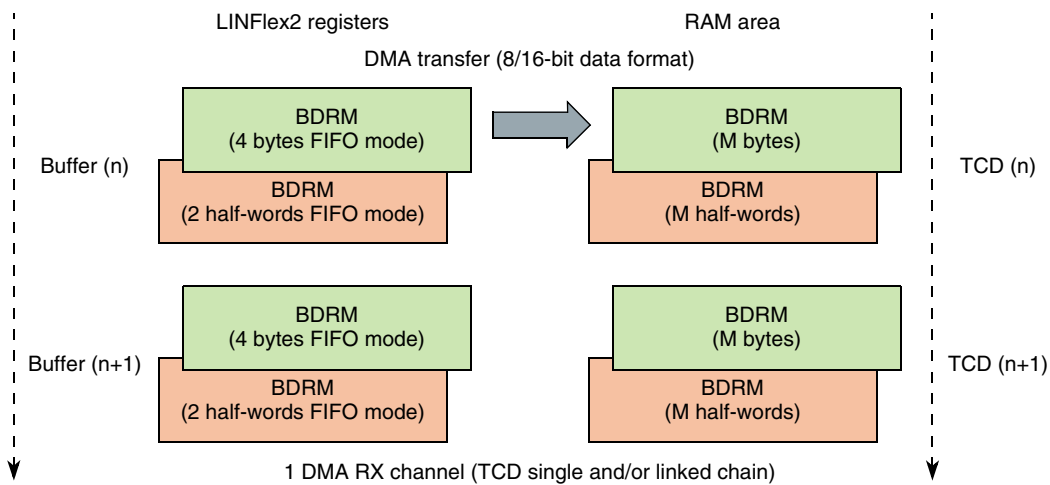
The TCD settings (typical case) are shown in [Table 24-47](#). All other TCD fields = 0. The minor loop transfers a single byte/half-word as soon a free entry is available in the Tx FIFO.

**Table 24-47. TCD settings (UART node, TX mode)**

| TCD Field       | Value        |             | Description   |
|-----------------|--------------|-------------|---|
|                 | 8-bit data   | 16-bit data |   |
| CITER[14:0]     | M            |             | Multiple iterations for the major loop  |
| BITER[14:0]     | M            |             | Multiple iterations for the major loop  |
| NBYTES[31:0]    | 1            | 2           | Minor loop transfer = 1 or 2 bytes  |
| SADDR[31:0]     | RAM address  |             |   |
| SOFF[15:0]      | 1            | 2           | Byte/half-word increment  |
| SSIZE[2:0]      | 0            | 1           | Byte/half-word transfer   |
| SLAST[31:0]     | -M           | -M × 2      |   |
| DADDR[31:0]     | BDRL address |             | DADDR = BDRL + 0x3 for byte transfer<br>DADDR = BDRL + 0x2 for half-word transfer |
| DOFF[15:0]      | 0            |             | No increment (FIFO)   |
| DSIZE[2:0]      | 0            | 1           | Byte/Half-word transfer   |
| DLAST_SGA[31:0] | 0            |             | No scatter/gather processing  |
| INT_MAJ         | 0/1          |             | Interrupt disabled/enabled  |
| D_REQ           | 1            |             | Only on the last TCD of the chain.  |
| START           | 0            |             | No software request   |

### 24.11.6 UART node, RX mode

In UART RX mode, the DMA interface requires a DMA RX channel. A single TCD can control the reception of an entire Rx buffer. The memory map associated with the TCD chain (RAM area and LINFlexD registers) is shown in [Figure 24-53](#).



**Figure 24-53. TCD chain memory map (UART node, RX mode)**

The UART RX buffer must be configured in FIFO mode in order to:

- Allow the transfer of large data buffer by a single TCD
- Adsorb the latency, following a DMA request (due to the DMA arbitration), to move data from the FIFO to the RAM
- Use low priority DMA channels
- Support high UART baud rate (at least 2 Mb/s) without overrun events

The Rx FIFO size is:

- 4 bytes in 8-bit data format
- 2 half-words in 16-bit data format

This is sufficient because just one byte allows a reaction time of about 3.8  $\mu$ s (at 2 Mbit/s), corresponding to about 450 clock cycles at 120 MHz, before the transmission is affected. A DMA request is triggered by FIFO not empty (RX) status signals.

The concept FSM to control the DMA Rx interface is shown in [Figure 24-54](#). DMA Rx FSM will move to idle state if DMARXE[0] = 0.

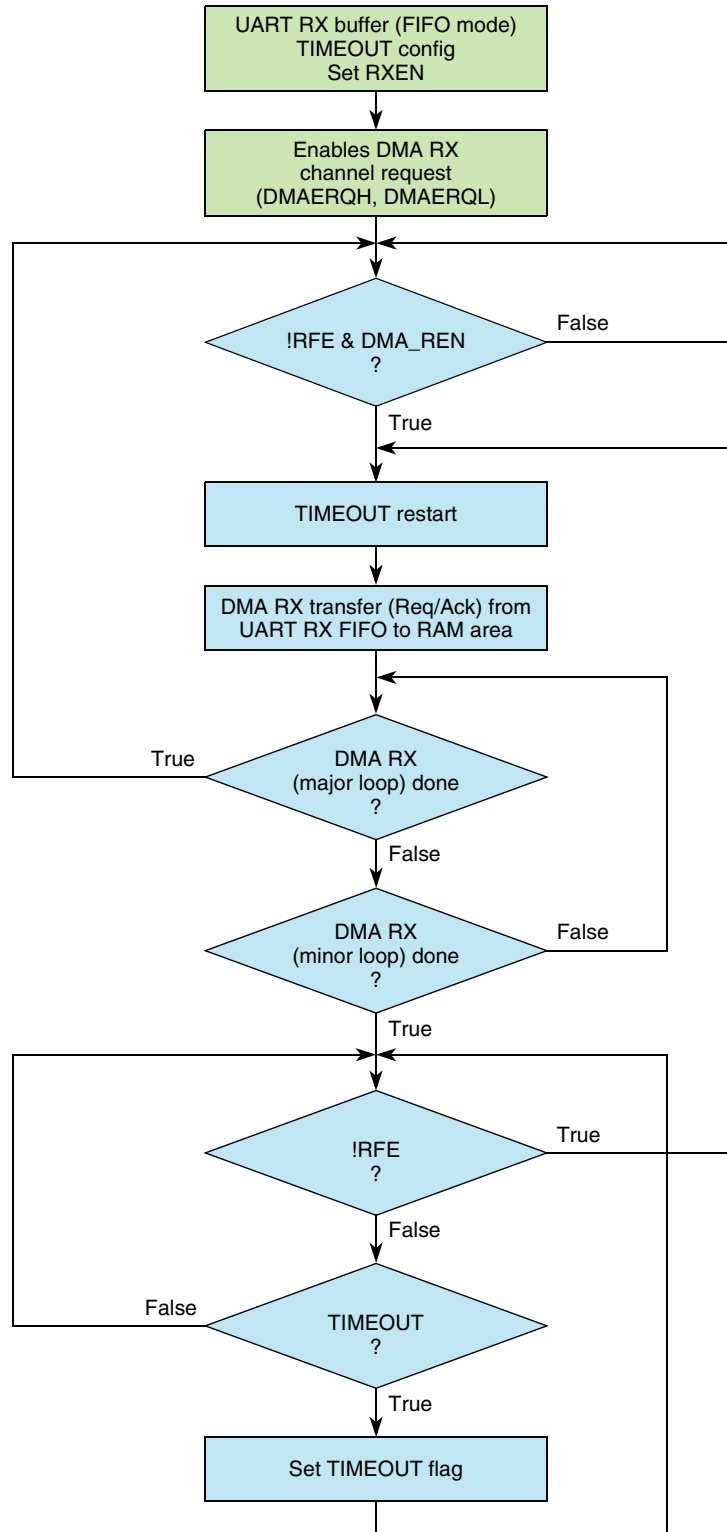


Figure 24-54. FSM to control the DMA RX interface (UART node)

The TCD settings (typical case) are shown in Table 24-48. All other TCD fields = 0. The minor loop transfers a single byte/half-word as soon an entry is available in the Rx FIFO. A new software reset bit is

required that allows the LINFlexD FSMs to be reset in case this timeout state is reached or in any other case. Timeout counter can be rewritten by software at any time to extend timeout period.

**Table 24-48. TCD settings (UART node, RX mode)**

| TCD Field       | Value        |              | Description   |
|-----------------|--------------|--------------|---|
|                 | 8 bits data  | 16 bits data |   |
| CITER[14:0]     | M            |              | Multiple iterations for the major loop  |
| BITER[14:0]     | M            |              | Multiple iterations for the major loop  |
| NBYTES[31:0]    | 1            | 2            | Minor loop transfer = 1 or 2 bytes  |
| SADDR[31:0]     | BDRM address |              | SADDR = BDRM + 0x3 for byte transfer<br>SADDR = BDRM + 0x2 for half-word transfer |
| SOFF[15:0]      | 0            |              | No increment (FIFO)   |
| SSIZE[2:0]      | 0            | 1            | Byte/half-word transfer   |
| SLAST[31:0]     | 0            |              |   |
| DADDR[31:0]     | RAM address  |              |   |
| DOFF[15:0]      | 1            | 2            | Byte/half-word increment  |
| DSIZE[2:0]      | 0            | 1            | Byte/half-word transfer   |
| DLAST_SGA[31:0] | -M           | -M × 2       | No scatter/gather processing  |
| INT_MAJ         | 0/1          |              | Interrupt disabled/enabled  |
| D_REQ           | 1            |              | Only on the last TCD of the chain.  |
| START           | 0            |              | No software request   |

### 24.11.7 Use cases and limitations

- In LIN slave mode, the DMA capability can be used only if the ID filtering mode is activated. The number of ID filters enabled must be equal to the number of DMA channels enabled. The correspondence between channel # and ID filter is based on IFMI (identifier filter match index).
- In LIN master mode, both the DMA channels (TX and RX) must be enabled in case the DMA capability is required.
- In UART mode, the DMA capability can be used only if the UART Tx/Rx buffers are configured as FIFOs.
- DMA and CPU operating modes are mutually exclusive for the data/frame transfer on a UART or LIN node. Once a DMA transfer is finished, the CPU can handle subsequent accesses.
- Error management must be always executed via CPU enabling the related error interrupt sources. The DMA capability does not provide support for the error management. Error management means checking status bits, handling IRQs, and potentially canceling DMA transfers.
- The DMA programming model must be coherent with the TCD setting defined in this document.

## 24.12 Functional description

### 24.12.1 8-bit timeout counter

#### 24.12.1.1 LIN timeout mode

Clearing the LTOM bit (setting its value to 0) in the LINTCSR enables the LIN timeout mode. The LINOCCR becomes read-only, and OC1 and OC2 output compare values in the LINOCCR are automatically updated by hardware.

This configuration detects header timeout, response timeout, and frame timeout.

Depending on the LIN mode (selected by the MME bit in LINCR1), the 8-bit timeout counter will behave differently.

LIN timeout mode must not be enabled during LIN extended frames transmission or reception (that is, if the data field length in the BIDR is configured with a value higher than 8 data bytes).

##### 24.12.1.1.1 LIN Master mode

Field RTO in the LINTOCR can be used to tune response timeout and frame timeout values. Header timeout value is fixed to HTO = 28-bit time.

Field OC1 checks THeader and TResponse and field OC2 checks TFrame (refer to [Figure 24-55 \(Header and response timeout\)](#)).

When LINFlexD moves from Break delimiter state to Synch Field state (refer to [Section 24.10.3, LIN status register \(LINSR\)](#)):

- OC1 is updated with the value of OCHeader ( $\text{OCHeader} = \text{CNT} + 28$ ),
- OC2 is updated with the value of OCFrame ( $\text{OCFrame} = \text{CNT} + 28 + \text{RTO} \times 9$  (frame timeout value for an 8-byte frame))
- the TOCE bit is set.

On the start bit of the first response data byte (and if no error occurred during the header reception), OC1 is updated with the value of OCResponse ( $\text{OCResponse} = \text{CNT} + \text{RTO} \times 9$  (response timeout value for an 8-byte frame)).

On the first response byte is received, OC1 and OC2 are automatically updated to check TResponse and TFrame according to RTO (tolerance) and DFL.

On the checksum reception or in case of error in the header or response, the TOCE bit is reset.

If there is no response, frame timeout value does not take into account the DFL value, and an 8-byte response ( $\text{DFL} = 7$ ) is always assumed.

##### 24.12.1.1.2 LIN Slave mode

Field RTO in the LINTOCR can be used to tune response timeout and frame timeout values. Header timeout value is fixed to HTO.



OC1 checks THeader and TResponse and OC2 checks TFrame (refer to [Figure 24-55 \(Header and response timeout\)](#)).

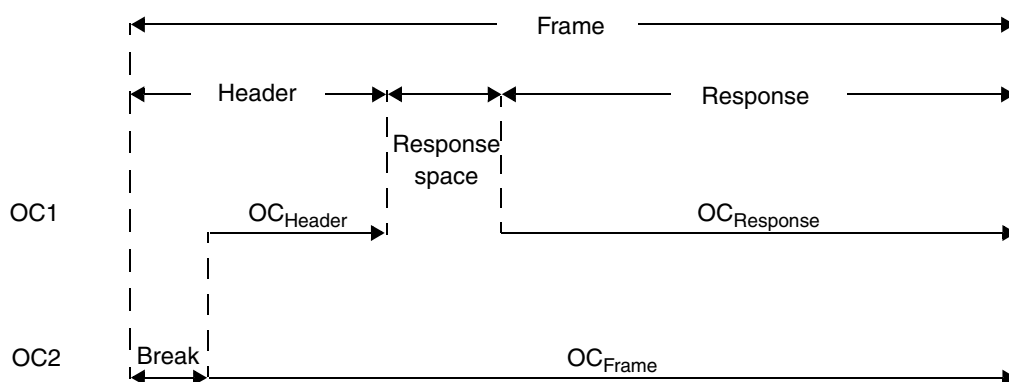
When LINFlexD moves from Break state to Break Delimiter state (refer to [Section 24.10.3, LIN status register \(LINSR\)](#)):

- OC1 is updated with the value of OCHeader ( $OCHeader = CNT + HTO$ ),
- OC2 is updated with the value of OCFrame ( $OCFrame = CNT + HTO + RTO \times 9$  (frame timeout value for an 8-byte frame)),
- The TOCE bit is set.

On the start bit of the first response data byte (and if no error occurred during the header reception), OC1 is updated with the value of OCResponse ( $OCResponse = CNT + RTO \times 9$  (response timeout value for an 8-byte frame)).

Once the first response byte is received, OC1 and OC2 are automatically updated to check TResponse and TFrame according to RTO (tolerance) and DFL.

On the checksum reception or in case of error in the header or data field, the TOCE bit is reset.



**Figure 24-55. Header and response timeout**

### 24.12.1.2 Output compare mode

Setting `LINTCSR[LTOM] = 1` enables the output compare mode. This mode allows the user to fully customize the use of the counter.

OC1 and OC2 output compare values can be updated in the `LINTOCR` by software.

## 24.12.2 Interrupts

**Table 24-49. LINFlexD interrupt control**

| Interrupt event            | Event flag bit | Enable control bit | Interrupt vector |
|----------------------------|----------------|--------------------|------------------|
| Header Received interrupt  | HRF            | HRIE               | RXI <sup>1</sup> |
| Data Transmitted interrupt | DTF            | DTIE               | TXI              |

Table 24-49. LINFlexD interrupt control (continued)

| Interrupt event                  | Event flag bit | Enable control bit | Interrupt vector |
|----------------------------------|----------------|--------------------|------------------|
| Data Received interrupt          | DRF            | DRIE               | RXI              |
| Data Buffer Empty interrupt      | DBEF           | DBEIE              | TXI              |
| Data Buffer Full interrupt       | DBFF           | DBFIE              | RXI              |
| Wake-up interrupt                | WUPF           | WUPIE              | RXI              |
| LIN State interrupt <sup>2</sup> | LSF            | LSIE               | RXI              |
| Buffer Overrun interrupt         | BOF            | BOIE               | ERR              |
| Framing Error interrupt          | FEF            | FEIE               | ERR              |
| Header Error interrupt           | HEF            | HEIE               | ERR              |
| Checksum Error interrupt         | CEF            | CEIE               | ERR              |
| Bit Error interrupt              | BEF            | BEIE               | ERR              |
| Output Compare interrupt         | OCF            | OCIE               | ERR              |
| Stuck at Zero interrupt          | SZF            | SZIE               | ERR              |

<sup>1</sup> In Slave mode, if at least one filter is configured as TX and enabled, header received interrupt vector is RXI or TXI depending on the value of identifier received.

<sup>2</sup> For debug and validation purposes.

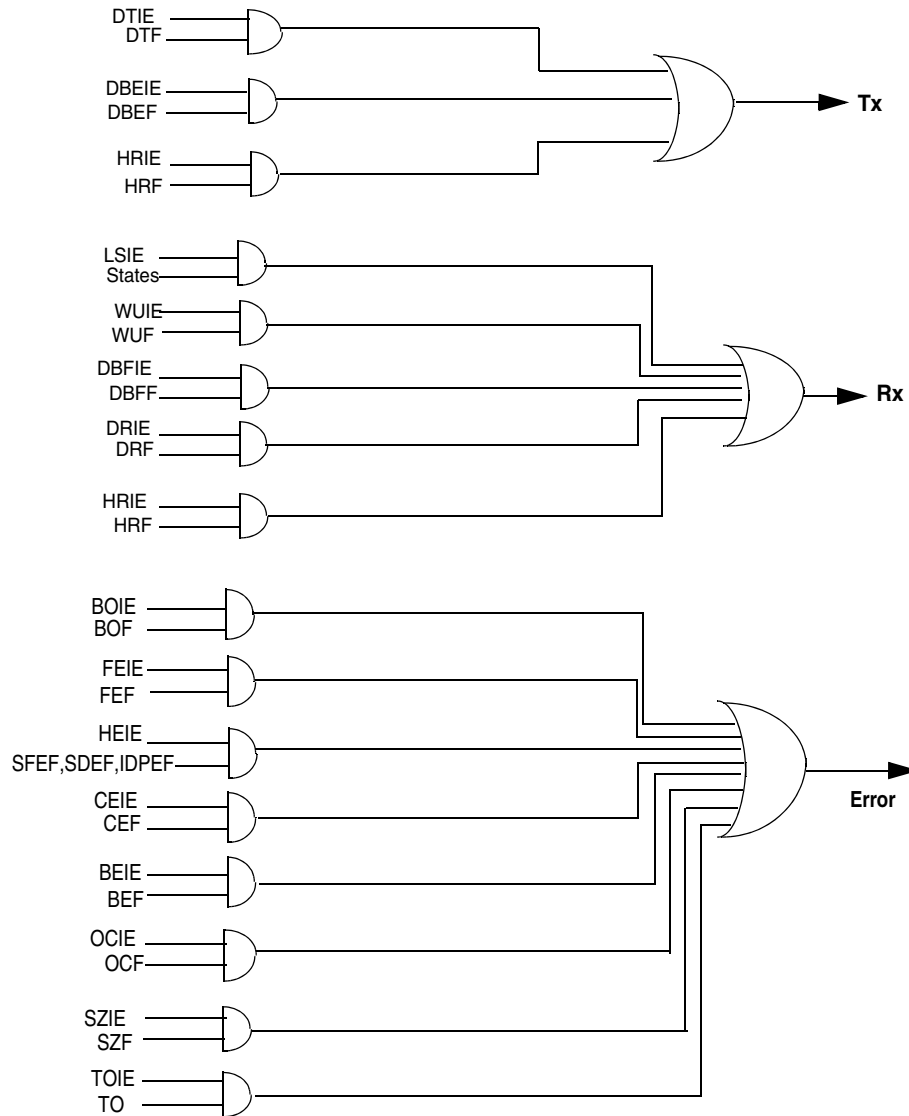


Figure 24-56. Interrupt diagram

### 24.12.3 Fractional baud rate generation

The baud rates for the receiver and transmitter are both set to the same value as programmed in the Mantissa (LINIBRR) and Fraction (LINFBR) registers.

$$\text{Tx/Rx baud} = \frac{f_{\text{ipg\_clock\_lin}}}{(16 \times \text{LFDIV})}$$

LFDIV is an unsigned fixed point number. The 20-bit mantissa is coded in the LINIBRR register and the fraction is coded in the LINFBR register.

The following examples show how to derive LFDIV from LINIBRR and LINFBR register values:

**Example 24-1.**

If LINIBRR = 27d and LINFBR = 12d, then

Mantissa (LFDIV) = 27d

Fraction (LFDIV) = 12/16 = 0.75d

Therefore LFDIV = 27.75d

**Example 24-2.**

To program LFDIV = 25.62d,

LINFBR =  $16 \times 0.62 = 9.92$ , nearest real number 10d = Ah

LINIBRR = mantissa(25.62d) = 25d = 19h

**NOTE**

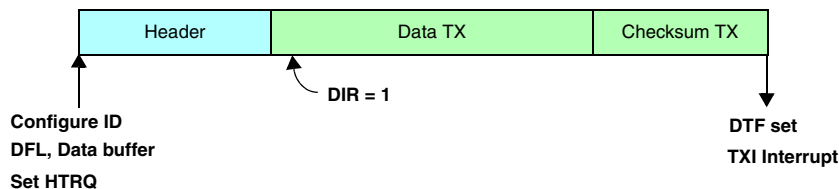
The Baud Counters are updated with the new value of the Baud Registers after a write to LINIBRR. Hence the Baud Register value must not be changed during a transaction. The LINFBR (containing the Fraction bits) must be programmed before LINIBRR.

**NOTE**

LFDIV must be greater than or equal to 1.5d, for example, LINIBRR = 1 and LINFBR = 8. Therefore, the maximum possible baud rate is  $f_{periph\_set\_1\_clk} / 24$ .

**24.13 Programming considerations**

This section describes the various configurations in which the LINFlexD can be used.

**24.13.1 Master node**

**Figure 24-57. Programming consideration: master node, transmitter**

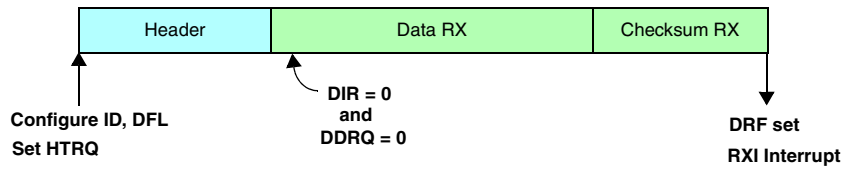


Figure 24-58. Programming consideration: master node, receiver

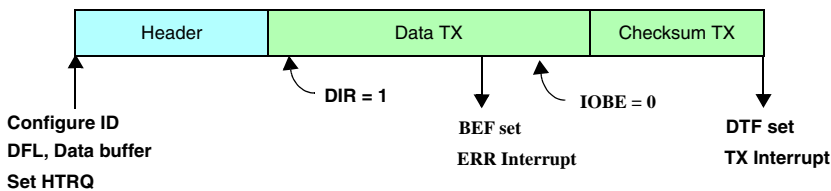
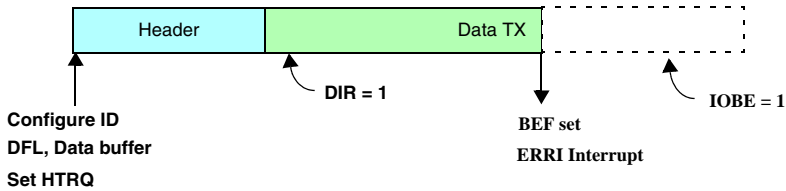


Figure 24-59. Programming consideration: master node, transmitter, bit error

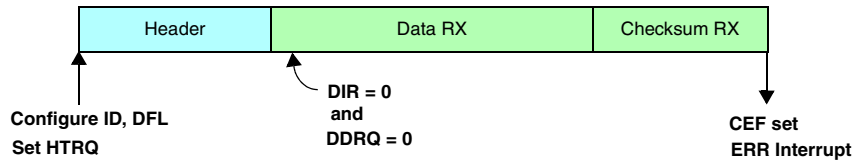


Figure 24-60. Programming consideration: master node, receiver, checksum error

### 24.13.2 Slave node

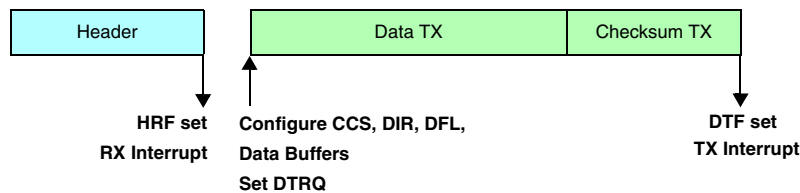


Figure 24-61. Programming consideration: slave node, transmitter, no filters

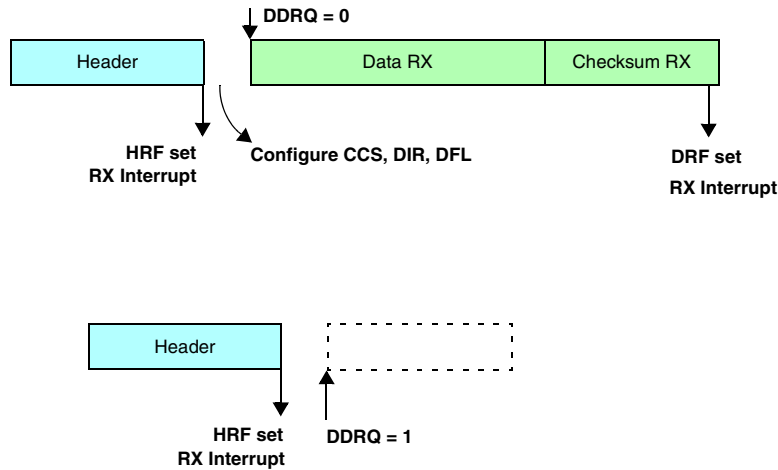


Figure 24-62. Programming consideration: slave node, receiver, no filters

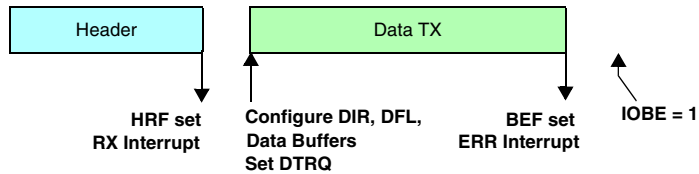


Figure 24-63. Programming consideration: slave node, transmitter, no filters, bit error

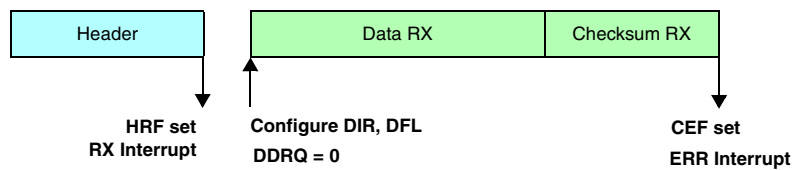
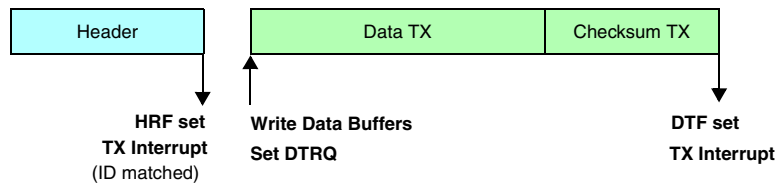


Figure 24-64. Programming consideration: slave node, receiver, no filters, checksum error



**Note:** This configuration can be used in case the slave never receives data (for example, as with a sensor).

Figure 24-65. Programming consideration: slave node, at least one TX filter, BF is reset, ID matches filter

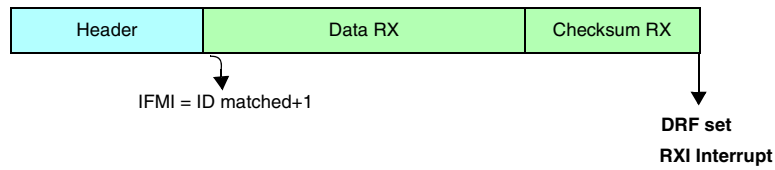


Figure 24-66. Programming consideration: slave node, at least one RX filter, BF is reset, ID matches filter

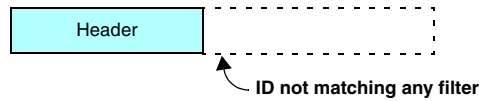
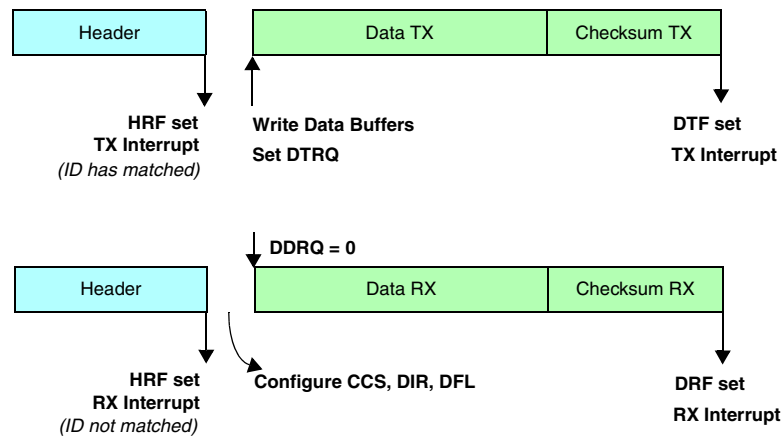


Figure 24-67. Programming consideration: slave node, RX only, TX only, RX and TX filters, ID not matching filter, BF is reset



**Note:** This configuration is used when:

- a) All TX IDs are managed by filters
- b) The number of other filters is not enough to manage all reception IDs

Figure 24-68. Programming consideration: slave node, TX filter, BF is set

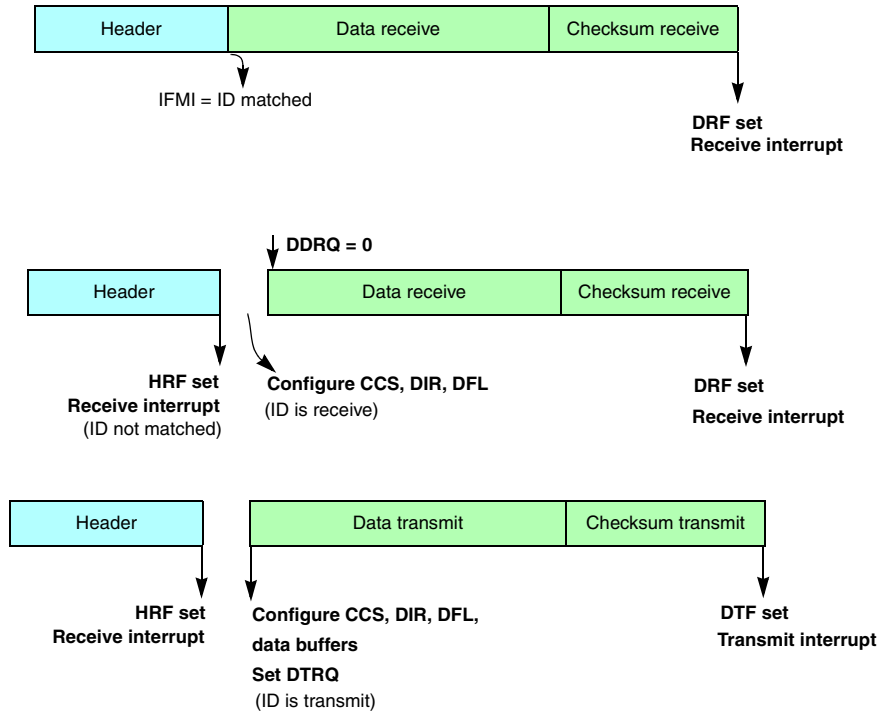
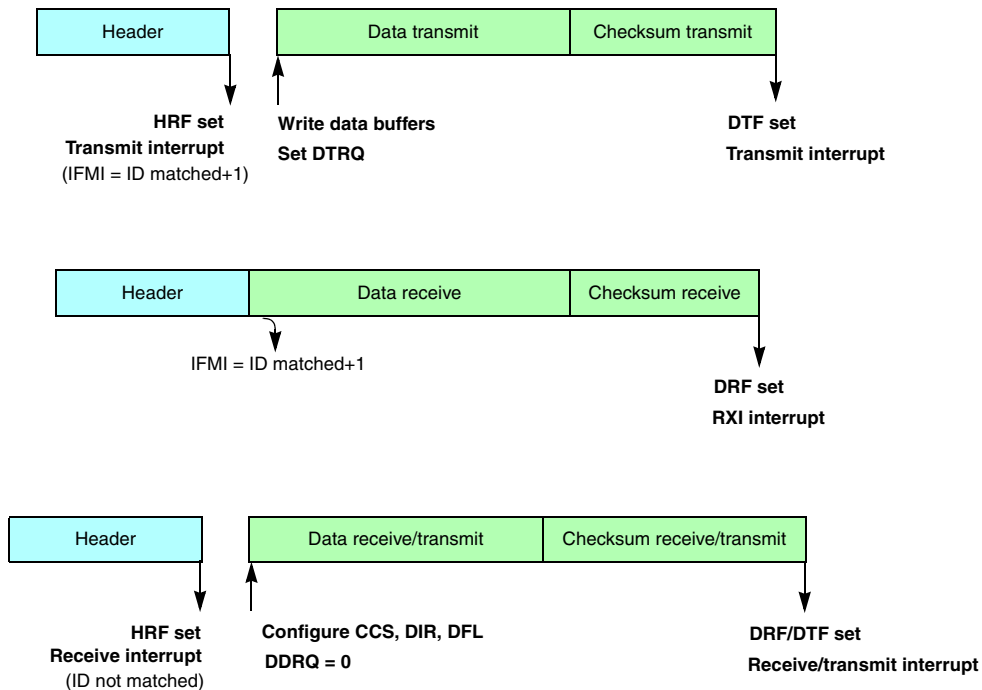


Figure 24-69. Programming consideration: slave node, RX filter, BF is set



**Note:** This configuration is used when:

- a) The number of filters is not enough
- b) Filters are used for most frequently used IDs to reduce CPU usage

Figure 24-70. Programming consideration: slave node, TX filter, RX filter, BF is set



### 24.13.3 Extended frames

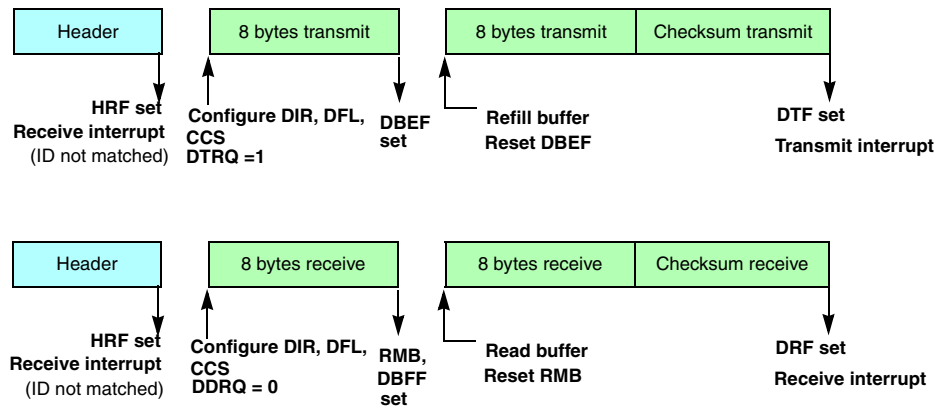


Figure 24-71. Programming consideration: extended frames

### 24.13.4 Timeout

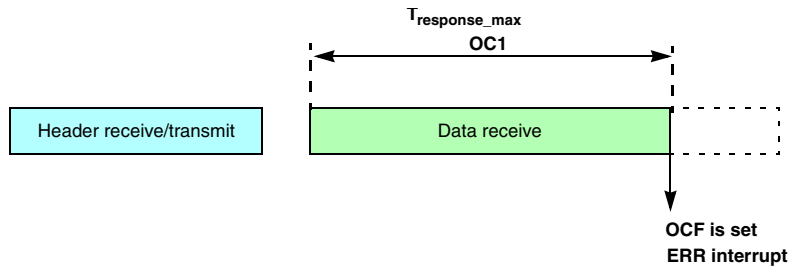


Figure 24-72. Programming consideration: response timeout

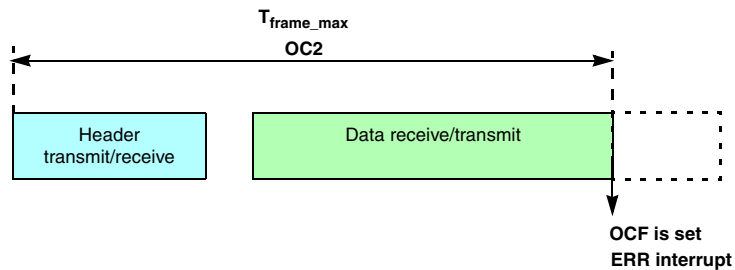


Figure 24-73. Programming consideration: frame timeout

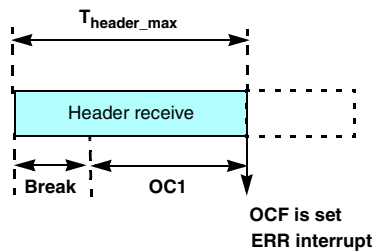


Figure 24-74. Programming consideration: header timeout

### 24.13.5 UART mode

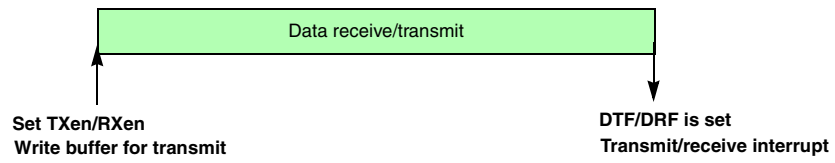


Figure 24-75. Programming consideration: UART mode

This page is intentionally left blank.



# Chapter 25

## FlexCAN

### 25.1 Information specific to this device

This section presents device-specific parameterization and customization information not specifically referenced in the remainder of this chapter.

#### 25.1.1 Device-specific features

The device has six Controller Area Network (FlexCAN) blocks.

- Each block supports 64 Message Buffers (MB).
- DMA support is not provided.
- It is possible to operate the FlexCAN bit timing logic with either system clock or 4–40 MHz fast external crystal oscillator clock (FXOSC).
- In the case of safe mode entry, the pad associated with CANTX can optionally be put into a high-impedance state (not recessive state)
- Modes of operation:
  - Four functional modes: Normal (User and Supervisor), Freeze, Listen-Only, and Loop-Back
  - One low-power mode (Disable mode)
- 1056 bytes (64 MBs) of RAM used for MB storage
- 256 bytes (64 MBs) of RAM used for individual Rx Mask registers
- Hardware cancellation on Tx message buffers
- Module Configuration Register (MCR): Bits 5, 9, 12, and 13 are reserved
- Error and Status Register (ESR): Bit 31 is reserved

### 25.2 Introduction

The FlexCAN module is a communication controller implementing the CAN protocol according to the CAN 2.0B protocol specification. A general block diagram is shown in [Figure 25-1](#), which describes the main sub-blocks implemented in the FlexCAN module, including two embedded memories, one for storing Message Buffers (MB) and another one for storing Rx Individual Mask registers. Support for as many as 64 Message Buffers is provided. The functions of the sub-modules are described in subsequent sections.

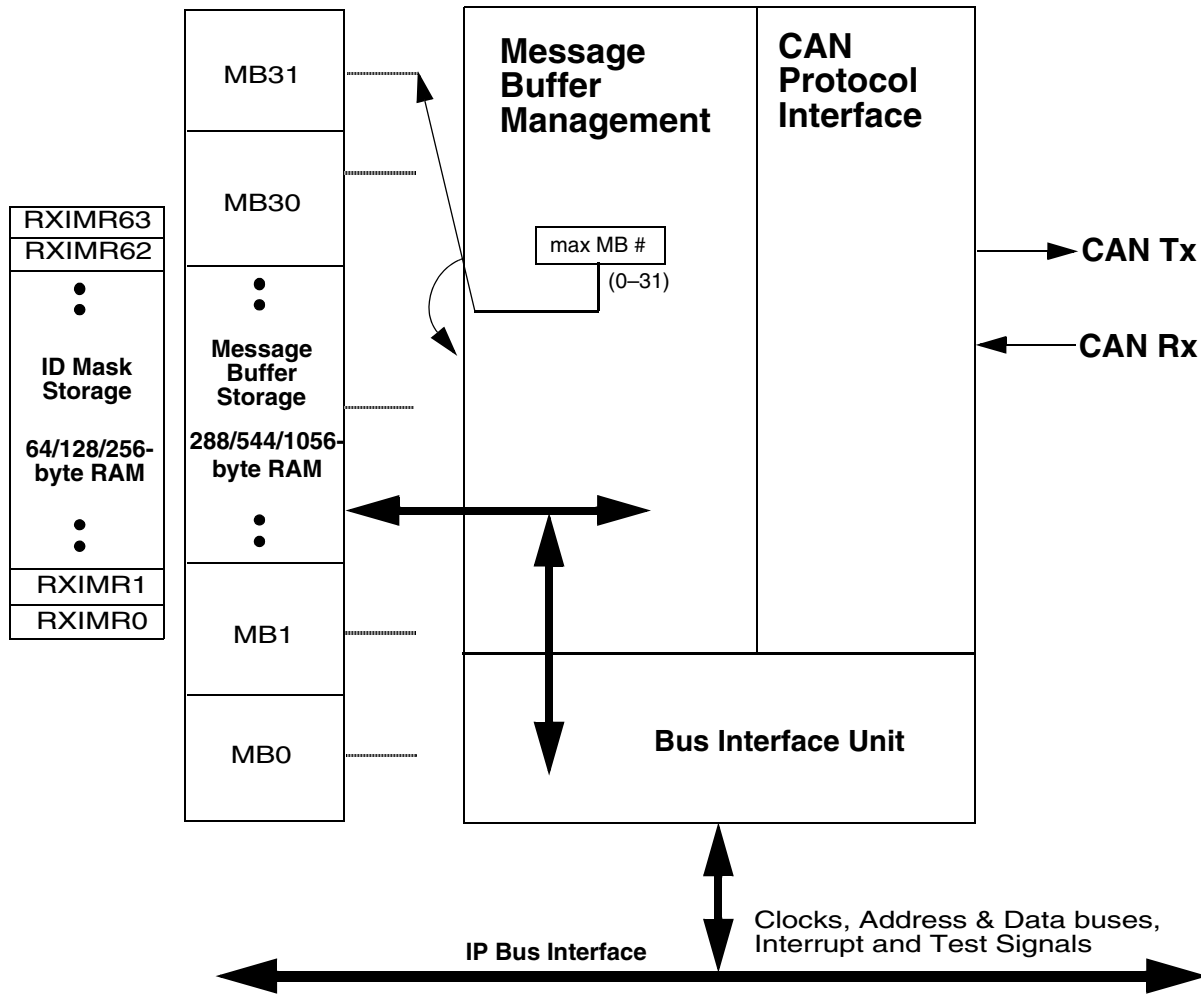


Figure 25-1. FlexCAN block diagram

### 25.2.1 Overview

The CAN protocol was primarily, but not only, designed to be used as a vehicle serial data bus, meeting the specific requirements of this field: real-time processing, reliable operation in the EMI environment of a vehicle, cost-effectiveness, and required bandwidth. The FlexCAN module is a full implementation of the CAN protocol specification, Version 2.0 B, which supports both standard and extended message frames. A flexible number of Message Buffers (16, 32 or 64) is also supported. The Message Buffers are stored in an embedded RAM dedicated to the FlexCAN module.

The CAN Protocol Interface (CPI) submodule manages the serial communication on the CAN bus, requesting RAM access for receiving and transmitting message frames, validating received messages, and performing error handling. The Message Buffer Management (MBM) submodule handles Message Buffer selection for reception and transmission, taking care of arbitration and ID matching algorithms. The Bus Interface Unit (BIU) submodule controls the access to and from the internal interface bus in order to

establish connection to the CPU and to other blocks. Clocks, address and data buses, interrupt outputs, and test signals are accessed through the Bus Interface Unit.

## 25.2.2 FlexCAN module features

The FlexCAN module includes these distinctive features:

- Full Implementation of the CAN protocol specification, Version 2.0B
  - Standard data and remote frames
  - Extended data and remote frames
  - 0 to 8 bytes data length
  - Programmable bit rate as fast as 1 Mbit/s
  - Content-related addressing
- Flexible Message Buffers (as many as 64) of 0 to 8 bytes data length
- Each MB configurable as Rx or Tx, all supporting standard and extended messages
- Individual Rx Mask registers per Message Buffer (MB)
- Includes either 1056 bytes (64 MBs), 544 bytes (32 MBs), or 288 bytes (16 MBs) of RAM used for MB storage
- Includes either 256 bytes (64 MBs), 128 bytes (32 MBs), or 64 bytes (16 MBs) of RAM used for individual Rx Mask registers
- Full featured Rx FIFO with storage capacity for 6 frames and internal pointer handling
- Powerful Rx FIFO ID filtering, capable of matching incoming IDs against either 8 extended, 16 standard, or 32 partial (8 bits) IDs, with individual masking capability
- Selectable backwards compatibility with previous FlexCAN version
- Programmable clock source to the CAN Protocol Interface, either bus clock or crystal oscillator
- Unused MB and Rx Mask register space can be used as general purpose RAM space
- Listen only mode capability
- Programmable loopback mode supporting self-test operation
- Programmable transmission priority scheme: lowest ID, lowest buffer number or highest priority
- Time Stamp based on 16-bit free-running timer
- Global network time, synchronized by a specific message
- Maskable interrupts
- Independent of the transmission medium (an external transceiver is assumed)
- Short latency time due to an arbitration scheme for high-priority messages
- Low power modes, with programmable wake up on bus activity

## 25.2.3 Modes of operation

The FlexCAN module has four functional modes: Normal mode (User and Supervisor), Freeze mode, Listen-Only mode, and Loop-Back mode. There is also a low-power mode (Disable mode).

- Normal mode (User or Supervisor):  
In Normal Mode, the module operates receiving and/or transmitting message frames, errors are handled normally and all the CAN Protocol functions are enabled. User and Supervisor Modes differ in the access to some restricted control registers.
- Freeze mode:  
It is enabled when the FRZ bit in MCR is asserted. If enabled, Freeze Mode is entered when the HALT bit in MCR is set or when Debug Mode is requested at MCU level. In this mode, no transmission or reception of frames is done and synchronicity to the CAN bus is lost. See [Section 25.5.10.1, Freeze mode](#), for more information.
- Listen-Only mode:  
The module enters this mode when the LOM bit in CTRL is asserted. In this mode, transmission is disabled, all error counters are frozen and the module operates in a CAN Error Passive mode. Only messages acknowledged by another CAN station will be received. If FlexCAN detects a message that has not been acknowledged, it will flag a BIT0 error (without changing the REC), as if it was trying to acknowledge the message.
- Loop-Back mode:  
The module enters this mode when the LPB bit in CTRL is asserted. In this mode, FlexCAN performs an internal loop back that can be used for self test operation. The bit stream output of the transmitter is internally fed back to the receiver input. The Rx CAN input pin is ignored and the Tx CAN output goes to the recessive state (logic 1). FlexCAN behaves as it normally does when transmitting and treats its own transmitted message as a message received from a remote node. In this mode, FlexCAN ignores the bit sent during the ACK slot in the CAN frame acknowledge field to ensure proper reception of its own message. Both transmit and receive interrupts are generated.
- Module Disable mode:  
This low power mode is entered when the MCR[MDIS] bit is asserted by the CPU. When disabled, the module requests to disable the clocks to the CAN Protocol Interface and Message Buffer Management submodules. Exit from this mode is done by negating the MDIS bit in MCR. See [Section 25.5.10.2, Module Disable mode](#), for more information.

## 25.3 External signal description

### 25.3.1 Overview

The FlexCAN module has two I/O signals connected to the external MCU pins. These signals are summarized in [Table 25-1](#) and described in more detail in the next subsections.

**Table 25-1. FlexCAN Signals**

| Signal Name <sup>1</sup> | Direction | Description      |
|--------------------------|-----------|------------------|
| CAN Rx                   | Input     | CAN Receive Pin  |
| CAN Tx                   | Output    | CAN Transmit Pin |

<sup>1</sup> The actual MCU pins may have different names.



## 25.3.2 Signal descriptions

### 25.3.2.1 CAN Rx

This pin is the receive pin from the CAN bus transceiver. Dominant state is represented by logic level 0. Recessive state is represented by logic level 1.

### 25.3.2.2 CAN Tx

This pin is the transmit pin to the CAN bus transceiver. Dominant state is represented by logic level 0. Recessive state is represented by logic level 1.

## 25.4 Memory map/register definition

This section describes the registers and data structures in the FlexCAN module. The base address of the module depends on the particular memory map of the MCU. The addresses presented here are relative to the base address.

The address space occupied by FlexCAN has 96 bytes for registers starting at the module base address, followed by MB storage space in embedded RAM starting at address 0x0060, and an extra ID Mask storage space in a separate embedded RAM starting at address 0x0880.

### 25.4.1 FlexCAN memory mapping

The complete memory map for a FlexCAN module with 64 MBs capability is shown in [Table 25-2](#).

All registers except for the MCR can be configured to have either supervisor or unrestricted access by programming the MCR[SUPV] bit.

The IFLAG2 and IMASK2 registers are considered reserved space when FlexCAN is configured with 16 or 32 MBs. The Rx Global Mask (RXGMASK), Rx Buffer 14 Mask (RX14MASK), and the Rx Buffer 15 Mask (RX15MASK) registers are provided for backwards compatibility and are not used when the BCC bit in MCR is asserted.

The address ranges 0x0060–0x047F and 0x0880–0x097F are occupied by two separate embedded memories. These two ranges are completely occupied by RAM (1056 and 256 bytes, respectively) only when FlexCAN is configured with 64 MBs. When it is configured with 16 MBs, the memory sizes are 288 and 64 bytes, so the address ranges 0x0180–0x047F and 0x08C0–0x097F are considered reserved space. When it is configured with 32 MBs, the memory sizes are 544 and 128 bytes, so the address ranges 0x0280–0x047F and 0x0900–0x097F are considered reserved space. Furthermore, if the BCC bit in MCR is negated, then the whole Rx Individual Mask registers address range (0x0880–0x097F) is considered reserved space.

**Table 25-2. FlexCAN memory map**

| Base addresses:<br>0xFFFC_0000 (FlexCAN_0)<br>0xFFFC_4000 (FlexCAN_1)<br>0xFFFC_8000 (FlexCAN_2)<br>0xFFFC_C000 (FlexCAN_3)<br>0xFFFD_0000 (FlexCAN_4)<br>0xFFFD_4000 (FlexCAN_5) |  |                             |
|---|--|-----------------------------|
| Address offset  | Register                                     | Location                    |
| 0x0000  | Module Configuration (MCR)                   | <a href="#">on page 557</a> |
| 0x0004  | Control Register (CTRL)                      | <a href="#">on page 561</a> |
| 0x0008  | Free Running Timer (TIMER)                   | <a href="#">on page 564</a> |
| 0x000C  | Reserved                                     |                             |
| 0x0010  | Rx Global Mask (RXGMASK)                     | <a href="#">on page 565</a> |
| 0x0014  | Rx Buffer 14 Mask (RX14MASK)                 | <a href="#">on page 567</a> |
| 0x0018  | Rx Buffer 15 Mask (RX15MASK)                 | <a href="#">on page 567</a> |
| 0x001C  | Error Counter Register (ECR)                 | <a href="#">on page 567</a> |
| 0x0020  | Error and Status Register (ESR)              | <a href="#">on page 569</a> |
| 0x0024  | Interrupt Masks 2 (IMASK2)                   | <a href="#">on page 572</a> |
| 0x0028  | Interrupt Masks 1 (IMASK1)                   | <a href="#">on page 573</a> |
| 0x002C  | Interrupt Flags 2 (IFLAG2)                   | <a href="#">on page 573</a> |
| 0x0030  | Interrupt Flags 1 (IFLAG1)                   | <a href="#">on page 574</a> |
| 0x0034–0x007F   | Reserved                                     |                             |
| 0x0080–0x017F   | Message Buffers MB0–MB15                     | —                           |
| 0x0180–0x027F   | Message Buffers MB16–MB31                    | —                           |
| 0x0280–0x047F   | Message Buffers MB32–MB63                    | —                           |
| 0x0480–087F   | Reserved                                     |                             |
| 0x0880–0x08BC   | Rx Individual Mask Registers RXIMR0–RXIMR15  | <a href="#">on page 575</a> |
| 0x08C0–0x08FC   | Rx Individual Mask Registers RXIMR16–RXIMR31 | <a href="#">on page 575</a> |
| 0x0900–0x097C   | Rx Individual Mask Registers RXIMR32–RXIMR63 | <a href="#">on page 575</a> |

The FlexCAN module stores CAN messages for transmission and reception using a Message Buffer structure. Each individual MB is formed by 16 bytes mapped on memory as described in [Table 25-3](#). [Table 25-3](#) shows a Standard/Extended Message Buffer (MB0) memory map, using 16 bytes total (0x80–0x8F space).

**Table 25-3. Message Buffer MB0 memory mapping**

| Address Offset | MB Field                 |
|----------------|--------------------------|
| 0x80           | Control and Status (C/S) |

Table 25-3. Message Buffer MBO memory mapping

|           |   |
|-----------|---|
| 0x84      | Identifier Field                          |
| 0x88–0x8F | Data Field 0 – Data Field 7 (1 byte each) |

## 25.4.2 Message Buffer Structure

The Message Buffer structure used by the FlexCAN module is represented in [Figure 25-2](#). Both Extended and Standard Frames (29-bit Identifier and 11-bit Identifier, respectively) used in the CAN specification (Version 2.0 Part B) are represented.

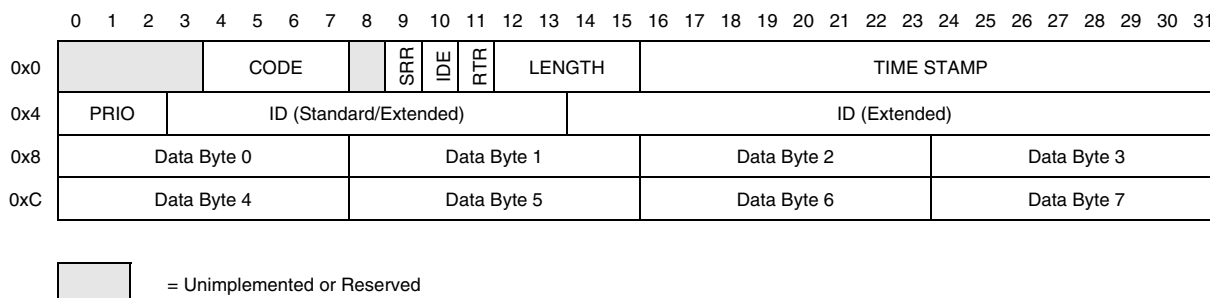


Figure 25-2. Message Buffer Structure

Table 25-4. Message Buffer Structure field description

| Field | Description   |
|-------|---|
| CODE  | <p><b>Message Buffer Code</b></p> <p>This 4-bit field can be accessed (read or write) by the CPU and by the FlexCAN module itself, as part of the message buffer matching and arbitration process. The encoding is shown in <a href="#">Table 25-5</a> and <a href="#">Table 25-6</a>. See <a href="#">Section 25.5, Functional description</a>, for additional information.</p>  |
| SRR   | <p><b>Substitute Remote Request</b></p> <p>Fixed recessive bit, used only in extended format. It must be set to 1 by the user for transmission (Tx Buffers) and will be stored with the value received on the CAN bus for Rx receiving buffers. It can be received as either recessive or dominant. If FlexCAN receives this bit as dominant, then it is interpreted as arbitration loss.</p> <p>0 Dominant is not a valid value for transmission in Extended Format frames.<br/>1 Recessive value is compulsory for transmission in Extended Format frames.</p>  |
| IDE   | <p><b>ID Extended Bit</b></p> <p>This bit identifies whether the frame format is standard or extended.</p> <p>0 Frame format is standard.<br/>1 Frame format is extended.</p>   |
| RTR   | <p><b>Remote Transmission Request</b></p> <p>This bit is used for requesting transmissions of a data frame. If FlexCAN transmits this bit as 1 (recessive) and receives it as 0 (dominant), it is interpreted as arbitration loss. If this bit is transmitted as 0 (dominant), then if it is received as 1 (recessive), the FlexCAN module treats it as bit error. If the value received matches the value transmitted, it is considered as a successful bit transmission.</p> <p>0 Indicates the current MB has a Data Frame to be transmitted.<br/>1 Indicates the current MB has a Remote Frame to be transmitted.</p> <p><b>Note:</b> Do not configure the last Message Buffer to be the RTR frame.</p> |

**Table 25-4. Message Buffer Structure field description (continued)**

| Field      | Description   |
|------------|---|
| LENGTH     | <b>Length of Data in Bytes</b><br>This 4-bit field is the length (in bytes) of the Rx or Tx data, which is located in offset 0x8 through 0xF of the MB space (see <a href="#">Figure 25-2</a> ). In reception, this field is written by the FlexCAN module, copied from the DLC (Data Length Code) field of the received frame. In transmission, this field is written by the CPU and corresponds to the DLC field value of the frame to be transmitted. When RTR=1, the Frame to be transmitted is a Remote Frame and does not include the data field, regardless of the Length field. |
| TIME STAMP | <b>Free-Running Counter Time Stamp</b><br>This 16-bit field is a copy of the Free-Running Timer, captured for Tx and Rx frames at the time when the beginning of the Identifier field appears on the CAN bus.   |
| PRIO       | <b>Local priority</b><br>This 3-bit field is only used when LPRIO_EN bit is set in MCR and it only makes sense for Tx buffers. These bits are not transmitted. They are appended to the regular ID to define the transmission priority. See <a href="#">Section 25.5.4, Arbitration process</a> .   |
| ID         | <b>Frame Identifier</b><br>In Standard Frame format, only the 11 most significant bits (3 to 13) are used for frame identification in both receive and transmit cases. The 18 least significant bits are ignored. In Extended Frame format, all bits are used for frame identification in both receive and transmit cases.  |
| DATA       | <b>Data Field</b><br>As many as 8 bytes can be used for a data frame. For Rx frames, the data is stored as it is received from the CAN bus. For Tx frames, the CPU prepares the data field to be transmitted within the frame.  |

**Table 25-5. Message Buffer Code for Rx buffers**

| Rx Code BEFORE Rx New Frame | Description                    | Rx Code AFTER Rx New Frame | Comment   |
|-----------------------------|--------------------------------|----------------------------|---|
| 0000                        | INACTIVE: MB is not active.    | —                          | MB does not participate in the matching process.  |
| 0100                        | EMPTY: MB is active and empty. | 0010                       | MB participates in the matching process. When a frame is received successfully, the code is automatically updated to FULL.  |
| 0010                        | FULL: MB is full.              | 0010                       | The act of reading the C/S word followed by unlocking the MB does not make the code return to EMPTY. It remains FULL. If a new frame is written to the MB after the C/S word was read and the MB was unlocked, the code still remains FULL. |
|                             |                                | 0110                       | If the MB is FULL and a new frame is overwritten to this MB before the CPU had time to read it, the code is automatically updated to OVERRUN. See <a href="#">Section 25.5.6, Matching process</a> , for details about overrun behavior.    |

Table 25-5. Message Buffer Code for Rx buffers

| Rx Code BEFORE Rx New Frame | Description   | Rx Code AFTER Rx New Frame | Comment  |
|-----------------------------|---|----------------------------|--|
| 0110                        | OVERRUN: a frame was overwritten into a full buffer.                              | 0010                       | If the code indicates OVERRUN but the CPU reads the C/S word and then unlocks the MB, when a new frame is written to the MB the code returns to FULL.  |
|                             |   | 0110                       | If the code already indicates OVERRUN, and yet another new frame must be written, the MB will be overwritten again, and the code will remain OVERRUN. See <a href="#">Section 25.5.6, Matching process</a> , for details about overrun behavior. |
| 0XY1 <sup>1</sup>           | BUSY: Flexcan is updating the contents of the MB. The CPU must not access the MB. | 0010                       | An EMPTY buffer was written with a new frame (XY was 01).  |
|                             |   | 0110                       | A FULL/OVERRUN buffer was overwritten (XY was 11).   |

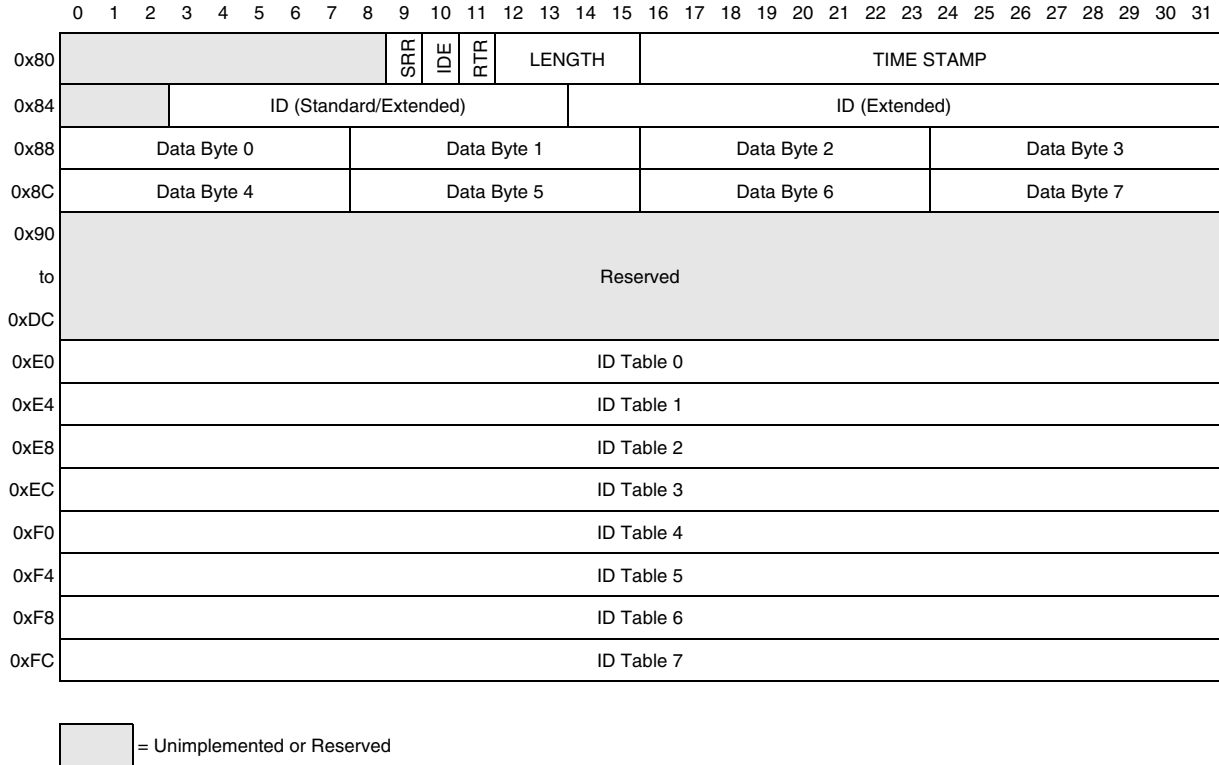
<sup>1</sup> Note that for Tx MBs (see [Table 25-6](#)), the BUSY bit should be ignored upon read, except when AEN bit is set in the MCR register.

Table 25-6. Message Buffer Code for Tx buffers

| RTR | Initial Tx code | Code after successful transmission | Description  |
|-----|-----------------|------------------------------------|--|
| X   | 1000            | —                                  | INACTIVE: MB does not participate in the arbitration process.  |
| X   | 1001            | —                                  | ABORT: MB was configured as Tx and CPU aborted the transmission. This code is only valid when AEN bit in MCR is asserted. MB does not participate in the arbitration process.  |
| 0   | 1100            | 1000                               | Transmit data frame unconditionally once. After transmission, the MB automatically returns to the INACTIVE state.  |
| 1   | 1100            | 0100                               | Transmit remote frame unconditionally once. After transmission, the MB automatically becomes an Rx MB with the same ID.  |
| 0   | 1010            | 1010                               | Transmit a data frame whenever a remote request frame with the same ID is received. This MB participates simultaneously in both the matching and arbitration processes. The matching process compares the ID of the incoming remote request frame with the ID of the MB. If a match occurs this MB is allowed to participate in the current arbitration process and the Code field is automatically updated to 1110 to allow the MB to participate in future arbitration runs. When the frame is eventually transmitted successfully, the Code automatically returns to 1010 to restart the process again. |
| 0   | 1110            | 1010                               | This is an intermediate code that is automatically written to the MB by the MBM as a result of match to a remote request frame. The data frame will be transmitted unconditionally once and then the code will automatically return to 1010. The CPU can also write this code with the same effect.  |

### 25.4.3 Rx FIFO structure

When the FEN bit is set in the MCR, the memory area from 0x80 to 0xFC (which is normally occupied by MBs 0 to 7) is used by the reception FIFO engine. [Figure 25-3](#) shows the Rx FIFO data structure. The region 0x80–0x8C contains an MB structure, which is the port through which the CPU reads data from the FIFO (the oldest frame received and not read yet). The region 0x90–0xDC is reserved for internal use of the FIFO engine. The region 0xE0–0xFC contains an 8-entry ID table that specifies filtering criteria for accepting frames into the FIFO. [Figure 25-4](#) shows the three different formats that the elements of the ID table can assume, depending on the IDAM field of the MCR. Note that all elements of the table must have the same format. See [Section 25.5.8, Rx FIFO](#), for more information.



**Figure 25-3. Rx FIFO structure**

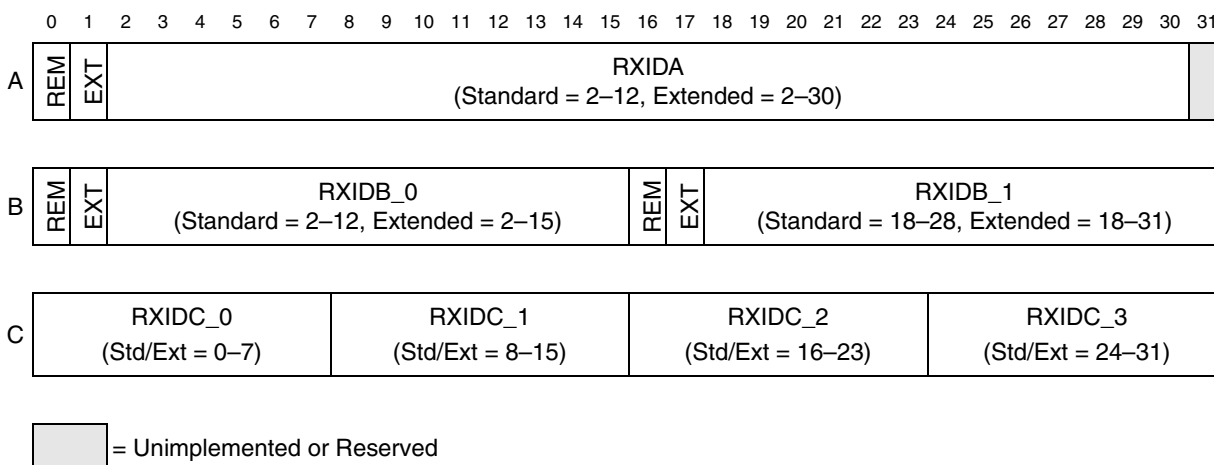


Figure 25-4. ID Table 0-7

Table 25-7. Rx FIFO Structure field description

| Field                                       | Description   |
|---|---|
| REM   | <b>Remote Frame</b><br>This bit specifies if Remote Frames are accepted into the FIFO if they match the target ID.<br>0 Remote Frames are rejected and data frames can be accepted<br>1 Remote Frames can be accepted and data frames are rejected  |
| EXT   | <b>Extended Frame</b><br>Specifies whether extended or standard frames are accepted into the FIFO if they match the target ID.<br>0 Extended frames are rejected and standard frames can be accepted<br>1 Extended frames can be accepted and standard frames are rejected  |
| RXIDA                                       | <b>Rx Frame Identifier (Format A)</b><br>Specifies an ID to be used as acceptance criteria for the FIFO. In the standard frame format, only the 11 most significant bits (3 to 13) are used for frame identification. In the extended frame format, all bits are used.  |
| RXIDB_0,<br>RXIDB_1                         | <b>Rx Frame Identifier (Format B)</b><br>Specifies an ID to be used as acceptance criteria for the FIFO. In the standard frame format, the 11 most significant bits (a full standard ID) (3 to 13) are used for frame identification. In the extended frame format, all 14 bits of the field are compared to the 14 most significant bits of the received ID. |
| RXIDC_0,<br>RXIDC_1,<br>RXIDC_2,<br>RXIDC_3 | <b>Rx Frame Identifier (Format C)</b><br>Specifies an ID to be used as acceptance criteria for the FIFO. In both standard and extended frame formats, all 8 bits of the field are compared to the 8 most significant bits of the received ID.   |

## 25.4.4 Register descriptions

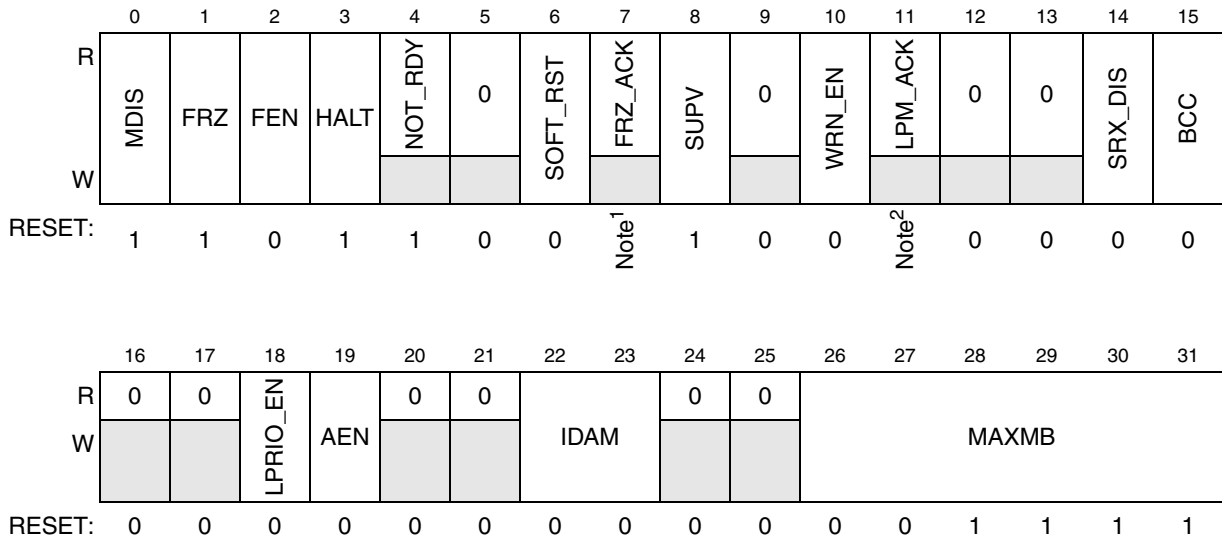
The FlexCAN registers are described in this section in ascending address order.

### 25.4.4.1 Module Configuration Register (MCR)

This register defines global system configurations, such as the module operation mode (e.g., low power) and maximum message buffer configuration. This register can be accessed at any time, however some fields must be changed only during Freeze Mode. Find more information in the fields descriptions ahead.

Offset: 0x0000

Access: Supervisor read/write



**Figure 25-5. Module Configuration Register (MCR)**

- <sup>1</sup> Different on various platforms, but it is always the opposite of the MDIS reset value.
- <sup>2</sup> Different on various platforms, but it is always the same as the MDIS reset value.

**Table 25-8. MCR field descriptions**

| Field | Description   |
|-------|---|
| MDIS  | <p>Module Disable</p> <p>This bit controls whether FlexCAN is enabled or not. When disabled, FlexCAN shuts down the clocks to the CAN Protocol Interface and Message Buffer Management submodules. This is the only bit in MCR not affected by soft reset. See <a href="#">Section 25.5.10.2, Module Disable mode</a>, for more information.</p> <p>0 Enable the FlexCAN module<br/>1 Disable the FlexCAN module</p>  |
| FRZ   | <p>Freeze Enable</p> <p>The FRZ bit specifies the FlexCAN behavior when the HALT bit in MCR is set or when Debug Mode is requested at MCU level. When FRZ is asserted, FlexCAN is enabled to enter Freeze Mode. Negation of this bit field causes FlexCAN to exit from Freeze Mode.</p> <p>0 Not enabled to enter Freeze Mode<br/>1 Enabled to enter Freeze Mode</p>  |
| FEN   | <p>FIFO Enable</p> <p>This bit controls whether the FIFO feature is enabled or not. When FEN is set, MBs 0 to 7 cannot be used for normal reception and transmission because the corresponding memory region (0x80–0xFF) is used by the FIFO engine. See <a href="#">Section 25.4.3, Rx FIFO structure</a>, and <a href="#">Section 25.5.8, Rx FIFO</a>, for more information. This bit must be written in Freeze mode only.</p> <p>0 FIFO not enabled<br/>1 FIFO enabled</p> |



Table 25-8. MCR field descriptions (continued)

| Field    | Description   |
|----------|---|
| HALT     | <p>Halt FlexCAN</p> <p>Assertion of this bit puts the FlexCAN module into Freeze Mode. The CPU should clear it after initializing the Message Buffers and CTRL. No reception or transmission is performed by FlexCAN before this bit is cleared. While in Freeze Mode, the CPU has write access to ECR, which is otherwise read-only. Freeze Mode cannot be entered while FlexCAN is in any of the low power modes. See <a href="#">Section 25.5.10.1, Freeze mode</a>, for more information.</p> <p>0 No Freeze Mode request.<br/>1 Enters Freeze Mode if the FRZ bit is asserted.</p>   |
| NOT_RDY  | <p>FlexCAN Not Ready</p> <p>This read-only bit indicates that FlexCAN is in Disable Mode or Freeze Mode. It is negated once FlexCAN has exited these modes.</p> <p>0 FlexCAN module is in Normal Mode, Listen-Only Mode or Loop-Back Mode<br/>1 FlexCAN module is in Disable Mode or Freeze Mode</p>  |
| SOFT_RST | <p>Soft Reset</p> <p>When this bit is asserted, FlexCAN resets its internal state machines and some of the memory mapped registers. The following registers are reset: MCR (except the MDIS bit), TIMER, ECR, ESR, IMASK1, IMASK2, IFLAG1, IFLAG2. Configuration registers that control the interface to the CAN bus are not affected by soft reset. The following registers are unaffected:</p> <p>CTRL<br/>RXIMR0–RXIMR63<br/>RXGMASK, RX14MASK, RX15MASK<br/>all Message Buffers</p> <p>The SOFT_RST bit can be asserted directly by the CPU when it writes to MCR, but it is also asserted when global soft reset is requested at MCU level. Since soft reset is synchronous and has to follow a request/acknowledge procedure across clock domains, it may take some time to fully propagate its effect. The SOFT_RST bit remains asserted while reset is pending, and is automatically negated when reset completes. Therefore, software can poll this bit to know when the soft reset has completed.</p> <p>Soft reset cannot be applied while clocks are shut down in any of the low power modes. The module should be first removed from low power mode, and then soft reset can be applied.</p> <p>0 No reset request<br/>1 Resets the registers marked as affected by soft reset in <a href="#">Table 25-2</a></p> |
| FRZ_ACK  | <p>Freeze Mode Acknowledge</p> <p>This read-only bit indicates that FlexCAN is in Freeze Mode and its prescaler is stopped. The Freeze Mode request cannot be granted until current transmission or reception processes have finished. Therefore the software can poll the FRZ_ACK bit to know when FlexCAN has actually entered Freeze Mode. If Freeze Mode request is negated, then this bit is negated once the FlexCAN prescaler is running again. If Freeze Mode is requested while FlexCAN is in any of the low power modes, then the FRZ_ACK bit will only be set when the low power mode is exited. See <a href="#">Section 25.5.10.1, Freeze mode</a>, for more information.</p> <p>0 FlexCAN not in Freeze Mode, prescaler running<br/>1 FlexCAN in Freeze Mode, prescaler stopped</p>  |
| SUPV     | <p>Supervisor Mode</p> <p>This bit configures some of the FlexCAN registers to be either in Supervisor or Unrestricted memory space. The registers affected by this bit are marked as S/U in the Access Type column of <a href="#">Table 25-2</a>. Reset value of this bit is 1, so the affected registers start with Supervisor access restrictions. This bit should be written in Freeze mode only.</p> <p>0 Affected registers are in Unrestricted memory space<br/>1 Affected registers are in Supervisor memory space. Any access without supervisor permission behaves as though the access was done to an unimplemented register location</p>  |

Table 25-8. MCR field descriptions (continued)

| Field    | Description   |
|----------|---|
| WRN_EN   | <p>Warning Interrupt Enable</p> <p>When asserted, this bit enables the generation of the TWRN_INT and RWRN_INT flags in the Error and Status Register (ESR). If WRN_EN is negated, the TWRN_INT and RWRN_INT flags will always be zero, independent of the values of the error counters, and no warning interrupt will ever be generated. This bit must be written in Freeze mode only.</p> <p>0 TWRN_INT and RWRN_INT bits are zero, independent of the values in the error counters.<br/> 1 TWRN_INT and RWRN_INT bits are set when the respective error counter transition from <math>&lt; 96</math> to <math>\geq 96</math>.</p>  |
| LPM_ACK  | <p>Low Power Mode Acknowledge</p> <p>This read-only bit indicates that FlexCAN is in Disable Mode. This mode cannot be entered until all current transmission or reception processes have finished, so the CPU can poll the LPM_ACK bit to know when FlexCAN has actually entered low power mode. See <a href="#">Section 25.5.10.2, Module Disable mode</a>, for more information.</p> <p>0 FlexCAN not in any low-power mode<br/> 1 FlexCAN is in Disable Mode</p>  |
| SRX_DIS  | <p>Self Reception Disable</p> <p>This bit defines whether FlexCAN is allowed to receive frames transmitted by itself. If this bit is asserted, frames transmitted by the module will not be stored in any MB, regardless if the MB is programmed with an ID that matches the transmitted frame, and no interrupt flag or interrupt signal will be generated due to the frame reception. This bit must be written in Freeze mode only.</p> <p>0 Self reception enabled<br/> 1 Self reception disabled</p>  |
| BCC      | <p>Backwards Compatibility Configuration</p> <p>This bit is provided to support Backwards Compatibility with previous FlexCAN versions. On this device, FlexCAN supports individual Rx ID masking using RXIMR0–63. Setting this bit enables individual Rx ID masking.</p> <p>When this bit is cleared, FlexCAN uses a backwards compatible masking scheme with RXGMASK, RX14MASK, and RX15MASK; and the reception queue feature is disabled. Upon receiving a message, if the first MB with a matching ID that is found is still occupied by a previous unread message, FlexCAN will not look for another matching MB. It will override this MB with the new message and set the CODE field to '0110' (overrun).</p> <p>This bit is cleared on reset, allowing legacy software to work without modification. This bit must be written in Freeze mode only.</p> <p>0 Individual Rx masking and queue feature are disabled.<br/> 1 Individual Rx masking and queue feature are enabled.</p> |
| LPRIO_EN | <p>Local Priority Enable</p> <p>This bit is provided for backwards compatibility reasons. It controls whether the local priority feature is enabled or not. It is used to extend the ID used during the arbitration process. With this extended ID concept, the arbitration process is done based on the full 32-bit word, but the actual transmitted ID still has 11-bit for standard frames and 29-bit for extended frames. This bit must be written in Freeze mode only.</p> <p>0 Local Priority disabled<br/> 1 Local Priority enabled</p>  |
| AEN      | <p>Abort Enable</p> <p>This bit is supplied for backwards compatibility reasons. When asserted, it enables the Tx abort feature. This feature guarantees a safe procedure for aborting a pending transmission, so that no frame is sent in the CAN bus without notification. This bit must be written in Freeze mode only.</p> <p>0 Abort disabled<br/> 1 Abort enabled</p>   |

Table 25-8. MCR field descriptions (continued)

| Field | Description   |
|-------|---|
| IDAM  | <p>ID Acceptance Mode</p> <p>This 2-bit field identifies the format of the elements of the Rx FIFO filter table, as shown in <a href="#">Table 25-9</a>. Note that all elements of the table are configured at the same time by this field (they are all the same format). See <a href="#">Section 25.4.3, Rx FIFO structure</a>. This bit must be written in Freeze mode only.</p>   |
| MAXMB | <p>Maximum Number of Message Buffers</p> <p>This 6-bit field defines the maximum number of message buffers that will take part in the matching and arbitration processes. The reset value (0x0F) is equivalent to 16 MB configuration. This field must be changed only while the module is in Freeze Mode.</p> <p>Maximum MBs in use = MAXMB + 1.</p> <p><b>Note:</b> MAXMB must be programmed with a value smaller or equal to the number of available Message Buffers, otherwise FlexCAN can transmit and receive wrong messages.</p> |

Table 25-9. IDAM coding

| IDAM | Format | Explanation  |
|------|--------|--|
| 0b00 | A      | One full ID (standard or extended) per filter element.                       |
| 0b01 | B      | Two full standard IDs or two partial 14-bit extended IDs per filter element. |
| 0b10 | C      | Four partial 8-bit IDs (standard or extended) per filter element.            |
| 0b11 | D      | All frames rejected.   |

#### 25.4.4.2 Control (CTRL) register

This register is defined for specific FlexCAN control features related to the CAN bus, such as bit rate, programmable sampling point within an Rx bit, Loop Back Mode, Listen Only Mode, Bus Off recovery behavior, and interrupt enabling (Bus-Off, Error, Warning). It also determines the Division Factor for the clock prescaler. Most of the fields in this register should only be changed while the module is in Disable Mode or in Freeze Mode. Exceptions are the BOFF\_MSK, ERR\_MSK, TWRN\_MSK, RWRN\_MSK, and BOFF\_REC bits, which can be accessed at any time.

Offset: 0x0004

Access: Read/write

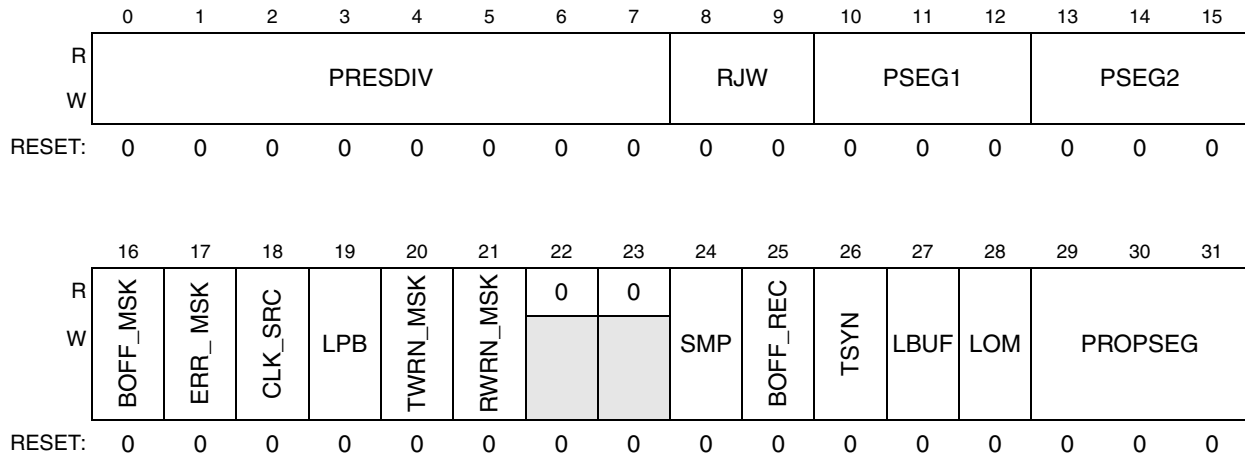


Figure 25-6. Control (CTRL) register

Table 25-10. CTRL field descriptions

| Field    | Description  |
|----------|--|
| PRESDIV  | Prescaler Division Factor<br>This 8-bit field defines the ratio between the CPI clock frequency and the Serial Clock (Sclock) frequency. The Sclock period defines the time quantum of the CAN protocol. For the reset value, the Sclock frequency is equal to the CPI clock frequency. The Maximum value of this register is 0xFF, that gives a minimum Sclock frequency equal to the CPI clock frequency divided by 256. For more information refer to <a href="#">Section 25.5.9.4, Protocol timing</a> . This bit must be written in Freeze mode only.<br>Sclock frequency = CPI clock frequency / (PRESDIV + 1) |
| RJW      | Resync Jump Width<br>This 2-bit field defines the maximum number of time quanta <sup>1</sup> that a bit time can be changed by one resynchronization. The valid programmable values are 0–3. This bit must be written in Freeze mode only.<br>Resync Jump Width = RJW + 1.   |
| PSEG1    | Phase Segment 1<br>This 3-bit field defines the length of Phase Buffer Segment 1 in the bit time. The valid programmable values are 0–7. This bit must be written in Freeze mode only.<br>Phase Buffer Segment 1 = (PSEG1 + 1) × Time Quanta.  |
| PSEG2    | Phase Segment 2<br>This 3-bit field defines the length of Phase Buffer Segment 2 in the bit time. The valid programmable values are 1–7. This bit must be written in Freeze mode only.<br>Phase Buffer Segment 2 = (PSEG2 + 1) × Time Quanta.  |
| BOFF_MSK | Bus Off Mask<br>This bit provides a mask for the Bus Off Interrupt.<br>0 Bus Off interrupt disabled<br>1 Bus Off interrupt enabled   |
| ERR_MSK  | Error Mask<br>This bit provides a mask for the Error Interrupt.<br>0 Error interrupt disabled<br>1 Error interrupt enabled   |

Table 25-10. CTRL field descriptions (continued)

| Field    | Description  |
|----------|--|
| CLK_SRC  | <p>CAN Engine Clock Source</p> <p>This bit selects the clock source to the CAN Protocol Interface (CPI) to be either the peripheral clock (driven by the PLL) or the crystal oscillator clock. The selected clock is the one fed to the prescaler to generate the Serial Clock (Sclock). In order to guarantee reliable operation, this bit must only be changed while the module is in Disable Mode. See <a href="#">Section 25.5.9.4, Protocol timing</a>, for more information.</p> <p>0 The CAN engine clock source is the oscillator clock<br/>1 The CAN engine clock source is the bus clock</p>   |
| TWRN_MSK | <p>Tx Warning Interrupt Mask</p> <p>This bit provides a mask for the Tx Warning Interrupt associated with the TWRN_INT flag in ESR. This bit has no effect if the WRN_EN bit in MCR is negated and it is read as zero when WRN_EN is negated.</p> <p>0 Tx Warning Interrupt disabled<br/>1 Tx Warning Interrupt enabled</p>  |
| RWRN_MSK | <p>Rx Warning Interrupt Mask</p> <p>This bit provides a mask for the Rx Warning Interrupt associated with the RWRN_INT flag in ESR. This bit has no effect if the WRN_EN bit in MCR is negated and it is read as zero when WRN_EN is negated.</p> <p>0 Rx Warning Interrupt disabled<br/>1 Rx Warning Interrupt enabled</p>  |
| LPB      | <p>Loop Back</p> <p>This bit configures FlexCAN to operate in Loop-Back Mode. In this mode, FlexCAN performs an internal loop back that can be used for self test operation. The bit stream output of the transmitter is fed back internally to the receiver input. The Rx CAN input pin is ignored and the Tx CAN output goes to the recessive state (logic 1). FlexCAN behaves as it normally does when transmitting, and treats its own transmitted message as a message received from a remote node. In this mode, FlexCAN ignores the bit sent during the ACK slot in the CAN frame acknowledge field, generating an internal acknowledge bit to ensure proper reception of its own message. Both transmit and receive interrupts are generated. This bit must be written in Freeze mode only.</p> <p>0 Loop Back disabled<br/>1 Loop Back enabled</p>  |
| SMP      | <p>Sampling Mode</p> <p>This bit defines the sampling mode of CAN bits at the Rx input. This bit must be written in Freeze mode only.</p> <p>0 Just one sample is used to determine the bit value<br/>1 Three samples are used to determine the value of the received bit: the regular one (sample point) and two preceding samples, a majority rule is used</p>   |
| BOFF_REC | <p>Bus Off Recovery Mode</p> <p>This bit defines how FlexCAN recovers from Bus Off state. If this bit is negated, automatic recovering from Bus Off state occurs according to the CAN Specification 2.0B. If the bit is asserted, automatic recovering from Bus Off is disabled and the module remains in Bus Off state until the bit is negated by the user. If the negation occurs before 128 sequences of 11 recessive bits are detected on the CAN bus, then Bus Off recovery happens as if the BOFF_REC bit had never been asserted. If the negation occurs after 128 sequences of 11 recessive bits occurred, then FlexCAN will resynchronize to the bus by waiting for 11 recessive bits before joining the bus. After negation, the BOFF_REC bit can be reasserted again during Bus Off, but it will only be effective the next time the module enters Bus Off. If BOFF_REC was negated when the module entered Bus Off, asserting it during Bus Off will not be effective for the current Bus Off recovery.</p> <p>0 Automatic recovering from Bus Off state enabled, according to CAN Spec 2.0 part B<br/>1 Automatic recovering from Bus Off state disabled</p> |

Table 25-10. CTRL field descriptions (continued)

| Field   | Description  |
|---------|--|
| TSYN    | <p>Timer Sync Mode</p> <p>This bit enables a mechanism that resets the free-running timer each time a message is received in Message Buffer 0. This feature provides means to synchronize multiple FlexCAN stations with a special SYNC message (i.e., global network time). If the FEN bit in MCR is set (FIFO enabled), MB8 is used for timer synchronization instead of MB0. This bit must be written in Freeze mode only.</p> <p>0 Timer Sync feature disabled<br/>1 Timer Sync feature enabled</p>  |
| LBUF    | <p>Lowest Buffer Transmitted First</p> <p>This bit defines the ordering mechanism for Message Buffer transmission. When asserted, the LPRIO_EN bit does not affect the priority arbitration. This bit must be written in Freeze mode only.</p> <p>0 Buffer with highest priority is transmitted first<br/>1 Lowest number buffer is transmitted first</p>  |
| LOM     | <p>Listen-Only Mode</p> <p>This bit configures FlexCAN to operate in Listen Only Mode. In this mode, transmission is disabled, all error counters are frozen and the module operates in a CAN Error Passive mode [Ref. 1]. Only messages acknowledged by another CAN station will be received. If FlexCAN detects a message that has not been acknowledged, it will flag a BIT0 error (without changing the REC), as if it was trying to acknowledge the message. This bit must be written in Freeze mode only.</p> <p>0 Listen Only Mode is deactivated<br/>1 FlexCAN module operates in Listen Only Mode</p> |
| PROPSEG | <p>Propagation Segment</p> <p>This 3-bit field defines the length of the Propagation Segment in the bit time. The valid programmable values are 0–7. This bit must be written in Freeze mode only.</p> <p>Propagation Segment Time = (PROPSEG + 1) × Time Quanta.<br/>Time Quantum = one Sclock period.</p>  |

<sup>1</sup> One time quantum is equal to the Sclock period.

### 25.4.4.3 Free Running Timer (TIMER) register

This register represents a 16-bit free running counter that can be read and written by the CPU.

The timer is clocked by the FlexCAN bit clock (which defines the baud rate on the CAN bus). During a message transmission/reception, it increments by one for each bit that is received or transmitted. When there is no message on the bus, it counts using the previously programmed baud rate. During Freeze Mode, the timer is not incremented.

The timer value is captured at the beginning of the identifier field of any frame on the CAN bus. This captured value is written into the Time Stamp entry in a message buffer after a successful reception or transmission of a message.

Writing to the timer is an indirect operation. The data is first written to an auxiliary register and then an internal request/acknowledge procedure across clock domains is executed. All this is transparent to the user, except for the fact that the data will take some time to be actually written to the register. If desired, software can poll the register to discover when the data was actually written.

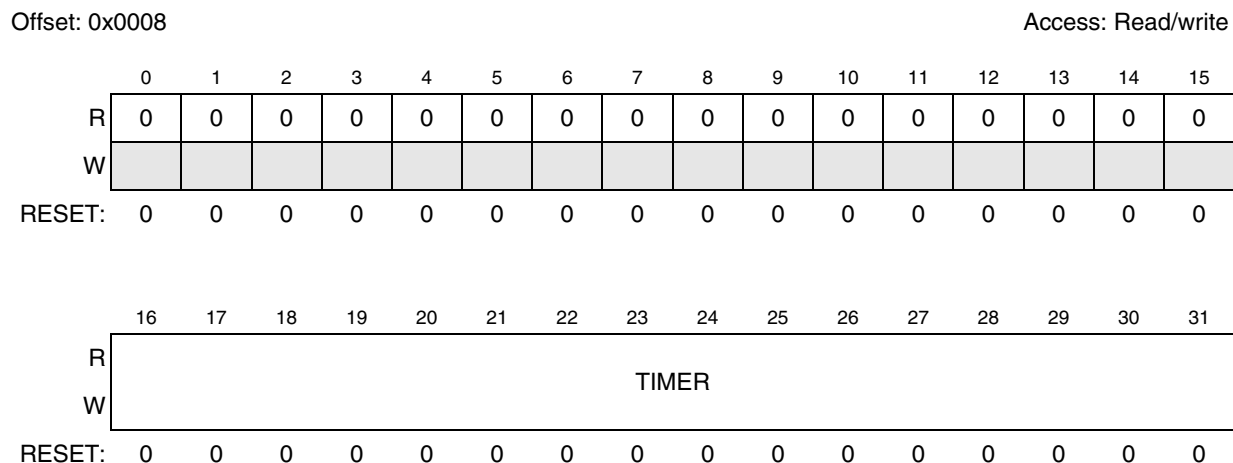


Figure 25-7. Free Running Timer (TIMER) register

Table 25-11. TIMER field descriptions

| Field | Description  |
|-------|--|
| TIMER | Free-running timer counter. The timer starts from 0x0000 after reset, counts linearly to 0xFFFF, and wraps around. |

#### 25.4.4.4 Rx Global Mask (RXGMASK) register

This register is provided for legacy support and for MCUs that do not have the individual masking per Message Buffer feature. Setting the BCC bit in MCR causes the RXGMASK register to have no effect on the module operation.

RXGMASK is used as acceptance mask for all Rx MBs, excluding MBs 14–15, which have individual mask registers. When the FEN bit in MCR is set (FIFO enabled), the RXGMASK also applies to all elements of the ID filter table, except elements 6–7, which have individual masks.

See [Section 25.5.8, Rx FIFO](#), for important details on usage of RXGMASK on filtering process for Rx FIFO.

The contents of this register must be programmed while the module is in Freeze Mode, and must not be modified when the module is transmitting or receiving frames.

During CAN messages reception by FlexCAN, the RXGMASK (Rx Global Mask) is used as acceptance mask for most of the Rx Message Buffers (MB). When the FIFO Enable bit in the FlexCAN Module Configuration Register (CANx\_MCR[FEN], bit 2) is set, the RXGMASK also applies to most of the elements of the ID filter table. However, there is a misalignment between the position of the ID field in the Rx MB and in RXIDA, RXIDB, and RXIDC fields of the ID Tables. In fact, the RXIDA filter in the ID Tables is shifted one bit to the left from Rx MBs ID position as shown below:

- Rx MB ID = bits 3–31 of ID word corresponding to message ID bits 0–28
- RXIDA = bits 2–30 of ID Table corresponding to message ID bits 0–28

The mask bits one-to-one correspondence occurs with the filters bits, not with the incoming message ID bits. This leads the RXGMASK to affect Rx MB and Rx FIFO filtering in different ways.

For example, if the user intends to mask out the bit 24 of the ID filter of Message Buffers then the RXGMASK will be configured as 0xffff\_ffef. As result, bit 24 of the ID field of the incoming message will be ignored during filtering process for Message Buffers. This very same configuration of RXGMASK would lead bit 24 of RXIDA to be “don’t care” and thus bit 25 of the ID field of the incoming message would be ignored during filtering process for Rx FIFO.

Similarly, both RXIDB and RXIDC filters have multiple misalignments with regards to position of ID field in Rx MBs, which can lead to erroneous masking during filtering process for either Rx FIFO or MBs.

RX14MASK (Rx 14 Mask) and RX15MASK (Rx 15 Mask) have the same structure as the RXGMASK. This includes the misalignment problem between the position of the ID field in the Rx MBs, and in RXIDA, RXIDB, and RXIDC fields of the ID Tables.

Therefore it is recommended that one of the following actions be taken to avoid problems:

- Do not enable the Rx FIFO. If  $CAN_x\_MCR[FEN]=0$  then the Rx FIFO is disabled and thus the masks RXGMASK, RX14MASK, and RX15MASK do not affect it.
- Enable Rx Individual Mask Registers. If the Backwards Compatibility Configuration bit in the FlexCAN Module Configuration Register ( $CAN_x\_MCR[BCC]$ , bit 15) is set then the Rx Individual Mask Registers (RXIMR0–63) are enabled, and thus the masks RXGMASK, RX14MASK, and RX15MASK are not used.
- Do not use masks RXGMASK, RX14MASK, and RX15MASK (that is, leave them at their reset value, which is 0xFFFF\_FFFF) when  $CAN_x\_MCR[FEN]=1$  and  $CAN_x\_MCR[BCC]=0$ . In this case, filtering processes for both Rx MBs and Rx FIFO are not affected by those masks.
- Do not configure any MB as Rx (i.e., let all MBs as either Tx or inactive) when  $CAN_x\_MCR[FEN]=1$  and  $CAN_x\_MCR[BCC]=0$ . In this case, the masks RXGMASK, RX14MASK, and RX15MASK can be used to affect ID Tables without affecting filtering process for Rx MBs.

Offset: 0x0010

Access: Read/write

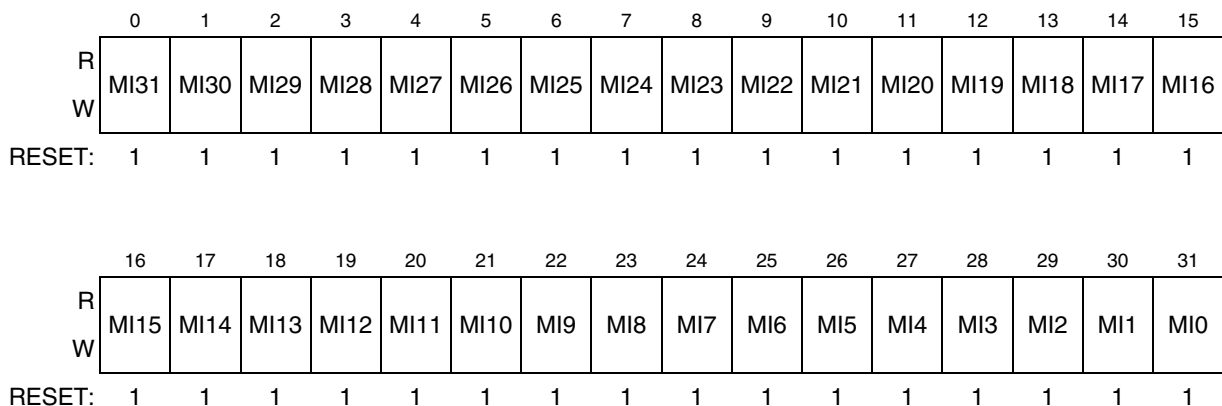


Figure 25-8. Rx Global Mask (RXGMASK) register



Table 25-12. RXGMASK field descriptions

| Field | Description  |
|-------|--|
| MIn   | Mask Bits<br>For normal Rx MBs, the mask bits affect the ID filter programmed on the MB. For the Rx FIFO, the mask bits affect all bits programmed in the filter table (ID, IDE, RTR).<br>0 The corresponding bit in the filter is “don’t care”<br>1 The corresponding bit in the filter is checked against the one received |

#### 25.4.4.5 Rx 14 Mask (RX14MASK) register

This register is provided for legacy support and for MCUs that do not have the individual masking per Message Buffer feature. Setting the BCC bit in MCR causes the RX14MASK register to have no effect on the module operation.

RX14MASK is used as acceptance mask for the Identifier in Message Buffer 14. When the FEN bit in MCR is set (FIFO enabled), the RXG14MASK also applies to element 6 of the ID filter table. This register has the same structure as the Rx Global Mask register.

See [Section 25.5.8, Rx FIFO](#), for important details on usage of RX14MASK on filtering process for Rx FIFO.

The contents of this register must be programmed while the module is in Freeze Mode, and must not be modified when the module is transmitting or receiving frames.

- Address offset: 0x14
- Reset value: 0xFFFF\_FFFF

#### 25.4.4.6 Rx 15 Mask (RX15MASK) register

This register is provided for legacy support and for MCUs that do not have the individual masking per Message Buffer feature. Setting the BCC bit in MCR causes the RX15MASK register to have no effect on the module operation.

When the BCC bit is negated, RX15MASK is used as acceptance mask for the Identifier in Message Buffer 15. When the FEN bit in MCR is set (FIFO enabled), the RXG15MASK also applies to element 7 of the ID filter table. This register has the same structure as the Rx Global Mask register.

Refer to [Section 25.5.8, Rx FIFO](#), for important details on usage of RX15MASK on filtering process for Rx FIFO.

The contents of this register must be programmed while the module is in Freeze Mode, and must not be modified when the module is transmitting or receiving frames.

- Address offset: 0x18
- Reset value: 0xFFFF\_FFFF

#### 25.4.4.7 Error Counter Register (ECR)

This register has two 8-bit fields reflecting the value of two FlexCAN error counters: Transmit Error Counter (TX\_ERR\_COUNTER field) and Receive Error Counter (RX\_ERR\_COUNTER field). The rules

for increasing and decreasing these counters are described in the CAN protocol and are completely implemented in the FlexCAN module. Both counters are read only except in Freeze Mode, where they can be written by the CPU.

Writing to the Error Counter Register while in Freeze Mode is an indirect operation. The data is first written to an auxiliary register and then an internal request/acknowledge procedure across clock domains is executed. All this is transparent to the user, except for the fact that the data will take some time to be actually written to the register. If desired, software can poll the register to discover when the data was actually written.

FlexCAN responds to any bus state as described in the protocol, e.g., transmit Error Active or Error Passive flag, delay its transmission start time (Error Passive) and avoid any influence on the bus when in Bus Off state. The following are the basic rules for FlexCAN bus state transitions.

- If the value of TX\_ERR\_COUNTER or RX\_ERR\_COUNTER increases to be greater than or equal to 128, the FLT\_CONF field in the Error and Status Register is updated to reflect Error Passive state.
- If the FlexCAN state is Error Passive, and either TX\_ERR\_COUNTER or RX\_ERR\_COUNTER decrements to a value less than or equal to 127 while the other already satisfies this condition, the FLT\_CONF field in the Error and Status Register is updated to reflect Error Active state.
- If the value of TX\_ERR\_COUNTER increases to be greater than 255, the FLT\_CONF field in the Error and Status Register is updated to reflect Bus Off state, and an interrupt may be issued. The value of TX\_ERR\_COUNTER is then reset to zero.
- If the FlexCAN is in Bus Off state, then Tx\_Err\_Counter is cascaded together with another internal counter to count the 128th occurrences of 11 consecutive recessive bits on the bus. Hence, TX\_ERR\_COUNTER is reset to 0 and counts in a manner where the internal counter counts 11 such bits and then wraps around while incrementing the TX\_ERR\_COUNTER. When TX\_ERR\_COUNTER reaches the value of 128, the FLT\_CONF field in the Error and Status Register is updated to be Error Active and both error counters are reset to 0. At any instance of dominant bit following a stream of less than 11 consecutive recessive bits, the internal counter resets itself to 0 without affecting the TX\_ERR\_COUNTER value.
- If during system start-up, only one node is operating, then its TX\_ERR\_COUNTER increases in each message it is trying to transmit, as a result of acknowledge errors (indicated by the ACK\_ERR bit in the Error and Status Register). After the transition to Error Passive state, the TX\_ERR\_COUNTER does not increment anymore by acknowledge errors. Therefore the device never goes to the Bus Off state.
- If the RX\_ERR\_COUNTER increases to a value greater than 127, it is not incremented further, even if more errors are detected while being a receiver. At the next successful message reception, the counter is set to a value between 119 and 127 to resume to Error Active state.

Offset: 0x001C

Access: Read/write

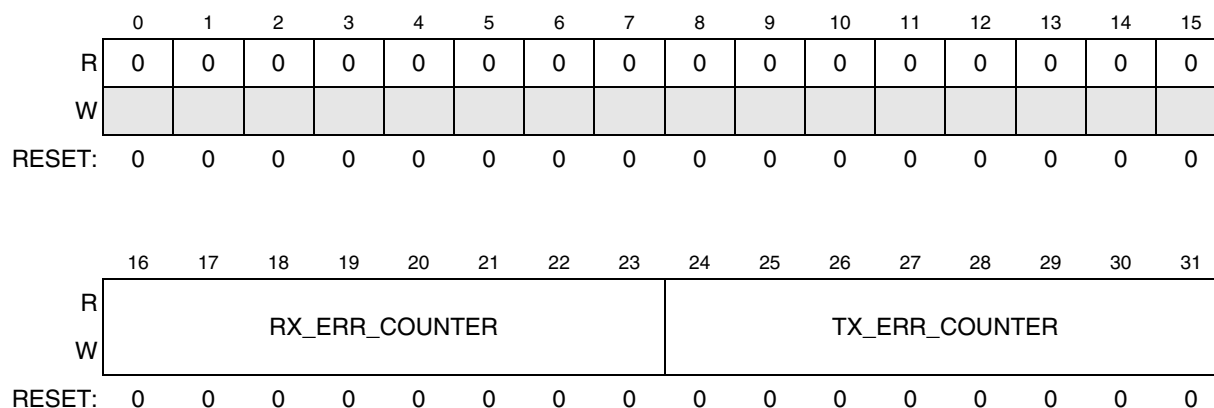


Figure 25-9. Error Counter Register (ECR)

Table 25-13. ECR field descriptions

| Field            | Description   |
|------------------|---|
| RX_ERROR_COUNTER | Receive Error Counter. See the text of this section for a detailed description of this field and how it interacts with TX_ERROR_COUNTER.  |
| TX_ERROR_COUNTER | Transmit Error Counter. See the text of this section for a detailed description of this field and how it interacts with RX_ERROR_COUNTER. |

#### 25.4.4.8 Error and Status Register (ESR)

This register reflects various error conditions, some general status of the device and it is the source of four interrupts to the CPU. The reported error conditions (bits 16–21) are those that occurred since the last time the CPU read this register. The CPU read action clears bits 16–23. Bits 22–28 are status bits.

Most bits in this register are read-only, except TWRN\_INT, RWRN\_INT, BOFF\_INT, and ERR\_INT, which are interrupt flags that can be cleared by writing 1 to them (writing 0 has no effect). See [Section 25.5.11, Interrupts](#), for more details.

Offset: 0x0020

Access: Read/write

|        |   |   |   |   |   |   |   |   |   |   |    |    |    |    |          |          |
|--------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----------|----------|
|        | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14       | 15       |
| R      | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | TWRN_INT | RWRN_INT |
| W      |   |   |   |   |   |   |   |   |   |   |    |    |    |    |          |          |
| RESET: | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0        | 0        |

|        |          |          |         |         |         |         |        |        |      |      |          |    |          |         |    |    |
|--------|----------|----------|---------|---------|---------|---------|--------|--------|------|------|----------|----|----------|---------|----|----|
|        | 16       | 17       | 18      | 19      | 20      | 21      | 22     | 23     | 24   | 25   | 26       | 27 | 28       | 29      | 30 | 31 |
| R      | BIT1_ERR | BIT0_ERR | ACK_ERR | CRC_ERR | FRM_ERR | STF_ERR | TX_WRN | RX_WRN | IDLE | TXRX | FLT_CONF | 0  | BOFF_INT | ERR_INT | 0  |    |
| W      |          |          |         |         |         |         |        |        |      |      |          |    |          |         |    |    |
| RESET: | 0        | 0        | 0       | 0       | 0       | 0       | 0      | 0      | 0    | 0    | 0        | 0  | 0        | 0       | 0  |    |

Figure 25-10. Error and Status Register (ESR)

Table 25-14. ESR field descriptions

| Field    | Description   |
|----------|---|
| TWRN_INT | <p>TWRN_INT — Tx Warning Interrupt Flag</p> <p>If the WRN_EN bit in MCR is asserted, the TWRN_INT bit is set when the TX_WRN flag transition from 0 to 1, meaning that the Tx error counter reached 96. If the corresponding mask bit in the Control Register (TWRN_MSK) is set, an interrupt is generated to the CPU. This bit is cleared by writing it to 1. Writing 0 has no effect.</p> <p>0 No such occurrence<br/>1 The Tx error counter transition from &lt; 96 to ≥ 96</p>  |
| RWRN_INT | <p>RWRN_INT — Rx Warning Interrupt Flag</p> <p>If the WRN_EN bit in MCR is asserted, the RWRN_INT bit is set when the RX_WRN flag transition from 0 to 1, meaning that the Rx error counters reached 96. If the corresponding mask bit in the Control Register (RWRN_MSK) is set, an interrupt is generated to the CPU. This bit is cleared by writing it to 1. Writing 0 has no effect.</p> <p>0 No such occurrence<br/>1 The Rx error counter transition from &lt; 96 to ≥ 96</p> |
| BIT1_ERR | <p>BIT1_ERR — Bit1 Error</p> <p>This bit indicates when an inconsistency occurs between the transmitted and the received bit in a message.</p> <p>0 No such occurrence<br/>1 At least one bit sent as recessive is received as dominant</p> <p><b>Note:</b> This bit is not set by a transmitter in case of arbitration field or ACK slot, or in case of a node sending a passive error flag that detects dominant bits.</p>  |
| BIT0_ERR | <p>BIT0_ERR — Bit0 Error</p> <p>This bit indicates when an inconsistency occurs between the transmitted and the received bit in a message.</p> <p>0 No such occurrence<br/>1 At least one bit sent as dominant is received as recessive</p>   |

Table 25-14. ESR field descriptions (continued)

| Field    | Description  |
|----------|--|
| ACK_ERR  | <p>ACK_ERR — Acknowledge Error</p> <p>This bit indicates that an Acknowledge Error has been detected by the transmitter node, i.e., a dominant bit has not been detected during the ACK SLOT.</p> <p>0 No such occurrence<br/>1 An ACK error occurred since last read of this register</p>   |
| CRC_ERR  | <p>CRC_ERR — Cyclic Redundancy Check Error</p> <p>This bit indicates that a CRC Error has been detected by the receiver node, i.e., the calculated CRC is different from the received.</p> <p>0 No such occurrence<br/>1 A CRC error occurred since last read of this register.</p>  |
| FRM_ERR  | <p>FRM_ERR — Form Error</p> <p>This bit indicates that a Form Error has been detected by the receiver node, i.e., a fixed-form bit field contains at least one illegal bit.</p> <p>0 No such occurrence<br/>1 A Form Error occurred since last read of this register</p>   |
| STF_ERR  | <p>STF_ERR — Stuffing Error</p> <p>This bit indicates that a Stuffing Error has been detected.</p> <p>0 No such occurrence.<br/>1 A Stuffing Error occurred since last read of this register.</p>  |
| TX_WRN   | <p>TX Error Warning</p> <p>This bit indicates when repetitive errors are occurring during message transmission.</p> <p>0 No such occurrence<br/>1 TX_Err_Counter <math>\geq</math> 96</p>  |
| RX_WRN   | <p>Rx Error Counter</p> <p>This bit indicates when repetitive errors are occurring during message reception.</p> <p>0 No such occurrence<br/>1 Rx_Err_Counter <math>\geq</math> 96</p>   |
| IDLE     | <p>CAN bus IDLE state</p> <p>This bit indicates when CAN bus is in IDLE state.</p> <p>0 No such occurrence<br/>1 CAN bus is now IDLE</p>   |
| TXRX     | <p>Current FlexCAN status (transmitting/receiving)</p> <p>This bit indicates if FlexCAN is transmitting or receiving a message when the CAN bus is not in IDLE state. This bit has no meaning when IDLE is asserted.</p> <p>0 FlexCAN is receiving a message (IDLE=0)<br/>1 FlexCAN is transmitting a message (IDLE=0)</p>   |
| FLT_CONF | <p>Fault Confinement State</p> <p>This 2-bit field indicates the Confinement State of the FlexCAN module, as shown in <a href="#">Table 25-15</a>. If the LOM bit in the Control Register is asserted, the FLT_CONF field will indicate “Error Passive”. Since the Control Register is not affected by soft reset, the FLT_CONF field will not be affected by soft reset if the LOM bit is asserted.</p> |
| BOFF_INT | <p>Bus Off Interrupt</p> <p>This bit is set when FlexCAN enters Bus Off state. If the corresponding mask bit in the Control Register (BOFF_MSK) is set, an interrupt is generated to the CPU. This bit is cleared by writing it to 1. Writing 0 has no effect.</p> <p>0 No such occurrence<br/>1 FlexCAN module entered Bus Off state</p>  |

Table 25-14. ESR field descriptions (continued)

| Field   | Description   |
|---------|---|
| ERR_INT | <p>Error Interrupt</p> <p>This bit indicates that at least one of the Error Bits (bits 16-21) is set. If the corresponding mask bit in the Control Register (ERR_MSK) is set, an interrupt is generated to the CPU. This bit is cleared by writing it to 1. Writing 0 has no effect.</p> <p>0 No such occurrence</p> <p>1 Indicates setting of any Error Bit in the Error and Status Register</p> |

Table 25-15. Fault confinement state

| Value | Meaning       |
|-------|---------------|
| 00    | Error Active  |
| 01    | Error Passive |
| 1X    | Bus Off       |

### 25.4.4.9 Interrupt Masks 2 (IMASK2) register

This register allows any number of a range of 32 Message Buffer Interrupts to be enabled or disabled. It contains one interrupt mask bit per buffer, enabling the CPU to determine which buffer generates an interrupt after a successful transmission or reception (i.e., when the corresponding IFLAG2 bit is set).

Offset: 0x0024

Access: Read/write

|        | 0   | 1   | 2   | 3   | 4   | 5   | 6   | 7   | 8   | 9   | 10  | 11  | 12  | 13  | 14  | 15  |
|--------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| R      | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF |
| W      | 63M | 62M | 61M | 60M | 59M | 58M | 57M | 56M | 55M | 54M | 53M | 52M | 51M | 50M | 49M | 48M |
| RESET: | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |

|        | 16  | 17  | 18  | 19  | 20  | 21  | 22  | 23  | 24  | 25  | 26  | 27  | 28  | 29  | 30  | 31  |
|--------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| R      | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF |
| W      | 47M | 46M | 45M | 44M | 43M | 42M | 41M | 40M | 39M | 38M | 37M | 36M | 35M | 34M | 33M | 32M |
| RESET: | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |

Figure 25-11. Interrupt Masks 2 (IMASK2) register

Table 25-16. MASK2 field descriptions

| Field              | Description  |
|--------------------|--|
| BUF <sub>n</sub> M | <p>Buffer MB<sub>n</sub> Mask</p> <p>Each bit enables or disables the respective FlexCAN Message Buffer (MB32 to MB63) Interrupt.</p> <p>0 The corresponding buffer Interrupt is disabled</p> <p>1 The corresponding buffer Interrupt is enabled</p> <p><b>Note:</b> Setting or clearing a bit in the IMASK2 register can assert or negate an interrupt request, if the corresponding IFLAG2 bit is set.</p> |

### 25.4.4.10 Interrupt Masks 1 (IMASK1) register

This register allows to enable or disable any number of a range of 32 Message Buffer Interrupts. It contains one interrupt mask bit per buffer, enabling the CPU to determine which buffer generates an interrupt after a successful transmission or reception (i.e., when the corresponding IFLAG1 bit is set).

Offset: 0x0028

Access: Read/write

|        | 0   | 1   | 2   | 3   | 4   | 5   | 6   | 7   | 8   | 9   | 10  | 11  | 12  | 13  | 14  | 15  |
|--------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| R      | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF |
| W      | 31M | 30M | 29M | 28M | 27M | 26M | 25M | 24M | 23M | 22M | 21M | 20M | 19M | 18M | 17M | 16M |
| RESET: | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |

|        | 16  | 17  | 18  | 19  | 20  | 21  | 22  | 23  | 24  | 25  | 26  | 27  | 28  | 29  | 30  | 31  |
|--------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| R      | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF |
| W      | 15M | 14M | 13M | 12M | 11M | 10M | 9M  | 8M  | 7M  | 6M  | 5M  | 4M  | 3M  | 2M  | 1M  | 0M  |
| RESET: | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |

Figure 25-12. Interrupt Masks 1 (IMASK1) register

Table 25-17. IMASK1 field descriptions

| Field              | Description  |
|--------------------|--|
| BUF <sub>n</sub> M | Buffer MB <sub>n</sub> Mask<br>Each bit enables or disables the respective FlexCAN Message Buffer (MB0 to MB31) Interrupt.<br>0 The corresponding buffer Interrupt is disabled<br>1 The corresponding buffer Interrupt is enabled<br><br><b>Note:</b> Setting or clearing a bit in IMASK1 can assert or negate an interrupt request, if the corresponding IFLAG1 bit is set. |

### 25.4.4.11 Interrupt Flags 2 (IFLAG2) register

This register defines the flags for 32 Message Buffer interrupts. It contains one interrupt flag bit per buffer. Each successful transmission or reception sets the corresponding IFLAG2 bit. If the corresponding IMASK2 bit is set, an interrupt will be generated. The interrupt flag must be cleared by writing it to 1. Writing 0 has no effect.

When the AEN bit in the MCR is set (Abort enabled), while the IFLAG2 bit is set for a MB configured as Tx, the writing access done by CPU into the corresponding MB will be blocked.

Offset: 0x002C

Access: Read/write

|        | 0   | 1   | 2   | 3   | 4   | 5   | 6   | 7   | 8   | 9   | 10  | 11  | 12  | 13  | 14  | 15  |
|--------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| R      | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF |
| W      | 63I | 62I | 61I | 60I | 59I | 58I | 57I | 56I | 55I | 54I | 53I | 52I | 51I | 50I | 49I | 48I |
| RESET: | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |

|        | 16  | 17  | 18  | 19  | 20  | 21  | 22  | 23  | 24  | 25  | 26  | 27  | 28  | 29  | 30  | 31  |
|--------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| R      | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF |
| W      | 47I | 46I | 45I | 44I | 43I | 42I | 41I | 40I | 39I | 38I | 37I | 36I | 35I | 34I | 33I | 32I |
| RESET: | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |

Figure 25-13. Interrupt Flags 2 (IFLAG2) register

Table 25-18. IFLAG2 field descriptions

| Field     | Description  |
|-----------|--|
| BUF $n$ I | Buffer MB $n$ Interrupt<br>Each bit flags the respective FlexCAN Message Buffer (MB32 to MB63) interrupt.<br>0 No such occurrence<br>1 The corresponding buffer has successfully completed transmission or reception |

#### 25.4.4.12 Interrupt Flags 1 (IFLAG1) register

This register defines the flags for 32 Message Buffer interrupts and FIFO interrupts. It contains one interrupt flag bit per buffer. Each successful transmission or reception sets the corresponding IFLAG1 bit. If the corresponding IMASK1 bit is set, an interrupt will be generated. The Interrupt flag must be cleared by writing it to 1. Writing 0 has no effect.

When the MCR[AEN] bit is set (Abort enabled), while the IFLAG1 bit is set for a MB configured as Tx, the writing access done by CPU into the corresponding MB will be blocked.

When the MCR[FEN] bit is set (FIFO enabled), the function of the 8 least significant interrupt flags (BUF7I–BUF0I) is changed to support the FIFO operation. BUF7I, BUF6I, and BUF5I indicate operating conditions of the FIFO, while BUF4I to BUF0I are not used.



Offset: 0x002C

Access: Read/write

|        |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|--------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
|        | 0   | 1   | 2   | 3   | 4   | 5   | 6   | 7   | 8   | 9   | 10  | 11  | 12  | 13  | 14  | 15  |
| R      | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF |
| W      | 31I | 30I | 29I | 28I | 27I | 26I | 25I | 24I | 23I | 22I | 21I | 20I | 19I | 18I | 17I | 16I |
| RESET: | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |

|        |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|--------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
|        | 16  | 17  | 18  | 19  | 20  | 21  | 22  | 23  | 24  | 25  | 26  | 27  | 28  | 29  | 30  | 31  |
| R      | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF | BUF |
| W      | 15I | 14I | 13I | 12I | 11I | 10I | 9I  | 8I  | 7I  | 6I  | 5I  | 4I  | 3I  | 2I  | 1I  | 0I  |
| RESET: | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |

Figure 25-14. Interrupt Flags 1 (IFLAG1) register

Table 25-19. IFLAG1 field descriptions

| Field        | Description  |
|--------------|--|
| BUF31I–BUF8I | Buffer MB <sub>n</sub> Interrupt<br>Each bit flags the respective FlexCAN Message Buffer (MB8 to MB31) interrupt.<br>0 No such occurrence<br>1 The corresponding MB has successfully completed transmission or reception   |
| BUF7I        | Buffer MB7 Interrupt or FIFO Overflow<br>If the FIFO is not enabled, this bit flags the interrupt for MB7. If the FIFO is enabled, this flag indicates an overflow condition in the FIFO (frame lost because FIFO is full).<br>0 No such occurrence<br>1 MB7 completed transmission/reception or FIFO overflow                     |
| BUF6I        | Buffer MB6 Interrupt or FIFO Warning<br>If the FIFO is not enabled, this bit flags the interrupt for MB6. If the FIFO is enabled, this flag indicates that 5 out of 6 buffers of the FIFO are already occupied (FIFO almost full).<br>0 No such occurrence<br>1 MB6 completed transmission/reception or FIFO almost full           |
| BUF5I        | Buffer MB5 Interrupt or Frames available in FIFO<br>If the FIFO is not enabled, this bit flags the interrupt for MB5. If the FIFO is enabled, this flag indicates that at least one frame is available to be read from the FIFO.<br>0 No such occurrence<br>1 MB5 completed transmission/reception or frames available in the FIFO |
| BUF4I–BUF0I  | Buffer MB <sub>i</sub> Interrupt or reserved<br>If the FIFO is not enabled, these bits flag the interrupts for MB0 to MB4. If the FIFO is enabled, these flags are not used and must be considered as reserved locations.<br>0 No such occurrence<br>1 Corresponding MB completed transmission/reception                           |

#### 25.4.4.13 Rx Individual Mask Registers (RXIMR0–RXIMR63)

These registers are used as acceptance masks for ID filtering in Rx MBs and the FIFO. If the FIFO is not enabled, one mask register is provided for each available Message Buffer, providing ID masking capability on a per Message Buffer basis. When the FIFO is enabled (FEN bit in MCR is set), the first 8 Mask

Registers apply to the 8 elements of the FIFO filter table (on a one-to-one correspondence), while the rest of the registers apply to the regular MBs, starting from MB8.

The Individual Rx Mask Registers are implemented in RAM, so they are not affected by reset and must be explicitly initialized prior to any reception. Furthermore, they can only be accessed by the CPU while the module is in Freeze Mode. Out of Freeze Mode, write accesses are blocked and read accesses will return all zeros. Furthermore, if the BCC bit in MCR is negated, any read or write operation to these registers results in access error.

Offsets: 0x0880–0x097F (64 registers)

Access: Read/write

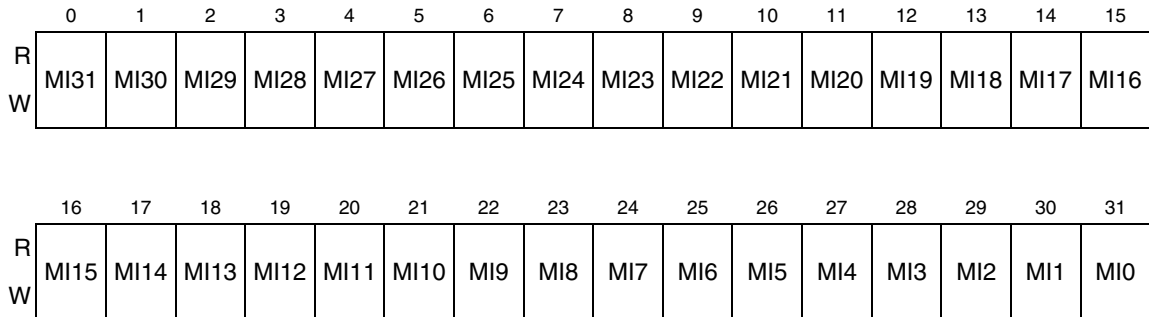


Figure 25-15. Rx Individual Mask Registers (RXIMR0–RXIMR63)

Table 25-20. RXIMR0–RXIMR63 field descriptions

| Field  | Description  |
|--------|--|
| MI $n$ | Mask Bits<br>For normal Rx MBs, the mask bits affect the ID filter programmed on the MB. For the Rx FIFO, the mask bits affect all bits programmed in the filter table (ID, IDE, RTR).<br>0 The corresponding bit in the filter is “don’t care”<br>1 The corresponding bit in the filter is checked against the one received |

## 25.5 Functional description

### 25.5.1 Overview

The FlexCAN module is a CAN protocol engine with a very flexible mailbox system for transmitting and receiving CAN frames. The mailbox system is composed by a set of as many as 64 Message Buffers (MB) that store configuration and control data, time stamp, message ID, and data (see [Section 25.4.2, Message Buffer Structure](#)). The memory corresponding to the first 8 MBs can be configured to support a FIFO reception scheme with a powerful ID filtering mechanism, capable of checking incoming frames against a table of IDs (as many as 8 extended IDs or 16 standard IDs or 32 8-bit ID slices), each one with its own individual mask register. Simultaneous reception through FIFO and mailbox is supported. For mailbox reception, a matching algorithm makes it possible to store received frames only into MBs that have the same ID programmed on its ID field. A masking scheme makes it possible to match the ID programmed on the MB with a range of IDs on received CAN frames. For transmission, an arbitration algorithm decides the prioritization of MBs to be transmitted based on the message ID (optionally augmented by 3 local priority bits) or the MB ordering.

Before proceeding with the functional description, an important concept must be explained. A Message Buffer is said to be active at a given time if it can participate in the matching and arbitration algorithms that are happening at that time. An Rx MB with a 0000 code is inactive (refer to [Table 25-5](#)). Similarly, a Tx MB with a 1000 or 1001 code is also inactive (refer to [Table 25-6](#)). An MB programmed with 0000, 1000, (inactive), or 1001 (abort) will be temporarily deactivated (will not participate in the current arbitration or matching run) when the CPU writes to the C/S field of that MB (see [Section 25.5.7.2, Message buffer deactivation](#)).

## 25.5.2 Local priority transmission

The term local priority refers to the priority of transmit messages of the host node. This allows increased control over the priority mechanism for transmitting messages. [Figure 25-2](#) shows the placement of PRIO in the ID part of the message buffer.

An additional 3-bit field (PRIO) in the long-word ID part of the message buffer structure has been added for local priority determination. They are prefixed to the regular ID to define the transmission priority. These bits are not transmitted and are intended only for Tx buffers.

Perform the following to use the local priority feature:

1. Set the LPRIO\_EN bit in the CANx\_MCR.
2. Write the additional PRIO bits in the ID long-word of Tx message buffers when configuring the Tx buffers.

With this extended ID concept, the arbitration process is based on the full 32-bit word. However, the actual transmitted ID continues to have 11 bits for standard frames and 29 bits for extended frames.

## 25.5.3 Transmit process

In order to transmit a CAN frame, the CPU must prepare a Message Buffer for transmission by executing the following procedure:

1. If the MB is active (transmission pending), write an ABORT code (1001) to the Code field of the Control and Status word to request an abortion of the transmission, then read back the Code field and the IFLAG register to check if the transmission was aborted (see [Section 25.5.7.1, Transmission abort mechanism](#)). If backwards compatibility is desired (AEN in MCR negated), write 1000 to the Code field to inactivate the MB but then the pending frame may be transmitted without notification (see [Section 25.5.7.2, Message buffer deactivation](#)).
2. Write the ID word.
3. Write the data bytes.
4. Write the Length, Control, and Code fields of the Control and Status word to activate the MB.

Once the MB is activated in the fourth step, it will participate into the arbitration process and eventually be transmitted according to its priority. At the end of the successful transmission, the value of the Free Running Timer is written into the Time Stamp field, the Code field in the Control and Status word is updated, a status flag is set in the Interrupt Flag register and an interrupt is generated if allowed by the corresponding Interrupt Mask register bit. The new Code field after transmission depends on the code that was used to activate the MB in step four (see [Table 25-5](#) and [Table 25-6](#) in [Section 25.4.2, Message Buffer](#)

**Structure**). When the Abort feature is enabled (AEN in MCR is asserted), after the Interrupt Flag is asserted for a MB configured as transmit buffer, the MB is blocked, therefore the CPU is not able to update it until the Interrupt Flag be negated by CPU. It means that the CPU must clear the corresponding IFLAG before starting to prepare this MB for a new transmission or reception.

## 25.5.4 Arbitration process

The arbitration process is an algorithm executed by the MBM that scans the whole MB memory looking for the highest priority message to be transmitted. All MBs programmed as transmit buffers will be scanned to find the lowest ID<sup>1</sup> or the lowest MB number or the highest priority, depending on the LBUF and LPRIO\_EN bits on the Control Register. The arbitration process is triggered in the following events:

- During the CRC field of the CAN frame
- During the error delimiter field of the CAN frame
- During Intermission, if the winner MB defined in a previous arbitration was deactivated, or if there was no MB to transmit, but the CPU wrote to the C/S word of any MB after the previous arbitration finished
- When MBM is in Idle or Bus Off state and the CPU writes to the C/S word of any MB
- Upon leaving Freeze mode

When LBUF is asserted, the LPRIO\_EN bit has no effect and the lowest number buffer is transmitted first. When LBUF and LPRIO\_EN are both negated, the MB with the lowest ID is transmitted first but. If LBUF is negated and LPRIO\_EN is asserted, the PRIO bits augment the ID used during the arbitration process. With this extended ID concept, arbitration is done based on the full 32-bit ID and the PRIO bits define which MB should be transmitted first, therefore MBs with PRIO = 000 have higher priority. If two or more MBs have the same priority, the regular ID will determine the priority of transmission. If two or more MBs have the same priority (3 extra bits) and the same regular ID, the lowest MB will be transmitted first.

Once the highest priority MB is selected, it is transferred to a temporary storage space called Serial Message Buffer (SMB), which has the same structure as a normal MB but is not user accessible. This operation is called move-out and after it is done, write access to the corresponding MB is blocked (if the AEN bit in MCR is asserted). The write access is released in the following events:

- After the MB is transmitted
- FlexCAN enters in HALT or BUS OFF
- FlexCAN loses the bus arbitration or there is an error during the transmission

At the first opportunity window on the CAN bus, the message on the SMB is transmitted according to the CAN protocol rules. FlexCAN transmits as many as 8 data bytes, even if the DLC (Data Length Code) value is bigger.

1. Actually, if LBUF is negated, the arbitration considers not only the ID, but also the RTR and IDE bits placed inside the ID at the same positions they are transmitted in the CAN frame.

## 25.5.5 Receive process

To be able to receive CAN frames into the mailbox MBs, the CPU must prepare one or more Message Buffers for reception by executing the following steps:

1. If the MB has a pending transmission, write an ABORT code (1001) to the Code field of the Control and Status word to request an abortion of the transmission, then read back the Code field and the IFLAG register to check if the transmission was aborted (see [Section 25.5.7.1, Transmission abort mechanism](#)). If backwards compatibility is desired (AEN in MCR negated), just write 1000 to the Code field to inactivate the MB, but then the pending frame may be transmitted without notification (see [Section 25.5.7.2, Message buffer deactivation](#)). If the MB already programmed as a receiver, just write 0000 to the Code field of the Control and Status word to keep the MB inactive.
2. Write the ID word.
3. Write 0100 to the Code field of the Control and Status word to activate the MB.

Once the MB is activated in the third step, it will be able to receive frames that match the programmed ID. At the end of a successful reception, the MB is updated by the MBM as follows:

1. The value of the Free Running Timer is written into the Time Stamp field.
2. The received ID, Data (8 bytes at most), and Length fields are stored.
3. The Code field in the Control and Status word is updated (see [Table 25-5](#) and [Table 25-6](#) in [Section 25.4.2, Message Buffer Structure](#)).
4. A status flag is set in the Interrupt Flag register and an interrupt is generated if allowed by the corresponding Interrupt Mask register bit.

Upon receiving the MB interrupt, the CPU should service the received frame using the following procedure:

1. Read the Control and Status word (mandatory – activates an internal lock for this buffer).
2. Read the ID field (optional – needed only if a mask was used).
3. Read the Data field.
4. Read the Free Running Timer (optional – releases the internal lock).

Upon reading the Control and Status word, if the BUSY bit is set in the Code field, then the CPU should defer the access to the MB until this bit is negated. Reading the Free Running Timer is not mandatory. If not executed the MB remains locked, unless the CPU reads the C/S word of another MB. Note that only a single MB is locked at a time. The only mandatory CPU read operation is the one on the Control and Status word to assure data coherency (see [Section 25.5.7, Data coherence](#)).

The CPU should synchronize to frame reception by the status flag bit for the specific MB in one of the IFLAG registers and not by the Code field of that MB. Polling the Code field does not work because once a frame was received and the CPU services the MB (by reading the C/S word followed by unlocking the MB), the Code field will not return to EMPTY. It will remain FULL, as explained in [Table 25-5](#). If the CPU tries to workaround this behavior by writing to the C/S word to force an EMPTY code after reading the MB, the MB is actually deactivated from any currently ongoing matching process. As a result, a newly received frame matching the ID of that MB may be lost. In summary: *never do polling by reading directly the C/S word of the MBs. Instead, read the IFLAG registers.*

Note that the received ID field is always stored in the matching MB, thus the contents of the ID field in an MB may change if the match was due to masking. Note also that FlexCAN does not receive frames transmitted by itself if there exists an Rx matching MB, provided the SRX\_DIS bit in the MCR is not asserted. If SRX\_DIS is asserted, FlexCAN will not store frames transmitted by itself in any MB, even if it contains a matching MB, and no interrupt flag or interrupt signal will be generated due to the frame reception.

To be able to receive CAN frames through the FIFO, the CPU must enable and configure the FIFO during Freeze Mode (see [Section 25.5.8, Rx FIFO](#)). Upon receiving the frames available interrupt from FIFO, the CPU should service the received frame using the following procedure:

1. Read the Control and Status word (optional – needed only if a mask was used for IDE and RTR bits).
2. Read the ID field (optional – needed only if a mask was used).
3. Read the Data field.
4. Clear the frames available interrupt (mandatory – release the buffer and allow the CPU to read the next FIFO entry).

### 25.5.6 Matching process

The matching process is an algorithm executed by the MBM that scans the MB memory looking for Rx MBs programmed with the same ID as the one received from the CAN bus. If the FIFO is enabled, the 8-entry ID table from FIFO is scanned first. If a match is not found within the FIFO table, the other MBs are scanned. In the event that the FIFO is full, the matching algorithm will always look for a matching MB outside the FIFO region.

When the frame is received, it is temporarily stored in a hidden auxiliary MB called Serial Message Buffer (SMB). The matching process takes place during the CRC field of the received frame. If a matching ID is found in the FIFO table or in one of the regular MBs, the contents of the SMB will be transferred to the FIFO or to the matched MB during the 6th bit of the End-Of-Frame field of the CAN protocol. This operation is called move-in. If any protocol error (CRC, ACK, etc.) is detected, then the move-in operation does not happen.

For the regular mailbox MBs, an MB is said to be free to receive a new frame if the following conditions are satisfied:

- The MB is not locked (see [Section 25.5.7.3, Message buffer lock mechanism](#))
- The Code field is either EMPTY or else it is FULL or OVERRUN but the CPU has already serviced the MB (read the C/S word and then unlocked the MB)

If the first MB with a matching ID is not free to receive the new frame, then the matching algorithm keeps looking for another free MB until it finds one. If it cannot find one that is free, then it will overwrite the last matching MB (unless it is locked) and set the Code field to OVERRUN (refer to [Table 25-5](#) and [Table 25-6](#)). If the last matching MB is locked, then the new message remains in the SMB, waiting for the MB to be unlocked (see [Section 25.5.7.3, Message buffer lock mechanism](#)).

Suppose, for example, that the FIFO is disabled and there are two MBs with the same ID, and FlexCAN starts receiving messages with that ID. Let us say that these MBs are the second and the fifth in the array. When the first message arrives, the matching algorithm will find the first match in MB number 2. The code

of this MB is EMPTY, so the message is stored there. When the second message arrives, the matching algorithm will find MB number 2 again, but it is not free to receive, so it will keep looking and find MB number 5 and store the message there. If yet another message with the same ID arrives, the matching algorithm finds out that there are no matching MBs that are free to receive, so it decides to overwrite the last matched MB, which is number 5. In doing so, it sets the Code field of the MB to indicate OVERRUN.

The ability to match the same ID in more than one MB can be exploited to implement a reception queue (in addition to the full featured FIFO) to allow more time for the CPU to service the MBs. By programming more than one MB with the same ID, received messages will be queued into the MBs. The CPU can examine the Time Stamp field of the MBs to determine the order in which the messages arrived.

The matching algorithm described above can be changed to be the same one used in previous versions of the FlexCAN module. When the BCC bit in MCR is negated, the matching algorithm stops at the first MB with a matching ID that it finds, whether this MB is free or not. As a result, the message queuing feature does not work if the BCC bit is negated.

Matching to a range of IDs is possible by using ID Acceptance Masks. FlexCAN supports individual masking per MB. Please refer to [Section 25.4.4.13, Rx Individual Mask Registers \(RXIMR0–RXIMR63\)](#). During the matching algorithm, if a mask bit is asserted, then the corresponding ID bit is compared. If the mask bit is negated, the corresponding ID bit is “don’t care”. Please note that the Individual Mask registers are implemented in RAM, so they are not initialized out of reset. Also, they can only be programmed if the BCC bit is asserted and while the module is in Freeze Mode.

FlexCAN also supports an alternate masking scheme with only three mask registers (RGXMASK, RX14MASK and RX15MASK) for backwards compatibility. This alternate masking scheme is enabled when the BCC bit in the MCR register is negated.

## 25.5.7 Data coherence

In order to maintain data coherency and proper FlexCAN operation, the CPU must obey the rules described in [Section 25.5.3, Transmit process](#) and [Section 25.5.5, Receive process](#). Any form of CPU accessing an MB structure within FlexCAN other than those specified may cause FlexCAN to behave in an unpredictable way.

### 25.5.7.1 Transmission abort mechanism

The abort mechanism provides a safe way to request the abortion of a pending transmission. A feedback mechanism is provided to inform the CPU if the transmission was aborted or if the frame could not be aborted and was transmitted instead. In order to maintain backwards compatibility, the abort mechanism must be explicitly enabled by asserting the AEN bit in the MCR.

In order to abort a transmission, the CPU must write a specific abort code (1001) to the Code field of the Control and Status word. When the abort mechanism is enabled, the active MBs configured as transmission must be aborted first and then they may be updated. If the abort code is written to an MB that is currently being transmitted, or to an MB that was already loaded into the SMB for transmission, the write operation is blocked and the MB is not deactivated, but the abort request is captured and kept pending until one of the following conditions is satisfied:

- The module loses the bus arbitration

- There is an error during the transmission
- The module is put into Freeze Mode

If none of these conditions is reached, the MB is transmitted correctly, the interrupt flag is set in the IFLAG register, and an interrupt to the CPU is generated (if enabled). The abort request is automatically cleared when the interrupt flag is set. On the other hand, if one of the above conditions is reached, the frame is not transmitted, therefore the abort code is written into the Code field, the interrupt flag is set in the IFLAG, and an interrupt is (optionally) generated to the CPU.

If the CPU writes the abort code before the transmission begins internally, then the write operation is not blocked, therefore the MB is updated and no interrupt flag is set. In this way the CPU just needs to read the abort code to make sure the active MB was deactivated. Although the AEN bit is asserted and the CPU wrote the abort code, in this case the MB is deactivated and not aborted, because the transmission did not start yet. One MB is only aborted when the abort request is captured and kept pending until one of the previous conditions are satisfied.

The abort procedure can be summarized as follows:

1. CPU writes 1001 into the code field of the C/S word.
2. CPU reads the CODE field and compares it to the value that was written.
3. If the CODE field that was read is different from the value that was written, the CPU must read the corresponding IFLAG to check if the frame was transmitted or it is being currently transmitted. If the corresponding IFLAG is set, the frame was transmitted. If the corresponding IFLAG is reset, the CPU must wait for it to be set, and then the CPU must read the CODE field to check if the MB was aborted (CODE=1001) or it was transmitted (CODE=1000).

### 25.5.7.2 Message buffer deactivation

Deactivation is mechanism provided to maintain data coherence when the CPU writes to the Control and Status word of active MBs out of Freeze Mode. Any CPU write access to the Control and Status word of an MB causes that MB to be excluded from the transmit or receive processes during the current matching or arbitration round. The deactivation is temporary, affecting only for the current match/arbitration round.

The purpose of deactivation is data coherency. The match/arbitration process scans the MBs to decide which MB to transmit or receive. If the CPU updates the MB in the middle of a match or arbitration process, the data of that MB may no longer be coherent, therefore deactivation of that MB is done.

Even with the coherence mechanism described above, writing to the Control and Status word of active MBs when not in Freeze Mode may produce undesirable results. Examples are:

- Matching and arbitration are one-pass processes. If MBs are deactivated after they are scanned, no reevaluation is done to determine a new match/winner. If an Rx MB with a matching ID is deactivated during the matching process after it was scanned, then this MB is marked as invalid to receive the frame, and FlexCAN will keep looking for another matching MB within the ones it has not scanned yet. If it cannot find one, then the message will be lost. Suppose, for example, that two MBs have a matching ID to a received frame, and the user deactivated the first matching MB after FlexCAN has scanned the second. The received frame will be lost even if the second matching MB was free to receive.



- If a Tx MB containing the lowest ID is deactivated after FlexCAN has scanned it, then FlexCAN will look for another winner within the MBs that it has not scanned yet. Therefore, it may transmit an MB with ID that may not be the lowest at the time because a lower ID might be present in one of the MBs that it had already scanned before the deactivation.
- There is a point in time until which the deactivation of a Tx MB causes it not to be transmitted (end of move-out). After this point, it is transmitted but no interrupt is issued and the Code field is not updated. In order to avoid this situation, the abort procedures described in [Section 25.5.7.1, Transmission abort mechanism](#), should be used.

### 25.5.7.3 Message buffer lock mechanism

Besides MB deactivation, FlexCAN has another data coherence mechanism for the receive process. When the CPU reads the Control and Status word of an “active not empty” Rx MB, FlexCAN assumes that the CPU wants to read the whole MB in an atomic operation, and thus it sets an internal lock flag for that MB. The lock is released when the CPU reads the Free Running Timer (global unlock operation), or when it reads the Control and Status word of another MB. The MB locking is done to prevent a new frame to be written into the MB while the CPU is reading it.

#### NOTE

The locking mechanism only applies to Rx MBs that have a code different than INACTIVE (0000) or EMPTY<sup>1</sup> (0100). Also, Tx MBs cannot be locked.

Suppose, for example, that the FIFO is disabled and the second and the fifth MBs of the array are programmed with the same ID, and FlexCAN has already received and stored messages into these two MBs. Suppose now that the CPU decides to read MB number 5 and at the same time another message with the same ID is arriving. When the CPU reads the Control and Status word of MB number 5, this MB is locked. The new message arrives and the matching algorithm finds out that there are no free to receive MBs, so it decides to override MB number 5. However, this MB is locked, so the new message cannot be written there. It will remain in the SMB waiting for the MB to be unlocked, and only then will be written to the MB. If the MB is not unlocked in time and yet another new message with the same ID arrives, then the new message overwrites the one on the SMB and there will be no indication of lost messages either in the Code field of the MB or in the Error and Status Register.

While the message is being moved in from the SMB to the MB, the BUSY bit on the Code field is asserted. If the CPU reads the Control and Status word and finds out that the BUSY bit is set, it should defer accessing the MB until the BUSY bit is negated.

#### NOTE

If the BUSY bit is asserted or if the MB is empty, then reading the Control and Status word does not lock the MB.

Deactivation takes precedence over locking. If the CPU deactivates a locked Rx MB, then its lock status is negated and the MB is marked as invalid for the current matching round. Any pending message on the SMB will not be transferred anymore to the MB.

<sup>1</sup> In previous FlexCAN versions, reading the C/S word locked the MB even if it was EMPTY. This behavior will be honored when the BCC bit is negated.

## 25.5.8 Rx FIFO

The receive-only FIFO is enabled by asserting the FEN bit in the MCR. The reset value of this bit is zero to maintain software backwards compatibility with previous versions of the module that did not have the FIFO feature. When the FIFO is enabled, the memory region normally occupied by the first 8 MBs (0x80–0xFF) is now reserved for use of the FIFO engine (see [Section 25.4.3, Rx FIFO structure](#)). Management of read and write pointers is done internally by the FIFO engine. The CPU can read the received frames sequentially, in the order they were received, by repeatedly accessing a Message Buffer structure at the beginning of the memory.

The FIFO can store as many as 6 frames pending service by the CPU. An interrupt is sent to the CPU when new frames are available in the FIFO. Upon receiving the interrupt, the CPU must read the frame (accessing an MB in the 0x80 address) and then clear the interrupt. The act of clearing the interrupt triggers the FIFO engine to replace the MB in 0x80 with the next frame in the queue, and then issue another interrupt to the CPU. If the FIFO is full and more frames continue to be received, an OVERFLOW interrupt is issued to the CPU and subsequent frames are not accepted until the CPU creates space in the FIFO by reading one or more frames. A warning interrupt is also generated when five frames are accumulated in the FIFO.

A powerful filtering scheme is provided to accept only frames intended for the target application, thus reducing the interrupt servicing work load. The filtering criteria is specified by programming a table of eight 32-bit registers that can be configured to one of the following formats (see also [Section 25.4.3, Rx FIFO structure](#)):

- Format A: 8 extended or standard IDs (including IDE and RTR)
- Format B: 16 standard IDs or 16 extended 14-bit ID slices (including IDE and RTR)
- Format C: 32 standard or extended 8-bit ID slices

### NOTE

A chosen format is applied to all eight registers of the filter table. It is not possible to mix formats within the table.

The eight elements of the filter table are individually affected by the first eight Individual Mask registers (RXIMR0–RXIMR7), allowing very powerful filtering criteria to be defined. The rest of the RXIMR, starting from RXIM8, continue to affect the regular MBs, starting from MB8. If the BCC bit is negated (or if the RXIMR are not available for the particular MCU), then the FIFO filter table is affected by the legacy mask registers as follows: element 6 is affected by RX14MASK, element 7 is affected by RX15MASK and the other elements (0 to 5) are affected by RXGMASK.

## 25.5.9 CAN protocol related features

### 25.5.9.1 Remote frames

Remote frame is a special kind of frame. The user can program a MB to be a Request Remote Frame by writing the MB as Transmit with the RTR bit set to 1. After the Remote Request frame is transmitted successfully, the MB becomes a Receive Message Buffer, with the same ID as before.

When a Remote Request frame is received by FlexCAN, its ID is compared to the IDs of the transmit message buffers with the Code field 1010. If there is a matching ID, then this MB frame will be transmitted. Note that if the matching MB has the RTR bit set, then FlexCAN will transmit a Remote Frame as a response.

A received Remote Request Frame is not stored in a receive buffer. It is only used to trigger a transmission of a frame in response. The mask registers are not used in remote frame matching, and all ID bits (except RTR) of the incoming received frame should match.

In the case that a Remote Request Frame was received and matched an MB, this message buffer immediately enters the internal arbitration process, but is considered as normal Tx MB, with no higher priority. The data length of this frame is independent of the DLC field in the remote frame that initiated its transmission.

If the Rx FIFO is enabled (bit FEN set in MCR), FlexCAN will not generate an automatic response for Remote Request Frames that match the FIFO filtering criteria. If the remote frame matches one of the target IDs, it will be stored in the FIFO and presented to the CPU. Note that for filtering formats A and B, it is possible to select whether remote frames are accepted or not. For format C, remote frames are always accepted (if they match the ID).

### 25.5.9.2 Overload frames

FlexCAN does transmit overload frames due to detection of following conditions on CAN bus:

- Detection of a dominant bit in the first/second bit of Intermission
- Detection of a dominant bit at the 7th bit (last) of End of Frame field (Rx frames)
- Detection of a dominant bit at the 8th bit (last) of Error Frame Delimiter or Overload Frame Delimiter

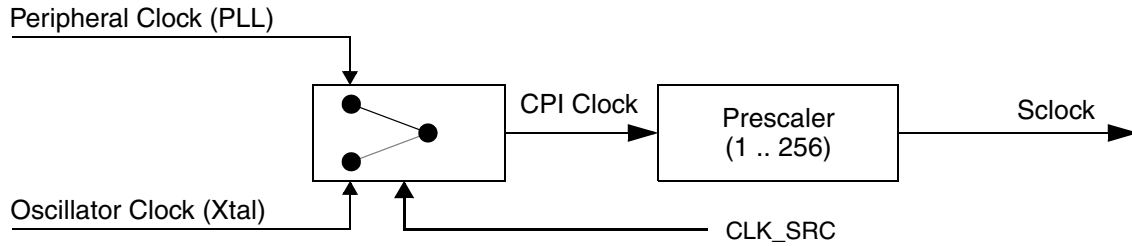
### 25.5.9.3 Time stamp

The value of the Free Running Timer is sampled at the beginning of the Identifier field on the CAN bus, and is stored at the end of move-in in the TIME STAMP field, providing network behavior with respect to time.

Note that the Free Running Timer can be reset upon a specific frame reception, enabling network time synchronization. Refer to TSYN description in [Section 25.4.4.2, Control \(CTRL\) register](#).

### 25.5.9.4 Protocol timing

[Figure 25-16](#) shows the structure of the clock generation circuitry that feeds the CAN Protocol Interface (CPI) submodule. The clock source bit (CLK\_SRC) in the CTRL register defines whether the internal clock is connected to the output of a crystal oscillator (Oscillator Clock) or to the Peripheral Clock (generally from a PLL). In order to guarantee reliable operation, the clock source should be selected while the module is in Disable Mode (bit MDIS set in MCR).



**Figure 25-16. CAN Engine Clocking Scheme**

The crystal oscillator clock should be selected whenever a tight tolerance (up to 0.1%) is required in the CAN bus timing. The crystal oscillator clock has better jitter performance than PLL generated clocks.

The FlexCAN module supports a variety of means to set up bit timing parameters that are required by the CAN protocol. The Control Register has various fields used to control bit timing parameters: PRESDIV, PROPSEG, PSEG1, PSEG2, and RJW. See [Section 25.4.4.2, Control \(CTRL\) register](#).

The PRESDIV field controls a prescaler that generates the Serial Clock (Sclock), whose period defines the time quantum used to compose the CAN waveform. A time quantum is the atomic unit of time handled by the CAN engine.

**Eqn. 25-1**

$$f_{Tq} = \frac{f_{CANCLK}}{\text{Prescaler value}}$$

A bit time is subdivided into three segments<sup>1</sup> (reference [Figure 25-17](#) and [Table 25-21](#)):

- SYNC\_SEG: This segment has a fixed length of one time quantum. Signal edges are expected to happen within this section.
- Time Segment 1: This segment includes the Propagation Segment and the Phase Segment 1 of the CAN standard. It can be programmed by setting the PROPSEG and the PSEG1 fields of CTRL so that their sum (plus 2) is in the range of 4 to 16 time quanta.
- Time Segment 2: This segment represents the Phase Segment 2 of the CAN standard. It can be programmed by setting the PSEG2 field of CTRL (plus 1) to be 2 to 8 time quanta long.

**Eqn. 25-2**

$$\text{Bit Rate} = \frac{f_{Tq}}{\text{(number of Time Quanta)}}$$

1. For further explanation of the underlying concepts please refer to ISO/DIS 11519-1, Section 10.3. Reference also the Bosch CAN 2.0A/B protocol specification dated September 1991 for bit timing.

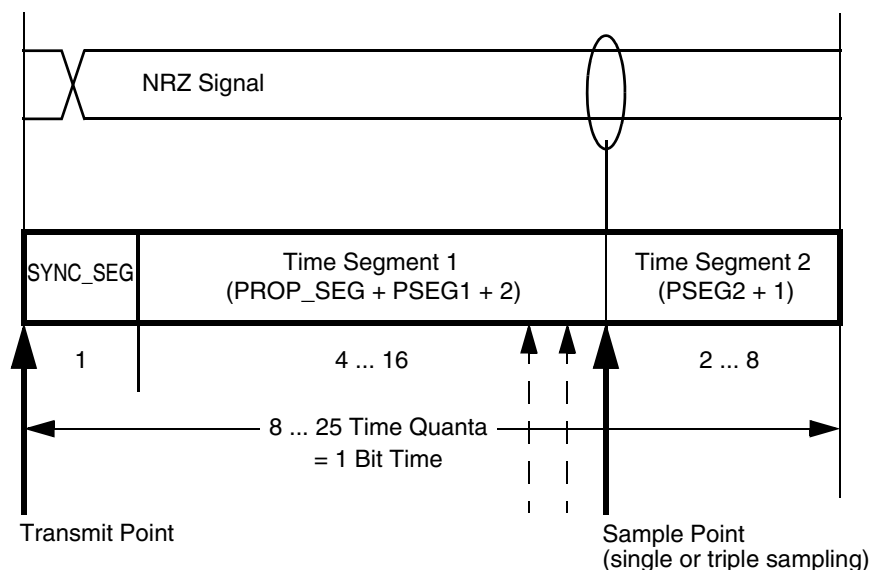


Figure 25-17. Segments within the Bit Time

Table 25-21. Time Segment Syntax

| Syntax         | Description  |
|----------------|--|
| SYNC_SEG       | System expects transitions to occur on the bus during this period.   |
| Transmit Point | A node in transmit mode transfers a new value to the CAN bus at this point.  |
| Sample Point   | A node samples the bus at this point. If the three samples per bit option is selected, then this point marks the position of the third sample. |

Table 25-22 gives an overview of the CAN compliant segment settings and the related parameter values.

Table 25-22. Bosch CAN 2.0B standard compliant bit time segment settings

| Time Segment 1 | Time Segment 2 | Resynchronization Jump Width |
|----------------|----------------|------------------------------|
| 5–10           | 2              | 1–2                          |
| 4–11           | 3              | 1–3                          |
| 5–12           | 4              | 1–4                          |
| 6–13           | 5              | 1–4                          |
| 7–14           | 6              | 1–4                          |
| 8–15           | 7              | 1–4                          |
| 9–16           | 8              | 1–4                          |

#### NOTE

Other combinations of Time Segment 1 and Time Segment 2 can be valid. It is the user's responsibility to ensure the bit time settings are in compliance with the CAN standard. For bit time calculations, use an IPT (Information Processing Time) of 2, which is the value implemented in the FlexCAN module.

### 25.5.9.5 Arbitration and matching timing

During normal transmission or reception of frames, the arbitration, matching, move-in, and move-out processes are executed during certain time windows inside the CAN frame, as shown in Figure 25-18.

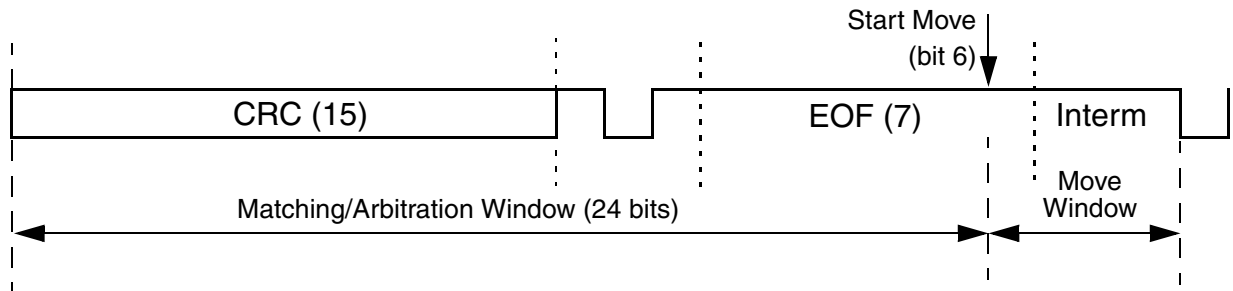


Figure 25-18. Arbitration, match, and move time windows

When doing matching and arbitration, FlexCAN needs to scan the whole Message Buffer memory during the available time slot. In order to have sufficient time to do that, the following requirements must be observed:

- A valid CAN bit timing must be programmed, as indicated in Table 25-22
- The peripheral clock frequency cannot be smaller than the oscillator clock frequency, i.e., the PLL cannot be programmed to divide down the oscillator clock
- There must be a minimum ratio between the peripheral clock frequency and the CAN bit rate, as specified in Table 25-23

Table 25-23. Minimum ratio between peripheral clock frequency and CAN bit rate

| Number of Message Buffers | Minimum ratio |
|---------------------------|---------------|
| 16                        | 8             |
| 32                        | 8             |
| 64                        | 16            |

A direct consequence of the first requirement is that the minimum number of time quanta per CAN bit must be 8, so the oscillator clock frequency should be at least 8 times the CAN bit rate. The minimum frequency ratio specified in Table 25-23 can be achieved by choosing a high enough peripheral clock frequency when compared to the oscillator clock frequency, or by adjusting one or more of the bit timing parameters (PRES DIV, PROPSEG, PSEG1, PSEG2). As an example, taking the case of 64 MBs, if the oscillator and peripheral clock frequencies are equal and the CAN bit timing is programmed to have 8 time quanta per bit, then the prescaler factor (PRES DIV + 1) should be at least 2. For prescaler factor equal to one and CAN bit timing with 8 time quanta per bit, the ratio between peripheral and oscillator clock frequencies should be at least 2.

## 25.5.10 Modes of operation details

### 25.5.10.1 Freeze mode

This mode is entered by asserting the HALT bit in MCR or when the MCU is put into Debug mode. In both cases it is also necessary that the FRZ bit is asserted in MCR and the module is not in a low-power mode

(Disable mode). When Freeze Mode is requested during transmission or reception, FlexCAN does the following:

- Waits to be in either Intermission, Passive Error, Bus Off, or Idle state
- Waits for all internal activities like arbitration, matching, move-in, and move-out to finish
- Ignores the Rx input pin and drives the Tx pin as recessive
- Stops the prescaler, thus halting all CAN protocol activities
- Grants write access to the Error Counters Register, which is read-only in other modes
- Sets the NOT\_RDY and FRZ\_ACK bits in MCR

After requesting Freeze Mode, the user must wait for the FRZ\_ACK bit to be asserted in MCR before executing any other action, otherwise FlexCAN may operate in an unpredictable way. In Freeze mode, all memory mapped registers are accessible.

Exiting Freeze Mode is done in one of the following ways:

- CPU negates the FRZ bit in MCR
- The MCU is removed from Debug mode and/or the HALT bit is negated

Once out of Freeze Mode, FlexCAN tries to resynchronize to the CAN bus by waiting for 11 consecutive recessive bits.

### 25.5.10.2 Module Disable mode

This low power mode is entered when the MCR[MDIS] bit is asserted. If the module is disabled during Freeze Mode, it requests to disable the clocks to the CAN Protocol Interface (CPI) and Message Buffer Management (MBM) submodules, sets the LPM\_ACK bit and negates the FRZ\_ACK bit. If the module is disabled during transmission or reception, FlexCAN does the following:

- Waits to be in either Idle or Bus Off state, or else waits for the third bit of Intermission and then checks it to be recessive
- Waits for all internal activities like arbitration, matching, move-in, and move-out to finish
- Ignores its Rx input pin and drives its Tx pin as recessive
- Shuts down the clocks to the CPI and MBM submodules
- Sets the NOT\_RDY and LPM\_ACK bits in MCR

The Bus Interface Unit continues to operate, enabling the CPU to access memory mapped registers, except the Free Running Timer, the Error Counter Register, and the Message Buffers, which cannot be accessed when the module is in Disable Mode. Exiting from this mode is done by negating the MDIS bit, which will resume the clocks and negate the LPM\_ACK bit.

### 25.5.11 Interrupts

The module can generate as many as 70 interrupt sources (64 interrupts due to message buffers and 6 interrupts due to ORed interrupts from MBs, Bus Off, Error, Tx Warning, Rx Warning, and Wake Up). The number of actual sources depends on the configured number of Message Buffers.

Each one of the message buffers can be an interrupt source, if its corresponding IMASK bit is set. There is no distinction between Tx and Rx interrupts for a particular buffer, under the assumption that the buffer is initialized for either transmission or reception. Each of the buffers has assigned a flag bit in the IFLAG registers. The bit is set when the corresponding buffer completes a successful transmission/reception and is cleared when the CPU writes it to 1 (unless another interrupt is generated at the same time).

### NOTE

It must be guaranteed that the CPU only clears the bit causing the current interrupt. For this reason, bit manipulation instructions (BSET) must not be used to clear interrupt flags. These instructions may cause accidental clearing of interrupt flags that are set after entering the current interrupt service routine.

If the Rx FIFO is enabled (bit FEN on MCR set), the interrupts corresponding to MBs 0 to 7 have a different behavior. Bit 7 of the IFLAG1 becomes the FIFO Overflow flag; bit 6 becomes the FIFO Warning flag, bit 5 becomes the Frames Available in FIFO flag, and bits 4–0 are unused. See [Section 25.4.4.12, Interrupt Flags 1 \(IFLAG1\) register](#) for more information.

A combined interrupt for all MBs is also generated by an Or of all the interrupt sources from MBs. This interrupt gets generated when any of the MBs generates an interrupt. In this case the CPU must read the IFLAG registers to determine which MB caused the interrupt.

The other 5 interrupt sources (Bus Off, Error, Tx Warning, Rx Warning, and Wake Up) generate interrupts like the MB ones, and can be read from the Error and Status Register. The Bus Off, Error, Tx Warning, and Rx Warning interrupt mask bits are located in CTRL, and the Wake-Up interrupt mask bit is located in MCR.

## 25.5.12 Bus interface

The CPU access to FlexCAN registers are subject to the following rules:

- Read and write access to supervisor registers in User Mode results in access error.
- Read and write access to unimplemented or reserved address space also results in access error. Any access to unimplemented MB or Rx Individual Mask Register locations results in access error. Any access to the Rx Individual Mask Register space when the BCC bit in MCR is negated results in access error.
- If MAXMB is programmed with a value smaller than the available number of MBs, then the unused memory space can be used as general purpose RAM space. Note that the Rx Individual Mask Registers can only be accessed in Freeze Mode, and this is still true for unused space within this memory. Note also that reserved words within RAM cannot be used. As an example, suppose FlexCAN is configured with 64 MBs and MAXMB is programmed with zero. The maximum number of MBs in this case becomes one. The MB memory starts at 0x0060, but the space from 0x0060 to 0x007F is reserved (for SMB usage), and the space from 0x0080 to 0x008F is used by the one MB. This leaves us with the available space from 0x0090 to 0x047F. The available memory in the Mask Registers space would be from 0x0884 to 0x097F.



**NOTE**

Unused MB space must not be used as general purpose RAM while FlexCAN is transmitting and receiving CAN frames.

## 25.6 Initialization/application information

This section provide instructions for initializing the FlexCAN module.

### 25.6.1 FlexCAN initialization sequence

The FlexCAN module may be reset in three ways:

- MCU level hard reset, which resets all memory mapped registers asynchronously
- MCU level soft reset, which resets some of the memory mapped registers synchronously (refer to [Table 25-2](#) to see what registers are affected by soft reset)
- SOFT\_RST bit in MCR, which has the same effect as the MCU level soft reset

Soft reset is synchronous and has to follow an internal request/acknowledge procedure across clock domains. Therefore, it may take some time to fully propagate its effects. The SOFT\_RST bit remains asserted while soft reset is pending, so software can poll this bit to know when the reset has completed. Also, soft reset cannot be applied while clocks are shut down in any of the low power modes. The low power mode should be exited and the clocks resumed before applying soft reset.

The clock source (CLK\_SRC bit) should be selected while the module is in Disable Mode. After the clock source is selected and the module is enabled (MDIS bit negated), FlexCAN automatically goes to Freeze Mode. In Freeze Mode, FlexCAN is unsynchronized from the CAN bus, the HALT and FRZ bits in MCR are set, the internal state machines are disabled, and the FRZ\_ACK and NOT\_RDY bits in MCR are set. The Tx pin is in recessive state and FlexCAN does not initiate any transmission or reception of CAN frames. Note that the Message Buffers and the Rx Individual Mask registers are not affected by reset, so they are not automatically initialized.

For any configuration change/initialization it is required that FlexCAN is put into Freeze Mode (see [Section 25.5.10.1, Freeze mode](#)). The following is a generic initialization sequence applicable to the FlexCAN module:

- Initialize MCR
  - Enable the individual filtering per MB and reception queue features by setting the BCC bit
  - Enable the warning interrupts by setting the WRN\_EN bit
  - If required, disable frame self reception by setting the SRX\_DIS bit
  - Enable the FIFO by setting the FEN bit
  - Enable the abort mechanism by setting the AEN bit
  - Enable the local priority feature by setting the LPRIO\_EN bit
- Initialize CTRL
  - Determine the bit timing parameters: PROPSEG, PSEG1, PSEG2, RJW
  - Determine the bit rate by programming the PRES DIV field

- Determine the internal arbitration mode (LBUF bit)
- Initialize the Message Buffers
  - The Control and Status word of all Message Buffers must be initialized
  - If FIFO was enabled, the 8-entry ID table must be initialized
  - Other entries in each Message Buffer should be initialized as required
- Initialize the Rx Individual Mask registers
- Set required interrupt mask bits in the IMASK registers (for all MB interrupts), in CTRL (for Bus Off and Error interrupts), and in MCR for Wake-Up interrupt
- Negate the HALT bit in MCR

Starting with the last event, FlexCAN attempts to synchronize to the CAN bus.

### 25.6.2 FlexCAN addressing and RAM size configurations

There are three RAM configurations that can be implemented within the FlexCAN module. The possible configurations are:

- For 16 MBs: 288 bytes for MB memory and 64 bytes for Individual Mask registers
- For 32 MBs: 544 bytes for MB memory and 128 bytes for Individual Mask registers
- For 64 MBs: 1056 bytes for MB memory and 256 bytes for Individual Mask registers

In each configuration the user can program the maximum number of MBs that will take part in the matching and arbitration processes using the MAXMB field in MCR. For 16 MB configuration, MAXMB can be any number between 0–15. For 32 MB configuration, MAXMB can be any number between 0–31. For 64 MB configuration, MAXMB can be any number between 0–63.

# Chapter 26

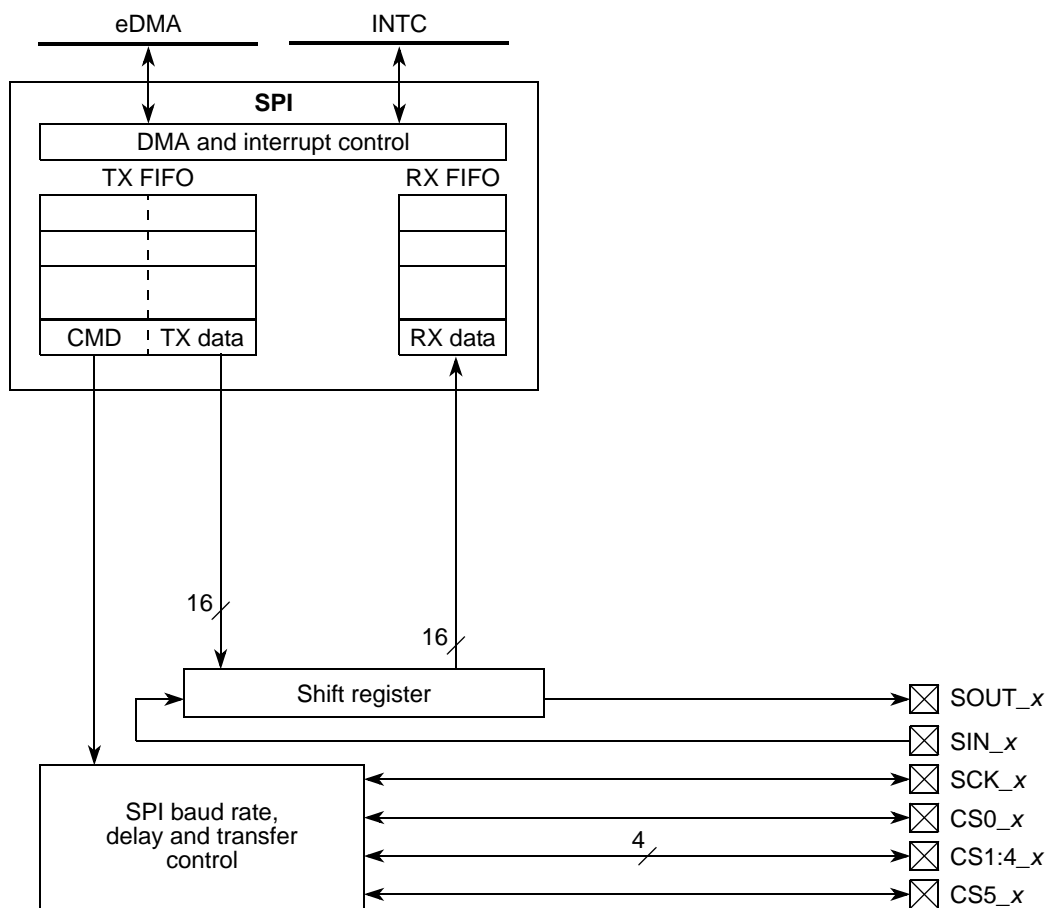
## Deserial Serial Peripheral Interface (DSPI)

### 26.1 Introduction

This chapter describes the Deserial Serial Peripheral Interface (DSPI), which provides a synchronous serial bus for communication between the MCU and an external peripheral device.

The MPC5606BK has six identical DSPI modules (DSPI\_0 – DSPI\_5). The “x” appended to signal names signifies the module to which the signal applies. Thus CS0\_x specifies that the CS0 signal applies to DSPI module 0, 1, etc.

A block diagram of the DSPI is shown in [Figure 26-1](#).



**Figure 26-1. DSPI block diagram**

The register content is transmitted using an SPI protocol.

For queued operations the SPI queues reside in internal SRAM that is external to the DSPI. Data transfers between the queues and the DSPI FIFOs are accomplished through the use of the eDMA controller or through host software.

Figure 26-2 shows a DSPI with external queues in internal SRAM.

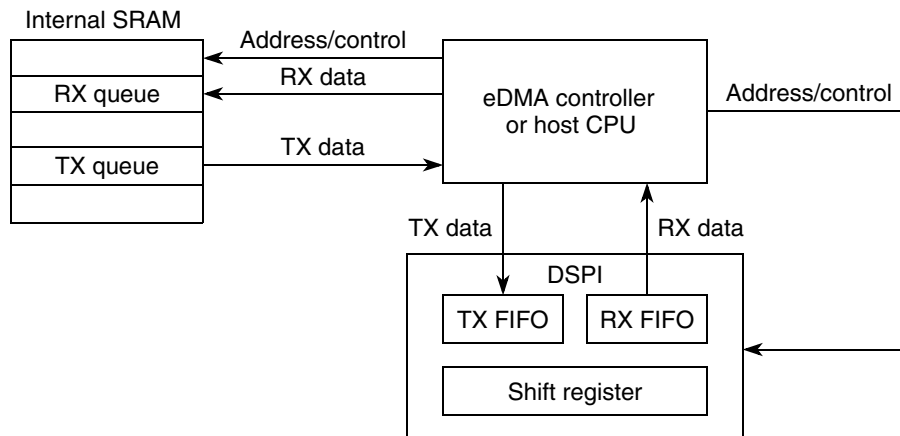


Figure 26-2. DSPI with queues and eDMA

## 26.2 Features

The DSPI supports these SPI features:

- Full-duplex, three-wire synchronous transfers
- Master and slave mode
- Buffered transmit and receive operation using the TX and RX FIFOs, with depths of four entries
- Visibility into TX and RX FIFOs for ease of debugging
- FIFO bypass mode for low-latency updates to SPI queues
- Programmable transfer attributes on a per-frame basis
  - Six clock and transfer attribute registers
  - Serial clock with programmable polarity and phase
  - Programmable delays
    - CS to SCK delay
    - SCK to CS delay
    - Delay between frames
  - Programmable serial frame size of 4 to 16 bits, expandable with software control
  - Continuously held chip select capability
- As many as six peripheral chip selects, expandable to 64 with external demultiplexer
- Deglitching support for as many as 32 peripheral chip selects with external demultiplexer
- Two DMA conditions for SPI queues residing in RAM or flash
  - TX FIFO is not full (TFFF)
  - RX FIFO is not empty (RFDF)
- Six interrupt conditions:

- End of queue reached (EOQF)
  - TX FIFO is not full (TFFF)
  - Transfer of current frame complete (TCF)
  - RX FIFO is not empty (RFDF)
  - FIFO overrun (attempt to transmit with an empty TX FIFO or serial frame received while RX FIFO is full) (RFOF) or (TFUF)
- Modified SPI transfer formats for communication with slower peripheral devices
  - Supports all functional modes from QSPI subblock of QSMCM (MPC500 family)
  - Continuous serial communications clock (SCK)

## 26.3 Modes of operation

The DSPI has five modes of operation. These modes can be divided into two categories:

- Module-specific: Master, Slave, and Module Disable modes
- MCU-specific: External Stop and Debug modes

The module-specific modes are entered by host software writing to a register. The MCU-specific modes are controlled by signals external to the DSPI. An MCU-specific mode is a mode that the entire device may enter, in parallel to the DSPI being in one of its module-specific modes.

### 26.3.1 Master mode

Master mode allows the DSPI to initiate and control serial communication. In this mode, the SCK, CS<sub>n</sub>, and SOUT signals are controlled by the DSPI and configured as outputs.

For more information, see [Section 26.6.1.1, Master mode](#).

### 26.3.2 Slave mode

Slave mode allows the DSPI to communicate with SPI bus masters. In this mode the DSPI responds to externally controlled serial transfers. The DSPI cannot initiate serial transfers in slave mode. In slave mode, the SCK signal and the CS<sub>0\_x</sub> signal are configured as inputs and provided by a bus master. CS<sub>0\_x</sub> must be configured as input and pulled high. If the internal pullup is being used then the appropriate bits in the relevant SIU\_PCR must be set (SIU\_PCR [WPE = 1], [WPS = 1]).

For more information, see [Section 26.6.1.2, Slave mode](#).

### 26.3.3 Module Disable mode

The module disable mode is used for MCU power management. The clock to the non-memory mapped logic in the DSPI is stopped while in module disable mode. The DSPI enters the module disable mode when the MDIS bit in DSPI<sub>x</sub>\_MCR is set.

For more information, see [Section 26.6.1.3, Module Disable mode](#).

## 26.3.4 Debug mode

Debug mode is used for system development and debugging. If the device enters debug mode while the FRZ bit in the DSPI<sub>x</sub>\_MCR is set, the DSPI halts operation on the next frame boundary. If the device enters debug mode while the FRZ bit is cleared, the DSPI behavior is unaffected and remains dictated by the module-specific mode and configuration of the DSPI.

For more information, see [Section 26.6.1.4, Debug mode](#).

## 26.4 External signal description

### 26.4.1 Signal overview

Table 26-1 lists off-chip DSPI signals.

Table 26-1. Signal properties

| Name    | I/O type       | Function  |                      |
|---------|----------------|---|----------------------|
|         |                | Master mode   | Slave mode           |
| CS0_x   | Output / input | Peripheral chip select 0                                    | Slave select         |
| CS1:3_x | Output         | Peripheral chip select 1–3                                  | Unused <sup>1</sup>  |
| CS4_x   | Output         | Peripheral chip select 4                                    | Master trigger       |
| CS5_x   | Output         | Peripheral chip select 5 /<br>Peripheral chip select strobe | Unused <sup>1</sup>  |
| SIN_x   | Input          | Serial data in  | Serial data in       |
| SOUT_x  | Output         | Serial data out   | Serial data out      |
| SCK_x   | Output / input | Serial clock (output)                                       | Serial clock (input) |

<sup>1</sup> The SIUL allows you to select alternate pin functions for the device.

### 26.4.2 Signal names and descriptions

#### 26.4.2.1 Peripheral Chip Select / Slave Select (CS0\_x)

In master mode, the CS0\_x signal is a peripheral chip select output that selects the slave device to which the current transmission is intended.

In slave mode, the CS0\_x signal is a slave select input signal that allows an SPI master to select the DSPI as the target for transmission. CS0\_x must be configured as input and pulled high. If the internal pullup is being used then the appropriate bits in the relevant SIU\_PCR must be set (SIU\_PCR [WPE = 1], [WPS = 1]).

Set the IBE and OBE bits in the SIU\_PCR for all CS0\_x pins when the DSPI chip select or slave select primary function is selected for that pin. When the pin is used for DSPI master mode as a chip select output, set the OBE bit. When the pin is used in DSPI slave mode as a slave select input, set the IBE bit.

### 26.4.2.2 Peripheral Chip Selects 1–3 (CS1:3\_x)

CS1:3\_x are peripheral chip select output signals in master mode. In slave mode these signals are not used.

### 26.4.2.3 Peripheral Chip Select 4 (CS4\_x)

CS4\_x is a peripheral chip select output signal in master mode.

### 26.4.2.4 Peripheral Chip Select 5 / Peripheral Chip Select Strobe (CS5\_x)

CS5\_x is a peripheral chip select output signal. When the DSPI is in master mode and PCSSE bit in the DSPIx\_MCR is cleared, the CS5\_x signal is used to select the slave device that receives the current transfer.

CS5\_x is a strobe signal used by external logic for deglitching of the CS signals. When the DSPI is in master mode and the PCSSE bit in the DSPIx\_MCR is set, the CS5\_x signal indicates the timing to decode CS0:4\_x signals, which prevents glitches from occurring.

CS5\_x is not used in slave mode.

### 26.4.2.5 Serial Input (SIN\_x)

SIN\_x is a serial data input signal.

### 26.4.2.6 Serial Output (SOUT\_x)

SOUT\_x is a serial data output signal.

### 26.4.2.7 Serial Clock (SCK\_x)

SCK\_x is a serial communication clock signal. In master mode, the DSPI generates the SCK. In slave mode, SCK\_x is an input from an external bus master.

## 26.5 Memory map and register description

### 26.5.1 Memory map

[Table 26-2](#) shows the DSPI memory map.

Table 26-2. DSPI memory map

| Base addresses:<br>0xFFFF9_0000 (DSPI_0)<br>0xFFFF9_4000 (DSPI_1)<br>0xFFFF9_8000 (DSPI_2)<br>0xFFFF9_C000 (DSPI_3)<br>0xFFFFA_0000 (DSPI_4)<br>0xFFFFA_4000 (DSPI_5) |  |                             |
|---|--|-----------------------------|
| Address offset  | Register   | Location                    |
| 0x00  | DSPI Module Configuration Register (DSPIx_MCR)                       | <a href="#">on page 598</a> |
| 0x04  | Reserved   |                             |
| 0x08  | DSPI Transfer Count Register (DSPIx_TCR)                             | <a href="#">on page 601</a> |
| 0x0C  | DSPI Clock and Transfer Attributes Register 0 (DSPIx_CTAR0)          | <a href="#">on page 602</a> |
| 0x10  | DSPI Clock and Transfer Attributes Register 1 (DSPIx_CTAR1)          | <a href="#">on page 602</a> |
| 0x14  | DSPI Clock and Transfer Attributes Register 2 (DSPIx_CTAR2)          | <a href="#">on page 602</a> |
| 0x18  | DSPI Clock and Transfer Attributes Register 3 (DSPIx_CTAR3)          | <a href="#">on page 602</a> |
| 0x1C  | DSPI Clock and Transfer Attributes Register 4 (DSPIx_CTAR4)          | <a href="#">on page 602</a> |
| 0x20  | DSPI Clock and Transfer Attributes Register 5 (DSPIx_CTAR5)          | <a href="#">on page 602</a> |
| 0x24–0x28   | Reserved   |                             |
| 0x2C  | DSPI Status Register (DSPIx_SR)                                      | <a href="#">on page 610</a> |
| 0x30  | DSPI DMA / Interrupt Request Select and Enable Register (DSPIx_RSER) | <a href="#">on page 612</a> |
| 0x34  | DSPI Push TX FIFO Register (DSPIx_PUSHR)                             | <a href="#">on page 614</a> |
| 0x38  | DSPI Pop RX FIFO Register (DSPIx_POPR)                               | <a href="#">on page 616</a> |
| 0x3C  | DSPI Transmit FIFO Register 0 (DSPIx_TXFR0)                          | <a href="#">on page 617</a> |
| 0x40  | DSPI Transmit FIFO Register 1 (DSPIx_TXFR1)                          | <a href="#">on page 617</a> |
| 0x44  | DSPI Transmit FIFO Register 2 (DSPIx_TXFR2)                          | <a href="#">on page 617</a> |
| 0x48  | DSPI Transmit FIFO Register 3 (DSPIx_TXFR3)                          | <a href="#">on page 617</a> |
| 0x4C–0x78   | Reserved   |                             |
| 0x7C  | DSPI Receive FIFO Register 0 (DSPIx_RXFR0)                           | <a href="#">on page 617</a> |
| 0x80  | DSPI Receive FIFO Register 1 (DSPIx_RXFR1)                           | <a href="#">on page 617</a> |
| 0x84  | DSPI Receive FIFO Register 2 (DSPIx_RXFR2)                           | <a href="#">on page 617</a> |
| 0x88  | DSPI Receive FIFO Register 3 (DSPIx_RXFR3)                           | <a href="#">on page 617</a> |

## 26.5.2 DSPI Module Configuration Register (DSPIx\_MCR)

The DSPIx\_MCR contains bits that configure attributes of the DSPI operation. The values of the HALT and MDIS bits can be changed at any time, but their effect begins on the next frame boundary. The HALT and MDIS bits in the DSPIx\_MCR are the only bit values software can change while the DSPI is running.



Offset: 0x00

Access: Read/write

|       |      |           |         |         |         |         |         |      |    |    |       |       |       |       |       |       |
|-------|------|-----------|---------|---------|---------|---------|---------|------|----|----|-------|-------|-------|-------|-------|-------|
|       | 0    | 1         | 2       | 3       | 4       | 5       | 6       | 7    | 8  | 9  | 10    | 11    | 12    | 13    | 14    | 15    |
| R     |      |           |         |         |         |         |         |      | 0  | 0  |       |       |       |       |       |       |
| W     | MSTR | CONT_SCKE | DCONF   |         | FRZ     | MTEF    | PCSSE   | ROOE |    |    | PCSI5 | PCSI4 | PCSI3 | PCSI2 | PCSI1 | PCSI0 |
| Reset | 0    | 0         | 0       | 0       | 0       | 0       | 0       | 0    | 0  | 0  | 0     | 0     | 0     | 0     | 0     | 0     |
|       | 16   | 17        | 18      | 19      | 20      | 21      | 22      | 23   | 24 | 25 | 26    | 27    | 28    | 29    | 30    | 31    |
| R     | 0    |           |         |         | 0       | 0       |         |      | 0  | 0  | 0     | 0     | 0     | 0     | 0     |       |
| W     |      | MDIS      | DIS_TXF | DIS_RXF | CLR_TXF | CLR_RXF | SMPL_PT |      |    |    |       |       |       |       |       | HALT  |
| Reset | 0    | 1         | 0       | 0       | 0       | 0       | 0       | 0    | 0  | 0  | 0     | 0     | 0     | 0     | 0     | 1     |

Figure 26-3. DSPI Module Configuration Register (DSPIx\_MCR)

Table 26-3. DSPIx\_MCR field descriptions

| Field     | Description  |       |               |    |     |    |               |    |               |    |               |
|-----------|--|-------|---------------|----|-----|----|---------------|----|---------------|----|---------------|
| MSTR      | Master/slave mode select<br>Configures the DSPI for master mode or slave mode.<br><br>0 DSPI is in slave mode<br>1 DSPI is in master mode  |       |               |    |     |    |               |    |               |    |               |
| CONT_SCKE | Continuous SCK enable<br>Enables the serial communication clock (SCK) to run continuously. See <a href="#">Section 26.6.6, Continuous serial communications clock</a> , for details.<br>0 Continuous SCK disabled<br><b>Note:</b> 1Continuous SCK enabled  |       |               |    |     |    |               |    |               |    |               |
| DCONF     | DSPI configuration<br>The following table lists the DCONF values for the various configurations. <table border="1" data-bbox="711 1402 1117 1640"> <thead> <tr> <th>DCONF</th> <th>Configuration</th> </tr> </thead> <tbody> <tr> <td>00</td> <td>SPI</td> </tr> <tr> <td>01</td> <td>Invalid value</td> </tr> <tr> <td>10</td> <td>Invalid value</td> </tr> <tr> <td>11</td> <td>Invalid value</td> </tr> </tbody> </table> | DCONF | Configuration | 00 | SPI | 01 | Invalid value | 10 | Invalid value | 11 | Invalid value |
| DCONF     | Configuration  |       |               |    |     |    |               |    |               |    |               |
| 00        | SPI  |       |               |    |     |    |               |    |               |    |               |
| 01        | Invalid value  |       |               |    |     |    |               |    |               |    |               |
| 10        | Invalid value  |       |               |    |     |    |               |    |               |    |               |
| 11        | Invalid value  |       |               |    |     |    |               |    |               |    |               |

Table 26-3. DSPIx\_MCR field descriptions (continued)

| Field      | Description  |
|------------|--|
| FRZ        | <p>Freeze<br/>Enables the DSPI transfers to be stopped on the next frame boundary when the device enters debug mode.</p> <p>0 Do not halt serial transfers<br/>1 Halt serial transfers</p>   |
| MTFE       | <p>Modified timing format enable<br/>Enables a modified transfer format to be used. See <a href="#">Section 26.6.5.4, Modified SPI transfer format (MTFE = 1, CPHA = 1)</a>, for more information.</p> <p>0 Modified SPI transfer format disabled<br/>1 Modified SPI transfer format enabled</p>   |
| PCSSE      | <p>Peripheral chip select strobe enable<br/>Enables the CS5_x to operate as a CS strobe output signal.<br/>See <a href="#">Section 26.6.4.5, Peripheral chip select strobe enable (CS5_x)</a>, for more information.</p> <p>0 CS5_x is used as the Peripheral chip select 5 signal<br/>1 CS5_x as an active-low CS strobe signal</p>   |
| ROOE       | <p>Receive FIFO overflow overwrite enable<br/>Enables an RX FIFO overflow condition to ignore the incoming serial data or to overwrite existing data. If the RX FIFO is full and new data is received, the data from the transfer that generated the overflow is ignored or put in the shift register.</p> <p>If the ROOE bit is set, the incoming data is put in the shift register. If the ROOE bit is cleared, the incoming data is ignored. See <a href="#">Section 26.6.7.6, Receive FIFO Overflow Interrupt Request (RFOF)</a>, for more information.</p> <p>0 Incoming data is ignored<br/>1 Incoming data is put in the shift register</p> |
| PCSI $S_n$ | <p>Peripheral chip select inactive state<br/>Determines the inactive state of the CS0_x signal. CS0_x must be configured as inactive high for slave mode operation.</p> <p>0 The inactive state of CS0_x is low<br/>1 The inactive state of CS0_x is high</p>  |
| MDIS       | <p>Module disable<br/>Allows the clock to stop to the non-memory mapped logic in the DSPI, effectively putting the DSPI in a software controlled power-saving state. See <a href="#">Section 26.6.8, Power saving features</a> for more information.</p> <p>0 Enable DSPI clocks<br/>1 Allow external logic to disable DSPI clocks</p>   |
| DIS_TXF    | <p>Disable transmit FIFO<br/>Enables and disables the TX FIFO. When the TX FIFO is disabled, the transmit part of the DSPI operates as a simplified double-buffered SPI. See <a href="#">Section 26.6.3.3, FIFO disable operation</a> for details.</p> <p>0 TX FIFO is enabled<br/>1 TX FIFO is disabled</p>   |

**Table 26-3. DSPIx\_MCR field descriptions (continued)**

| Field   | Description   |         |  |    |   |    |   |    |   |    |          |
|---------|---|---------|--|----|---|----|---|----|---|----|----------|
| DIS_RXF | <p>Disable receive FIFO</p> <p>Enables and disables the RX FIFO. When the RX FIFO is disabled, the receive part of the DSPI operates as a simplified double-buffered SPI. See <a href="#">Section 26.6.3.3, FIFO disable operation</a> for details.</p> <p>0 RX FIFO is enabled<br/>1 RX FIFO is disabled</p>   |         |  |    |   |    |   |    |   |    |          |
| CLR_TXF | <p>Clear TX FIFO. CLR_TXF is used to flush the TX FIFO. Writing a 1 to CLR_TXF clears the TX FIFO Counter. The CLR_TXF bit is always read as zero.</p> <p>0 Do not clear the TX FIFO Counter<br/>1 Clear the TX FIFO Counter</p>  |         |  |    |   |    |   |    |   |    |          |
| CLR_RXF | <p>Clear RX FIFO. CLR_RXF is used to flush the RX FIFO. Writing a 1 to CLR_RXF clears the RX Counter. The CLR_RXF bit is always read as zero.</p> <p>0 Do not clear the RX FIFO Counter<br/>1 Clear the RX FIFO Counter</p>   |         |  |    |   |    |   |    |   |    |          |
| SMPL_PT | <p>Sample point</p> <p>Allows the host software to select when the DSPI master samples SIN in modified transfer format. <a href="#">Figure 26-18</a> shows where the master can sample the SIN pin. The following table lists the delayed sample points.</p> <table border="1" data-bbox="542 894 1305 1161"> <thead> <tr> <th>SMPL_PT</th> <th>Number of system clock cycles between odd-numbered edge of SCK_x and sampling of SIN_x</th> </tr> </thead> <tbody> <tr> <td>00</td> <td>0</td> </tr> <tr> <td>01</td> <td>1</td> </tr> <tr> <td>10</td> <td>2</td> </tr> <tr> <td>11</td> <td>Reserved</td> </tr> </tbody> </table> | SMPL_PT | Number of system clock cycles between odd-numbered edge of SCK_x and sampling of SIN_x | 00 | 0 | 01 | 1 | 10 | 2 | 11 | Reserved |
| SMPL_PT | Number of system clock cycles between odd-numbered edge of SCK_x and sampling of SIN_x  |         |  |    |   |    |   |    |   |    |          |
| 00      | 0   |         |  |    |   |    |   |    |   |    |          |
| 01      | 1   |         |  |    |   |    |   |    |   |    |          |
| 10      | 2   |         |  |    |   |    |   |    |   |    |          |
| 11      | Reserved  |         |  |    |   |    |   |    |   |    |          |
| HALT    | <p>Halt</p> <p>Provides a mechanism for software to start and stop DSPI transfers. See <a href="#">Section 26.6.2, Start and stop of DSPI transfers</a>, for details on the operation of this bit.</p> <p>0 Start transfers<br/>1 Stop transfers</p>  |         |  |    |   |    |   |    |   |    |          |

### 26.5.3 DSPI Transfer Count Register (DSPIx\_TCR)

The DSPIx\_TCR contains a counter that indicates the number of SPI transfers made. The transfer counter is intended to assist in queue management. The user must not write to the DSPIx\_TCR while the DSPI is running.

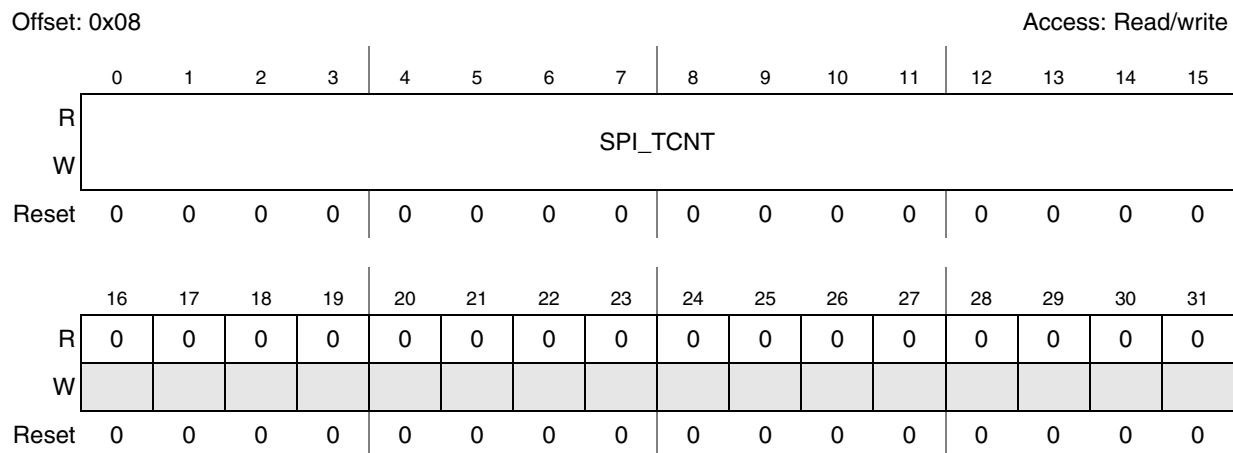


Figure 26-4. DSPI Transfer Count Register (DSPIx\_TCR)

Table 26-4. DSPIx\_TCR field descriptions

| Field    | Description   |
|----------|---|
| SPI_TCNT | <p>SPI transfer counter</p> <p>Counts the number of SPI transfers the DSPI makes. The SPI_TCNT field is incremented every time the last bit of an SPI frame is transmitted. A value written to SPI_TCNT presets the counter to that value. SPI_TCNT is reset to zero at the beginning of the frame when the CTCNT field is set in the executing SPI command. The transfer counter wraps around, incrementing the counter past 65535 resets the counter to zero.</p> |

## 26.5.4 DSPI Clock and Transfer Attributes Registers 0–5 (DSPIx\_CTAR $n$ )

The DSPI modules each contain six clock and transfer attribute registers (DSPIx\_CTAR $n$ ), which are used to define different transfer attribute configurations. Each DSPIx\_CTAR controls:

- Frame size
- Baud rate and transfer delay values
- Clock phase
- Clock polarity
- MSB or LSB first

DSPIx\_CTARs support compatibility with the QSPI module in the MPC5606BK family of MCUs. At the initiation of an SPI transfer, control logic selects the DSPIx\_CTAR that contains the transfer's attributes. Do not write to the DSPIx\_CTARs while the DSPI is running.

In master mode, the DSPIx\_CTAR $n$  registers define combinations of transfer attributes such as frame size, clock phase, and polarity, data bit ordering, baud rate, and various delays. In slave mode, a subset of the bit fields in the DSPIx\_CTAR0 and DSPIx\_CTAR1 registers are used to set the slave transfer attributes. See the individual bit descriptions for details on which bits are used in slave modes.

When the DSPI is configured as an SPI master, the CTAS field in the command portion of the TX FIFO entry selects which of the DSPIx\_CTAR registers is used on a per-frame basis. When the DSPI is configured as an SPI bus slave, the DSPIx\_CTAR0 register is used.

Offsets: 0x0C–0x20 (6 registers)

Access: Read/write

|       |       |    |    |    |      |    |    |    |      |      |       |        |    |      |    |     |   |     |   |   |
|-------|-------|----|----|----|------|----|----|----|------|------|-------|--------|----|------|----|-----|---|-----|---|---|
|       | 0     | 1  | 2  | 3  | 4    | 5  | 6  | 7  | 8    | 9    | 10    | 11     | 12 | 13   | 14 | 15  |   |     |   |   |
| R     | DBR   |    |    |    | FMSZ |    |    |    | CPOL | CPHA | LSBFE | PCSSCK |    | PASC |    | PDT |   | PBR |   |   |
| W     | DBR   |    |    |    | FMSZ |    |    |    | CPOL | CPHA | LSBFE | PCSSCK |    | PASC |    | PDT |   | PBR |   |   |
| Reset | 0     | 1  | 1  | 1  | 1    | 0  | 0  | 0  | 0    | 0    | 0     | 0      | 0  | 0    | 0  | 0   | 0 | 0   | 0 | 0 |
|       | 16    | 17 | 18 | 19 | 20   | 21 | 22 | 23 | 24   | 25   | 26    | 27     | 28 | 29   | 30 | 31  |   |     |   |   |
| R     | CSSCK |    |    |    | ASC  |    |    |    | DT   |      |       |        | BR |      |    |     |   |     |   |   |
| W     | CSSCK |    |    |    | ASC  |    |    |    | DT   |      |       |        | BR |      |    |     |   |     |   |   |
| Reset | 0     | 0  | 0  | 0  | 0    | 0  | 0  | 0  | 0    | 0    | 0     | 0      | 0  | 0    | 0  | 0   | 0 | 0   | 0 | 0 |

Figure 26-5. DSPI Clock and Transfer Attributes Registers 0–5 (DSPIx\_CTARn)

Table 26-5. DSPIx\_CTARn field descriptions

| Field | Descriptions   |
|-------|--|
| DBR   | <p>Double Baud Rate</p> <p>The DBR bit doubles the effective baud rate of the Serial Communications Clock (SCK). This field is only used in Master Mode. It effectively halves the Baud Rate division ratio supporting faster frequencies and odd division ratios for the Serial Communications Clock (SCK). When the DBR bit is set, the duty cycle of the Serial Communications Clock (SCK) depends on the value in the Baud Rate Prescaler and the Clock Phase bit as listed in <a href="#">Table 26-12</a>. See the BR[0:3] field description for details on how to compute the baud rate. If the overall baud rate is divide by two or divide by three of the system clock then neither the Continuous SCK Enable or the Modified Timing Format Enable bits should be set.</p> <p>0 The baud rate is computed normally with a 50/50 duty cycle<br/> 1 The baud rate is doubled with the duty cycle depending on the Baud Rate Prescaler</p> |
| FMSZ  | <p>Frame Size</p> <p>The FMSZ field selects the number of bits transferred per frame. The FMSZ field is used in Master Mode and Slave Mode. <a href="#">Table 26-13</a> lists the frame size encodings.</p>  |
| CPOL  | <p>Clock Polarity</p> <p>The CPOL bit selects the inactive state of the Serial Communications Clock (SCK). This bit is used in both Master and Slave Mode. For successful communication between serial devices, the devices must have identical clock polarities. When the Continuous Selection Format is selected, switching between clock polarities without stopping the DSPI can cause errors in the transfer due to the peripheral device interpreting the switch of clock polarity as a valid clock edge.</p> <p>0 The inactive state value of SCK is low<br/> 1 The inactive state value of SCK is high</p>   |
| CPHA  | <p>Clock Phase</p> <p>The CPHA bit selects which edge of SCK causes data to change and which edge causes data to be captured. This bit is used in both Master and Slave Mode. For successful communication between serial devices, the devices must have identical clock phase settings. Continuous SCK is only supported for CPHA = 1.</p> <p>0 Data is captured on the leading edge of SCK and changed on the following edge<br/> 1 Data is changed on the leading edge of SCK and captured on the following edge</p>  |

Table 26-5. DSPIx\_CTARn field descriptions (continued)

| Field  | Descriptions  |        |                                      |    |   |    |   |    |   |    |   |
|--------|---|--------|--------------------------------------|----|---|----|---|----|---|----|---|
| LSBFE  | <p>LSB First</p> <p>The LSBFE bit selects if the LSB or MSB of the frame is transferred first. This bit is only used in Master Mode.</p> <p>0 Data is transferred MSB first</p> <p>1 Data is transferred LSB first</p>  |        |                                      |    |   |    |   |    |   |    |   |
| PCSSCK | <p>PCS to SCK Delay Prescaler</p> <p>The PCSSCK field selects the prescaler value for the delay between assertion of PCS and the first edge of the SCK. This field is only used in Master Mode. The table below lists the prescaler values. See the CSSCK field description for details on how to compute the PCS to SCK delay.</p> <table border="1"> <thead> <tr> <th>PCSSCK</th> <th>PCS to SCK delay prescaler value</th> </tr> </thead> <tbody> <tr> <td>00</td> <td>1</td> </tr> <tr> <td>01</td> <td>3</td> </tr> <tr> <td>10</td> <td>5</td> </tr> <tr> <td>11</td> <td>7</td> </tr> </tbody> </table>  | PCSSCK | PCS to SCK delay prescaler value     | 00 | 1 | 01 | 3 | 10 | 5 | 11 | 7 |
| PCSSCK | PCS to SCK delay prescaler value  |        |                                      |    |   |    |   |    |   |    |   |
| 00     | 1   |        |                                      |    |   |    |   |    |   |    |   |
| 01     | 3   |        |                                      |    |   |    |   |    |   |    |   |
| 10     | 5   |        |                                      |    |   |    |   |    |   |    |   |
| 11     | 7   |        |                                      |    |   |    |   |    |   |    |   |
| PASC   | <p>After SCK Delay Prescaler</p> <p>The PASC field selects the prescaler value for the delay between the last edge of SCK and the negation of PCS. This field is only used in Master Mode. The table below lists the prescaler values. See the ASC[0:3] field description for details on how to compute the After SCK delay.</p> <table border="1"> <thead> <tr> <th>PASC</th> <th>After SCK delay prescaler value</th> </tr> </thead> <tbody> <tr> <td>00</td> <td>1</td> </tr> <tr> <td>01</td> <td>3</td> </tr> <tr> <td>10</td> <td>5</td> </tr> <tr> <td>11</td> <td>7</td> </tr> </tbody> </table>  | PASC   | After SCK delay prescaler value      | 00 | 1 | 01 | 3 | 10 | 5 | 11 | 7 |
| PASC   | After SCK delay prescaler value   |        |                                      |    |   |    |   |    |   |    |   |
| 00     | 1   |        |                                      |    |   |    |   |    |   |    |   |
| 01     | 3   |        |                                      |    |   |    |   |    |   |    |   |
| 10     | 5   |        |                                      |    |   |    |   |    |   |    |   |
| 11     | 7   |        |                                      |    |   |    |   |    |   |    |   |
| PDT    | <p>Delay after Transfer Prescaler</p> <p>The PDT field selects the prescaler value for the delay between the negation of the PCS signal at the end of a frame and the assertion of PCS at the beginning of the next frame. The PDT field is only used in Master Mode. The table below lists the prescaler values. See the DT[0:3] field description for details on how to compute the delay after transfer.</p> <table border="1"> <thead> <tr> <th>PDT</th> <th>Delay after transfer prescaler value</th> </tr> </thead> <tbody> <tr> <td>00</td> <td>1</td> </tr> <tr> <td>01</td> <td>3</td> </tr> <tr> <td>10</td> <td>5</td> </tr> <tr> <td>11</td> <td>7</td> </tr> </tbody> </table> | PDT    | Delay after transfer prescaler value | 00 | 1 | 01 | 3 | 10 | 5 | 11 | 7 |
| PDT    | Delay after transfer prescaler value  |        |                                      |    |   |    |   |    |   |    |   |
| 00     | 1   |        |                                      |    |   |    |   |    |   |    |   |
| 01     | 3   |        |                                      |    |   |    |   |    |   |    |   |
| 10     | 5   |        |                                      |    |   |    |   |    |   |    |   |
| 11     | 7   |        |                                      |    |   |    |   |    |   |    |   |

Table 26-5. DSPIx\_CTARn field descriptions (continued)

| Field | Descriptions  |     |                           |    |   |    |   |    |   |    |   |
|-------|---|-----|---------------------------|----|---|----|---|----|---|----|---|
| PBR   | <p>Baud Rate Prescaler</p> <p>The PBR field selects the prescaler value for the baud rate. This field is only used in Master Mode. The Baud Rate is the frequency of the Serial Communications Clock (SCK). The system clock is divided by the prescaler value before the baud rate selection takes place. The Baud Rate Prescaler values are listed in the table below. See the BR[0:3] field description for details on how to compute the baud rate.</p> <table border="1" data-bbox="558 451 1130 688"> <thead> <tr> <th>PBR</th> <th>Baud rate prescaler value</th> </tr> </thead> <tbody> <tr> <td>00</td> <td>2</td> </tr> <tr> <td>01</td> <td>3</td> </tr> <tr> <td>10</td> <td>5</td> </tr> <tr> <td>11</td> <td>7</td> </tr> </tbody> </table> | PBR | Baud rate prescaler value | 00 | 2 | 01 | 3 | 10 | 5 | 11 | 7 |
| PBR   | Baud rate prescaler value   |     |                           |    |   |    |   |    |   |    |   |
| 00    | 2   |     |                           |    |   |    |   |    |   |    |   |
| 01    | 3   |     |                           |    |   |    |   |    |   |    |   |
| 10    | 5   |     |                           |    |   |    |   |    |   |    |   |
| 11    | 7   |     |                           |    |   |    |   |    |   |    |   |
| CSSCK | <p>PCS to SCK Delay Scaler</p> <p>The CSSCK field selects the scaler value for the PCS to SCK delay. This field is only used in Master Mode. The PCS to SCK Delay is the delay between the assertion of PCS and the first edge of the SCK. <a href="#">Table 26-14</a> list the scaler values. The PCS to SCK Delay is a multiple of the system clock period and it is computed according to the following equation:</p> $t_{CSC} = \frac{1}{f_{SYS}} \times PCSSCK \times CSSCK \quad \text{Eqn. 26-1}$ <p>See <a href="#">Section 26.6.4.2, CS to SCK delay (tCSC)</a>, for more details.</p>   |     |                           |    |   |    |   |    |   |    |   |
| ASC   | <p>After SCK Delay Scaler</p> <p>The ASC field selects the scaler value for the After SCK Delay. This field is only used in Master Mode. The After SCK Delay is the delay between the last edge of SCK and the negation of PCS. <a href="#">Table 26-15</a> lists the scaler values. The After SCK Delay is a multiple of the system clock period, and it is computed according to the following equation:</p> $t_{ASC} = \frac{1}{f_{SYS}} \times PASC \times ASC \quad \text{Eqn. 26-2}$ <p>See <a href="#">Section 26.6.4.3, After SCK delay (tASC)</a>, for more details.</p>   |     |                           |    |   |    |   |    |   |    |   |
| DT    | <p>Delay after Transfer Scaler</p> <p>The DT field selects the Delay after Transfer Scaler. This field is only used in Master Mode. The Delay after Transfer is the time between the negation of the PCS signal at the end of a frame and the assertion of PCS at the beginning of the next frame. <a href="#">Table 26-16</a> lists the scaler values. In the Continuous Serial Communications Clock operation the DT value is fixed to one TSCK. The Delay after Transfer is a multiple of the system clock period and it is computed according to the following equation:</p> $t_{DT} = \frac{1}{f_{SYS}} \times PDT \times DT \quad \text{Eqn. 26-3}$ <p>See <a href="#">Section 26.6.4.4, Delay after transfer (tDT)</a>, for more details.</p>      |     |                           |    |   |    |   |    |   |    |   |

Table 26-5. DSPIx\_CTARn field descriptions (continued)

| Field | Descriptions   |
|-------|--|
| BR    | <p>Baud Rate Scaler</p> <p>The BR field selects the scaler value for the baud rate. This field is only used in Master Mode. The prescaled system clock is divided by the Baud Rate Scaler to generate the frequency of the SCK. Table 26-17 lists the Baud Rate Scaler values. The baud rate is computed according to the following equation:</p> $\text{SCK baud rate} = \frac{f_{\text{SYS}}}{\text{PBR}} \times \frac{1 + \text{DBR}}{\text{BR}} \quad \text{Eqn. 26-4}$ <p>See Section 26.6.4.2, CS to SCK delay (tCSC), for more details.</p> |

Table 26-6. DSPI SCK duty cycle

| DBR | CPHA | PBR | SCK duty cycle |
|-----|------|-----|----------------|
| 0   | any  | any | 50/50          |
| 1   | 0    | 00  | 50/50          |
| 1   | 0    | 01  | 33/66          |
| 1   | 0    | 10  | 40/60          |
| 1   | 0    | 11  | 43/57          |
| 1   | 1    | 00  | 50/50          |
| 1   | 1    | 01  | 66/33          |
| 1   | 1    | 10  | 60/40          |
| 1   | 1    | 11  | 57/43          |

Table 26-7. DSPI transfer frame size

| FMSZ | Frame size | FMSZ | Frame size |
|------|------------|------|------------|
| 0000 | Reserved   | 1000 | 9          |
| 0001 | Reserved   | 1001 | 10         |
| 0010 | Reserved   | 1010 | 11         |
| 0011 | 4          | 1011 | 12         |
| 0100 | 5          | 1100 | 13         |
| 0101 | 6          | 1101 | 14         |
| 0110 | 7          | 1110 | 15         |
| 0111 | 8          | 1111 | 16         |



Table 26-8. DSPI PCS to SCK delay scaler

| CSSCK | PCS to SCK delay scaler value | CSSCK | PCS to SCK delay scaler value |
|-------|-------------------------------|-------|-------------------------------|
| 0000  | 2                             | 1000  | 512                           |
| 0001  | 4                             | 1001  | 1024                          |
| 0010  | 8                             | 1010  | 2048                          |
| 0011  | 16                            | 1011  | 4096                          |
| 0100  | 32                            | 1100  | 8192                          |
| 0101  | 64                            | 1101  | 16384                         |
| 0110  | 128                           | 1110  | 32768                         |
| 0111  | 256                           | 1111  | 65536                         |

Table 26-9. DSPI After SCK delay scaler

| ASC  | After SCK delay scaler value | ASC  | After SCK delay scaler value |
|------|------------------------------|------|------------------------------|
| 0000 | 2                            | 1000 | 512                          |
| 0001 | 4                            | 1001 | 1024                         |
| 0010 | 8                            | 1010 | 2048                         |
| 0011 | 16                           | 1011 | 4096                         |
| 0100 | 32                           | 1100 | 8192                         |
| 0101 | 64                           | 1101 | 16384                        |
| 0110 | 128                          | 1110 | 32768                        |
| 0111 | 256                          | 1111 | 65536                        |

Table 26-10. DSPI delay after transfer scaler

| DT   | Delay after transfer scaler value | DT   | Delay after transfer scaler value |
|------|-----------------------------------|------|-----------------------------------|
| 0000 | 2                                 | 1000 | 512                               |
| 0001 | 4                                 | 1001 | 1024                              |
| 0010 | 8                                 | 1010 | 2048                              |
| 0011 | 16                                | 1011 | 4096                              |
| 0100 | 32                                | 1100 | 8192                              |
| 0101 | 64                                | 1101 | 16384                             |
| 0110 | 128                               | 1110 | 32768                             |
| 0111 | 256                               | 1111 | 65536                             |

Table 26-11. DSPI baud rate scaler

| BR   | Baud rate scaler value | BR   | Baud rate scaler value |
|------|------------------------|------|------------------------|
| 0000 | 2                      | 1000 | 256                    |
| 0001 | 4                      | 1001 | 512                    |
| 0010 | 6                      | 1010 | 1024                   |
| 0011 | 8                      | 1011 | 2048                   |
| 0100 | 16                     | 1100 | 4096                   |
| 0101 | 32                     | 1101 | 8192                   |
| 0110 | 64                     | 1110 | 16384                  |
| 0111 | 128                    | 1111 | 32768                  |

Table 26-12. DSPI SCK duty cycle

| DBR | CPHA | PBR | SCK duty cycle |
|-----|------|-----|----------------|
| 0   | any  | any | 50/50          |
| 1   | 0    | 00  | 50/50          |
| 1   | 0    | 01  | 33/66          |
| 1   | 0    | 10  | 40/60          |
| 1   | 0    | 11  | 43/57          |
| 1   | 1    | 00  | 50/50          |
| 1   | 1    | 01  | 66/33          |
| 1   | 1    | 10  | 60/40          |
| 1   | 1    | 11  | 57/43          |

Table 26-13. DSPI transfer frame size

| FMSZ | Frame size | FMSZ | Frame size |
|------|------------|------|------------|
| 0000 | Reserved   | 1000 | 9          |
| 0001 | Reserved   | 1001 | 10         |
| 0010 | Reserved   | 1010 | 11         |
| 0011 | 4          | 1011 | 12         |
| 0100 | 5          | 1100 | 13         |
| 0101 | 6          | 1101 | 14         |
| 0110 | 7          | 1110 | 15         |
| 0111 | 8          | 1111 | 16         |

Table 26-14. DSPI PCS to SCK delay scaler

| CSSCK | PCS to SCK delay scaler value | CSSCK | PCS to SCK delay scaler value |
|-------|-------------------------------|-------|-------------------------------|
| 0000  | 2                             | 1000  | 512                           |
| 0001  | 4                             | 1001  | 1024                          |
| 0010  | 8                             | 1010  | 2048                          |
| 0011  | 16                            | 1011  | 4096                          |
| 0100  | 32                            | 1100  | 8192                          |
| 0101  | 64                            | 1101  | 16384                         |
| 0110  | 128                           | 1110  | 32768                         |
| 0111  | 256                           | 1111  | 65536                         |

Table 26-15. DSPI After SCK delay scaler

| ASC  | After SCK delay scaler value | ASC  | After SCK delay scaler value |
|------|------------------------------|------|------------------------------|
| 0000 | 2                            | 1000 | 512                          |
| 0001 | 4                            | 1001 | 1024                         |
| 0010 | 8                            | 1010 | 2048                         |
| 0011 | 16                           | 1011 | 4096                         |
| 0100 | 32                           | 1100 | 8192                         |
| 0101 | 64                           | 1101 | 16384                        |
| 0110 | 128                          | 1110 | 32768                        |
| 0111 | 256                          | 1111 | 65536                        |

Table 26-16. DSPI delay after transfer scaler

| DT   | Delay after transfer scaler value | DT   | Delay after transfer scaler value |
|------|-----------------------------------|------|-----------------------------------|
| 0000 | 2                                 | 1000 | 512                               |
| 0001 | 4                                 | 1001 | 1024                              |
| 0010 | 8                                 | 1010 | 2048                              |
| 0011 | 16                                | 1011 | 4096                              |
| 0100 | 32                                | 1100 | 8192                              |
| 0101 | 64                                | 1101 | 16384                             |
| 0110 | 128                               | 1110 | 32768                             |
| 0111 | 256                               | 1111 | 65536                             |

Table 26-17. DSPI baud rate scaler

| BR   | Baud rate scaler value | BR   | Baud rate scaler value |
|------|------------------------|------|------------------------|
| 0000 | 2                      | 1000 | 256                    |
| 0001 | 4                      | 1001 | 512                    |
| 0010 | 6                      | 1010 | 1024                   |
| 0011 | 8                      | 1011 | 2048                   |
| 0100 | 16                     | 1100 | 4096                   |
| 0101 | 32                     | 1101 | 8192                   |
| 0110 | 64                     | 1110 | 16384                  |
| 0111 | 128                    | 1111 | 32768                  |

### 26.5.5 DSPI Status Register (DSPIx\_SR)

The DSPIx\_SR contains status and flag bits. The bits are set by the hardware and reflect the status of the DSPI and indicate the occurrence of events that can generate interrupt or DMA requests. Software can clear flag bits in the DSPIx\_SR by writing a 1 to clear it (w1c). Writing a 0 to a flag bit has no effect. This register may not be writable in Module Disable mode due to the use of power saving mechanisms.

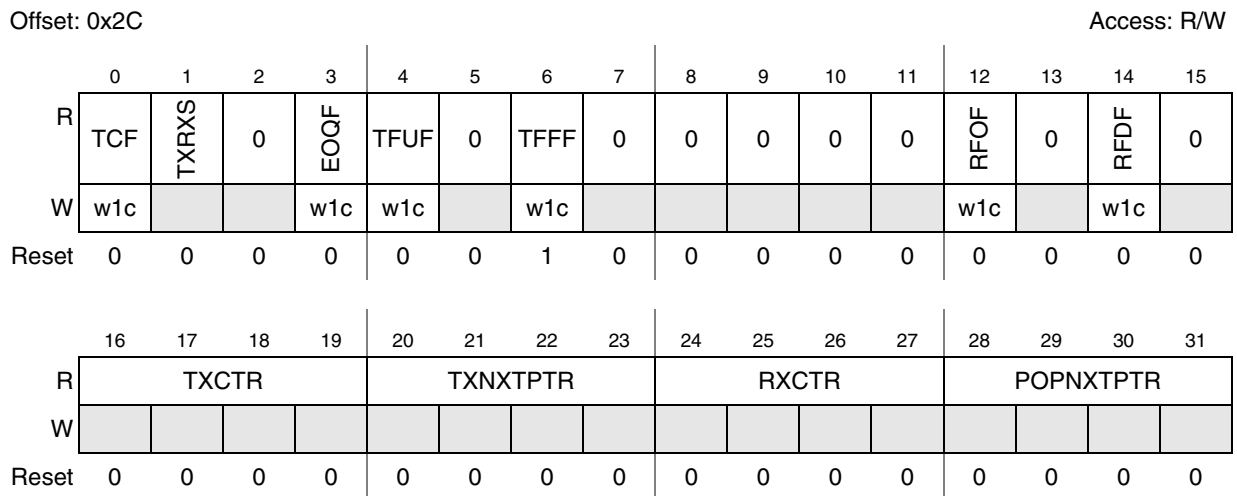


Figure 26-6. DSPI Status Register (DSPIx\_SR)

Table 26-18. DSPIx\_SR field descriptions

| Field | Description  |
|-------|--|
| TCF   | <p>Transfer complete flag<br/>Indicates that all bits in a frame have been shifted out. The TCF bit is set after the last incoming data bit is sampled, but before the <math>t_{ASC}</math> delay starts. See <a href="#">Section 26.6.5.1, Classic SPI transfer format (CPHA = 0)</a> for details.</p> <p>0 Transfer not complete<br/>1 Transfer complete</p>   |
| TXRXS | <p>TX and RX status<br/>Reflects the status of the DSPI. See <a href="#">Section 26.6.2, Start and stop of DSPI transfers</a> for information on what clears and sets this bit.</p> <p>0 TX and RX operations are disabled (DSPI is in STOPPED state)<br/>1 TX and RX operations are enabled (DSPI is in RUNNING state)</p>  |
| EOQF  | <p>End of queue flag<br/>Indicates that transmission in progress is the last entry in a queue. The EOQF bit is set when TX FIFO entry has the EOQ bit set in the command halfword and the end of the transfer is reached. See <a href="#">Section 26.6.5.1, Classic SPI transfer format (CPHA = 0)</a> for details.</p> <p>When the EOQF bit is set, the TXRXS bit is automatically cleared.</p> <p>0 EOQ is not set in the executing command<br/>1 EOQ bit is set in the executing SPI command<br/><b>Note:</b> EOQF does not function in slave mode.</p> |
| TFUF  | <p>Transmit FIFO underflow flag<br/>Indicates that an underflow condition in the TX FIFO has occurred. The transmit underflow condition is detected only for DSPI modules operating in slave mode and SPI configuration. The TFUF bit is set when the TX FIFO of a DSPI operating in SPI slave mode is empty, and a transfer is initiated by an external SPI master.</p> <p>0 TX FIFO underflow has not occurred<br/>1 TX FIFO underflow has occurred</p>  |
| TFFF  | <p>Transmit FIFO fill flag<br/>Indicates that the TX FIFO can be filled. Provides a method for the DSPI to request more entries to be added to the TX FIFO. The TFFF bit is set while the TX FIFO is not full. The TFFF bit can be cleared by writing 1 to it, or an by acknowledgement from the Edam controller when the TX FIFO is full.</p> <p>0 TX FIFO is full<br/>1 TX FIFO is not full</p>  |
| RFOF  | <p>Receive FIFO overflow flag<br/>Indicates that an overflow condition in the RX FIFO has occurred. The bit is set when the RX FIFO and shift register are full and a transfer is initiated.</p> <p>0 RX FIFO overflow has not occurred<br/>1 RX FIFO overflow has occurred</p>  |

Table 26-18. DSPIx\_SR field descriptions (continued)

| Field     | Description   |
|-----------|---|
| RFDF      | <p>Receive FIFO drain flag</p> <p>Indicates that the RX FIFO can be drained. Provides a method for the DSPI to request that entries be removed from the RX FIFO. The bit is set while the RX FIFO is not empty. The RFDF bit can be cleared by writing 1 to it, or by acknowledgement from the Edam controller when the RX FIFO is empty.</p> <p>0 RX FIFO is empty<br/>1 RX FIFO is not empty</p> <p><b>Note:</b> In the interrupt service routine, RFDF must be cleared only after the DSPIx_POPR register is read.</p> |
| TXCTR     | <p>TX FIFO counter</p> <p>Indicates the number of valid entries in the TX FIFO. The TXCTR is incremented every time the DSPI_PUSH register is written. The TXCTR is decremented every time an SPI command is executed and the SPI data is transferred to the shift register.</p>  |
| TXNXTPTR  | <p>Transmit next pointer</p> <p>Indicates which TX FIFO entry is transmitted during the next transfer. The TXNXTPTR field is updated every time SPI data is transferred from the TX FIFO to the shift register. See <a href="#">Section 26.6.3.4, Transmit First In First Out (TX FIFO) buffering mechanism</a> for more details.</p>   |
| RXCTR     | <p>RX FIFO counter</p> <p>Indicates the number of entries in the RX FIFO. The RXCTR is decremented every time the DSPI_POPR register is read. The RXCTR is incremented after the last incoming data bit is sampled, but before the <math>t_{ASC}</math> delay starts. See <a href="#">Section 26.6.5.1, Classic SPI transfer format (CPHA = 0)</a> for details.</p>   |
| POPNXTPTR | <p>Pop next pointer</p> <p>Contains a pointer to the RX FIFO entry that is returned when the DSPIx_POPR is read. The POPNXTPTR is updated when the DSPIx_POPR is read. See <a href="#">Section 26.6.3.5, Receive First In First Out (RX FIFO) buffering mechanism</a> for more details.</p>   |

### 26.5.6 DSPI DMA / Interrupt Request Select and Enable Register (DSPIx\_RSER)

The DSPIx\_RSER serves two purposes:

- It enables flag bits in the DSPIx\_SR to generate DMA requests or interrupt requests.
- It selects the type of request to generate.

See the bit descriptions for the type of requests that are supported.

Do not write to the DSPIx\_RSER while the DSPI is running.

Offset:0x30

Access: Read/write

|       |        |    |    |         |         |    |         |           |    |    |    |    |         |    |         |           |
|-------|--------|----|----|---------|---------|----|---------|-----------|----|----|----|----|---------|----|---------|-----------|
|       | 0      | 1  | 2  | 3       | 4       | 5  | 6       | 7         | 8  | 9  | 10 | 11 | 12      | 13 | 14      | 15        |
| R     |        | 0  | 0  | EOQF_RE | TFUF_RE | 0  | TFFF_RE | TFFF_DIRS | 0  | 0  | 0  | 0  | RFOF_RE | 0  | RDFE_RE | RDFE_DIRS |
| W     | TCF_RE |    |    |         |         |    |         |           |    |    |    |    |         |    |         |           |
| Reset | 0      | 0  | 0  | 0       | 0       | 0  | 0       | 0         | 0  | 0  | 0  | 0  | 0       | 0  | 0       | 0         |
|       | 16     | 17 | 18 | 19      | 20      | 21 | 22      | 23        | 24 | 25 | 26 | 27 | 28      | 29 | 30      | 31        |
| R     | 0      | 0  | 0  | 0       | 0       | 0  | 0       | 0         | 0  | 0  | 0  | 0  | 0       | 0  | 0       | 0         |
| W     |        |    |    |         |         |    |         |           |    |    |    |    |         |    |         |           |
| Reset | 0      | 0  | 0  | 0       | 0       | 0  | 0       | 0         | 0  | 0  | 0  | 0  | 0       | 0  | 0       | 0         |

Figure 26-7. DSPI DMA / Interrupt Request Select and Enable Register (DSPIx\_RSER)

Table 26-19. DSPIx\_RSER field descriptions

| Field     | Description   |
|-----------|---|
| TCF_RE    | Transmission complete request enable<br>Enables TCF flag in the DSPIx_SR to generate an interrupt request.<br><br>0 TCF interrupt requests are disabled<br>1 TCF interrupt requests are enabled   |
| EOQF_RE   | DSPI finished request enable<br>Enables the EOQF flag in the DSPIx_SR to generate an interrupt request.<br><br>0 EOQF interrupt requests are disabled<br>1 EOQF interrupt requests are enabled  |
| TFUF_RE   | Transmit FIFO underflow request enable<br>The TFUF_RE bit enables the TFUF flag in the DSPIx_SR to generate an interrupt request.<br><br>0 TFUF interrupt requests are disabled<br>1 TFUF interrupt requests are enabled  |
| TFFF_RE   | Transmit FIFO fill request enable<br>Enables the TFFF flag in the DSPIx_SR to generate a request. The TFFF_DIRS bit selects between generating an interrupt request or a DMA requests.<br><br>0 TFFF interrupt requests or DMA requests are disabled<br>1 TFFF interrupt requests or DMA requests are enabled   |
| TFFF_DIRS | Transmit FIFO fill DMA or interrupt request select<br>Selects between generating a DMA request or an interrupt request. When the TFFF flag bit in the DSPIx_SR is set, and the TFFF_RE bit in the DSPIx_RSER is set, this bit selects between generating an interrupt request or a DMA request.<br><br>0 Interrupt request is selected<br>1 DMA request is selected |

**Table 26-19. DSPIx\_RSER field descriptions (continued)**

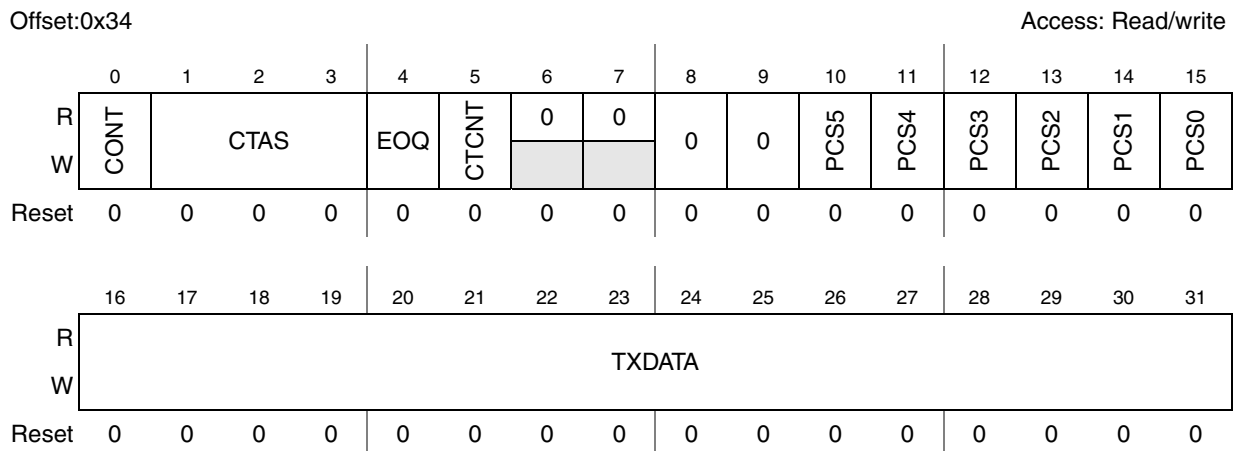
| Field     | Description  |
|-----------|--|
| RFOF_RE   | Receive FIFO overflow request enable<br>Enables the RFOF flag in the DSPIx_SR to generate an interrupt requests.<br><br>0 RFOF interrupt requests are disabled<br>1 RFOF interrupt requests are enabled  |
| RFDF_RE   | Receive FIFO drain request enable<br>Enables the RFDF flag in the DSPIx_SR to generate a request. The RFDF_DIRS bit selects between generating an interrupt request or a DMA request.<br><br>0 RFDF interrupt requests or DMA requests are disabled<br>1 RFDF interrupt requests or DMA requests are enabled   |
| RFDF_DIRS | Receive FIFO drain DMA or interrupt request select<br>Selects between generating a DMA request or an interrupt request. When the RFDF flag bit in the DSPIx_SR is set, and the RFDF_RE bit in the DSPIx_RSER is set, the RFDF_DIRS bit selects between generating an interrupt request or a DMA request.<br><br>0 Interrupt request is selected<br>1 DMA request is selected |

### 26.5.7 DSPI PUSH TX FIFO Register (DSPIx\_PUSHR)

The DSPIx\_PUSHR provides a means to write to the TX FIFO. Data written to this register is transferred to the TX FIFO. See [Section 26.6.3.4, Transmit First In First Out \(TX FIFO\) buffering mechanism](#), for more information. Write accesses of 8 or 16 bits to the DSPIx\_PUSHR transfers 32 bits to the TX FIFO.

**NOTE**

TXDATA is used in master and slave modes.



**Figure 26-8. DSPI PUSH TX FIFO Register (DSPIx\_PUSHR)**



Table 26-20. DSPIx\_PUSHR field descriptions

| Field | Description  |      |  |     |             |     |             |     |             |     |             |     |             |     |             |     |          |     |          |
|-------|--|------|--|-----|-------------|-----|-------------|-----|-------------|-----|-------------|-----|-------------|-----|-------------|-----|----------|-----|----------|
| CONT  | <p>Continuous peripheral chip select enable<br/>Selects a continuous selection format. The bit is used in SPI master mode. The bit enables the selected CS signals to remain asserted between transfers. See <a href="#">Section 26.6.5.5, Continuous selection format</a>, for more information.</p> <p>0 Return peripheral chip select signals to their inactive state between transfers<br/>1 Keep peripheral chip select signals asserted between transfers</p>  |      |  |     |             |     |             |     |             |     |             |     |             |     |             |     |          |     |          |
| CTAS  | <p>Clock and transfer attributes select<br/>Selects which of the DSPIx_CTARs is used to set the transfer attributes for the SPI frame. In SPI slave mode, DSPIx_CTAR0 is used. The following table shows how the CTAS values map to the DSPIx_CTARs. There are eight DSPIx_CTARs in the device DSPI implementation.<br/><b>Note:</b> Use in SPI master mode only.</p> <table border="1"> <thead> <tr> <th>CTAS</th> <th>Use clock and transfer attributes from</th> </tr> </thead> <tbody> <tr> <td>000</td> <td>DSPIx_CTAR0</td> </tr> <tr> <td>001</td> <td>DSPIx_CTAR1</td> </tr> <tr> <td>010</td> <td>DSPIx_CTAR2</td> </tr> <tr> <td>011</td> <td>DSPIx_CTAR3</td> </tr> <tr> <td>100</td> <td>DSPIx_CTAR4</td> </tr> <tr> <td>101</td> <td>DSPIx_CTAR5</td> </tr> <tr> <td>110</td> <td>Reserved</td> </tr> <tr> <td>111</td> <td>Reserved</td> </tr> </tbody> </table> | CTAS | Use clock and transfer attributes from | 000 | DSPIx_CTAR0 | 001 | DSPIx_CTAR1 | 010 | DSPIx_CTAR2 | 011 | DSPIx_CTAR3 | 100 | DSPIx_CTAR4 | 101 | DSPIx_CTAR5 | 110 | Reserved | 111 | Reserved |
| CTAS  | Use clock and transfer attributes from   |      |  |     |             |     |             |     |             |     |             |     |             |     |             |     |          |     |          |
| 000   | DSPIx_CTAR0  |      |  |     |             |     |             |     |             |     |             |     |             |     |             |     |          |     |          |
| 001   | DSPIx_CTAR1  |      |  |     |             |     |             |     |             |     |             |     |             |     |             |     |          |     |          |
| 010   | DSPIx_CTAR2  |      |  |     |             |     |             |     |             |     |             |     |             |     |             |     |          |     |          |
| 011   | DSPIx_CTAR3  |      |  |     |             |     |             |     |             |     |             |     |             |     |             |     |          |     |          |
| 100   | DSPIx_CTAR4  |      |  |     |             |     |             |     |             |     |             |     |             |     |             |     |          |     |          |
| 101   | DSPIx_CTAR5  |      |  |     |             |     |             |     |             |     |             |     |             |     |             |     |          |     |          |
| 110   | Reserved   |      |  |     |             |     |             |     |             |     |             |     |             |     |             |     |          |     |          |
| 111   | Reserved   |      |  |     |             |     |             |     |             |     |             |     |             |     |             |     |          |     |          |
| EOQ   | <p>End of queue<br/>Provides a means for host software to signal to the DSPI that the current SPI transfer is the last in a queue. At the end of the transfer the EOQF bit in the DSPIx_SR is set.</p> <p>0 The SPI data is not the last data to transfer<br/>1 The SPI data is the last data to transfer<br/><b>Note:</b> Use in SPI master mode only.</p>  |      |  |     |             |     |             |     |             |     |             |     |             |     |             |     |          |     |          |
| CTCNT | <p>Clear SPI_TCNT<br/>Provides a means for host software to clear the SPI transfer counter. The CTCNT bit clears the SPI_TCNT field in the DSPIx_TCR. The SPI_TCNT field is cleared before transmission of the current SPI frame begins.</p> <p>0 Do not clear SPI_TCNT field in the DSPIx_TCR<br/>1 Clear SPI_TCNT field in the DSPIx_TCR<br/><b>Note:</b> Use in SPI master mode only.</p>   |      |  |     |             |     |             |     |             |     |             |     |             |     |             |     |          |     |          |

**Table 26-20. DSPIx\_PUSHR field descriptions (continued)**

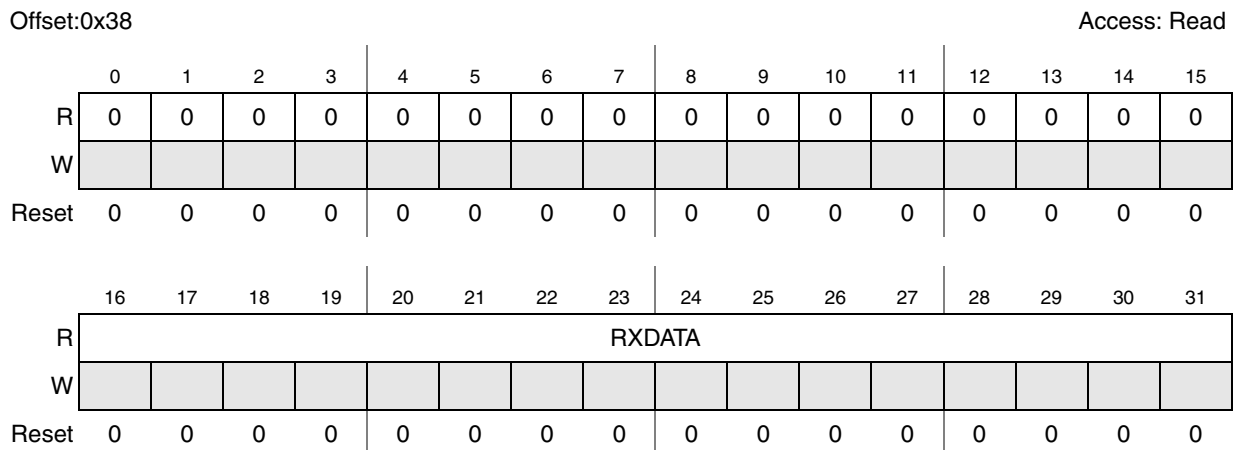
| Field  | Description   |
|--------|---|
| PCSx   | Peripheral chip select x<br>Selects which CSx signals are asserted for the transfer.<br><br>0 Negate the CSx signal<br>1 Assert the CSx signal<br><b>Note:</b> Use in SPI master mode only. |
| TXDATA | Transmit data<br>Holds SPI data for transfer according to the associated SPI command.<br><br><b>Note:</b> Use TXDATA in master and slave modes.   |

### 26.5.8 DSPI POP RX FIFO Register (DSPIx\_POPR)

The DSPIx\_POPR allows you to read the RX FIFO. See [Section 26.6.3.5, Receive First In First Out \(RX FIFO\) buffering mechanism](#) for a description of the RX FIFO operations. Eight- or 16-bit read accesses to the DSPIx\_POPR fetch the RX FIFO data, and update the counter and pointer.

**NOTE**

Reading the RX FIFO field fetches data from the RX FIFO. Once the RX FIFO is read, the read data pointer is moved to the next entry in the RX FIFO. Therefore, read DSPIx\_POPR only when you need the data. For compatibility, configure the TLB entry for DSPIx\_POPR as guarded.



**Figure 26-9. DSPI POP RX FIFO Register (DSPIx\_POPR)**

**Table 26-21. DSPIx\_POPR field descriptions**

| Field  | Description  |
|--------|--|
| RXDATA | Received data<br>The RXDATA field contains the SPI data from the RX FIFO entry pointed to by the pop next data pointer (POPNEXTPTR). |

## 26.5.9 DSPI Transmit FIFO Registers 0–3 (DSPIx\_TXFRn)

The DSPIx\_TXFRn registers provide visibility into the TX FIFO for debugging purposes. Each register is an entry in the TX FIFO. The registers are read-only and cannot be modified. Reading the DSPIx\_TXFRn registers does not alter the state of the TX FIFO. The MCU uses four registers to implement the TX FIFO, that is DSPIx\_TXFR0–DSPIx\_TXFR3 are used.

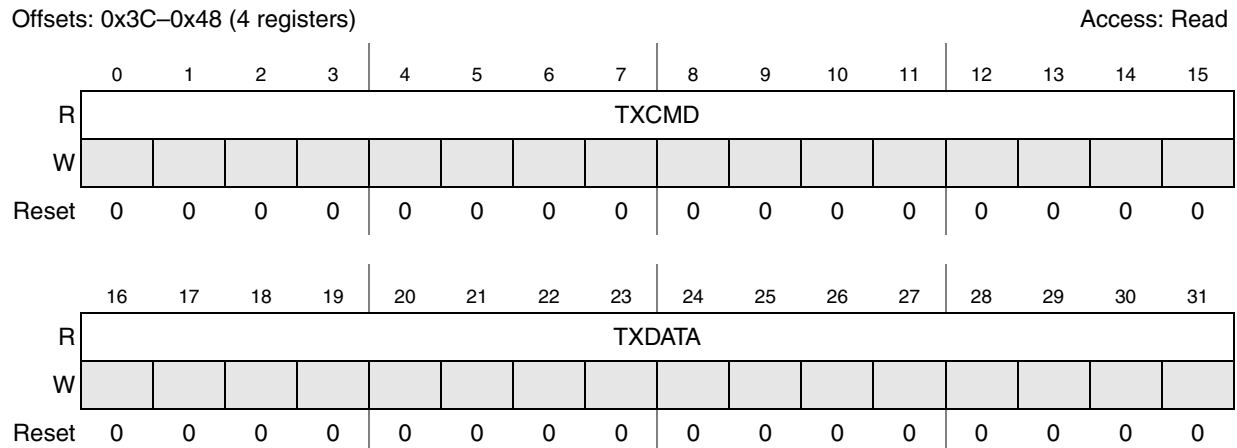


Figure 26-10. DSPI Transmit FIFO Register 0–3 (DSPIx\_TXFRn)

Table 26-22. DSPIx\_TXFRn field descriptions

| Field  | Description   |
|--------|---|
| TXCMD  | Transmit command<br>Contains the command that sets the transfer attributes for the SPI data. See <a href="#">Section 26.5.7, DSPI PUSH TX FIFO Register (DSPIx_PUSHR)</a> , for details on the command field. |
| TXDATA | Transmit data<br>Contains the SPI data to be shifted out.   |

### 26.5.9.1 DSPI Receive FIFO Registers 0–3 (DSPIx\_RXFRn)

The DSPIx\_RXFRn registers provide visibility into the RX FIFO for debugging purposes. Each register is an entry in the RX FIFO. The DSPIx\_RXFR registers are read-only. Reading the DSPIx\_RXFRn registers does not alter the state of the RX FIFO. The device uses four registers to implement the RX FIFO, that is DSPIx\_RXFR0–DSPIx\_RXFR3 are used.

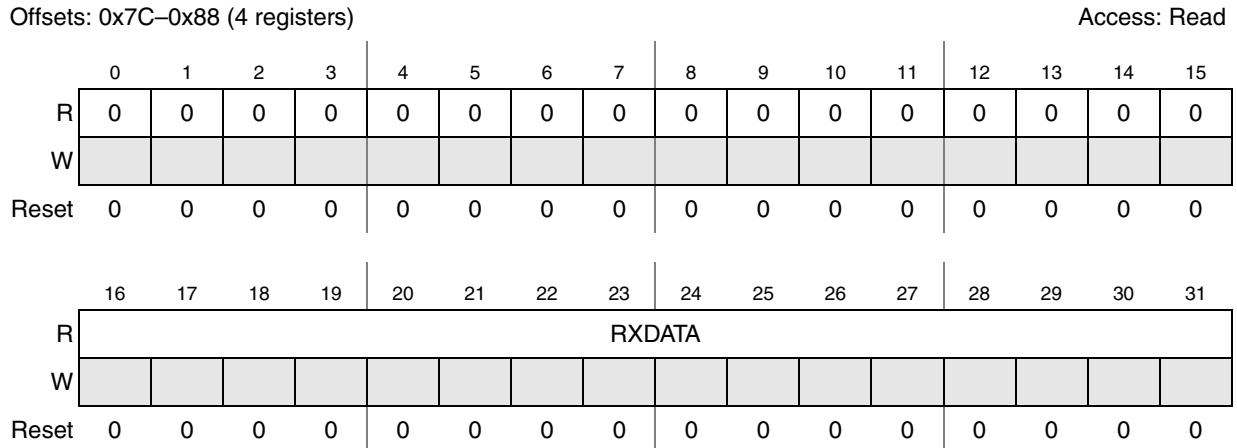


Figure 26-11. DSPI Receive FIFO Registers 0–3 (DSPIx\_RXFRn)

Table 26-23. DSPIx\_RXFRn field description

| Field  | Description                                     |
|--------|---|
| RXDATA | Receive data<br>Contains the received SPI data. |

## 26.6 Functional description

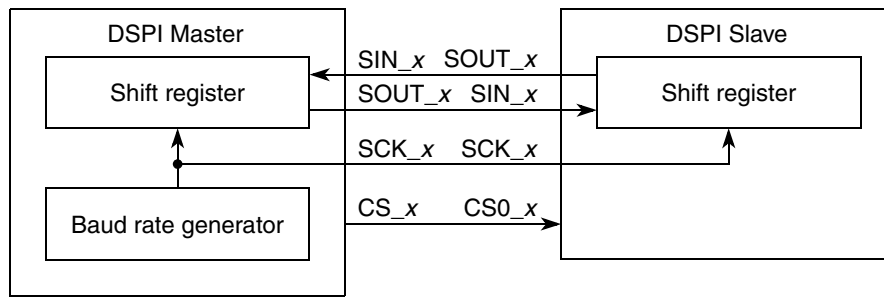
The DSPI supports full-duplex, synchronous serial communications between the MCU and peripheral devices. All communications are through an SPI-like protocol.

The DSPI has one configuration, namely serial peripheral interface (SPI), in which the DSPI operates as a basic SPI or a queued SPI.

The DCONF field in the DSPIx\_MCR register determines the DSPI configuration. See [Table 26-3](#) for the DSPI configuration values.

The DSPIx\_CTAR0–DSPIx\_CTAR5 registers hold clock and transfer attributes. The SPI configuration can select which CTAR to use on a frame by frame basis by setting the CTAS field in the DSPIx\_PUSHR.

The 16-bit shift register in the master and the 16-bit shift register in the slave are linked by the SOUT\_x and SIN\_x signals to form a distributed 32-bit register. When a data transfer operation is performed, data is serially shifted a predetermined number of bit positions. Because the registers are linked, data is exchanged between the master and the slave; the data that was in the master’s shift register is now in the shift register of the slave, and vice versa. At the end of a transfer, the TCF bit in the DSPIx\_SR is set to indicate a completed transfer. [Figure 26-12](#) illustrates how master and slave data is exchanged.



**Figure 26-12. SPI serial protocol overview**

The DSPI has six peripheral chip select ( $CS_x$ ) signals that are be used to select which of the slaves to communicate with.

Transfer protocols and timing properties are shared by the three DSPI configurations; these properties are described independently of the configuration in [Section 26.6.5, Transfer formats](#). The transfer rate and delay settings are described in [Section 26.6.4, DSPI baud rate and clock delay generation](#).

See [Section 26.6.8, Power saving features](#), for information on the power-saving features of the DSPI.

## 26.6.1 Modes of operation

The DSPI modules have the following available distinct modes:

- Master mode
- Slave mode
- Module Disable mode
- Debug mode

Master, slave, and module disable modes are module-specific modes whereas debug mode is device-specific.

The module-specific modes are determined by bits in the  $DSPI_x\_MCR$ . Debug mode is a mode that the entire device can enter in parallel with the DSPI being configured in one of its module-specific modes.

### 26.6.1.1 Master mode

In master mode the DSPI can initiate communications with peripheral devices. The DSPI operates as bus master when the MSTR bit in the  $DSPI_x\_MCR$  is set. The serial communications clock (SCK) is controlled by the master DSPI. All three DSPI configurations are valid in master mode.

In SPI configuration, master mode transfer attributes are controlled by the SPI command in the current TX FIFO entry. The CTAS field in the SPI command selects which of the  $DSPI_x\_CTARs$  are used to set the transfer attributes. Transfer attribute control is on a frame by frame basis.

See [Section 26.6.3, Serial peripheral interface \(SPI\) configuration](#) for more details.

### 26.6.1.2 Slave mode

In slave mode the DSPI responds to transfers initiated by an SPI master. The DSPI operates as bus slave when the MSTR bit in the DSPI<sub>x</sub>\_MCR is negated. The DSPI slave is selected by a bus master by having the slave's CS0<sub>x</sub> asserted. In slave mode the SCK is provided by the bus master. All transfer attributes are controlled by the bus master, except the clock polarity, clock phase, and the number of bits to transfer, which must be configured in the DSPI slave to communicate correctly.

### 26.6.1.3 Module Disable mode

The module disable mode is used for MCU power management. The clock to the non-memory mapped logic in the DSPI is stopped while in module disable mode. The DSPI enters the module disable mode when the MDIS bit in DSPI<sub>x</sub>\_MCR is set.

See [Section 26.6.8, Power saving features](#), for more details on the module disable mode.

### 26.6.1.4 Debug mode

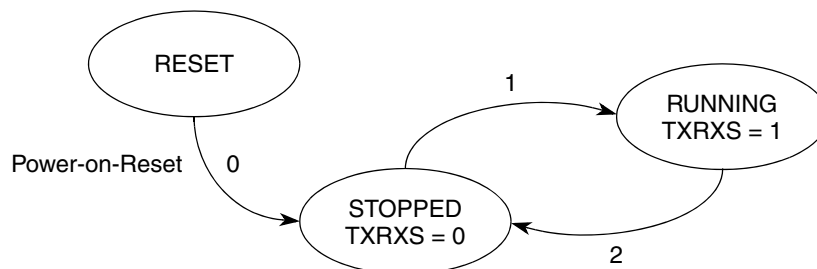
The debug mode is used for system development and debugging. If the MCU enters debug mode while the FRZ bit in the DSPI<sub>x</sub>\_MCR is set, the DSPI stops all serial transfers and enters a stopped state. If the MCU enters debug mode while the FRZ bit is cleared, the DSPI behavior is unaffected and remains dictated by the module-specific mode and configuration of the DSPI. The DSPI enters debug mode when a debug request is asserted by an external controller.

See [Figure 26-13](#) for a state diagram.

## 26.6.2 Start and stop of DSPI transfers

The DSPI has two operating states: STOPPED and RUNNING. The states are independent of DSPI configuration. The default state of the DSPI is STOPPED. In the STOPPED state, no serial transfers are initiated in master mode and no transfers are responded to in slave mode. The STOPPED state is also a safe state for writing the various configuration registers of the DSPI without causing undetermined results. The TXRXS bit in the DSPI<sub>x</sub>\_SR is cleared in this state. In the RUNNING state, serial transfers take place. The TXRXS bit in the DSPI<sub>x</sub>\_SR is set in the RUNNING state.

[Figure 26-13](#) shows a state diagram of the start and stop mechanism.



**Figure 26-13. DSPI start and stop state diagram**

The transitions are described in [Table 26-24](#).

**Table 26-24. State transitions for start and stop of DSPI transfers**

| Transition No. | Current state | Next state | Description   |
|----------------|---------------|------------|---|
| 0              | RESET         | STOPPED    | Generic power-on-reset transition   |
| 1              | STOPPED       | RUNNING    | The DSPI starts (transitions from STOPPED to RUNNING) when all of the following conditions are true: <ul style="list-style-type: none"> <li>• EOQF bit is clear</li> <li>• Debug mode is unselected or the FRZ bit is clear</li> <li>• HALT bit is clear</li> </ul>           |
| 2              | RUNNING       | STOPPED    | The DSPI stops (transitions from RUNNING to STOPPED) after the current frame for any one of the following conditions: <ul style="list-style-type: none"> <li>• EOQF bit is set</li> <li>• Debug mode is selected and the FRZ bit is set</li> <li>• HALT bit is set</li> </ul> |

State transitions from RUNNING to STOPPED occur on the next frame boundary if a transfer is in progress, or on the next system clock cycle if no transfers are in progress.

### 26.6.3 Serial peripheral interface (SPI) configuration

The SPI configuration transfers data serially using a shift register and a selection of programmable transfer attributes. The DSPI is in SPI configuration when the DCONF field in the DSPIx\_MCR is 0b00. The SPI frames can be from 4 to 16 bits long. The data to be transmitted can come from queues stored in SRAM external to the DSPI. Host software or an eDMA controller can transfer the SPI data from the queues to a first-in first-out (FIFO) buffer. The received data is stored in entries in the receive FIFO (RX FIFO) buffer. Host software or an eDMA controller transfers the received data from the RX FIFO to memory external to the DSPI.

The FIFO buffer operations are described in [Section 26.6.3.4, Transmit First In First Out \(TX FIFO\) buffering mechanism](#), and [Section 26.6.3.5, Receive First In First Out \(RX FIFO\) buffering mechanism](#).

The interrupt and DMA request conditions are described in [Section 26.6.7, Interrupt/DMA requests](#).

The SPI configuration supports two module-specific modes; master mode and slave mode. The FIFO operations are similar for the master mode and slave mode. The main difference is that in master mode the DSPI initiates and controls the transfer according to the fields in the SPI command field of the TX FIFO entry. In slave mode the DSPI only responds to transfers initiated by a bus master external to the DSPI and the SPI command field of the TX FIFO entry is ignored.

#### 26.6.3.1 SPI Master mode

In SPI master mode the DSPI initiates the serial transfers by controlling the serial communications clock (SCK<sub>x</sub>) and the peripheral chip select (CS<sub>x</sub>) signals. The SPI command field in the executing TX FIFO entry determines which CTARs are used to set the transfer attributes and which CS<sub>x</sub> signal to assert. The command field also contains various bits that help with queue management and transfer protocol. The data field in the executing TX FIFO entry is loaded into the shift register and shifted out on the serial out

(SOUT<sub>x</sub>) pin. In SPI master mode, each SPI frame to be transmitted has a command associated with it allowing for transfer attribute control on a frame by frame basis.

See [Section 26.5.7, DSPI PUSH TX FIFO Register \(DSPI<sub>x</sub>\\_PUSHR\)](#), for details on the SPI command fields.

### 26.6.3.2 SPI Slave mode

In SPI slave mode the DSPI responds to transfers initiated by an SPI bus master. The DSPI does not initiate transfers. Certain transfer attributes such as clock polarity, clock phase, and frame size must be set for successful communication with an SPI master. The SPI slave mode transfer attributes are set in the DSPI<sub>x</sub>\_CTAR0.

### 26.6.3.3 FIFO disable operation

The FIFO disable mechanisms allow SPI transfers without using the TX FIFO or RX FIFO. The DSPI operates as a double-buffered simplified SPI when the FIFOs are disabled. The TX and RX FIFOs are disabled separately. The TX FIFO is disabled by writing a 1 to the DIS\_TXF bit in the DSPI<sub>x</sub>\_MCR. The RX FIFO is disabled by writing a 1 to the DIS\_RXF bit in the DSPI<sub>x</sub>\_MCR.

The FIFO disable mechanisms are transparent to the user and to host software; transmit data and commands are written to the DSPI<sub>x</sub>\_PUSHR and received data is read from the DSPI<sub>x</sub>\_POPR. When the TX FIFO is disabled, the TFFF, TFUF, and TXCTR fields in DSPI<sub>x</sub>\_SR behave as if there is a one-entry FIFO but the contents of the DSPI<sub>x</sub>\_TXFRs and TXNXTPTR are undefined. When the RX FIFO is disabled, the RFDF, RFOF, and RXCTR fields in the DSPI<sub>x</sub>\_SR behave as if there is a one-entry FIFO but the contents of the DSPI<sub>x</sub>\_RXFRs and POPNXTPTR are undefined.

Disable the TX and RX FIFOs only if the FIFO must be disabled as a requirement of the application's operating mode. A FIFO must be disabled before it is accessed. Failure to disable a FIFO prior to a first FIFO access is not supported, and can result in incorrect results.

### 26.6.3.4 Transmit First In First Out (TX FIFO) buffering mechanism

The TX FIFO functions as a buffer of SPI data and SPI commands for transmission. The TX FIFO holds four entries, each consisting of a command field and a data field. SPI commands and data are added to the TX FIFO by writing to the DSPI push TX FIFO register (DSPI<sub>x</sub>\_PUSHR). TX FIFO entries can only be removed from the TX FIFO by being shifted out or by flushing the TX FIFO. For more information on DSPI<sub>x</sub>\_PUSHR, see [Section 26.5.7, DSPI PUSH TX FIFO Register \(DSPI<sub>x</sub>\\_PUSHR\)](#).

The TX FIFO counter field (TXCTR) in the DSPI status register (DSPI<sub>x</sub>\_SR) indicates the number of valid entries in the TX FIFO. The TXCTR is updated every time the DSPI<sub>x</sub>\_PUSHR is written or SPI data is transferred into the shift register from the TX FIFO.

See [Section 26.5.5, DSPI Status Register \(DSPI<sub>x</sub>\\_SR\)](#) for more information on DSPI<sub>x</sub>\_SR.

The TXNXTPTR field indicates which TX FIFO entry is transmitted during the next transfer. The TXNXTPTR contains the positive offset from DSPI<sub>x</sub>\_TXFR0 in number of 32-bit registers. For example, TXNXTPTR equal to two means that the DSPI<sub>x</sub>\_TXFR2 contains the SPI data and command for the next



transfer. The TXNXTPTR field is incremented every time SPI data is transferred from the TX FIFO to the shift register.

#### 26.6.3.4.1 Filling the TX FIFO

Host software or the eDMA controller can add (push) entries to the TX FIFO by writing to the DSPIx\_PUSHR. When the TX FIFO is not full, the TX FIFO fill flag (TFFF) in the DSPIx\_SR is set. The TFFF bit is cleared when the TX FIFO is full and the eDMA controller indicates that a write to DSPIx\_PUSHR is complete or alternatively by host software writing a 1 to the TFFF in the DSPIx\_SR. The TFFF can generate a DMA request or an interrupt request.

See [Section 26.6.7.2, Transmit FIFO Fill Interrupt or DMA Request \(TFFF\)](#), for details.

The DSPI ignores attempts to push data to a full TX FIFO; that is, the state of the TX FIFO is unchanged. No error condition is indicated.

#### 26.6.3.4.2 Draining the TX FIFO

The TX FIFO entries are removed (drained) by shifting SPI data out through the shift register. Entries are transferred from the TX FIFO to the shift register and shifted out as long as there are valid entries in the TX FIFO. Every time an entry is transferred from the TX FIFO to the shift register, the TX FIFO counter is decremented by one. At the end of a transfer, the TCF bit in the DSPIx\_SR is set to indicate the completion of a transfer. The TX FIFO is flushed by writing a 1 to the CLR\_TXF bit in DSPIx\_MCR.

If an external SPI bus master initiates a transfer with a DSPI slave while the slave's DSPI TX FIFO is empty, the transmit FIFO underflow flag (TFUF) in the slave's DSPIx\_SR is set.

See [Section 26.6.7.4, Transmit FIFO Underflow Interrupt Request \(TFUF\)](#), for details.

#### 26.6.3.5 Receive First In First Out (RX FIFO) buffering mechanism

The RX FIFO functions as a buffer for data received on the SIN pin. The RX FIFO holds four received SPI data frames. SPI data is added to the RX FIFO at the completion of a transfer when the received data in the shift register is transferred into the RX FIFO. SPI data is removed (popped) from the RX FIFO by reading the DSPIx\_POPR register. RX FIFO entries can only be removed from the RX FIFO by reading the DSPIx\_POPR or by flushing the RX FIFO.

See [Section 26.5.8, DSPI POP RX FIFO Register \(DSPIx\\_POPR\)](#) for more information on the DSPIx\_POPR.

The RX FIFO counter field (RXCTR) in the DSPI status register (DSPIx\_SR) indicates the number of valid entries in the RX FIFO. The RXCTR is updated every time the DSPIx\_POPR is read or SPI data is copied from the shift register to the RX FIFO.

The POPNXTPTR field in the DSPIx\_SR points to the RX FIFO entry that is returned when the DSPIx\_POPR is read. The POPNXTPTR contains the positive, 32-bit word offset from DSPIx\_RXFR0. For example, POPNXTPTR equal to two means that the DSPIx\_RXFR2 contains the received SPI data that is returned when DSPIx\_POPR is read. The POPNXTPTR field is incremented every time the DSPIx\_POPR is read. POPNXTPTR rolls over every four frames on the MCU.

### 26.6.3.5.1 Filling the RX FIFO

The RX FIFO is filled with the received SPI data from the shift register. While the RX FIFO is not full, SPI frames from the shift register are transferred to the RX FIFO. Every time an SPI frame is transferred to the RX FIFO the RX FIFO counter is incremented by one.

If the RX FIFO and shift register are full and a transfer is initiated, the RFOF bit in the DSPLx\_SR is set indicating an overflow condition. Depending on the state of the ROOE bit in the DSPLx\_MCR, the data from the transfer that generated the overflow is ignored or put in the shift register. If the ROOE bit is set, the incoming data is put in the shift register. If the ROOE bit is cleared, the incoming data is ignored.

### 26.6.3.5.2 Draining the RX FIFO

Host software or the eDMA can remove (pop) entries from the RX FIFO by reading the DSPLx\_POPR. A read of the DSPLx\_POPR decrements the RX FIFO counter by one. Attempts to pop data from an empty RX FIFO are ignored, the RX FIFO counter remains unchanged. The data returned from reading an empty RX FIFO is undetermined.

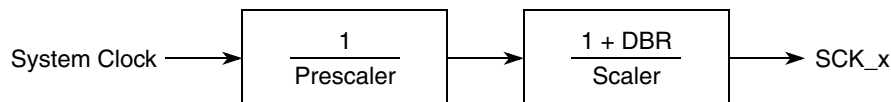
See [Section 26.5.8, DSPI POP RX FIFO Register \(DSPLx\\_POPR\)](#) for more information on DSPLx\_POPR.

When the RX FIFO is not empty, the RX FIFO drain flag (RFDF) in the DSPLx\_SR is set. The RFDF bit is cleared when the RX\_FIFO is empty and the eDMA controller indicates that a read from DSPLx\_POPR is complete; alternatively the RFDF bit can be cleared by the host writing a 1 to it.

## 26.6.4 DSPI baud rate and clock delay generation

The SCK\_x frequency and the delay values for serial transfer are generated by dividing the system clock frequency by a prescaler and a scaler with the option of doubling the baud rate.

[Figure 26-14](#) shows conceptually how the SCK signal is generated.



**Figure 26-14. Communications clock prescalers and scalers**

### 26.6.4.1 Baud rate generator

The baud rate is the frequency of the serial communication clock (SCK\_x). The system clock is divided by a baud rate prescaler (defined by DSPLx\_CTAR[PBR]) and baud rate scaler (defined by DSPLx\_CTAR[BR]) to produce SCK\_x with the possibility of doubling the baud rate. The DBR, PBR, and BR fields in the DSPLx\_CTARs select the frequency of SCK\_x using the following formula:

$$\text{SCK baud rate} = \frac{f_{\text{SYS}}}{\text{PBRPrescalerValue}} \times \frac{1 + \text{DBR}}{\text{BRScalerValue}}$$

[Table 26-25](#) shows an example of a computed baud rate.

Table 26-25. Baud rate computation example

| $f_{SYS}$ | PBR  | Prescaler value | BR     | Scaler value | DBR value | Baud rate |
|-----------|------|-----------------|--------|--------------|-----------|-----------|
| 64 MHz    | 0b00 | 2               | 0b0000 | 2            | 0         | 16 Mbit/s |
| 20 MHz    | 0b00 | 2               | 0b0000 | 2            | 1         | 10 Mbit/s |

#### 26.6.4.2 CS to SCK delay ( $t_{CSC}$ )

The CS<sub>x</sub> to SCK<sub>x</sub> delay is the length of time from assertion of the CS<sub>x</sub> signal to the first SCK<sub>x</sub> edge. See [Figure 26-16](#) for an illustration of the CS<sub>x</sub> to SCK<sub>x</sub> delay. The PCSSCK and CSSCK fields in the DSPLx\_CTAR<sub>n</sub> registers select the CS<sub>x</sub> to SCK<sub>x</sub> delay, and the relationship is expressed by the following formula:

$$t_{CSC} = \frac{1}{f_{SYS}} \times PCSSCK \times CSSCK$$

[Table 26-26](#) shows an example of the computed CS to SCK<sub>x</sub> delay.

Table 26-26. CS to SCK delay computation example

| PCSSCK | Prescaler value | CSSCK  | Scaler value | $f_{SYS}$ | CS to SCK delay |
|--------|-----------------|--------|--------------|-----------|-----------------|
| 0b01   | 3               | 0b0100 | 32           | 64 MHz    | 1.5 $\mu$ s     |

#### 26.6.4.3 After SCK delay ( $t_{ASC}$ )

The after SCK<sub>x</sub> delay is the length of time between the last edge of SCK<sub>x</sub> and the negation of CS<sub>x</sub>. See [Figure 26-16](#) and [Figure 26-17](#) for illustrations of the after SCK<sub>x</sub> delay. The PASC and ASC fields in the DSPLx\_CTAR<sub>n</sub> registers select the after SCK delay. The relationship between these variables is given in the following formula:

$$t_{ASC} = \frac{1}{f_{SYS}} \times PASC \times ASC$$

[Table 26-27](#) shows an example of the computed after SCK delay.

Table 26-27. After SCK delay computation example

| PASC | Prescaler value | ASC    | Scaler value | $f_{SYS}$ | After SCK delay |
|------|-----------------|--------|--------------|-----------|-----------------|
| 0b01 | 3               | 0b0100 | 32           | 64 MHz    | 1.5 $\mu$ s     |

#### 26.6.4.4 Delay after transfer ( $t_{DT}$ )

The delay after transfer is the length of time between negation of the CS<sub>x</sub> signal for a frame and the assertion of the CS<sub>x</sub> signal for the next frame. The PDT and DT fields in the DSPLx\_CTAR<sub>n</sub> registers select the delay after transfer.

See [Figure 26-16](#) for an illustration of the delay after transfer.

The following formula expresses the PDT/DT/delay after transfer relationship:

$$t_{DT} = \frac{1}{f_{SYS}} \times PDT \times DT$$

Table 26-28 shows an example of the computed delay after transfer.

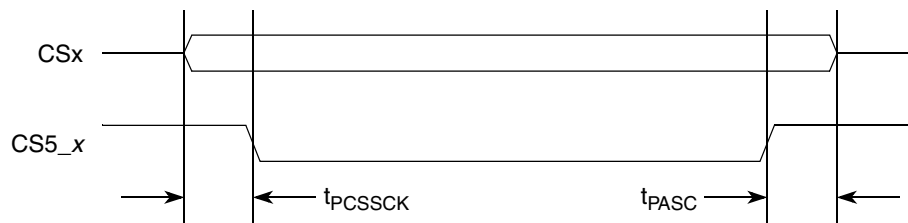
**Table 26-28. Delay after transfer computation example**

| PDT  | Prescaler value | DT     | Scaler value | f <sub>SYS</sub> | Delay after transfer |
|------|-----------------|--------|--------------|------------------|----------------------|
| 0b01 | 3               | 0b1110 | 32768        | 64 MHz           | 1.54 ms              |

#### 26.6.4.5 Peripheral chip select strobe enable (CS5\_x)

The CS5\_x signal provides a delay to allow the CSx signals to settle after transitioning thereby avoiding glitches. When the DSPI is in master mode and PCSSE bit is set in the DSPIx\_MCR, CS5\_x provides a signal for an external demultiplexer to decode the CS4\_x signals into as many as 32 glitchfree CSx signals.

Figure 26-15 shows the timing of the CS5\_x signal relative to CS signals.



**Figure 26-15. Peripheral chip select strobe timing**

The delay between the assertion of the CSx signals and the assertion of CS5\_x is selected by the PCSSCK field in the DSPIx\_CTAR based on the following formula:

$$t_{PCSSCK} = \frac{1}{f_{SYS}} \times PCSSCK$$

At the end of the transfer the delay between CS5\_x negation and CSx negation is selected by the PASC field in the DSPIx\_CTAR based on the following formula:

$$t_{PASC} = \frac{1}{f_{SYS}} \times PASC$$

Table 26-29 shows an example of the computed t<sub>PCSSCK</sub> delay.

**Table 26-29. Peripheral chip select strobe assert computation example**

| PCSSCK | Prescaler | f <sub>SYS</sub> | Delay before transfer |
|--------|-----------|------------------|-----------------------|
| 0b11   | 7         | 64 MHz           | 109.4 ns              |

Table 26-30 shows an example of the computed the t<sub>PASC</sub> delay.

**Table 26-30. Peripheral chip select strobe negate computation example**

| PASC | Prescaler | $f_{SYS}$ | Delay after transfer |
|------|-----------|-----------|----------------------|
| 0b11 | 7         | 64 MHz    | 109.4 ns             |

## 26.6.5 Transfer formats

The SPI serial communication is controlled by the serial communications clock (SCK<sub>x</sub>) signal and the CS<sub>x</sub> signals. The SCK<sub>x</sub> signal provided by the master device synchronizes shifting and sampling of the data by the SIN<sub>x</sub> and SOUT<sub>x</sub> pins. The CS<sub>x</sub> signals serve as enable signals for the slave devices.

When the DSPI is the bus master, the CPOL and CPHA bits in the DSPI clock and transfer attributes registers (DSPI<sub>x</sub>\_CTAR<sub>n</sub>) select the polarity and phase of the serial clock, SCK<sub>x</sub>. The polarity bit selects the idle state of the SCK<sub>x</sub>. The clock phase bit selects if the data on SOUT<sub>x</sub> is valid before or on the first SCK<sub>x</sub> edge.

When the DSPI is the bus slave, CPOL and CPHA bits in the DSPI<sub>x</sub>\_CTAR<sub>0</sub> (SPI slave mode) select the polarity and phase of the serial clock. Even though the bus slave does not control the SCK signal, clock polarity, clock phase and number of bits to transfer must be identical for the master device and the slave device to ensure proper transmission.

The DSPI supports four different transfer formats:

- Classic SPI with CPHA = 0
- Classic SPI with CPHA = 1
- Modified transfer format with CPHA = 0
- Modified transfer format with CPHA = 1

A modified transfer format is supported to allow for high-speed communication with peripherals that require longer setup times. The DSPI can sample the incoming data later than halfway through the cycle to give the peripheral more setup time. The MTFE bit in the DSPI<sub>x</sub>\_MCR selects between classic SPI format and modified transfer format. The classic SPI formats are described in [Section 26.6.5.1, Classic SPI transfer format \(CPHA = 0\)](#), and [Section 26.6.5.2, Classic SPI transfer format \(CPHA = 1\)](#). The modified transfer formats are described in [Section 26.6.5.3, Modified SPI transfer format \(MTFE = 1, CPHA = 0\)](#), and [Section 26.6.5.4, Modified SPI transfer format \(MTFE = 1, CPHA = 1\)](#).

In the SPI configuration, the DSPI provides the option of keeping the CS signals asserted between frames. See [Section 26.6.5.5, Continuous selection format](#) for details.

### 26.6.5.1 Classic SPI transfer format (CPHA = 0)

The transfer format shown in Figure 26-16 is used to communicate with peripheral SPI slave devices where the first data bit is available on the first clock edge. In this format, the master and slave sample their SIN<sub>x</sub> pins on the odd-numbered SCK<sub>x</sub> edges and change the data on their SOUT<sub>x</sub> pins on the even-numbered SCK<sub>x</sub> edges.

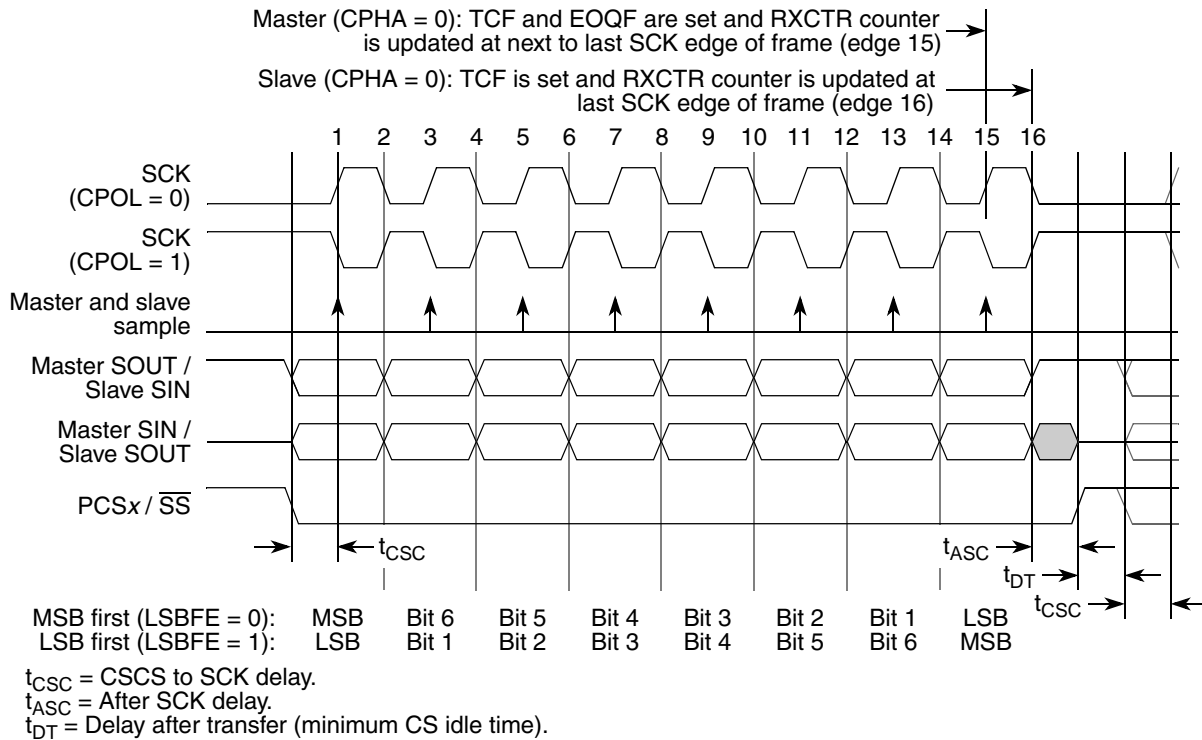


Figure 26-16. DSPI transfer timing diagram (MTFE = 0, CPHA = 0, FMSZ = 8)

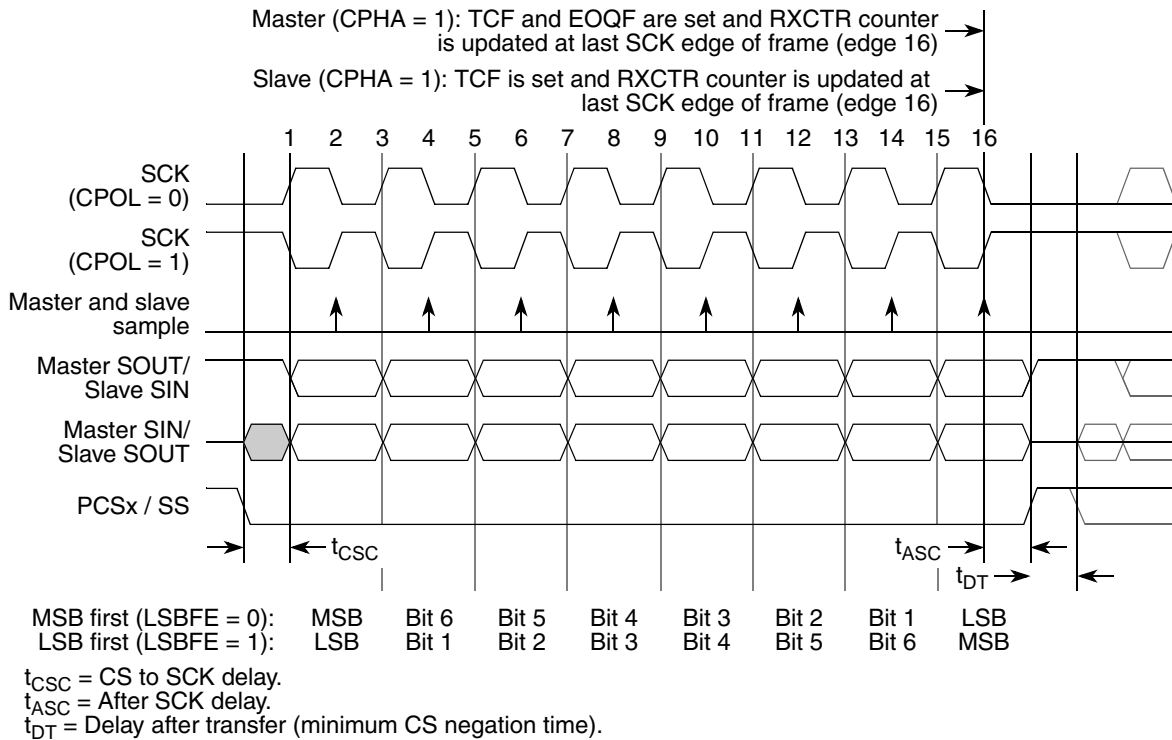
The master initiates the transfer by placing its first data bit on the SOUT<sub>x</sub> pin and asserting the appropriate peripheral chip select signals to the slave device. The slave responds by placing its first data bit on its SOUT<sub>x</sub> pin. After the  $t_{CSC}$  delay has elapsed, the master outputs the first edge of SCK<sub>x</sub>. This is the edge used by the master and slave devices to sample the first input data bit on their serial data input signals. At the second edge of the SCK<sub>x</sub> the master and slave devices place their second data bit on their serial data output signals. For the rest of the frame the master and the slave sample their SIN<sub>x</sub> pins on the odd-numbered clock edges and changes the data on their SOUT<sub>x</sub> pins on the even-numbered clock edges. After the last clock edge occurs a delay of  $t_{ASC}$  is inserted before the master negates the CS signals. A delay of  $t_{DT}$  is inserted before a new frame transfer can be initiated by the master.

For the CPHA = 0 condition of the master, TCF and EOQF are set and the RXCTR counter is updated at the next to last serial clock edge of the frame (edge 15) of Figure 26-16.

For the CPHA = 0 condition of the slave, TCF is set and the RXCTR counter is updated at the last serial clock edge of the frame (edge 16) of Figure 26-16.

### 26.6.5.2 Classic SPI transfer format (CPHA = 1)

This transfer format shown in Figure 26-17 is used to communicate with peripheral SPI slave devices that require the first SCK<sub>x</sub> edge before the first data bit becomes available on the slave SOUT<sub>x</sub> pin. In this format the master and slave devices change the data on their SOUT<sub>x</sub> pins on the odd-numbered SCK<sub>x</sub> edges and sample the data on their SIN<sub>x</sub> pins on the even-numbered SCK<sub>x</sub> edges.



**Figure 26-17. DSPI transfer timing diagram (MTFE = 0, CPHA = 1, FMSZ = 8)**

The master initiates the transfer by asserting the CS<sub>x</sub> signal to the slave. After the  $t_{CSC}$  delay has elapsed, the master generates the first SCK<sub>x</sub> edge and at the same time places valid data on the master SOUT<sub>x</sub> pin. The slave responds to the first SCK<sub>x</sub> edge by placing its first data bit on its slave SOUT<sub>x</sub> pin.

At the second edge of the SCK<sub>x</sub> the master and slave sample their SIN<sub>x</sub> pins. For the rest of the frame the master and the slave change the data on their SOUT<sub>x</sub> pins on the odd-numbered clock edges and sample their SIN<sub>x</sub> pins on the even-numbered clock edges. After the last clock edge occurs a delay of  $t_{ASC}$  is inserted before the master negates the CS<sub>x</sub> signal. A delay of  $t_{DT}$  is inserted before a new frame transfer can be initiated by the master.

For CPHA = 1 the master EOQF and TCF and slave TCF are set at the last serial clock edge (edge 16) of Figure 26-17. For CPHA = 1 the master and slave RXCTR counters are updated on the same clock edge.

### 26.6.5.3 Modified SPI transfer format (MTFE = 1, CPHA = 0)

In this modified transfer format both the master and the slave sample later in the SCK period than in classic SPI mode to allow for delays in device pads and board traces. These delays become a more significant fraction of the SCK period as the SCK period decreases with increasing baud rates.

#### NOTE

For the modified transfer format to operate correctly, you must thoroughly analyze the SPI link timing budget.

The master and the slave place data on the SOUT<sub>x</sub> pins at the assertion of the CS<sub>x</sub> signal. After the CS<sub>x</sub> to SCK<sub>x</sub> delay has elapsed the first SCK<sub>x</sub> edge is generated. The slave samples the master SOUT<sub>x</sub> signal on every odd numbered SCK<sub>x</sub> edge. The slave also places new data on the slave SOUT<sub>x</sub> on every odd numbered clock edge.

The master places its second data bit on the SOUT<sub>x</sub> line one system clock after odd numbered SCK<sub>x</sub> edge. The point where the master samples the slave SOUT<sub>x</sub> is selected by writing to the SMPL\_PT field in the DSPI<sub>x</sub>\_MCR. [Table 26-31](#) lists the number of system clock cycles between the active edge of SCK<sub>x</sub> and the master sample point for different values of the SMPL\_PT bit field. The master sample point can be delayed by one or two system clock cycles.

**Table 26-31. Delayed master sample point**

| SMPL_PT | Number of system clock cycles<br>between odd-numbered edge of SCK and sampling of SIN |
|---------|---|
| 00      | 0   |
| 01      | 1   |
| 10      | 2   |
| 11      | Invalid value   |

[Figure 26-18](#) shows the modified transfer format for CPHA = 0. Only the condition where CPOL = 0 is illustrated. The delayed master sample points are indicated with a lighter shaded arrow.



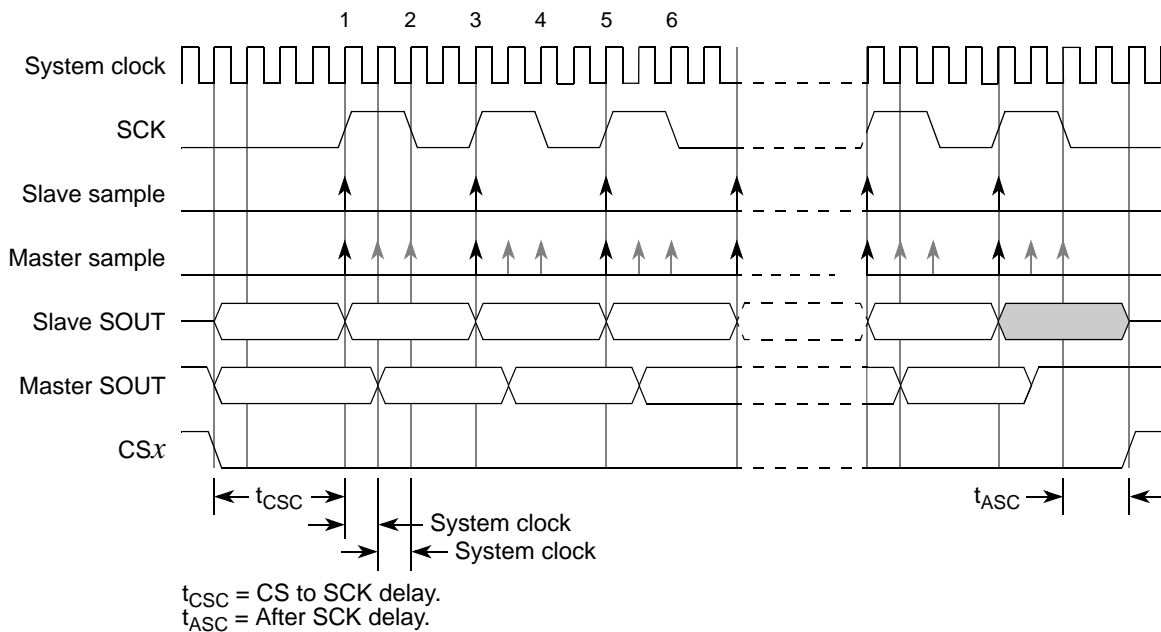


Figure 26-18. DSPI modified transfer format (MTFE = 1, CPHA = 0,  $f_{SCK} = f_{SYS} / 4$ )

#### 26.6.5.4 Modified SPI transfer format (MTFE = 1, CPHA = 1)

At the start of a transfer the DSPI asserts the CS signal to the slave device. After the CS to SCK delay has elapsed the master and the slave put data on their SOUT pins at the first edge of SCK. The slave samples the master SOUT signal on the even numbered edges of SCK. The master samples the slave SOUT signal on the odd numbered SCK edges starting with the third SCK edge. The slave samples the last bit on the last edge of the SCK. The master samples the last slave SOUT bit one half SCK cycle after the last edge of SCK. No clock edge is visible on the master SCK pin during the sampling of the last bit. The SCK to CS delay must be greater or equal to half of the SCK period.

#### NOTE

For the modified transfer format to operate correctly, you must thoroughly analyze the SPI link timing budget.

Figure 26-19 shows the modified transfer format for CPHA = 1. Only the condition where CPOL = 0 is described.

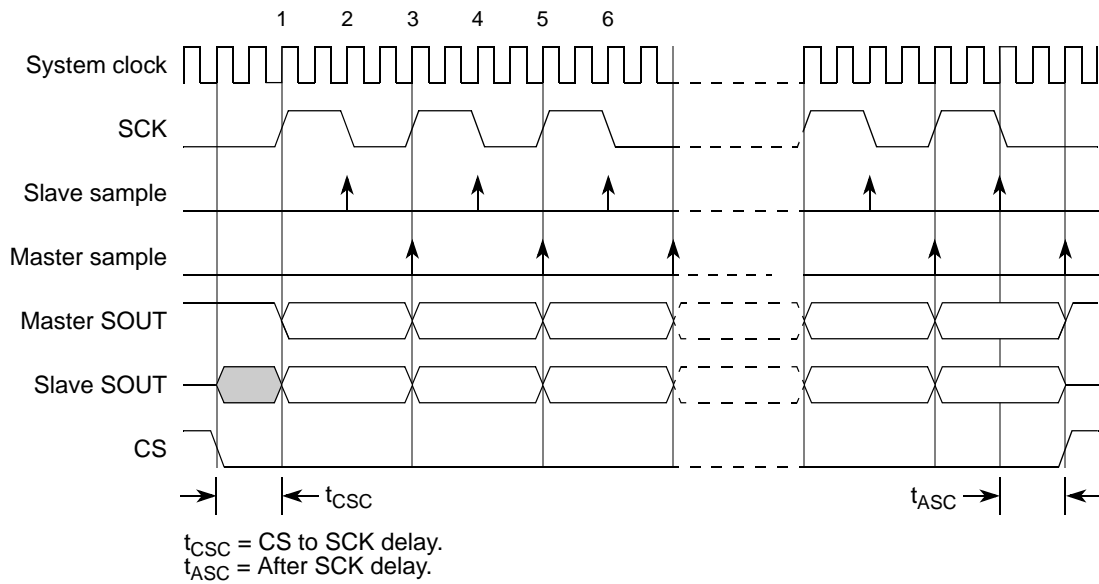


Figure 26-19. DSPI modified transfer format (MTFE = 1, CPHA = 1,  $f_{SCK} = f_{SYS} / 4$ )

### 26.6.5.5 Continuous selection format

Some peripherals must be deselected between every transfer. Other peripherals must remain selected between several sequential serial transfers. The continuous selection format provides the flexibility to handle both cases. The continuous selection format is enabled for the SPI configuration by setting the CONT bit in the SPI command.

When the CONT bit = 0, the DSPI drives the asserted chip select signals to their idle states in between frames. The idle states of the chip select signals are selected by the PCSIS field in the DSPIx\_MCR.

Figure 26-20 shows the timing diagram for two 4-bit transfers with CPHA = 1 and CONT = 0.

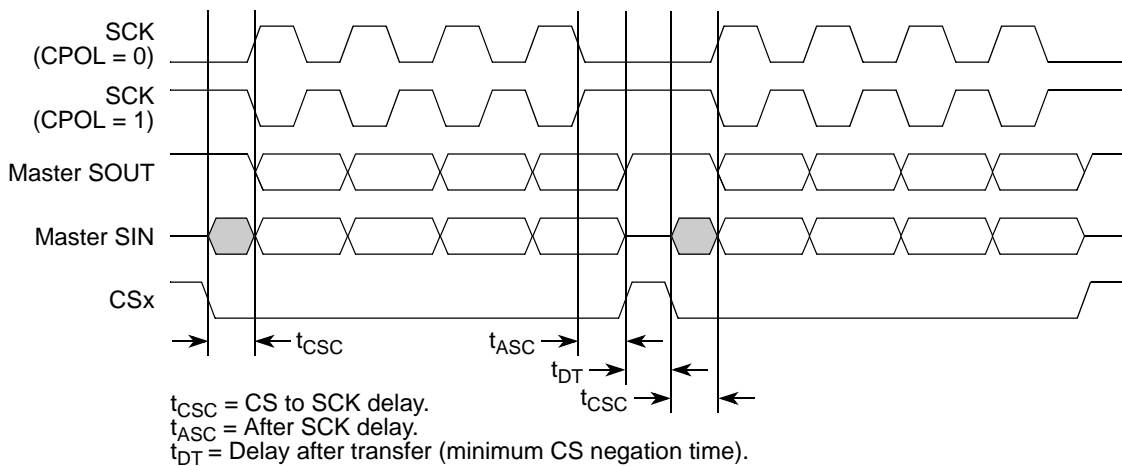
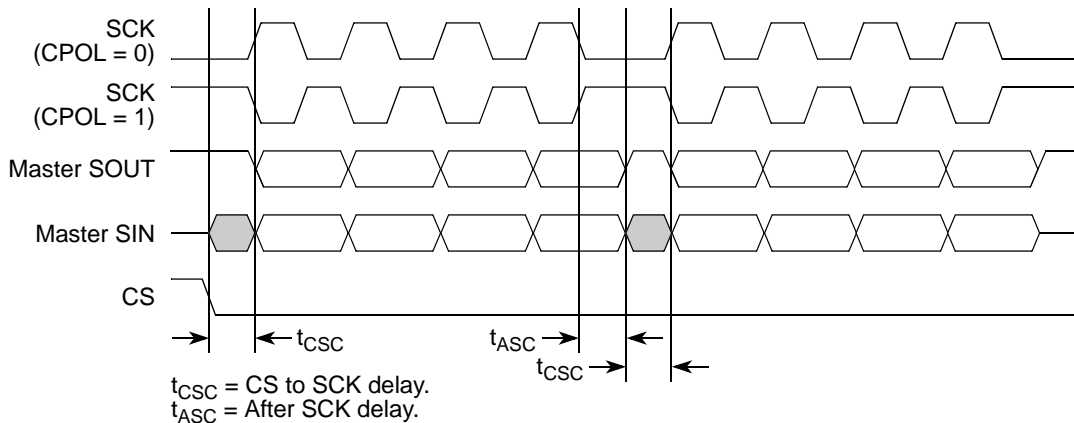


Figure 26-20. Example of non-continuous format (CPHA = 1, CONT = 0)

When the  $CONT = 1$  and the CS signal for the next transfer is the same as for the current transfer, the CS signal remains asserted for the duration of the two transfers. The delay between transfers ( $t_{DT}$ ) is not inserted between the transfers.

Figure 26-21 shows the timing diagram for two 4-bit transfers with  $CPHA = 1$  and  $CONT = 1$ .



**Figure 26-21. Example of continuous transfer ( $CPHA = 1$ ,  $CONT = 1$ )**

In Figure 26-21, the period length at the start of the next transfer is the sum of  $t_{ASC}$  and  $t_{CSC}$ ; that is, it does not include a half-clock period. The default settings for these provide a total of four system clocks. In many situations,  $t_{ASC}$  and  $t_{CSC}$  must be increased if a full half-clock period is required.

Switching CTARs between frames while using continuous selection can cause errors in the transfer. The CS signal must be negated before CTAR is switched.

When the  $CONT$  bit = 1 and the CS signals for the next transfer are different from the present transfer, the CS signals behave as if the  $CONT$  bit was not set.

#### NOTE

You must fill the TXFIFO with the number of entries that will be concatenated together under one PCS assertion for both master and slave before the TXFIFO becomes empty. For example; while transmitting in master mode, ensure that the last entry in the TXFIFO, after which TXFIFO becomes empty, has  $CONT = 0$  in the command frame.

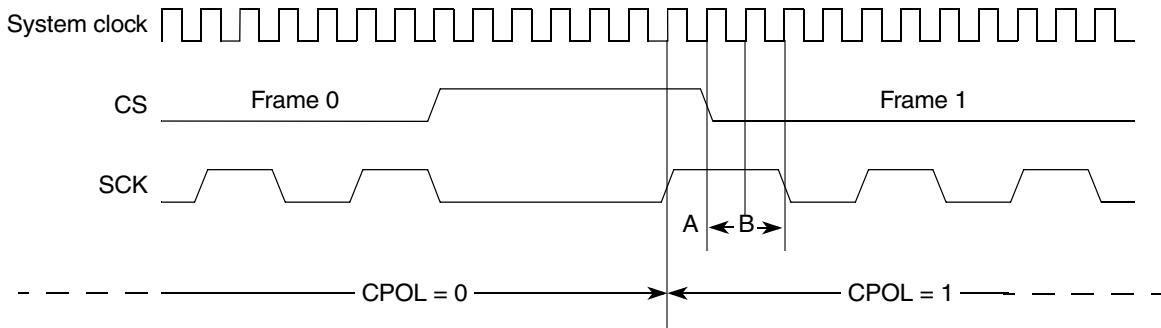
When operating in slave mode, ensure that when the last entry in the TXFIFO is completely transmitted (i.e. the corresponding TCF flag is asserted and TXFIFO is empty) the slave is deselected for any further serial communication; otherwise, an underflow error occurs.

#### 26.6.5.6 Clock polarity switching between DSPI transfers

If it is desired to switch polarity between non-continuous DSPI frames, the edge generated by the change in the idle state of the clock occurs one system clock before the assertion of the chip select for the next frame.

See [Section 26.5.4, DSPI Clock and Transfer Attributes Registers 0–5 \(DSPIx\\_CTARn\)](#).

In [Figure 26-22](#), time A shows the one clock interval. Time B is user programmable from a minimum of two system clocks.



**Figure 26-22. Polarity switching between frames**

### 26.6.6 Continuous serial communications clock

The DSPI provides the option of generating a continuous SCK signal for slave peripherals that require a continuous clock.

Continuous SCK is enabled by setting the CONT\_SCKE bit in the DSPIx\_MCR. Continuous SCK is valid in all configurations.

Continuous SCK is only supported for CPHA = 1. Setting CPHA = 0 is ignored if the CONT\_SCKE bit is set. Continuous SCK is supported for modified transfer format.

Clock and transfer attributes for the continuous SCK mode are set according to the following rules:

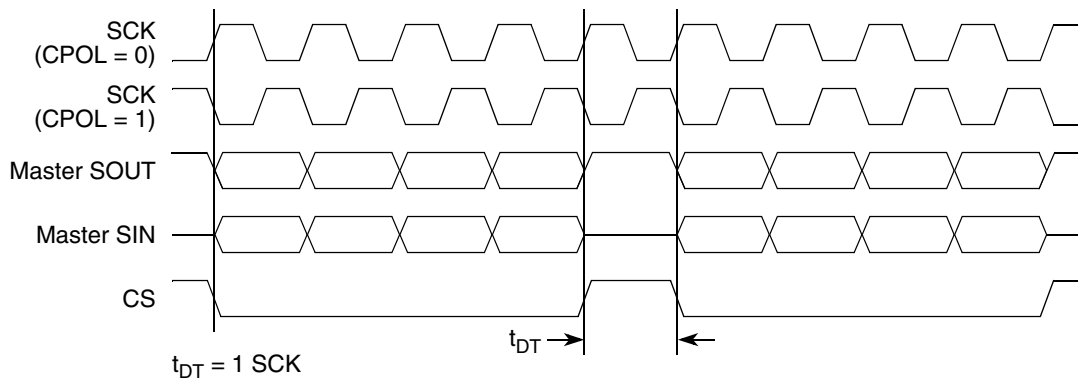
- The TX FIFO must be cleared before initiating any SPI configuration transfer.
- When the DSPI is in SPI configuration, CTAR0 is used initially. At the start of each SPI frame transfer, the CTAR specified by the CTAS for the frame should be CTAR0.
- In all configurations, the currently selected CTAR remains in use until the start of a frame with a different CTAR specified, or the continuous SCK mode is terminated.

The device is designed to use the same baud rate for all transfers made while using the continuous SCK. Switching clock polarity between frames while using continuous SCK can cause errors in the transfer. Continuous SCK operation is not guaranteed if the DSPI is put into module disable mode.

Enabling continuous SCK disables the CS to SCK delay and the After SCK delay. The delay after transfer is fixed at one SCK cycle. [Figure 26-23](#) shows timing diagram for continuous SCK format with continuous selection disabled.

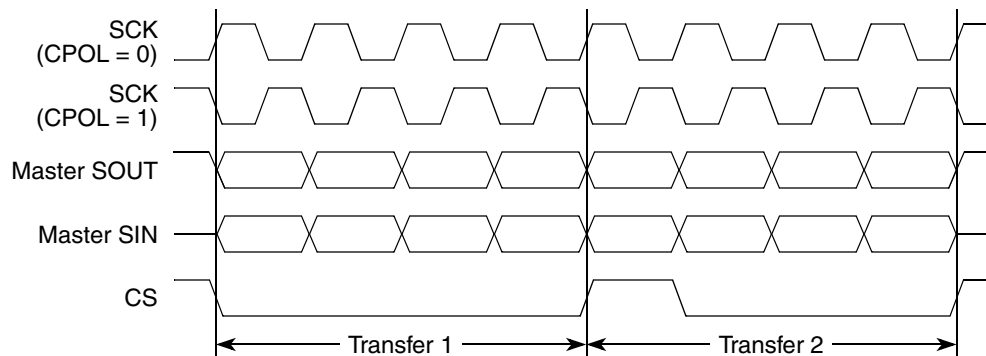
#### NOTE

When in Continuous SCK mode, always use CTAR0 for the SPI transfer, and clear the TXFIFO using the MCR[CLR\_TXF] field before initiating transfer.



**Figure 26-23. Continuous SCK timing diagram (CONT= 0)**

If the CONT bit in the TX FIFO entry is set, CS remains asserted between the transfers when the CS signal for the next transfer is the same as for the current transfer. [Figure 26-24](#) shows timing diagram for continuous SCK format with continuous selection enabled.



**Figure 26-24. Continuous SCK timing diagram (CONT=1)**

## 26.6.7 Interrupt/DMA requests

The DSPI has five conditions that can generate interrupt requests only, and two conditions that can generate interrupt or DMA requests.

[Table 26-32](#) lists the seven conditions.

**Table 26-32. Interrupt and DMA request conditions**

| Condition                                    | Flag | Interrupt | DMA |
|--|------|-----------|-----|
| End of transfer queue has been reached (EOQ) | EOQF | X         |     |
| TX FIFO is not full                          | TFFF | X         | X   |

**Table 26-32. Interrupt and DMA request conditions (continued)**

| Condition                            | Flag                | Interrupt | DMA |
|--------------------------------------|---------------------|-----------|-----|
| Current frame transfer is complete   | TCF                 | X         |     |
| TX FIFO underflow has occurred       | TFUF                | X         |     |
| RX FIFO is not empty                 | RFDF                | X         | X   |
| RX FIFO overflow occurred            | RFOF                | X         |     |
| A FIFO overrun occurred <sup>1</sup> | TFUF ORed with RFOF | X         |     |

<sup>1</sup> The FIFO overrun condition is created by ORing the TFUF and RFOF flags together.

Each condition has a flag bit and a request enable bit. The flag bits are described in the [Section 26.5.5, DSPI Status Register \(DSPIx\\_SR\)](#), and the request enable bits are described in the [Section 26.5.6, DSPI DMA / Interrupt Request Select and Enable Register \(DSPIx\\_RSER\)](#). The TX FIFO fill flag (TFFF) and RX FIFO drain flag (RFDF) generate interrupt requests or DMA requests depending on the TFFF\_DIRS and RFDF\_DIRS bits in the DSPIx\_RSER.

### 26.6.7.1 End of Queue Interrupt Request (EOQF)

The end of queue request indicates that the end of a transmit queue is reached. The end of queue request is generated when the EOQ bit in the executing SPI command is asserted and the EOQF\_RE bit in the DSPIx\_RSER is set. See the EOQ bit description in [Section 26.5.5, DSPI Status Register \(DSPIx\\_SR\)](#). See [Figure 26-16](#) and [Figure 26-17](#) that illustrate when EOQF is set.

### 26.6.7.2 Transmit FIFO Fill Interrupt or DMA Request (TFFF)

The transmit FIFO fill request indicates that the TX FIFO is not full. The transmit FIFO fill request is generated when the number of entries in the TX FIFO is less than the maximum number of possible entries, and the TFFF\_RE bit in the DSPIx\_RSER is set. The TFFF\_DIRS bit in the DSPIx\_RSER selects whether a DMA request or an interrupt request is generated.

### 26.6.7.3 Transfer Complete Interrupt Request (TCF)

The transfer complete request indicates the end of the transfer of a serial frame. The transfer complete request is generated at the end of each frame transfer when the TCF\_RE bit is set in the DSPIx\_RSER. See the TCF bit description in [Section 26.5.5, DSPI Status Register \(DSPIx\\_SR\)](#). See [Figure 26-16](#), and [Figure 26-17](#) that illustrate when TCF is set.

### 26.6.7.4 Transmit FIFO Underflow Interrupt Request (TFUF)

The transmit FIFO underflow request indicates that an underflow condition in the TX FIFO has occurred. The transmit underflow condition is detected only for DSPI modules operating in slave mode and SPI configuration. The TFUF bit is set when the TX FIFO of a DSPI operating in slave mode and SPI configuration is empty, and a transfer is initiated from an external SPI master. If the TFUF bit is set while the TFUF\_RE bit in the DSPIx\_RSER is set, an interrupt request is generated.

### 26.6.7.5 Receive FIFO Drain Interrupt or DMA Request (RFDF)

The receive FIFO drain request indicates that the RX FIFO is not empty. The receive FIFO drain request is generated when the number of entries in the RX FIFO is not zero, and the RFDF\_RE bit in the DSPLx\_RSER is set. The RFDF\_DIRS bit in the DSPLx\_RSER selects whether a DMA request or an interrupt request is generated.

### 26.6.7.6 Receive FIFO Overflow Interrupt Request (RFOF)

The receive FIFO overflow request indicates that an overflow condition in the RX FIFO has occurred. A receive FIFO overflow request is generated when RX FIFO and shift register are full and a transfer is initiated. The RFOF\_RE bit in the DSPLx\_RSER must be set for the interrupt request to be generated.

Depending on the state of the ROOE bit in the DSPLx\_MCR, the data from the transfer that generated the overflow is either ignored or shifted in to the shift register. If the ROOE bit is set, the incoming data is shifted in to the shift register. If the ROOE bit is negated, the incoming data is ignored.

### 26.6.7.7 FIFO Overrun Request (TFUF) or (RFOF)

The FIFO overrun request indicates that at least one of the FIFOs in the DSPI has exceeded its capacity. The FIFO overrun request is generated by logically ORing together the RX FIFO overflow and TX FIFO underflow signals.

## 26.6.8 Power saving features

The DSPI supports the following power-saving strategies:

- Module disable mode—clock gating of non-memory mapped logic
- Clock gating of slave interface signals and clock to memory-mapped logic

### 26.6.8.1 Module Disable mode

Module disable mode is a module-specific mode that the DSPI can enter to save power. Host software can initiate the module disable mode by writing a 1 to the MDIS bit in the DSPLx\_MCR. In module disable mode, the DSPI is in a dormant state, but the memory mapped registers are still accessible. Certain read or write operations have a different affect when the DSPI is in the module disable mode. Reading the RX FIFO pop register does not change the state of the RX FIFO. Likewise, writing to the TX FIFO push register does not change the state of the TX FIFO. Clearing either of the FIFOs does not have any effect in the module disable mode. Changes to the DIS\_TXF and DIS\_RXF fields of the DSPLx\_MCR does not have any affect in the module disable mode. In the module disable mode, all status bits and register flags in the DSPI return the correct values when read, but writing to them has no affect. Writing to the DSPLx\_TCR during module disable mode does not have an effect. Interrupt and DMA request signals cannot be cleared while in the module disable mode.

### 26.6.8.2 Slave interface signal gating

The DSPI module enable signal is used to gate slave interface signals such as address, byte enable, read/write, and data. This prevents toggling slave interface signals from consuming power unless the DSPI is accessed.

## 26.7 Initialization and application information

### 26.7.1 How to change queues

DSPI queues are not part of the DSPI module, but the DSPI includes features in support of queue management. Queues are primarily supported in SPI configuration. This section presents an example of how to change queues for the DSPI.

1. The last command word from a queue is executed. The EOQ bit in the command word is set to indicate to the DSPI that this is the last entry in the queue.
2. At the end of the transfer, corresponding to the command word with EOQ set is sampled, the EOQ flag (EOQF) in the DSPIx\_SR is set.
3. The setting of the EOQF flag disables both serial transmission and serial reception of data, putting the DSPI in the STOPPED state. The TXRXS bit is negated to indicate the STOPPED state.
4. The eDMA continues to fill TX FIFO until it is full or step 5 occurs.
5. Disable DSPI DMA transfers by disabling the DMA enable request for the DMA channel assigned to TX FIFO and RX FIFO. This is done by clearing the corresponding DMA enable request bits in the eDMA controller.
6. Ensure all received data in RX FIFO has been transferred to memory receive queue by reading the RXCNT in DSPIx\_SR or by checking RFDF in the DSPIx\_SR after each read operation of the DSPIx\_POPR.
7. Modify DMA descriptor of TX and RX channels for new queues.
8. Flush TX FIFO by writing a 1 to the CLR\_TXF bit in the DSPIx\_MCR register and flush the RX FIFO by writing a 1 to the CLR\_RXF bit in the DSPIx\_MCR register.
9. Clear transfer count either by setting CTCNT bit in the command word of the first entry in the new queue or via CPU writing directly to SPI\_TCNT field in the DSPIx\_TCR.
10. Enable DMA channel by enabling the DMA enable request for the DMA channel assigned to the DSPI TX FIFO, and RX FIFO by setting the corresponding DMA set enable request bit.
11. Enable serial transmission and serial reception of data by clearing the EOQF bit.

### 26.7.2 Baud rate settings

Table 26-33 shows the baud rate that is generated based on the combination of the baud rate prescaler PBR and the baud rate scaler BR in the DSPIx\_CTARs. The values are calculated at a 64 MHz system frequency.



Table 26-33. Baud rate values

|   |       | Baud rate divider prescaler values<br>(DSPI_CTAR[PBR]) |          |          |           |
|---|-------|--|----------|----------|-----------|
|   |       | 2  | 3        | 5        | 7         |
| Baud rate scaler values (DSPI_CTAR[BR]) | 2     | 16.0 MHz   | 10.7 MHz | 6.4 MHz  | 4.57 MHz  |
|   | 4     | 8 MHz  | 5.33 MHz | 3.2 MHz  | 2.28 MHz  |
|   | 6     | 5.33 MHz   | 3.56 MHz | 2.13 MHz | 1.52 MHz  |
|   | 8     | 4 MHz  | 2.67 MHz | 1.60 MHz | 1.15 MHz  |
|   | 16    | 2 MHz  | 1.33 MHz | 800 kHz  | 571 kHz   |
|   | 32    | 1 MHz  | 670 kHz  | 400 kHz  | 285 kHz   |
|   | 64    | 500 kHz  | 333 kHz  | 200 kHz  | 142 kHz   |
|   | 128   | 250 kHz  | 166 kHz  | 100 kHz  | 71.7 kHz  |
|   | 256   | 125 kHz  | 83.2 kHz | 50 kHz   | 35.71 kHz |
|   | 512   | 62.5 kHz   | 41.6 kHz | 25 kHz   | 17.86 kHz |
|   | 1024  | 31.2 kHz   | 20.8 kHz | 12.5 kHz | 8.96 kHz  |
|   | 2048  | 15.6 kHz   | 10.4 kHz | 6.25 kHz | 4.47 kHz  |
|   | 4096  | 7.81 kHz   | 5.21 kHz | 3.12 kHz | 2.23 kHz  |
|   | 8192  | 3.90 kHz   | 2.60 kHz | 1.56 kHz | 1.11 kHz  |
|   | 16384 | 1.95 kHz   | 1.31 kHz | 781 Hz   | 558 Hz    |
|   | 32768 | 979 Hz   | 653 Hz   | 390 Hz   | 279 Hz    |

### 26.7.3 Delay settings

Table 26-34 shows the values for the delay after transfer ( $t_{DT}$ ) that can be generated based on the prescaler values and the scaler values set in the DSPIx\_CTARs. The values calculated assume a 64 MHz system frequency.

Table 26-34. Delay values

|                                     |         | Delay prescaler values (DSPI_CTAR[PDT]) |             |              |              |
|-------------------------------------|---------|---|-------------|--------------|--------------|
|                                     |         | 1                                       | 3           | 5            | 7            |
| Delay scaler values (DSPI_CTAR[DT]) | 2       | 31.25 ns                                | 93.75 ns    | 156.25 ns    | 218.75 ns    |
|                                     | 4       | 62.5 ns                                 | 187.5 ns    | 312.5 ns     | 437.5 ns     |
|                                     | 8       | 125 ns                                  | 375 ns      | 625 ns       | 875 ns       |
|                                     | 16      | 250 ns                                  | 750 ns      | 1.25 $\mu$ s | 1.75 $\mu$ s |
|                                     | 32      | 0.5 $\mu$ s                             | 1.5 $\mu$ s | 2.5 $\mu$ s  | 3.5 $\mu$ s  |
|                                     | 64      | 1 $\mu$ s                               | 3 $\mu$ s   | 5 $\mu$ s    | 7 $\mu$ s    |
|                                     | 128     | 2 $\mu$ s                               | 6 $\mu$ s   | 10 $\mu$ s   | 14 $\mu$ s   |
|                                     | 256     | 4 $\mu$ s                               | 12 $\mu$ s  | 20 $\mu$ s   | 28 $\mu$ s   |
|                                     | 512     | 8 $\mu$ s                               | 24 $\mu$ s  | 40 $\mu$ s   | 56 $\mu$ s   |
|                                     | 1024    | 16 $\mu$ s                              | 48 $\mu$ s  | 80 $\mu$ s   | 112 $\mu$ s  |
|                                     | 2048    | 32 $\mu$ s                              | 96 $\mu$ s  | 160 $\mu$ s  | 224 $\mu$ s  |
|                                     | 4096    | 64 $\mu$ s                              | 192 $\mu$ s | 320 $\mu$ s  | 448 $\mu$ s  |
|                                     | 8192    | 128 $\mu$ s                             | 384 $\mu$ s | 640 $\mu$ s  | 896 $\mu$ s  |
|                                     | 16384   | 256 $\mu$ s                             | 768 $\mu$ s | 1.28 ms      | 1.79 ms      |
|                                     | 32768   | 512 $\mu$ s                             | 1.54 ms     | 2.56 ms      | 3.58 ms      |
| 65536                               | 1.02 ms | 3.07 ms                                 | 5.12 ms     | 7.17 ms      |              |

### 26.7.4 Calculation of FIFO pointer addresses

The user has complete visibility of the TX and RX FIFO contents through the FIFO registers, and valid entries can be identified through a memory mapped pointer and a memory mapped counter for each FIFO. The pointer to the first-in entry in each FIFO is memory mapped. For the TX FIFO the first-in pointer is the transmit next pointer (TXNXTPTR). For the RX FIFO the first-in pointer is the pop next pointer (POPNXTPTR).

See Section 26.6.3.4, [Transmit First In First Out \(TX FIFO\) buffering mechanism](#), and Section 26.6.3.5, [Receive First In First Out \(RX FIFO\) buffering mechanism](#), for details on the FIFO operation. The TX FIFO is chosen for the illustration, but the concepts carry over to the RX FIFO.

Figure 26-25 illustrates the concept of first-in and last-in FIFO entries along with the FIFO counter.

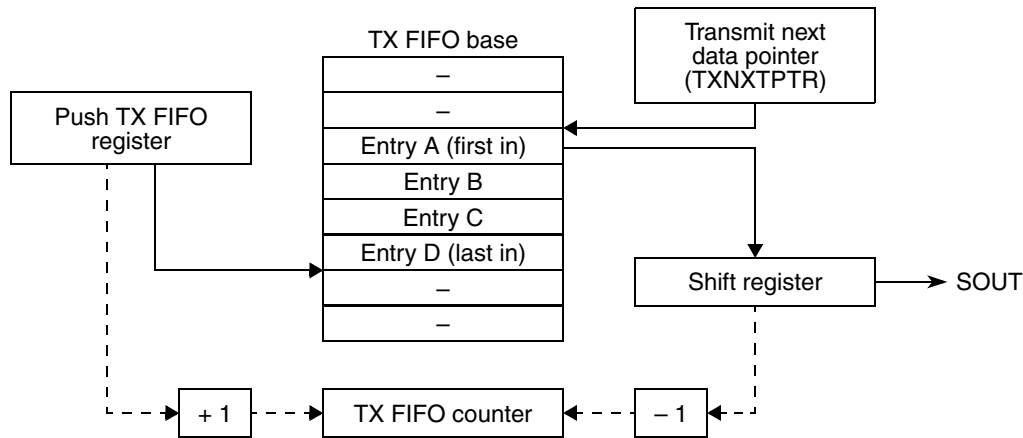


Figure 26-25. TX FIFO pointers and counter

#### 26.7.4.1 Address calculation for the first-in entry and last-in entry in the TX FIFO

The memory address of the first-in entry in the TX FIFO is computed by the following equation:

$$\text{First-in entry address} = \text{TXFIFO base} + 4 (\text{TXNXPTR})$$

The memory address of the last-in entry in the TX FIFO is computed by the following equation:

$$\text{Last-in entry address} = \text{TXFIFO base} + 4 \times [(\text{TXCTR} + \text{TXNXPTR} - 1) \text{ modulo TXFIFO depth}]$$

where:

TXFIFO base = base address of transmit FIFO

TXCTR = transmit FIFO counter

TXNXPTR = transmit next pointer

TX FIFO depth = transmit FIFO depth, implementation specific

#### 26.7.4.2 Address calculation for the first-in entry and last-in entry in the RX FIFO

The memory address of the first-in entry in the RX FIFO is computed by the following equation:

$$\text{First-in entry address} = \text{RXFIFO base} + 4 \times (\text{POPXPTR})$$

The memory address of the last-in entry in the RX FIFO is computed by the following equation:

$$\text{Last-in entry address} = \text{RXFIFO base} + 4 \times [(\text{RXCTR} + \text{POPXPTR} - 1) \text{ modulo RXFIFO depth}]$$

where:

RXFIFO base = base address of receive FIFO


RXCTR = receive FIFO counter

POPNXTPTR = pop next pointer

RX FIFO depth = receive FIFO depth, implementation specific



# —— Timers ——



This page is intentionally left blank.

# Chapter 27

## Timers

### 27.1 Introduction

This chapter describes the timer modules implemented on the microcontroller:

- [System Timer Module \(STM\)](#)
- [Enhanced Modular IO Subsystem \(eMIOS\)](#)
- [Periodic Interrupt Timer \(PIT\)](#)

The microcontroller also has a [Real Time Clock / Autonomous Periodic Interrupt \(RTC/API\)](#) module. The main purpose of this is to provide a periodic device wakeup source.

### 27.2 Technical overview

This section gives a technical overview of each of the timers as well as detailing the pins that can be used to access the timer peripherals if applicable.

[Figure 27-1](#) details the interaction between the timers and the eDMA, INTC, CTU, and ADC.



**Figure 27-1. Interaction between timers and relevant peripherals**



## 27.2.1 Overview of the STM

The STM is a 32-bit free running up-counter clocked by the system clock with a configurable 8-bit clock prescaler (divide by 1 to 256). The counter is disabled out of reset and must therefore be enabled by software prior to use. The counter value can be read at any time.

The STM has four 32-bit compare channels. Each channel can generate a unique interrupt on an exact match event with the free running counter.

The STM is often used to analyze code execution times. By starting the STM and reading the timer before and after a task or function, you can make an accurate measurement of the time taken in clock cycles to perform the task.

The STM can be configured to stop (freeze) or continue to run in debug mode and is available for use in all operating mode where the system clock is present (not STANDBY or certain STOP mode configurations)

There are no external pins associated with the STM.

## 27.2.2 Overview of the eMIOS

Each eMIOS offers a combination of PWM, Output Capture, and Input Compare functions. There are different types of channel implemented and not every channel supports every eMIOS function. The channel functionality also differs between each eMIOS module. See [Section 27.4, Enhanced Modular IO Subsystem \(eMIOS\)](#), for more details.

Each channel has its own independent 16-bit counter. To allow synchronization between channels, there are a number of shared counter busses that can be used as a common timing reference. These counter buses can be used in combination with the individual channel counters to provide advanced features such as centre aligned PWM with dead time insertion.

Once configured, the eMIOS needs very little CPU intervention. Interrupts, eDMA requests, and CTU trigger requests can be raised based on eMIOS flag and timeout events.

The eMIOS is clocked from the system clock via peripheral clock group 3 (with a maximum permitted clock frequency of 64 MHz). The eMIOS can be used in all modes where the system clock is available (which excludes STANDBY mode and STOP mode when the system clock is turned off). The eMIOS has an option to allow the eMIOS counters to freeze or to continue running in debug mode.

The CTU allows an eMIOS event to trigger a single ADC conversion via the CTU without any CPU intervention. Without the CTU, the eMIOS would have to trigger an interrupt request. The respective ISR would then perform a software triggered ADC conversion. This not only uses CPU resource, but also increases the latency between the eMIOS event and the ADC trigger.

The eMIOS Output Pulse Width Modulation with Trigger mode (see [Section 27.4.4.1.1.12, Output Pulse Width Modulation with Trigger \(OPWMT\) mode](#)) allows a customizable trigger point to be defined at any point in the waveform period. This is extremely useful for LED lighting applications where the trigger can be set to a point where the PWM output is high but after the initial inrush current to the LED has occurred. The PWM trigger can then cause the CTU to perform a single ADC conversion, which in turn measures the operating conditions of the LED to ensure it is working within specification. A watchdog feature on

the ADC allows channels to be monitored. If the results fall outside a specific range, an interrupt is triggered. This means that all of the measurement is without CPU intervention if the results are within range.

To make it easier to plan which pins to use for the eMIOS, [Table 27-1](#) and [Table 27-2](#) show the eMIOS channel numbers that are available on each pin. The color shading matches the channel configuration diagram in the eMIOS section.

Table 27-1. eMIOS\_0 channel to pin mapping

| Channel | Pin function  |       |        | Channel | Pin function   |        |       |
|---------|---------------|-------|--------|---------|----------------|--------|-------|
|         | ALT1          | ALT2  | ALT3   |         | ALT1           | ALT2   | ALT3  |
| UC[0]   | PA[0]         |       | PA[14] | UC[16]  | PE[0]          |        |       |
| UC[1]   | PA[1]         |       | PA[15] | UC[17]  | PE[1]          |        |       |
| UC[2]   | PA[2]         |       |        | UC[18]  | PE[2]          |        |       |
| UC[3]   | PA[3], PB[11] | PC[8] |        | UC[19]  | PE[3]          |        |       |
| UC[4]   | PA[4], PB[12] |       |        | UC[20]  | PE[4]          |        |       |
| UC[5]   | PA[5], PB[13] |       |        | UC[21]  | PE[5]          |        |       |
| UC[6]   | PA[6], PB[14] |       |        | UC[22]  | PE[6], PF[5]   | PE[8]  |       |
| UC[7]   | PA[7], PB[15] | PC[9] |        | UC[23]  | PE[7], PF[6]   | PE[9]  |       |
| UC[8]   | PA[8]         |       |        | UC[24]  | PE[11], PG[10] | PD[12] |       |
| UC[9]   | PA[9]         |       |        | UC[25]  | PG[11]         | PD[13] |       |
| UC[10]  | PA[10], PF[0] |       |        | UC[26]  | PG[12]         | PD[14] |       |
| UC[11]  | PA[11], PF[1] |       |        | UC[27]  | PG[13]         | PD[15] |       |
| UC[12]  | PC[12], PF[2] |       |        | UC[28]  | PI[0]          | PA[12] |       |
| UC[13]  | PC[13], PF[3] |       | PA[0]  | UC[29]  | PI[1]          | PA[13] |       |
| UC[14]  | PC[14], PF[4] | PA[8] |        | UC[30]  | PI[2]          | PB[0]  | PB[2] |
| UC[15]  | PC[15]        |       |        | UC[31]  | PB[3], PI[3]   | PB[1]  |       |

Table 27-2. eMIOS\_1 channel to pin mapping

| Channel | Pin function  |        |        | Channel | Pin function |        |        |
|---------|---------------|--------|--------|---------|--------------|--------|--------|
|         | ALT1          | ALT2   | ALT3   |         | ALT1         | ALT2   | ALT3   |
| UC[0]   | PG[14]        |        |        | UC[16]  | PG[7]        |        |        |
| UC[1]   | PF[9], PG[15] |        |        | UC[17]  | PG[8]        |        | PH[15] |
| UC[2]   | PH[0]         |        | PF[10] | UC[18]  | PG[9]        | PJ[4]  |        |
| UC[3]   | PH[1]         | PF[11] |        | UC[19]  |              | PE[12] |        |
| UC[4]   | PF[15], PH[2] |        |        | UC[20]  |              | PE[13] |        |
| UC[5]   | PH[3]         |        | PH[11] | UC[21]  |              | PE[14] |        |
| UC[6]   | PH[4]         |        |        | UC[22]  |              | PE[15] |        |
| UC[7]   | PH[5]         |        |        | UC[23]  |              | PG[0]  |        |
| UC[8]   | PH[6]         |        |        | UC[24]  |              | PG[1]  |        |
| UC[9]   | PH[7]         |        |        | UC[25]  | PF[12]       |        | PH[12] |
| UC[10]  | PH[8]         |        |        | UC[26]  | PF[13]       |        | PH[13] |
| UC[11]  | PG[2]         |        |        | UC[27]  |              | PF[14] | PH[14] |
| UC[12]  | PG[3]         |        |        | UC[28]  | PI[4]        | PC[6]  |        |

Table 27-2. eMIOS\_1 channel to pin mapping (continued)

| Channel | Pin function |      |      | Channel | Pin function |        |        |
|---------|--------------|------|------|---------|--------------|--------|--------|
|         | ALT1         | ALT2 | ALT3 |         | ALT1         | ALT2   | ALT3   |
| UC[13]  | PG[4]        |      |      | UC[29]  | PI[5]        | PC[7]  |        |
| UC[14]  | PG[5]        |      |      | UC[30]  | PI[6]        | PG[7]  | PE[10] |
| UC[15]  | PG[6]        |      |      | UC[31]  | PC[4], PI[7] | PG[10] |        |

### 27.2.3 Overview of the PIT

The PIT module consists of 8 Periodic Interrupt Timers (PITs) clocked from the system clock.

Out of reset, the PIT is disabled. There is a global disable control bit for all of the PIT timers. Before using the timers, software must clear the appropriate disabled bit. Each of the PIT timers are effectively standalone entities and each have their own timer and control registers.

The PIT timers are 32-bit count down timers. To use them, you must first program an initial value into the LDVAL register. The timer will then start to count down and can be read at any time. Once the timer reaches 0x0000\_0000, a flag is set and the previous value is automatically reloaded into the LDVAL register and the countdown starts again. The flag event can be routed to a dedicated INTC interrupt if desired.

The PIT is also used to trigger other events:

- Four of the PIT channels can be used as an eDMA trigger
- Two PIT channels can be used to trigger a CTU ADC conversion (single)
- Two PIT channels (one per ADC module) can be used to directly trigger injected conversions on the ADC

The timers can be configured to stop (freeze) or to continue to run in debug mode. The PIT is available in all modes where a system clock is generated.

There are no external pins associated with the PIT.

## 27.3 System Timer Module (STM)

### 27.3.1 Introduction

#### 27.3.1.1 Overview

The System Timer Module (STM) is a 32-bit timer designed to support commonly required system and application software timing functions. The STM includes a 32-bit up counter and four 32-bit compare channels with a separate interrupt source for each channel. The counter is driven by the system clock divided by an 8-bit prescale value (1 to 256).

### 27.3.1.2 Features

The STM has the following features:

- One 32-bit up counter with 8-bit prescaler
- Four 32-bit compare channels
- Independent interrupt source for each channel
- Counter can be stopped in debug mode

### 27.3.1.3 Modes of operation

The STM supports two device modes of operation: normal and debug. When the STM is enabled in normal mode, its counter runs continuously. In debug mode, operation of the counter is controlled by the FRZ bit in the STM\_CR register. If the FRZ bit is set, the counter is stopped in debug mode, otherwise it continues to run.

## 27.3.2 External signal description

The STM does not have any external interface signals.

## 27.3.3 Memory map and register definition

The STM programming model has fourteen 32-bit registers. The STM registers can only be accessed using 32-bit (word) accesses. Attempted references using a different size or to a reserved address generates a bus error termination.

### 27.3.3.1 Memory map

The STM memory map is shown in [Table 27-3](#).

**Table 27-3. STM memory map**

| Base address: 0xFFF3_C000 |   |                             |
|---------------------------|---|-----------------------------|
| Address offset            | Register                                    | Location                    |
| 0x0000                    | STM Control Register (STM_CR)               | <a href="#">on page 651</a> |
| 0x0004                    | STM Counter Value (STM_CNT)                 | <a href="#">on page 652</a> |
| 0x0008–0x000C             | Reserved                                    |                             |
| 0x0010                    | STM Channel 0 Control Register (STM_CCR0)   | <a href="#">on page 652</a> |
| 0x0014                    | STM Channel 0 Interrupt Register (STM_CIR0) | <a href="#">on page 653</a> |
| 0x0018                    | STM Channel 0 Compare Register (STM_CMP0)   | <a href="#">on page 654</a> |
| 0x001C                    | Reserved                                    |                             |
| 0x0020                    | STM Channel 1 Control Register (STM_CCR1)   | <a href="#">on page 652</a> |
| 0x0024                    | STM Channel 1 Interrupt Register (STM_CIR1) | <a href="#">on page 653</a> |
| 0x0028                    | STM Channel 1 Compare Register (STM_CMP1)   | <a href="#">on page 654</a> |

Table 27-3. STM memory map (continued)

| Base address: 0xFFF3_C000 |   |             |
|---------------------------|---|-------------|
| Address offset            | Register                                    | Location    |
| 0x002C                    | Reserved                                    |             |
| 0x0030                    | STM Channel 2 Control Register (STM_CCR2)   | on page 652 |
| 0x0034                    | STM Channel 2 Interrupt Register (STM_CIR2) | on page 653 |
| 0x0038                    | STM Channel 2 Compare Register (STM_CMP2)   | on page 654 |
| 0x003C                    | Reserved                                    |             |
| 0x0040                    | STM Channel 3 Control Register (STM_CCR3)   | on page 652 |
| 0x0044                    | STM Channel 3 Interrupt Register (STM_CIR3) | on page 653 |
| 0x0048                    | STM Channel 3 Compare Register (STM_CMP3)   | on page 654 |
| 0x004C–0x3FFF             | Reserved                                    |             |

### 27.3.3.2 Register descriptions

The following sections detail the individual registers within the STM programming model.

#### 27.3.3.2.1 STM Control Register (STM\_CR)

The STM Control Register (STM\_CR) includes the prescale value, freeze control, and timer enable bits.

Offset: 0x000 Access: Read/Write

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       |     |    |    |    |    |    |    |    |    |    |    |    |    |    |     |     |
|-------|-----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|-----|
|       | 16  | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30  | 31  |
| R     | CPS |    |    |    |    |    |    |    | 0  | 0  | 0  | 0  | 0  | 0  |     |     |
| W     |     |    |    |    |    |    |    |    |    |    |    |    |    |    | FRZ | TEN |
| Reset | 0   | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 0   |

Figure 27-2. STM Control Register (STM\_CR)

Table 27-4. STM\_CR field descriptions

| Field | Description  |
|-------|--|
| CPS   | Counter Prescaler. Selects the clock divide value for the prescaler (1–256).<br>0x00 = Divide system clock by 1<br>0x01 = Divide system clock by 2<br>...<br>0xFF = Divide system clock by 256 |
| FRZ   | Freeze. Allows the timer counter to be stopped when the device enters debug mode.<br>0 = STM counter continues to run in debug mode.<br>1 = STM counter is stopped in debug mode.              |
| TEN   | Timer Counter Enabled.<br>0 = Counter is disabled.<br>1 = Counter is enabled.  |

### 27.3.3.2.2 STM Count Register (STM\_CNT)

The STM Count Register (STM\_CNT) holds the timer count value.

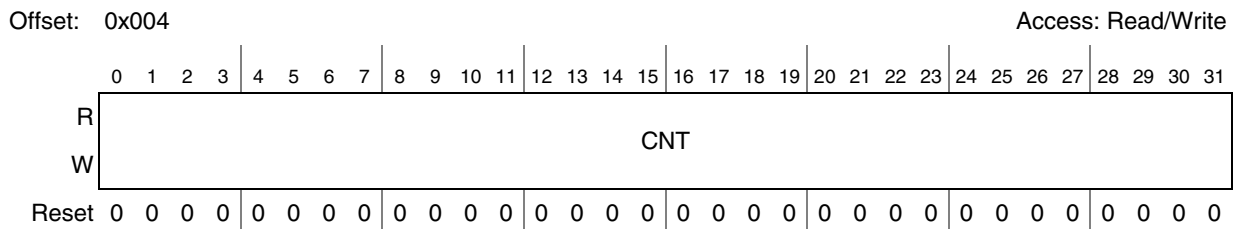


Figure 27-3. STM Count Register (STM\_CNT)

Table 27-5. STM\_CNT field descriptions

| Field | Description   |
|-------|---|
| CNT   | Timer count value used as the time base for all channels. When enabled, the counter increments at the rate of the system clock divided by the prescale value. |

### 27.3.3.2.3 STM Channel Control Register (STM\_CCRn)

The STM Channel Control Register (STM\_CCRn) has the enable bit for channel n of the timer.

Offset:  $0x10+0x10 \times n$ 

Access: Read/Write

|       |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |     |
|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|
|       | 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 | 14 | 15  |
| R     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   |
| W     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |     |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   |
|       | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31  |
| R     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | CEN |
| W     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |     |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   |

Figure 27-4. STM Channel Control Register (STM\_CCRn)

Table 27-6. STM\_CCRn field descriptions

| Field | Description  |
|-------|--|
| CEN   | Channel Enable.<br>0 = The channel is disabled.<br>1 = The channel is enabled. |

### 27.3.3.2.4 STM Channel Interrupt Register (STM\_CIRn)

The STM Channel Interrupt Register (STM\_CIRn) has the interrupt flag for channel n of the timer.

Offset:  $0x14+0x10 \times n$ 

Access: Read/Write

|       |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |     |
|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|
|       | 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 | 14 | 15  |
| R     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   |
| W     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |     |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   |
|       | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31  |
| R     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | CIF |
| W     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | w1c |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   |

Figure 27-5. STM Channel Interrupt Register (STM\_CIRn)

Table 27-7. STM\_CIRn field descriptions

| Field | Description   |
|-------|---|
| CIF   | Channel Interrupt Flag<br>0 = No interrupt request.<br>1 = Interrupt request due to a match on the channel. |

### 27.3.3.2.5 STM Channel Compare Register (STM\_CMPn)

The STM channel compare register (STM\_CMPn) holds the compare value for channel n.

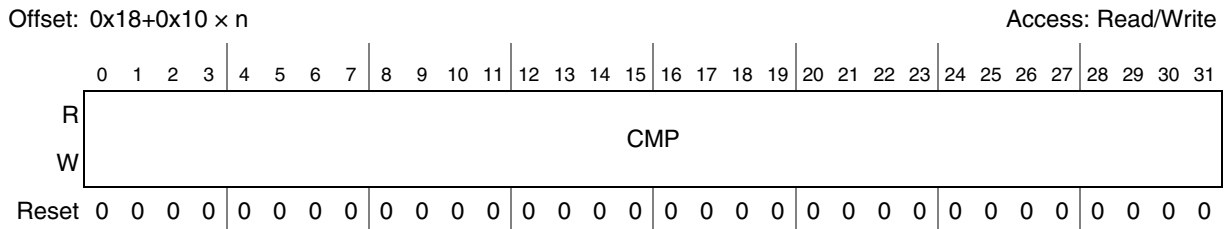


Figure 27-6. STM Channel Compare Register (STM\_CMPn)

Table 27-8. STM\_CMPn field descriptions

| Field | Description   |
|-------|---|
| CMP   | Compare value for channel n. If the STM_CCRn[CEN] bit is set and the STM_CMPn register matches the STM_CNT register, a channel interrupt request is generated and the STM_CIRn[CIF] bit is set. |

## 27.3.4 Functional description

The System Timer Module (STM) is a 32-bit timer designed to support commonly required system and application software timing functions. The STM includes a 32-bit up counter and four 32-bit compare channels with a separate interrupt source for each channel.

The STM has one 32-bit up counter (STM\_CNT) that is used as the time base for all channels. When enabled, the counter increments at the system clock frequency divided by a prescale value. The STM\_CR[CPS] field sets the divider to any value in the range from 1 to 256. The counter is enabled with the STM\_CR[TEN] bit. When enabled in normal mode the counter continuously increments. When enabled in debug mode the counter operation is controlled by the STM\_CR[FRZ] bit. When the STM\_CR[FRZ] bit is set, the counter is stopped in debug mode, otherwise it continues to run in debug mode. The counter rolls over at  $0xFFFF\_FFFF$  to  $0x0000\_0000$  with no restrictions at this boundary.

The STM has four identical compare channels. Each channel includes a channel control register (STM\_CCRn), a channel interrupt register (STM\_CIRn) and a channel compare register (STM\_CMPn). The channel is enabled by setting the STM\_CCRn[CEN] bit. When enabled, the channel will set the STM\_CIR[CIF] bit and generate an interrupt request when the channel compare register matches the timer counter. The interrupt request is cleared by writing a 1 to the STM\_CIRn[CIF] bit. A write of 0 to the STM\_CIRn[CIF] bit has no effect.

### NOTE

STM counter does not advance when the system clock is stopped.



## 27.4 Enhanced Modular IO Subsystem (eMIOS)

### 27.4.1 Introduction

#### 27.4.1.1 Overview of the eMIOS module

The eMIOS provides functionality to generate or measure time events. The eMIOS uses timer channels that are reduced versions of the unified channel (UC) module used on MPC555x devices. Each channel provides a subset of the functionality available in the unified channel, at a resolution of 16 bits, and provides a user interface that is consistent with previous eMIOS implementations.

#### 27.4.1.2 Features of the eMIOS module

- Two eMIOS blocks with 32 channels each
  - All 64 channels with OPWMT, which can be connected to the CTU
  - Both eMIOS blocks can be synchronized
- One global prescaler
- 16-bit data registers
- $10 \times 16$ -bit wide counter buses
  - Counter buses B, C, D, and E can be driven by Unified Channel 0, 8, 16, and 24, respectively
  - Counter bus A is driven by the Unified Channel 23
  - Several channels have their own time base, alternative to the counter buses
  - Shared timebases through the counter buses
  - Synchronization among timebases
- Control and Status bits grouped in a single register
- Shadow FLAG register
- State of the UC can be frozen for debug purposes
- Motor control capability

#### 27.4.1.3 Modes of operation

The Unified Channels can be configured to operate in the following modes:

- General purpose input/output
- Single Action Input Capture
- Single Action Output Compare
- Input Pulse Width Measurement
- Input Period Measurement
- Double Action Output Compare
- Modulus Counter
- Modulus Counter Buffered

- Output Pulse Width and Frequency Modulation Buffered
- Output Pulse Width Modulation Buffered
- Output Pulse Width Modulation with Trigger
- Center Aligned Output Pulse Width Modulation Buffered

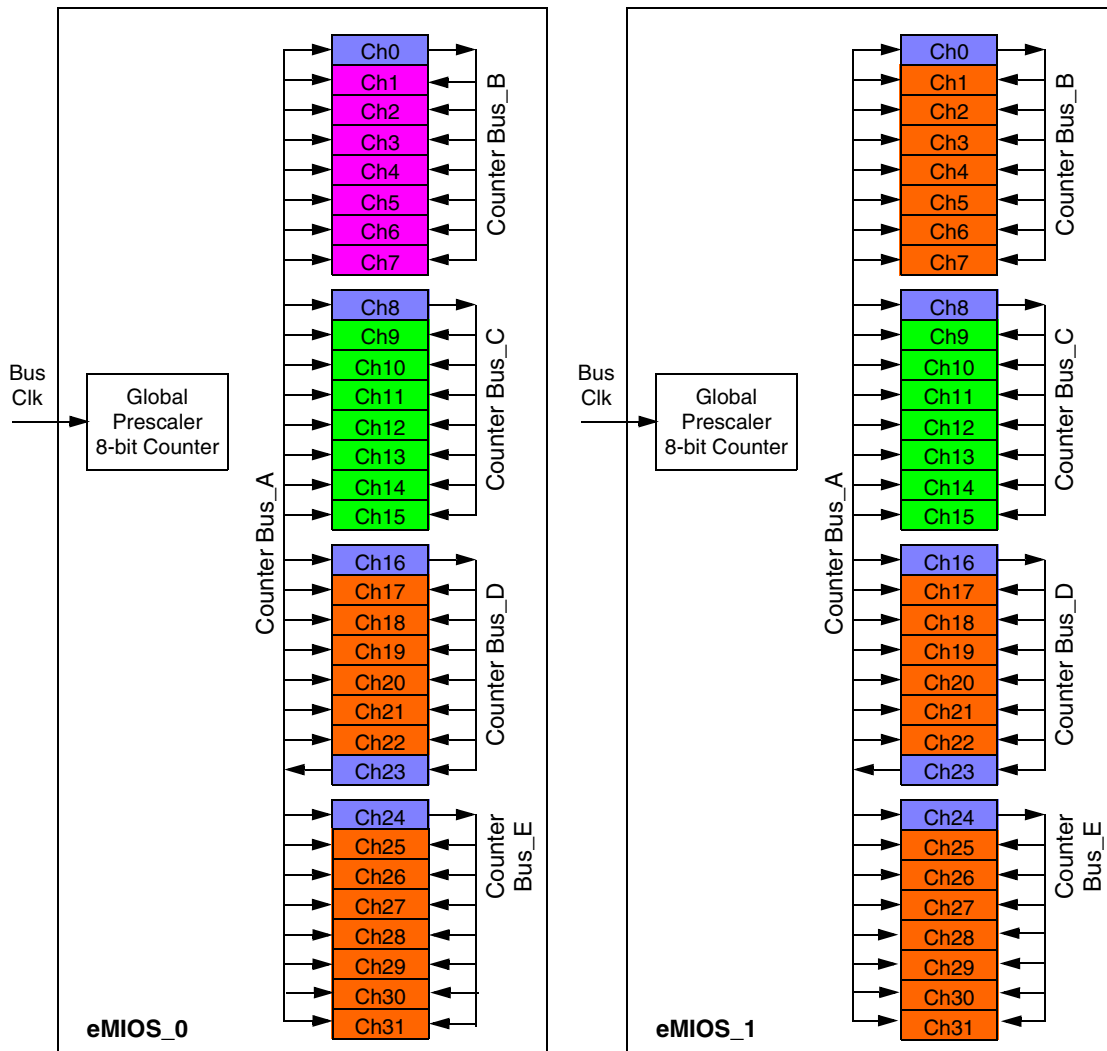
These modes are described in [Section 27.4.4.1.1, UC modes of operation](#).

Each channel can have a specific set of modes implemented, according to device requirements.

If an unimplemented mode (reserved) is selected, the results are unpredictable such as writing a reserved value to MODE[0:6] in [Section 27.4.3.2.8, eMIOS UC Control Register \(EMIOSC\[n\]\)](#).

#### **27.4.1.4 Channel implementation**

[Figure 27-7](#) shows the channel configuration of the eMIOS blocks.



**Channel Functionality**

**TYPE G**

- MCB
- OPWMT
- OPWMB
- OPWFMB
- OPWMCB
- IPWM, IPM
- DAOC
- SAIC, SAOC
- GPIO

**TYPE X**

- MC, MCB
- OPWMT
- OPWMB
- OPWFMB
- SAIC, SAOC
- GPIO

**TYPE H**

- OPWMT
- OPWMB
- IPWM, IPM
- DAOC
- SAIC, SAOC
- GPIO

**TYPE Y**

- OPWMT
- OPWMB
- SAIC, SAOC
- GPIO

**Key**

|        |  |
|--------|--|
| DAOC   | Dual Action Output Compare                           |
| GPIO   | General Purpose Input Output                         |
| IPM    | Input Period Measurement                             |
| IPWM   | Input Pulse Width Measurement                        |
| MC     | Modulus Counter                                      |
| MCB    | Buffered Modulus Counter                             |
| OPWMB  | Buffered Output Pulse Width Modulation               |
| OPWMT  | Buffered Output Pulse Width Modulation with Trigger  |
| OPWFMB | Buffered Output Pulse Width and Frequency Modulation |
| OPWMCB | Center Aligned Output PWM Buffered with Dead Time    |
| SAIC   | Single Action Input Capture                          |
| SAOC   | Single Action Output Compare                         |

Figure 27-7. Channel configuration

### 27.4.1.4.1 Channel mode selection

Channel modes are selected using the mode selection bits MODE[0:6] in the eMIOS UC Control Register (EMIOSC[n]). [Table 27-21](#) provides the specific mode selection settings for the eMIOS implementation on this device.

## 27.4.2 External signal description

For information on eMIOS external signals on this device, please refer to [Chapter 4, Signal description](#).

## 27.4.3 Memory map and register description

### 27.4.3.1 Memory maps

The overall address map organization is shown in [Table 27-9](#).

#### 27.4.3.1.1 Unified Channel memory map

Table 27-9. eMIOS memory map

| Base addresses:<br>0xC3FA_0000 (eMIOS_0)<br>0xC3FA_4000 (eMIOS_1) |   |                             |
|---|---|-----------------------------|
| Address offset  | Description                                       | Location                    |
| 0x000–0x003   | eMIOS Module Configuration Register (EMIOSMCR)    | <a href="#">on page 659</a> |
| 0x004–0x007   | eMIOS Global FLAG (EMIOSGFLAG) Register           | <a href="#">on page 660</a> |
| 0x008–0x00B   | eMIOS Output Update Disable (EMIOSOUDIS) Register | <a href="#">on page 661</a> |
| 0x00C–0x00F   | eMIOS Disable Channel (EMIOSUCDIS) Register       | <a href="#">on page 662</a> |
| 0x010–0x01F   | Reserved  | —                           |
| 0x020–0x11F   | Channel [0]<br>to<br>Channel [7]                  | —                           |
| 0x120–0x21F   | Channel [8]<br>to<br>Channel [15]                 | —                           |
| 0x220–0x31F   | Channel [16]<br>to<br>Channel [23]                | —                           |
| 0x320–0x41F   | Channel [24]<br>to<br>Channel [31]                | —                           |
| 0x420–0xFFFF  | Reserved  | —                           |

Addresses of Unified Channel registers are specified as offsets from the channel's base address; otherwise the eMIOS base address is used as reference.

Table 27-10 describes the Unified Channel memory map.

**Table 27-10. Unified Channel memory map**

| UC[n] base address | Description                                  | Location    |
|--------------------|--|-------------|
| 0x00               | eMIOS UC A Register (EMIOSA[n])              | on page 663 |
| 0x04               | eMIOS UC B Register (EMIOSB[n])              | on page 663 |
| 0x08               | eMIOS UC Counter Register (EMIOSCNT[n])      | on page 664 |
| 0x0C               | eMIOS UC Control Register (EMIOSC[n])        | on page 665 |
| 0x10               | eMIOS UC Status Register (EMIOSS[n])         | on page 669 |
| 0x14               | eMIOS UC Alternate A Register (EMIOSALTA[n]) | on page 670 |
| 0x18–0x1F          | Reserved                                     | —           |

### 27.4.3.2 Register description

All control registers are 32 bits wide. Data registers and counter registers are 16 bits wide.

#### 27.4.3.2.1 eMIOS Module Configuration Register (EMIOSMCR)

The EMIOSMCR contains global control bits for the eMIOS block.

Address: eMIOS base address +0x00

|       |      |      |     |      |    |       |    |    |    |    |    |    |    |    |    |    |
|-------|------|------|-----|------|----|-------|----|----|----|----|----|----|----|----|----|----|
|       | 0    | 1    | 2   | 3    | 4  | 5     | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 | 14 | 15 |
| R     | 0    |      |     | GTBE | 0  | GPREN | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |      | MDIS | FRZ |      |    |       |    |    |    |    |    |    |    |    |    |    |
| Reset | 0    | 0    | 0   | 0    | 0  | 0     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
|       | 16   | 17   | 18  | 19   | 20 | 21    | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| R     | GPRE |      |     |      |    |       |    |    | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |      |      |     |      |    |       |    |    |    |    |    |    |    |    |    |    |
| Reset | 0    | 0    | 0   | 0    | 0  | 0     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |

**Figure 27-8. eMIOS Module Configuration Register (EMIOSMCR)**

**Table 27-11. EMIOSMCR field descriptions**

| Field | Description   |
|-------|---|
| MDIS  | Module Disable<br>Puts the eMIOS in low power mode. The MDIS bit is used to stop the clock of the block, except the access to registers EMIOSMCR, EMIOSOUDIS, and EMIOSUCDIS.<br>1 = Enter low power mode<br>0 = Clock is running |

Table 27-11. EMIOSMCR field descriptions (continued)

| Field | Description  |
|-------|--|
| FRZ   | Freeze<br>Enables the eMIOS to freeze the registers of the Unified Channels when Debug Mode is requested at MCU level. Each Unified Channel should have FREN bit set in order to enter freeze state. While in Freeze state, the eMIOS continues to operate to allow the MCU access to the Unified Channels registers. The Unified Channel will remain frozen until the FRZ bit is written to 0 or the MCU exits Debug mode or the Unified Channel FREN bit is cleared.<br>1 = Stops Unified Channels operation when in Debug mode and the FREN bit is set in the EMIOSS[n] register<br>0 = Exit freeze state |
| GTBE  | Global Time Base Enable<br>The GTBE bit is used to export a Global Time Base Enable from the module and provide a method to start time bases of several blocks simultaneously.<br>1 = Global Time Base Enable Out signal asserted<br>0 = Global Time Base Enable Out signal negated<br><b>Note:</b> The Global Time Base Enable input pin controls the internal counters. When asserted, Internal counters are enabled. When negated, Internal counters disabled.  |
| GPREN | Global Prescaler Enable<br>The GPREN bit enables the prescaler counter.<br>1 = Prescaler enabled<br>0 = Prescaler disabled (no clock) and prescaler counter is cleared   |
| GPRE  | Global Prescaler<br>The GPRE bits select the clock divider value for the global prescaler, as shown in <a href="#">Table 27-12</a> .   |

Table 27-12. Global prescaler clock divider

| GPRE     | Divide ratio |
|----------|--------------|
| 00000000 | 1            |
| 00000001 | 2            |
| 00000010 | 3            |
| 00000011 | 4            |
| .        | .            |
| .        | .            |
| .        | .            |
| .        | .            |
| 11111110 | 255          |
| 11111111 | 256          |

#### 27.4.3.2.2 eMIOS Global FLAG (EMIOGFLAG) Register

The EMIOGFLAG is a read-only register that groups the flag bits (F[31:0]) from all channels. This organization improves interrupt handling on simpler devices. Each bit relates to one channel.

For Unified Channels these bits are mirrors of the FLAG bits in the EMIOSS[n] register.

Address: eMIOS base address +0x04

|       |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|-------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
|       | 0   | 1   | 2   | 3   | 4   | 5   | 6   | 7   | 8   | 9   | 10  | 11  | 12  | 13  | 14  | 15  |
| R     | F31 | F30 | F29 | F28 | F27 | F26 | F25 | F24 | F23 | F22 | F21 | F20 | F19 | F18 | F17 | F16 |
| W     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| Reset | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |

|       |     |     |     |     |     |     |    |    |    |    |    |    |    |    |    |    |
|-------|-----|-----|-----|-----|-----|-----|----|----|----|----|----|----|----|----|----|----|
|       | 16  | 17  | 18  | 19  | 20  | 21  | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| R     | F15 | F14 | F13 | F12 | F11 | F10 | F9 | F8 | F7 | F6 | F5 | F4 | F3 | F2 | F1 | F0 |
| W     |     |     |     |     |     |     |    |    |    |    |    |    |    |    |    |    |
| Reset | 0   | 0   | 0   | 0   | 0   | 0   | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |

Figure 27-9. eMIOS Global FLAG (EMIOGFLAG) Register

Table 27-13. EMIOGFLAG field descriptions

| Field          | Description          |
|----------------|----------------------|
| F <sub>n</sub> | Channel [n] Flag bit |

### 27.4.3.2.3 eMIOS Output Update Disable (EMIOSOUDIS) Register

Address: eMIOS base address +0x08

|       |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
|-------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|
|       | 0    | 1    | 2    | 3    | 4    | 5    | 6    | 7    | 8    | 9    | 10   | 11   | 12   | 13   | 14   | 15   |
| R     | OU31 | OU30 | OU29 | OU28 | OU27 | OU26 | OU25 | OU24 | OU23 | OU22 | OU21 | OU20 | OU19 | OU18 | OU17 | OU16 |
| W     | OU31 | OU30 | OU29 | OU28 | OU27 | OU26 | OU25 | OU24 | OU23 | OU22 | OU21 | OU20 | OU19 | OU18 | OU17 | OU16 |
| Reset | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    |

|       |      |      |      |      |      |      |     |     |     |     |     |     |     |     |     |     |
|-------|------|------|------|------|------|------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
|       | 16   | 17   | 18   | 19   | 20   | 21   | 22  | 23  | 24  | 25  | 26  | 27  | 28  | 29  | 30  | 31  |
| R     | OU15 | OU14 | OU13 | OU12 | OU11 | OU10 | OU9 | OU8 | OU7 | OU6 | OU5 | OU4 | OU3 | OU2 | OU1 | OU0 |
| W     | OU15 | OU14 | OU13 | OU12 | OU11 | OU10 | OU9 | OU8 | OU7 | OU6 | OU5 | OU4 | OU3 | OU2 | OU1 | OU0 |
| Reset | 0    | 0    | 0    | 0    | 0    | 0    | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |

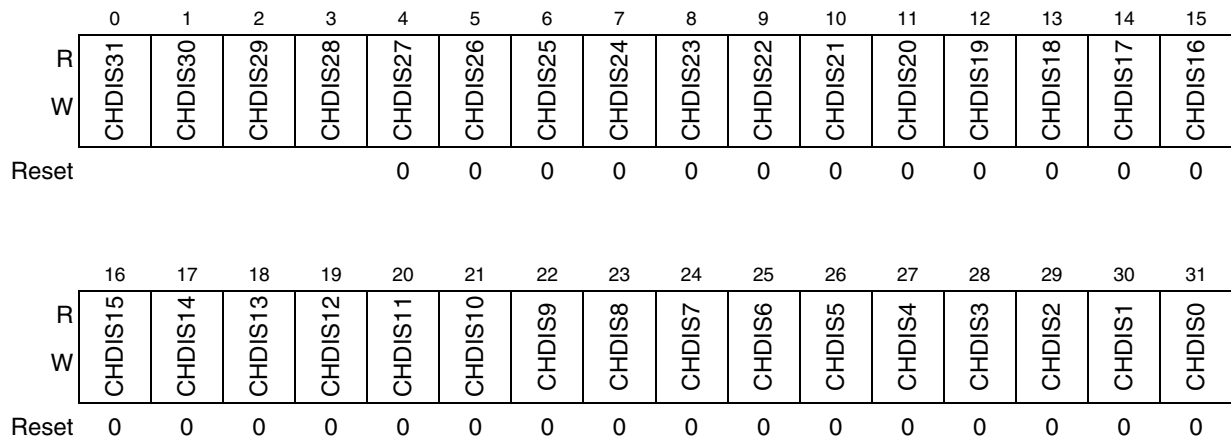
Figure 27-10. eMIOS Output Update Disable (EMIOSOUDIS) Register

**Table 27-14. EMIOSOUDIS field descriptions**

| Field | Description   |
|-------|---|
| OUn   | Channel [n] Output Update Disable bit<br>When running MC, MCB or an output mode, values are written to registers A2 and B2. OU[n] bits are used to disable transfers from registers A2 to A1 and B2 to B1. Each bit controls one channel.<br>1 = Transfers disabled<br>0 = Transfer enabled. Depending on the operation mode, transfer may occur immediately or in the next period. Unless stated otherwise, transfer occurs immediately. |

**27.4.3.2.4 eMIOS Disable Channel (EMIOSUCDIS) Register**

Address: eMIOS base address +0x0C



**Figure 27-11. eMIOS Enable Channel (EMIOSUCDIS) Register**

**Table 27-15. EMIOSUCDIS field descriptions**

| Field  | Description   |
|--------|---|
| CHDISn | Enable Channel [n] bit<br>The CHDIS[n] bit is used to disable each of the channels by stopping its respective clock.<br>1 = Channel [n] disabled<br>0 = Channel [n] enabled |



### 27.4.3.2.5 eMIOS UC A Register (EMIOSA[n])

Address: UC[n] base address + 0x00

|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| R     | A  |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| W     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |

**Figure 27-12. eMIOS UC A Register (EMIOSA[n])**

Depending on the mode of operation, internal registers A1 or A2, used for matches and captures, can be assigned to address EMIOSA[n]. Both A1 and A2 are cleared by reset. [Figure 27-16](#) summarizes the EMIOSA[n] writing and reading accesses for all operation modes. For more information see [Section 27.4.4.1.1, UC modes of operation](#).

### 27.4.3.2.6 eMIOS UC B Register (EMIOSB[n])

Address: UC[n] base address + 0x04

|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| R     | B  |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| W     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |

**Figure 27-13. eMIOS UC B Register (EMIOSB[n])**

Depending on the mode of operation, internal registers B1 or B2 can be assigned to address EMIOSB[n]. Both B1 and B2 are cleared by reset. [Table 27-16](#) summarizes the EMIOSB[n] writing and reading accesses for all operation modes. For more information see [Section 27.4.4.1.1, UC modes of operation](#).

Depending on the channel configuration, it may have EMIOSB register or not. This means that, if at least one mode that requires the register is implemented, then the register is present; otherwise it is absent.

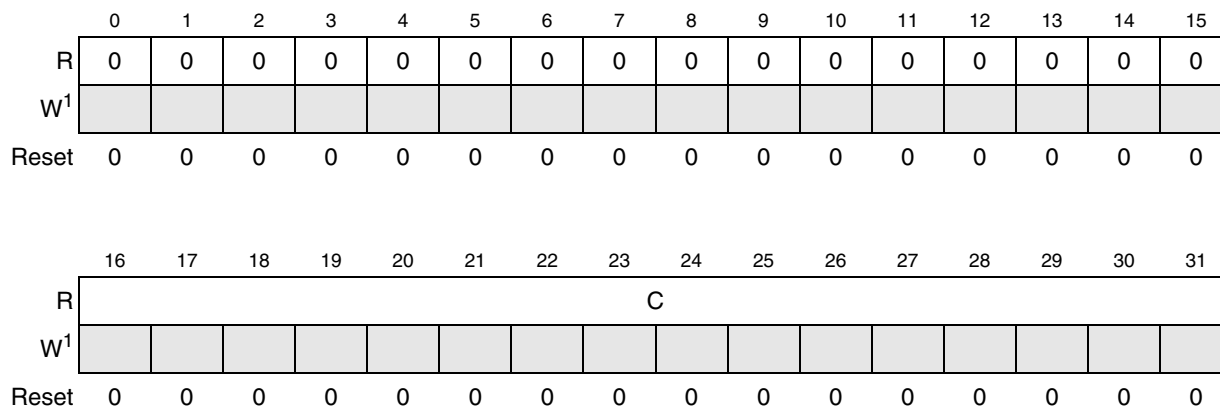
**Table 27-16. EMIOSA[n], EMIOSB[n] and EMIOSALTA[n] values assignment**

| Operation mode    | Register access |      |       |      |           |          |
|-------------------|-----------------|------|-------|------|-----------|----------|
|                   | write           | read | write | read | alt write | alt read |
| GPIO              | A1, A2          | A1   | B1,B2 | B1   | A2        | A2       |
| SAIC <sup>1</sup> | —               | A2   | B2    | B2   | —         | —        |
| SAOC <sup>1</sup> | A2              | A1   | B2    | B2   | —         | —        |
| IPWM              | —               | A2   | —     | B1   | —         | —        |
| IPM               | —               | A2   | —     | B1   | —         | —        |
| DAOC              | A2              | A1   | B2    | B1   | —         | —        |
| MC <sup>1</sup>   | A2              | A1   | B2    | B2   | —         | —        |
| OPWMT             | A1              | A1   | B2    | B1   | A2        | A2       |
| MCB <sup>1</sup>  | A2              | A1   | B2    | B2   | —         | —        |
| OPWFMB            | A2              | A1   | B2    | B1   | —         | —        |
| OPWMCB            | A2              | A1   | B2    | B1   | —         | —        |
| OPWMB             | A2              | A1   | B2    | B1   | —         | —        |

<sup>1</sup> In these modes, the register EMIOSB[n] is not used, but B2 can be accessed.

### 27.4.3.2.7 eMIOS UC Counter Register (EMIOSCNT[n])

Address: UC[n] base address + 0x08



**Figure 27-14. eMIOS UC Counter Register (EMIOSCNT[n])**

<sup>1</sup> In GPIO mode or Freeze action, this register is writable.

The EMIOSCNT[n] register contains the value of the internal counter. When GPIO mode is selected or the channel is frozen, the EMIOSCNT[n] register is read/write. For all others modes, the EMIOSCNT[n] is a read-only register. When entering some operation modes, this register is automatically cleared (refer to [Section 27.4.4.1.1, UC modes of operation](#), for details).

Depending on the channel configuration it may have an internal counter or not. It means that if at least one mode that requires the counter is implemented, then the counter is present; otherwise it is absent.

Channels of type X and G have the internal counter enabled, so their timebase can be selected by channel's BSL[1:0]=11:eMIOS\_A—channels 0 to 8, 16, 23, and 24, eMIOS\_B—channels 0, 8, 16, 23, and 24. Other channels from the above list don't have internal counters.

### 27.4.3.2.8 eMIOS UC Control Register (EMIOSC[n])

The Control register gathers bits reflecting the status of the UC input/output signals and the overflow condition of the internal counter, as well as several read/write control bits.

Address: UC[n] base address + 0x0C

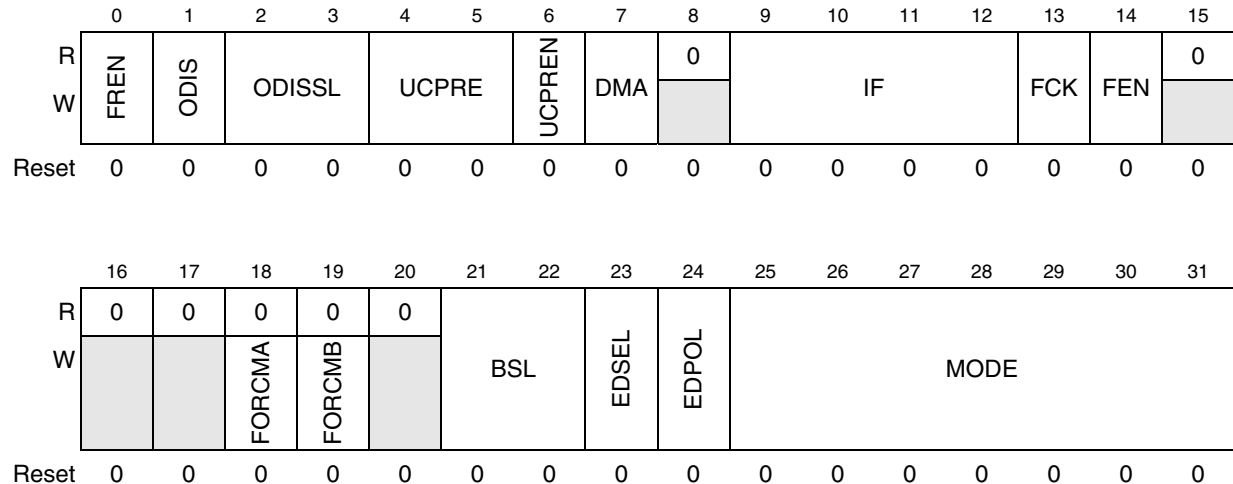


Figure 27-15. eMIOS UC Control Register (EMIOSC[n])

Table 27-17. EMIOSC[n] field descriptions

| Field  | Description  |
|--------|--|
| FREN   | Freeze Enable bit<br>The FREN bit, if set and validated by FRZ bit in EMIOSMCR register allows the channel to enter freeze state, freezing all registers values when in debug mode and allowing the MCU to perform debug functions.<br>1 = Freeze UC registers values<br>0 = Normal operation  |
| ODIS   | Output Disable<br>This bit allows disabling the output pin when running any of the output modes with the exception of GPIO mode.<br>1 = If the selected Output Disable Input signal is asserted, the output pin goes to EDPOL for OPWFMB and OPWMB modes and to the complement of EDPOL for other output modes, but the Unified Channel continues to operate normally (that is, it continues to produce FLAG and matches).<br>When the selected Output Disable Input signal is negated, the output pin operates normally.<br>0 = The output pin operates normally. |
| ODISSL | Output Disable select<br>This field selects one of the four output disable input signals, as follows:<br>00 = Output Disable Input 0<br>01 = Output Disable Input 1<br>10 = Output Disable Input 2<br>11 = Output Disable Input 3  |

Table 27-17. EMIOSC[n] field descriptions (continued)

| Field  | Description  |
|--------|--|
| UCPRE  | Prescaler bits<br>The UCPRE bits select the clock divider value for the internal prescaler of Unified Channel, as shown in <a href="#">Table 27-18</a> .   |
| UCPREN | Prescaler Enable bit<br>The UCPREN bit enables the prescaler counter.<br>1 = Prescaler enabled<br>0 = Prescaler disabled (no clock)  |
| DMA    | Direct Memory Access bit<br>The DMA bit selects if the FLAG generation will be used as an interrupt request, as a DMA request or as a CTU trigger. The choice between a DMA request or a CTU trigger is determined by the value of bit TM in the register CTU_EVTTCFGRx (refer to <a href="#">Chapter 29, Cross Triggering Unit (CTU)</a> ).<br>1 = Flag/overrun assigned to DMA request or CTU trigger<br>0 = Flag/overrun assigned to interrupt request                    |
| IF     | Input Filter<br>The IF field controls the programmable input filter, selecting the minimum input pulse width that can pass through the filter, as shown in <a href="#">Table 27-19</a> . For output modes, these bits have no meaning.   |
| FCK    | Filter Clock select bit<br>The FCK bit selects the clock source for the programmable input filter.<br>1 = Main clock<br>0 = Prescaled clock  |
| FEN    | FLAG Enable bit<br>The FEN bit allows the Unified Channel FLAG bit to generate an interrupt signal or a DMA request signal or a CTU trigger signal (The type of signal to be generated is defined by the DMA bit).<br>1 = Enable (FLAG will generate an interrupt request or DMA request or a CTU trigger)<br>0 = Disable (FLAG does not generate an interrupt request or DMA request or a CTU trigger)  |
| FORCMA | Force Match A bit<br>For output modes, the FORCMA bit is equivalent to a successful comparison on comparator A (except that the FLAG bit is not set). This bit is cleared by reset and is always read as zero. This bit is valid for every output operation mode that uses comparator A, otherwise it has no effect.<br>1 = Force a match at comparator A<br>0 = Has no effect<br><b>Note:</b> For input modes, the FORCMA bit is not used and writing to it has no effect.  |
| FORCMB | Force Match B bit<br>For output modes, the FORCMB bit is equivalent to a successful comparison on comparator B (except that the FLAG bit is not set). This bit is cleared by reset and is always read as zero. This bit is valid for every output operation mode that uses comparator B, otherwise it has no effect.<br>1 = Force a match at comparator B<br>0 = Has not effect<br><b>Note:</b> For input modes, the FORCMB bit is not used and writing to it has no effect. |
| BSL    | Bus Select<br>The BSL field is used to select either one of the counter buses or the internal counter to be used by the Unified Channel. Refer to <a href="#">Table 27-20</a> for details.   |

Table 27-17. EMIOSC[n] field descriptions (continued)

| Field | Description   |
|-------|---|
| EDSEL | <p>Edge Selection bit</p> <p>For input modes, the EDSEL bit selects whether the internal counter is triggered by both edges of a pulse or just by a single edge as defined by the EDPOL bit. When not shown in the mode of operation description, this bit has no effect.</p> <p>1 = Both edges triggering<br/>0 = Single edge triggering defined by the EDPOL bit</p> <p>For GPIO in mode, the EDSEL bit selects if a FLAG can be generated.</p> <p>1 = No FLAG is generated<br/>0 = A FLAG is generated as defined by the EDPOL bit</p> <p>For SAOC mode, the EDSEL bit selects the behavior of the output flip-flop at each match.</p> <p>1 = The output flip-flop is toggled<br/>0 = The EDPOL value is transferred to the output flip-flop</p> |
| EDPOL | <p>Edge Polarity bit</p> <p>For input modes, the EDPOL bit asserts which edge triggers either the internal counter or an input capture or a FLAG. When not shown in the mode of operation description, this bit has no effect.</p> <p>1 = Trigger on a rising edge<br/>0 = Trigger on a falling edge</p> <p>For output modes, the EDPOL bit is used to select the logic level on the output pin.</p> <p>1 = A match on comparator A sets the output flip-flop, while a match on comparator B clears it<br/>0 = A match on comparator A clears the output flip-flop, while a match on comparator B sets it</p>   |
| MODE  | <p>Mode selection</p> <p>The MODE field selects the mode of operation of the Unified Channel, as shown in <a href="#">Table 27-21</a>.</p> <p><b>Note:</b> If a reserved value is written to mode the results are unpredictable.</p>  |

Table 27-18. UC internal prescaler clock divider

| UCPRE | Divide ratio |
|-------|--------------|
| 00    | 1            |
| 01    | 2            |
| 10    | 3            |
| 11    | 4            |

Table 27-19. UC input filter bits

| IF <sup>1</sup> | Minimum input pulse width [FLT_CLK periods] |
|-----------------|---|
| 0000            | Bypassed <sup>2</sup>                       |
| 0001            | 02  |
| 0010            | 04  |
| 0100            | 08  |
| 1000            | 16  |
| all others      | Reserved                                    |

- 1 Filter latency is 3 clock edges.
- 2 The input signal is synchronized before arriving to the digital filter.

**Table 27-20. UC BSL bits**

| <b>BSL</b> | <b>Selected bus</b>   |
|------------|---|
| 00         | All channels: counter bus[A]  |
| 01         | Channels 0 to 7: counter bus[B]<br>Channels 8 to 15: counter bus[C]<br>Channels 16 to 23: counter bus[D]<br>Channels 24 to 27: counter bus[E] |
| 10         | Reserved  |
| 11         | All channels: internal counter  |

**Table 27-21. Channel mode selection**

| <b>MODE<sup>1</sup></b> | <b>Mode of operation</b>   |
|-------------------------|--|
| 0000000                 | General purpose Input/Output mode (input)  |
| 0000001                 | General purpose Input/Output mode (output)   |
| 0000010                 | Single Action Input Capture  |
| 0000011                 | Single Action Output Compare   |
| 0000100                 | Input Pulse Width Measurement  |
| 0000101                 | Input Period Measurement   |
| 0000110                 | Double Action Output Compare (with FLAG set on B match)                              |
| 0000111                 | Double Action Output Compare (with FLAG set on both match)                           |
| 0001000 – 0001111       | Reserved   |
| 001000b                 | Modulus Counter (Up counter with clear on match start)                               |
| 001001b                 | Modulus Counter (Up counter with clear on match end)                                 |
| 00101bb                 | Modulus Counter (Up/Down counter)  |
| 0011000 – 0100101       | Reserved   |
| 0100110                 | Output Pulse Width Modulation with Trigger   |
| 0100111 – 1001111       | Reserved   |
| 101000b                 | Modulus Counter Buffered (Up counter)  |
| 101001b                 | Reserved   |
| 10101bb                 | Modulus Counter Buffered (Up/Down counter)   |
| 10110b0                 | Output Pulse Width and Frequency Modulation Buffered                                 |
| 10110b1                 | Reserved   |
| 10111b0                 | Center Aligned Output Pulse Width Modulation Buffered (with trailing edge dead time) |
| 10111b1                 | Center Aligned Output Pulse Width Modulation Buffered (with leading edge dead time)  |

Table 27-21. Channel mode selection (continued)

| MODE <sup>1</sup> | Mode of operation                      |
|-------------------|--|
| 11000b0           | Output Pulse Width Modulation Buffered |
| 1100001 – 1111111 | Reserved                               |

<sup>1</sup> b = adjust parameters for the mode of operation. Refer to [Section 27.4.4.1.1, UC modes of operation](#) for details.

### 27.4.3.2.9 eMIOS UC Status Register (EMIOSS[n])

Address: UC[n] base address + 0x10

|       | 0   | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|-------|-----|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
| R     | OVR | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     | w1c |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0   | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       | 16   | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29   | 30    | 31   |
|-------|------|----|----|----|----|----|----|----|----|----|----|----|----|------|-------|------|
| R     | OVFL | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | UCIN | UCOUT | FLAG |
| W     | w1c  |    |    |    |    |    |    |    |    |    |    |    |    |      |       | w1c  |
| Reset | 0    | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0    | 0     | 0    |

Figure 27-16. eMIOS UC Status Register (EMIOSS[n])

Table 27-22. EMIOSS[n] field descriptions

| Field | Description  |
|-------|--|
| OVR   | <p>Overrun bit</p> <p>The OVR bit indicates that FLAG generation occurred when the FLAG bit was already set.</p> <p>1 = Overrun has occurred</p> <p>0 = Overrun has not occurred</p>   |
| OVFL  | <p>Overflow bit</p> <p>The OVFL bit indicates that an overflow has occurred in the internal counter. OVFL must be cleared by software writing a 1 to the OVFLC bit.</p> <p>1 = An overflow had occurred</p> <p>0 = No overflow</p>   |
| UCIN  | <p>Unified Channel Input pin bit</p> <p>The UCIN bit reflects the input pin state after being filtered and synchronized.</p>   |
| UCOUT | <p>UCOUT — Unified Channel Output pin bit</p> <p>The UCOUT bit reflects the output pin state.</p>  |
| FLAG  | <p>FLAG bit</p> <p>The FLAG bit is set when an input capture or a match event in the comparators occurred.</p> <p>1 = FLAG set event has occurred</p> <p>0 = FLAG cleared</p> <p><b>Note:</b> When DMA bit is set, the FLAG bit can be cleared by the DMA controller or the CTU.</p> |

### 27.4.3.2.10 eMIOS UC Alternate A Register (EMIOSALTA[n])

Address: UC[n] base address + 0x14

|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       | 16   | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|-------|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| R     | ALTA |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| W     |      |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| Reset | 0    | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |

Figure 27-17. eMIOS UC Alternate A register (EMIOSALTA[n])

The EMIOSALTA[n] register provides an alternate address to access A2 channel registers in restricted modes (GPIO, OPWMT) only. If EMIOSA[n] register is used along with EMIOSALTA[n], both A1 and A2 registers can be accessed in these modes. Figure 27-16 summarizes the EMIOSALTA[n] writing and reading accesses for all operation modes. Please, see Section 27.4.4.1.1.1, [General purpose Input/Output \(GPIO\) mode](#), Section 27.4.4.1.1.12, [Output Pulse Width Modulation with Trigger \(OPWMT\) mode](#) for a more detailed description of the use of EMIOSALTA[n] register.

## 27.4.4 Functional description

The four types of channels of the eMIOS (types X, Y, G, and H) can operate in the modes as listed in Figure 27-7. The eMIOS provides independently operating unified channels (UC) that can be configured and accessed by a host MCU. As many as three time bases<sup>1</sup> can be shared by the channels through five counter buses<sup>2</sup> and each unified channel can generate its own time base<sup>3</sup>. The eMIOS block is reset at positive edge of the clock (synchronous reset). All registers are cleared on reset.

### 27.4.4.1 Unified Channel (UC)

Each Unified Channel consists of:

- Counter bus selector, which selects the time base to be used by the channel for all timing functions
- A programmable clock prescaler

1. Time bases can be supplied by:

a) channel 23 to all unified channels

b) channel 0 to channels 0 to 7, by channel 8 to channels 8 to 15, by channel 16 to channels 16 to 23, by channel 24 to channels 24 to 31

c) channel's internal counter when available.

2. Internal eMIOS architecture have one global counter bus A and four local counter buses B, C, D, and E, that distribute the time bases described in Note 1 (a) and (b).

3. Channels of type X and G have the internal counter enabled, so their timebase can be selected by channel's BSL[1:0]=11: eMIOS\_A—channels 0 to 8, 16, 23, and 24 eMIOS\_B—channels 0, 8, 16, 23, and 24.



- Two double buffered data registers A and B that allow as many as two input capture and/or output compare events to occur before software intervention is needed.
- Two comparators (equal only) A and B, which compare the selected counter bus with the value in the data registers
- Internal counter, which can be used as a local time base or to count input events
- Programmable input filter, which ensures that only valid pin transitions are received by channel
- Programmable input edge detector, which detects the rising, falling, or either edges
- An output flip-flop, which holds the logic level to be applied to the output pin
- eMIOS Status and Control register

#### 27.4.4.1.1 UC modes of operation

The mode of operation of the Unified Channel is determined by the mode select bits MODE[0:6] in the eMIOS UC Control Register (EMIOSC[n]) (see [Figure 27-15](#) for details).

As the internal counter EMIOSCNT[n] continues to run in all modes (except for GPIO mode), it is possible to use this as a time base if the resource is not used in the current mode.

In order to provide smooth waveform generation even if A and B registers are changed on the fly, it is available the MCB, OPWFMB, OPWMB, and OPWMCB modes. In these modes A and B registers are double buffered.

##### 27.4.4.1.1.1 General purpose Input/Output (GPIO) mode

In GPIO mode, all input capture and output compare functions of the UC are disabled, the internal counter (EMIOSCNT[n] register) is cleared and disabled. All control bits remain accessible. In order to prepare the UC for a new operation mode, writing to registers EMIOSA[n] or EMIOSB[n] stores the same value in registers A1/A2 or B1/B2, respectively. Writing to register EMIOSALTA[n] stores a value only in register A2.

MODE[6] bit selects between input (MODE[6] = 0) and output (MODE[6] = 1) modes.

It is required that when changing MODE[0:6], the application software goes to GPIO mode first in order to reset the UC's internal functions properly. Failure to do this could lead to invalid and unexpected output compare or input capture results or the FLAGS being set incorrectly.

In GPIO input mode (MODE[0:6] = 0000000), the FLAG generation is determined according to EDPOL and EDSEL bits, and the input pin status can be determined by reading the UCIN bit.

In GPIO output mode (MODE[0:6] = 0000001), the Unified Channel is used as a single output port pin, and the value of the EDPOL bit is permanently transferred to the output flip-flop.

##### 27.4.4.1.1.2 Single Action Input Capture (SAIC) mode

In SAIC mode (MODE[0:6] = 0000010), when a triggering event occurs on the input pin, the value on the selected time base is captured into register A2. The FLAG bit is set along with the capture event to indicate that an input capture has occurred. Register EMIOSA[n] returns the value of register A2. As soon as the SAIC mode is entered coming out from GPIO mode the channel is ready to capture events. The events are

captured as soon as they occur thus reading register A always returns the value of the latest captured event. Subsequent captures are enabled with no need of further reads from EMIOSA[n] register. The FLAG is set at any time a new event is captured.

The input capture is triggered by a rising, falling or either edges in the input pin, as configured by EDPOL and EDSEL bits in EMIOSC[n] register.

Figure 27-18 and Figure 27-19 show how the Unified Channel can be used for input capture.

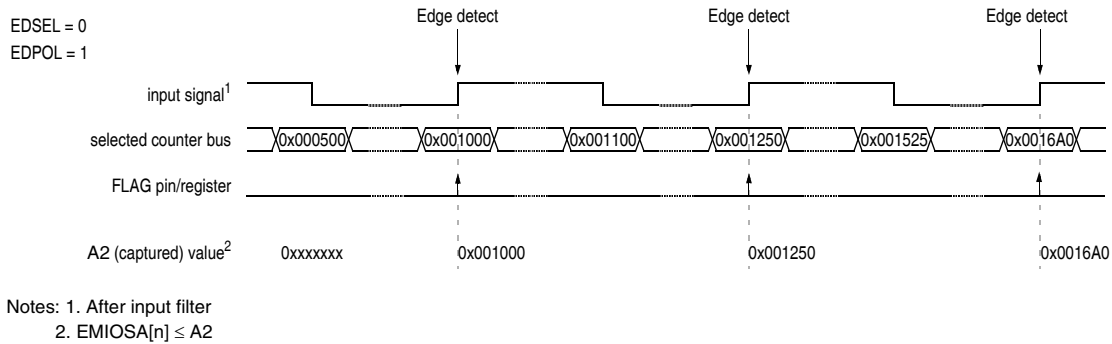


Figure 27-18. Single action input capture with rising edge triggering example

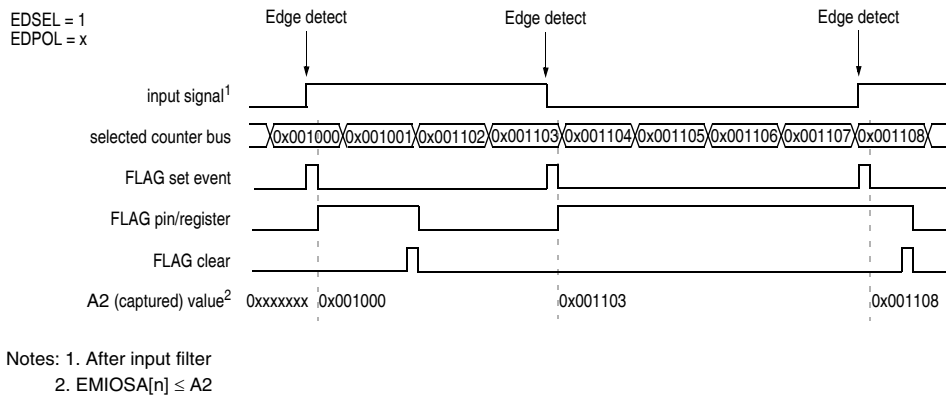


Figure 27-19. Single action input capture with both edges triggering example

### 27.4.4.1.1.3 Single Action Output Compare (SAOC) mode

In SAOC mode (MODE[0:6] = 0000011) a match value is loaded in register A2, and then immediately transferred to register A1 to be compared with the selected time base. When a match occurs, the EDSEL bit selects whether the output flip-flop is toggled or the value in EDPOL is transferred to it. Along with the match the FLAG bit is set to indicate that the output compare match has occurred. Writing to register EMIOSA[n] stores the value in register A2 and reading to register EMIOSA[n] returns the value of register A1.

An output compare match can be simulated in software by setting the FORCMA bit in EMIOSC[n] register. In this case, the FLAG bit is not set.

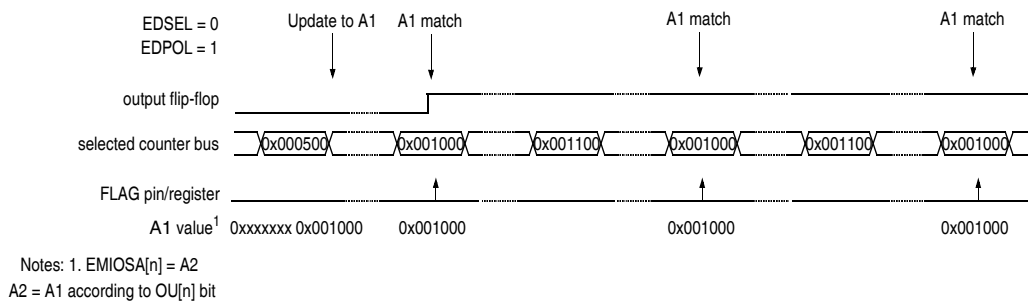
When SAOC mode is entered coming out from GPIO mode the output flip-flop is set to the complement of the EDPOL bit in the EMIOSC[n] register.

Counter bus can be either internal or external and is selected through bits BSL[0:1].

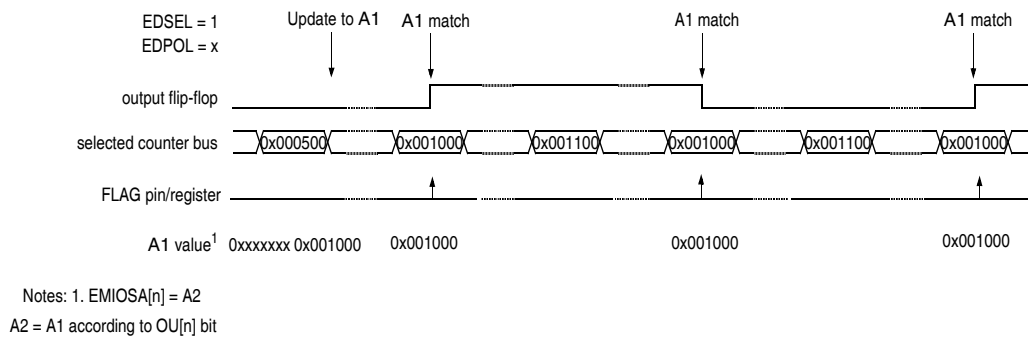
Figure 27-20 and Figure 27-21 show how the Unified Channel can be used to perform a single output compare with EDPOL value being transferred to the output flip-flop and toggling the output flip-flop at each match, respectively. Note that once in SAOC mode the matches are enabled thus the desired match value on register A1 must be written before the mode is entered. A1 register can be updated at any time, thus modifying the match value that will reflect in the output signal generated by the channel. Subsequent matches are enabled with no need of further writes to EMIOSA[n] register. The FLAG is set at the same time a match occurs (see Figure 27-22).

### NOTE

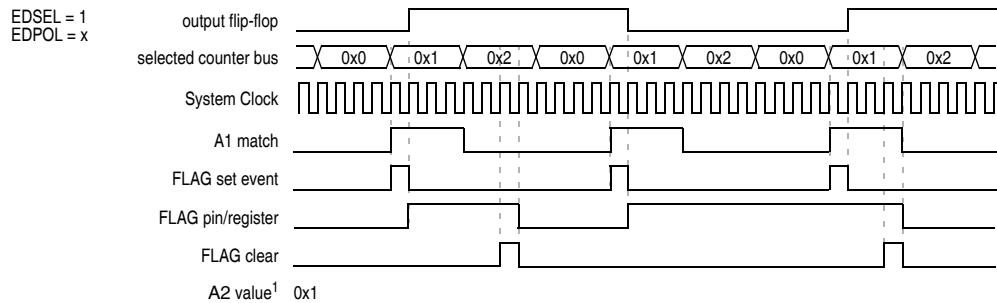
The channel internal counter in SAOC mode is free-running. It starts counting as soon as the SAOC mode is entered.



**Figure 27-20. SAOC example with EDPOL value being transferred to the output flip-flop**



**Figure 27-21. SAOC example toggling the output flip-flop**



Note: 1.  $EMIOSA[n] \leq A2$

Figure 27-22. SAOC example with flag behavior

#### 27.4.4.1.4 Input Pulse Width Measurement (IPWM) Mode

The IPWM mode ( $MODE[0:6] = 0000100$ ) allows the measurement of the width of a positive or negative pulse by capturing the leading edge on register B1 and the trailing edge on register A2. Successive captures are done on consecutive edges of opposite polarity. The leading edge sensitivity (that is, pulse polarity) is selected by EDPOL bit in the EMIOSC[n] register. Registers EMIOSA[n] and EMIOSB[n] return the values in register A2 and B1, respectively.

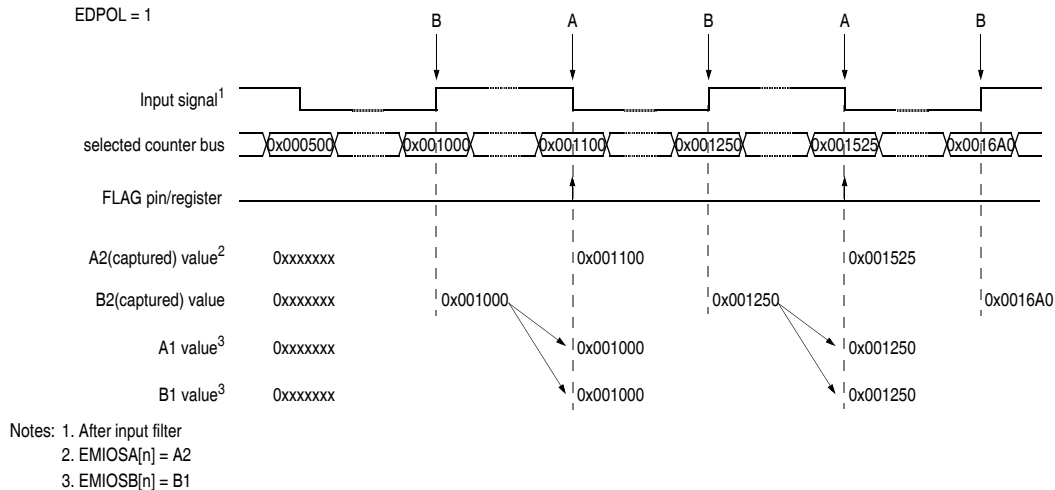
The capture function of register A2 remains disabled until the first leading edge triggers the first input capture on register B2. When this leading edge is detected, the count value of the selected time base is latched into register B2; the FLAG bit is not set. When the trailing edge is detected, the count value of the selected time base is latched into register A2 and, at the same time, the FLAG bit is set and the content of register B2 is transferred to register B1 and to register A1.

If subsequent input capture events occur while the corresponding FLAG bit is set, registers A2, B1, and A1 will be updated with the latest captured values and the FLAG will remain set. Registers EMIOSA[n] and EMIOSB[n] return the value in registers A2 and B1, respectively.

In order to guarantee coherent access, reading EMIOSA[n] forces B1 be updated with the content of register A1. At the same time transfers between B2 and B1 are disabled until the next read of EMIOSB[n] register. Reading EMIOSB[n] register forces B1 be updated with A1 register content and reenables transfers from B2 to B1, to take effect at the next trailing edge capture. Transfers from B2 to A1 are not blocked at any time.

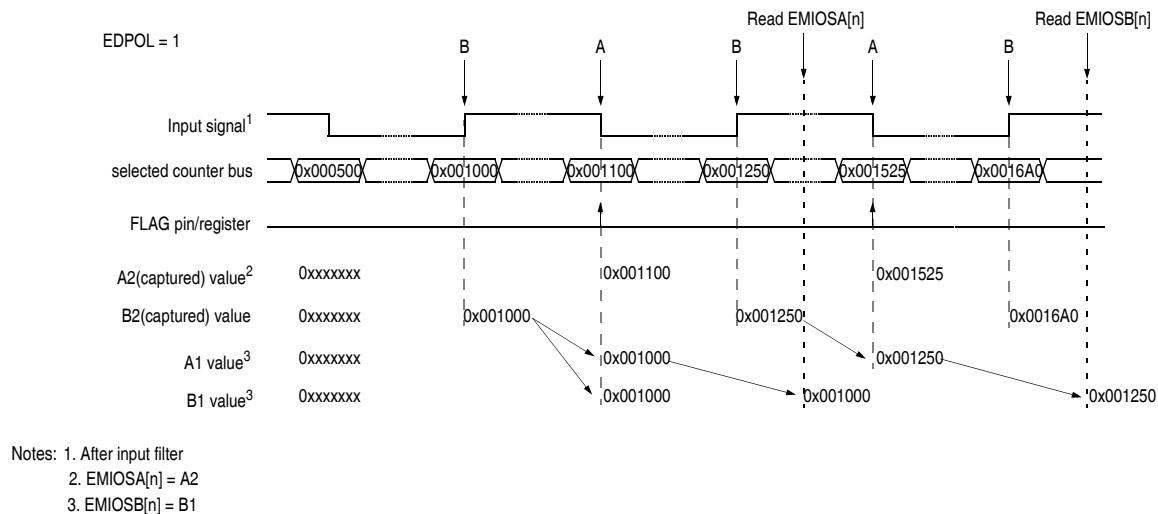
The input pulse width is calculated by subtracting the value in B1 from A2.

Figure 27-23 shows how the Unified Channel can be used for input pulse width measurement.



**Figure 27-23. Input pulse width measurement example**

Figure 27-24 shows the A1 and B1 updates when EMIOSA[n] and EMIOSB[n] register reads occur. Note that A1 register has always coherent data related to A2 register. Note also that when EMIOSA[n] read is performed B1 register is loaded with A1 register content. This guarantee that the data in register B1 has always the coherent data related to the last EMIOSA[n] read. The B1 register updates remains locked until EMIOSB[n] read occurs. If EMIOSA[n] read is performed B1 is updated with A1 register content even if B1 update is locked by a previous EMIOSA[n] read operation.



**Figure 27-24. B1 and A1 updates at EMIOSA[n] and EMIOSB[n] reads**

Reading EMIOSA[n] followed by EMIOSB[n] always provides coherent data. If not coherent data is required for any reason, the sequence of reads should be inverted, therefore EMIOSB[n] should be read prior to EMIOSA[n] register. Note that even in this case B1 register updates will be blocked after EMIOSA[n] read, thus a second EMIOSB[n] is required in order to release B1 register updates.

### 27.4.4.1.1.5 Input Period Measurement (IPM) mode

The IPM mode (MODE[0:6] = 0000101) allows the measurement of the period of an input signal by capturing two consecutive rising edges or two consecutive falling edges. Successive input captures are done on consecutive edges of the same polarity. The edge polarity is defined by the EDPOL bit in the EMIOSC[n] register.

When the first edge of selected polarity is detected, the selected time base is latched into the registers A2 and B2, and the data previously held in register B2 is transferred to register B1. On this first capture the FLAG line is not set, and the values in registers B1 is meaningless. On the second and subsequent captures, the FLAG line is set and data in register B2 is transferred to register B1.

When the second edge of the same polarity is detected, the counter bus value is latched into registers A2 and B2, the data previously held in register B2 is transferred to data register B1 and to register A1. The FLAG bit is set to indicate the start and end points of a complete period have been captured. This sequence of events is repeated for each subsequent capture. Registers EMIOSA[n] and EMIOSB[n] return the values in register A2 and B1, respectively.

In order to allow coherent data, reading EMIOSA[n] forces A1 content be transferred to B1 register and disables transfers between B2 and B1. These transfers are disabled until the next read of the EMIOSB[n] register. Reading EMIOSB[n] register forces A1 content to be transferred to B1 and reenables transfers from B2 to B1, to take effect at the next edge capture.

The input pulse period is calculated by subtracting the value in B1 from A2.

Figure 27-25 shows how the Unified Channel can be used for input period measurement.

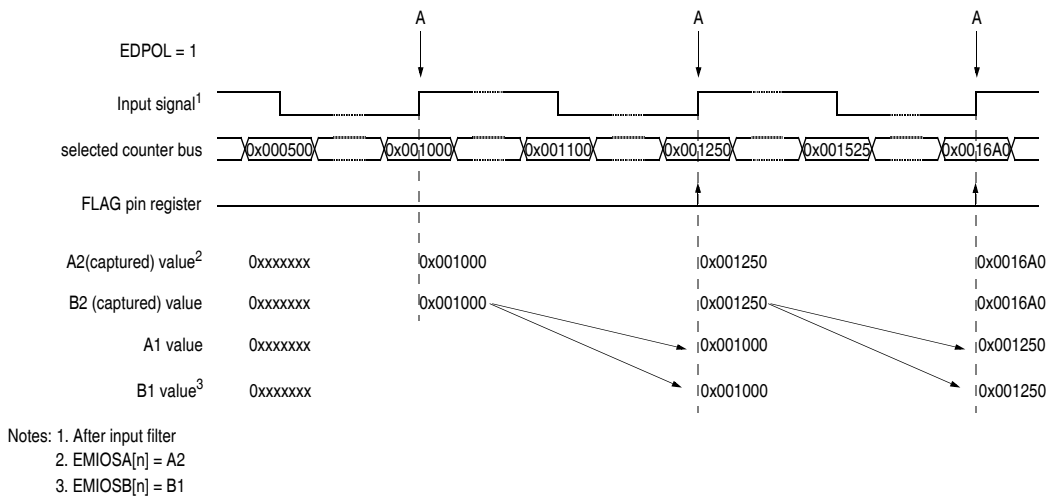
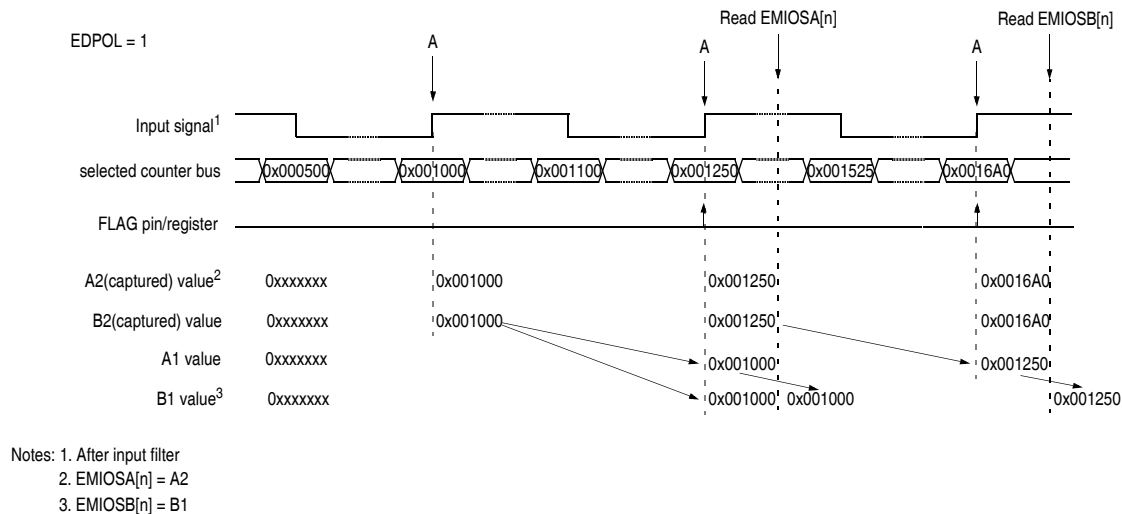


Figure 27-25. Input period measurement example

Figure 27-26 describes the A1 and B1 register updates when EMIOSA[n] and EMIOSB[n] read operations are performed. When EMIOSA[n] read occurs the content of A1 is transferred to B1 thus providing coherent data in A2 and B1 registers. Transfers from B2 to B1 are then blocked until EMIOSB[n] is read. After EMIOSB[n] is read, register A1 content is transferred to register B1 and the transfers from B2 to B1 are reenabled to occur at the transfer edges, which is the leading edge in the Figure 27-26 example.



**Figure 27-26. A1 and B1 updates at EMIOSA[n] and EMIOSB[n] reads**

#### 27.4.4.1.1.6 Double Action Output Compare (DAOC) mode

In the DAOC mode, the leading and trailing edges of the variable pulse width output are generated by matches occurring on comparators A and B. There is no restriction concerning the order in which A and B matches occur.

When the DAOC mode is entered, coming out from GPIO mode both comparators are disabled and the output flip-flop is set to the complement of the EDPOL bit in the EMIOSC[n] register.

Data written to A2 and B2 are transferred to A1 and B1, respectively, on the next system clock cycle if bit OU[n] of the EMIOSOUDIS register is cleared (see [Figure 27-29](#)). The transfer is blocked if bit OU[n] is set. Comparator A is enabled only after the transfer to A1 register occurs and is disabled on the next A match. Comparator B is enabled only after the transfer to B1 register occurs and is disabled on the next B match. Comparators A and B are enabled and disabled independently.

The output flip-flop is set to the value of EDPOL when a match occurs on comparator A and to the complement of EDPOL when a match occurs on comparator B.

MODE[6] controls if the FLAG is set on both matches (MODE[0:6] = 0000111) or just on the B match (MODE[0:6] = 0000110). FLAG bit assertion depends on comparator enabling.

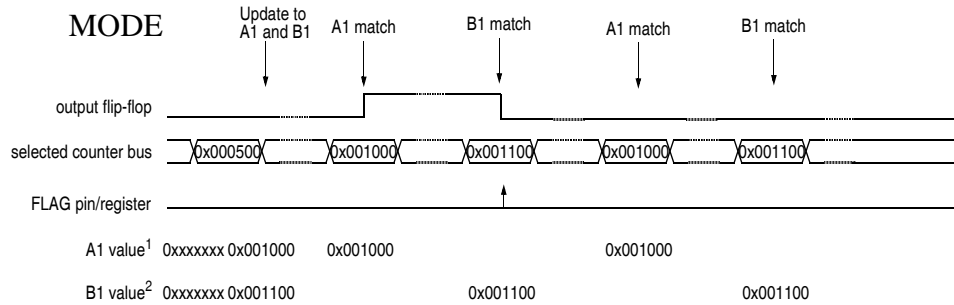
If subsequent enabled output compares occur on registers A1 and B1, pulses will continue to be generated, regardless of the state of the FLAG bit.

At any time, the FORCMA and FORCMB bits allow the software to force the output flip-flop to the level corresponding to a comparison event in comparator A or B, respectively. Note that the FLAG bit is not affected by these forced operations.

**NOTE**

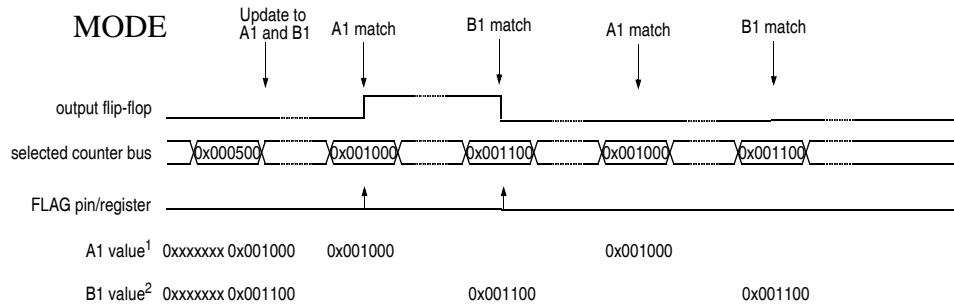
If both registers (A1 and B1) are loaded with the same value, the B match prevails concerning the output pin state (output flip-flop is set to the complement of EDPOL), the FLAG bit is set and both comparators are disabled.

Figure 27-27 and Figure 27-28 show how the Unified Channel can be used to generate a single output pulse with FLAG bit being set on the second match or on both matches, respectively.



Notes: 1. EMIOSA[n] = A1 (when reading)  
 2. EMIOB[n] = B1 (when reading)  
 A2 = A1 according to OU[n] bit  
 B2 = B1 according to OU[n] bit

**Figure 27-27. Double action output compare with FLAG set on the second match**



Notes: 1. EMIOSA[n] = A1 (when reading)  
 2. EMIOB[n] = B1 (when reading)  
 A2 = A1 according to OU[n] bit  
 B2 = B1 according to OU[n] bit

**Figure 27-28. Double action output compare with FLAG set on both matches**



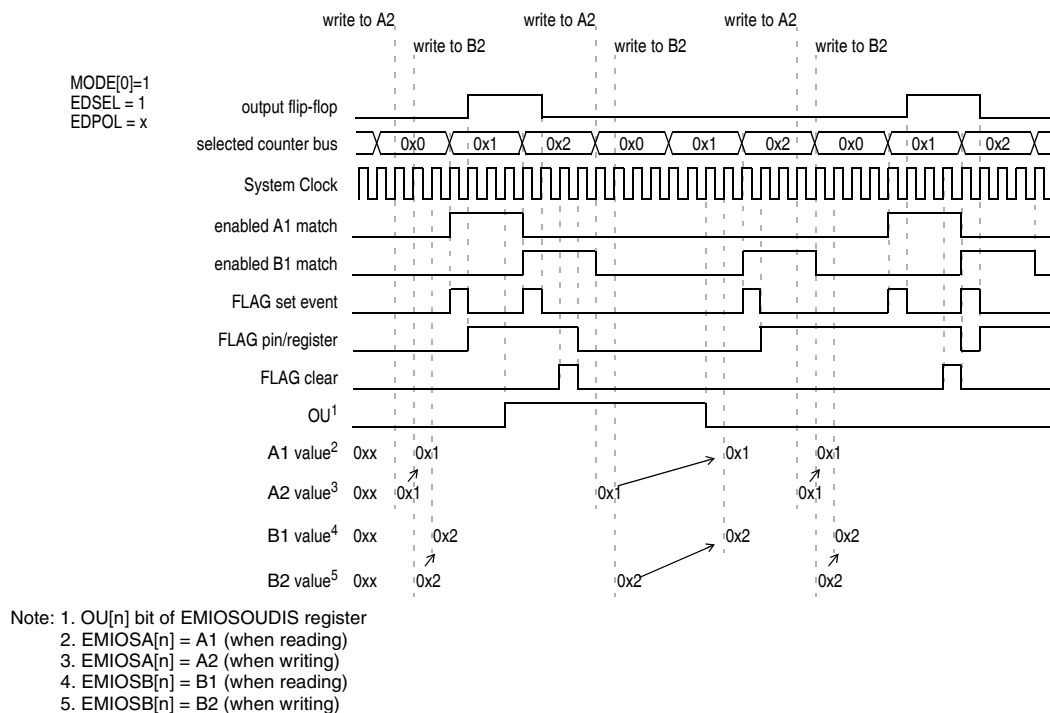


Figure 27-29. DAOC with transfer disabling example

### 27.4.4.1.1.7 Modulus Counter (MC) mode

The MC mode can be used to provide a time base for a counter bus or as a general purpose timer.

Bit MODE[6] selects internal or external clock source when cleared or set, respectively. When external clock is selected, the input signal pin is used as the source and the triggering polarity edge is selected by the EDPOL and EDSEL in the EMIOSC[n] register.

The internal counter counts up from the current value until it matches the value in register A1. Register B1 is cleared and is not accessible to the MCU. Bit MODE[4] selects up mode or up/down mode, when cleared or set, respectively.

When in up count mode, a match between the internal counter and register A1 sets the FLAG and clears the internal counter. The timing of those events varies according to the MC mode setup as follows:

- Internal counter clearing on match start (MODE[0:6] = 001000b)
  - External clock is selected if MODE[6] is set. In this case the internal counter clears as soon as the match signal occurs. The channel FLAG is set at the same time the match occurs. Note that by having the internal counter cleared as soon as the match occurs and incremented at the next input event a shorter zero count is generated. See [Figure 27-52](#) and [Figure 27-53](#).
  - Internal clock source is selected if MODE[6] is cleared. In this case the counter clears as soon as the match signal occurs. The channel FLAG is set at the same time the match occurs. At the next prescaler tick after the match the internal counter remains at zero and only resumes counting on the following tick. See [Figure 27-52](#) and [Figure 27-54](#).
- Internal counter clearing on match end (MODE[0:6] = 001001b)

- External clock is selected if MODE[6] is set. In this case the internal counter clears when the match signal is asserted and the input event occurs. The channel FLAG is set at the same time the counter is cleared. See Figure 27-52 and Figure 27-55.
- Internal clock source is selected if MODE[6] is cleared. In this case the internal counter clears when the match signal is asserted and the prescaler tick occurs. The channel FLAG is set at the same time the counter is cleared. See Figure 27-52 and Figure 27-55.

**NOTE**

If the internal clock source is selected and the prescaler of the internal counter is set to 1, the MC mode behaves the same way even in Clear on Match Start or Clear on Match End submodes.

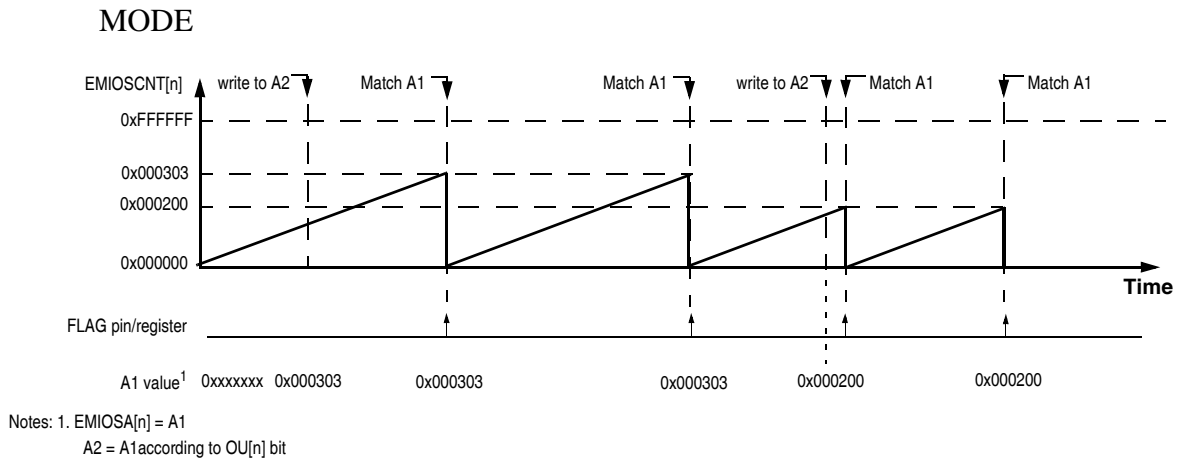
When in up/down count mode (MODE[0:6] = 00101bb), a match between the internal counter and register A1 sets the FLAG and changes the counter direction from increment to decrement. A match between register B1 and the internal counter changes the counter direction from decrement to increment and sets the FLAG only if MODE[5] bit is set.

Only values different than 0x0 must be written at A register. Loading 0x0 leads to unpredictable results.

Updates on A register or counter in MC mode may cause loss of match in the current cycle if the transfer occurs near the match. In this case, the counter may rollover and resume operation in the next cycle.

Register B2 has no effect in MC mode. Nevertheless, register B2 can be accessed for reads and writes by addressing EMIOSB.

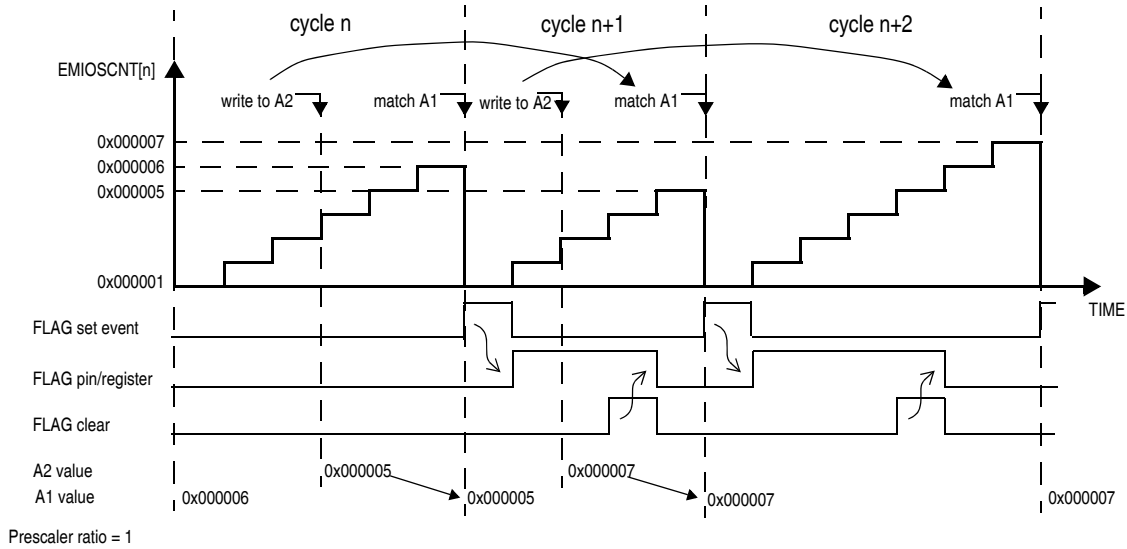
Figure 27-30 and Figure 27-31 show how the Unified Channel can be used as modulus counter in up mode and up/down mode, respectively.



**Figure 27-30. Modulus Counter Up mode example**

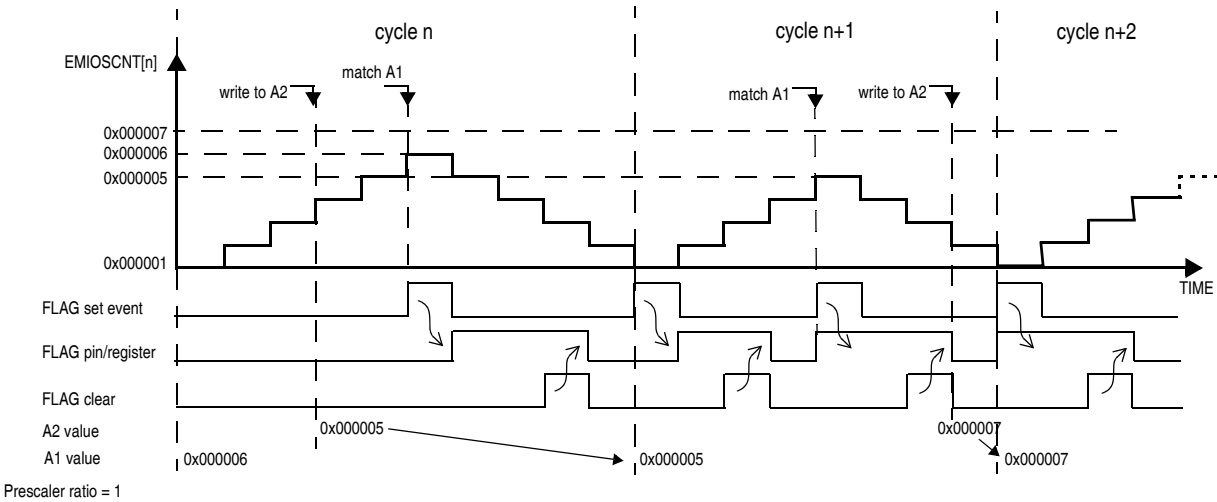


the next cycle boundary and therefore will be used on cycle **n+1**. The cycle boundary between cycle **n** and cycle **n+1** is defined as when the internal counter transitions from A1 value in cycle **n** to 0x1 in cycle **n+1**. Note that the FLAG is generated at the cycle boundary and has a synchronous operation, meaning that it is asserted one system clock cycle after the FLAG set event.



**Figure 27-32. Modulus Counter Buffered (MCB) Up Count mode**

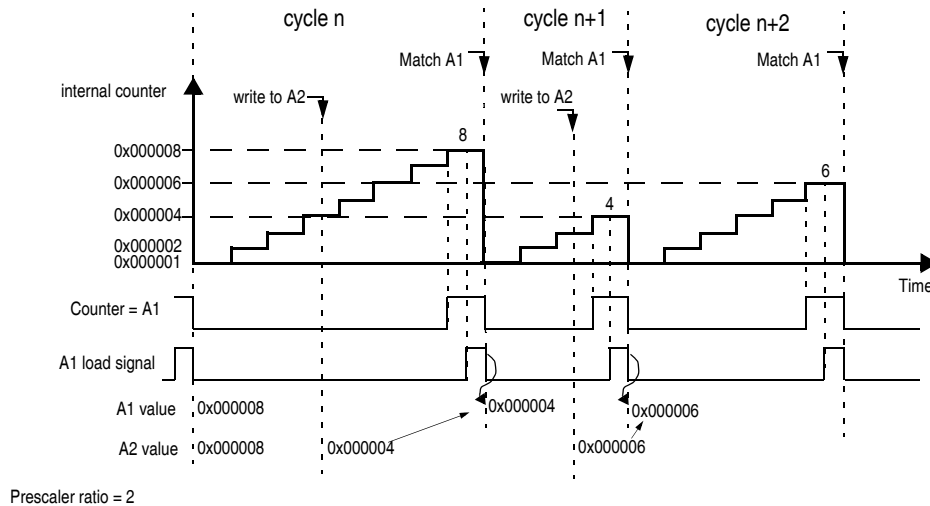
Figure 27-33 describes the MCB in up/down counter mode (MODE[0:6] = 10101bb). A1 register is updated at the cycle boundary. If A2 is written in cycle **n**, this new value will be used in cycle **n+1** for A1 match. Flags are generated only at A1 match start if MODE[5] is 0. If MODE[5] is set to 1 flags are also generated at the cycle boundary.



**Figure 27-33. Modulus Counter Buffered (MCB) Up/Down mode**

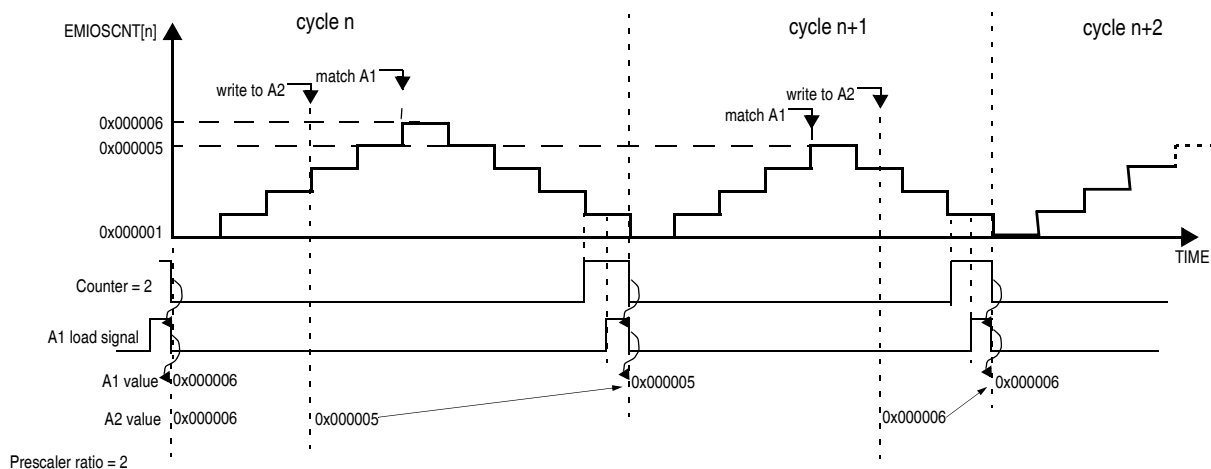
Figure 27-34 describes in more detail the A1 register update process in up counter mode. The A1 load signal is generated at the last system clock period of a counter cycle. Thus, A1 is updated with A2 value at the same time that the counter (EMIOSCNT[n]) is loaded with 0x1. The load signal pulse has the

duration of one system clock period. If A2 is written within cycle  $n$  its value is available at A1 at the first clock of cycle  $n+1$  and the new value is used for match at cycle  $n+1$ . The update disable bits  $OU[n]$  of EMIOSOUDIS register can be used to control the update of this register, thus allowing to delay the A1 register update for synchronization purposes.



**Figure 27-34. MCB Mode A1 Register Update in Up Counter mode**

Figure 27-35 describes the A1 register update in up/down counter mode. Note that A2 can be written at any time within cycle  $n$  in order to be used in cycle  $n+1$ . Thus A1 receives this new value at the next cycle boundary. Note that the update disable bits  $OU[n]$  of EMIOSOUDIS register can be used to disable the update of A1 register.



**Figure 27-35. MCB Mode A1 Register Update in Up/Down Counter mode**

#### 27.4.4.1.1.9 Output Pulse Width and Frequency Modulation Buffered (OPWFMB) mode

This mode ( $MODE[0:6] = 10110b0$ ) provides waveforms with variable duty cycle and frequency. The internal channel counter is automatically selected as the time base when this mode is selected. A1 register indicates the duty cycle and B1 register the frequency. Both A1 and B1 registers are double buffered to

allow smooth signal generation when changing the registers values on the fly. 0% and 100% duty cycles are supported.

At OPWFMB mode entry the output flip-flop is set to the value of the EDPOL bit in the EMIOSC[n] register.

If when entering OPWFMB mode coming out from GPIO mode the internal counter value is not within that range then the B match will not occur causing the channel internal counter to wrap at the maximum counter value, which is 0xFFFF for a 16-bit counter. After the counter wrap occurs it returns to 0x1 and resume normal OPWFMB mode operation. Thus in order to avoid the counter wrap condition make sure its value is within the 0x1 to B1 register value range when the OPWFMB mode is entered.

When a match on comparator A occurs the output register is set to the value of EDPOL. When a match on comparator B occurs the output register is set to the complement of EDPOL. B1 match also causes the internal counter to transition to 0x1, thus restarting the counter cycle.

Only values greater than 0x1 are allowed to be written to B1 register. Loading values other than those leads to unpredictable results. If you want to configure the module for OPWFMB mode, ensure that the B1 register is modified before the mode is set.

Figure 27-36 describes the operation of the OPWFMB mode regarding output pin transitions and A1/B1 registers match events. Note that the output pin transition occurs when the A1 or B1 match signal is deasserted, which is indicated by the A1 match negedge detection signal. If register A1 is set to 0x4 the output pin transitions 4 counter periods after the cycle had started, plus one system clock cycle. Note that in the example shown in Figure 27-36 the internal counter prescaler has a ratio of two.

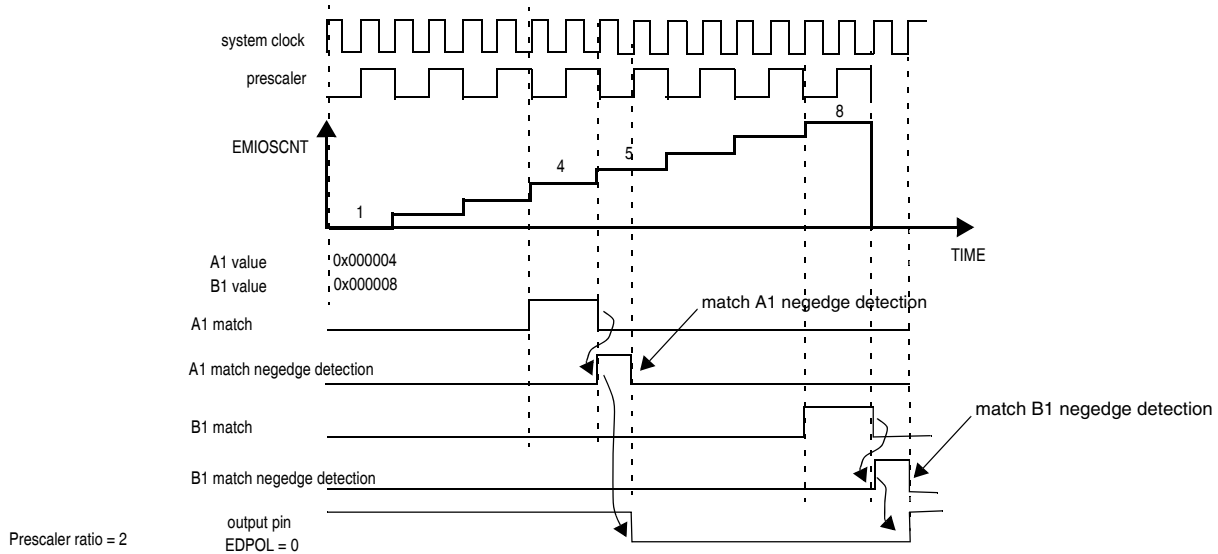
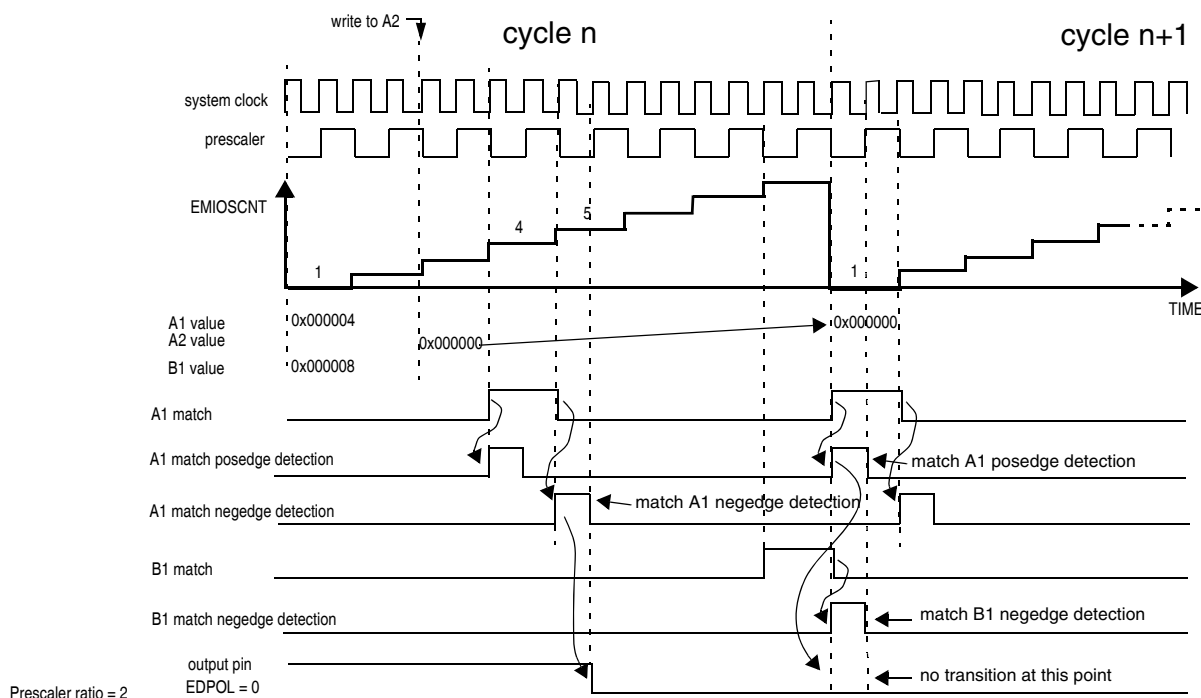


Figure 27-36. OPWFMB A1 and B1 match to Output Register Delay

Figure 27-37 describes the generated output signal if A1 is set to 0x0. Since the counter does not reach zero in this mode, the channel internal logic infers a match as if A1 = 0x1 with the difference that in this case, the posedge of the match signal is used to trigger the output pin transition instead of the negedge used when A1 = 0x1. Note that A1 posedge match signal from cycle  $n+1$  occurs at the same time as B1 negedge

match signal from cycle  $n$ . This allows to use the A1 posedge match to mask the B1 negedge match when they occur at the same time. The result is that no transition occurs on the output flip-flop and a 0% duty cycle is generated.



**Figure 27-37. OPWFMB Mode with A1 = 0 (0% duty cycle)**

Figure 27-38 describes the timing for the A1 and B1 registers load. The A1 and B1 load use the same signal, which is generated at the last system clock period of a counter cycle. Thus, A1 and B1 are updated respectively with A2 and B2 values at the same time that the counter (EMIOSCNT[n]) is loaded with 0x1. This event is defined as the cycle boundary. The load signal pulse has the duration of one system clock period. If A2 and B2 are written within cycle  $n$  their values are available at A1 and B1, respectively, at the first clock of cycle  $n+1$  and the new values are used for matches at cycle  $n+1$ . The update disable bits OU[n] of EMIOSOUDIS register can be used to control the update of these registers, thus allowing to delay the A1 and B1 registers update for synchronization purposes.

In Figure 27-38 it is assumed that both the channel and global prescalers are set to 0x1 (each divide ratio is two), meaning that the channel internal counter transitions at every four system clock cycles. FLAGS can be generated only on B1 matches when MODE[5] is cleared, or on both A1 and B1 matches when MODE[5] is set. Since B1 flag occurs at the cycle boundary, this flag can be used to indicate that A2 or B2 data written on cycle  $n$  were loaded to A1 or B1, respectively, thus generating matches in cycle  $n+1$ . Note that the FLAG has a synchronous operation, meaning that it is asserted one system clock cycle after the FLAG set event.

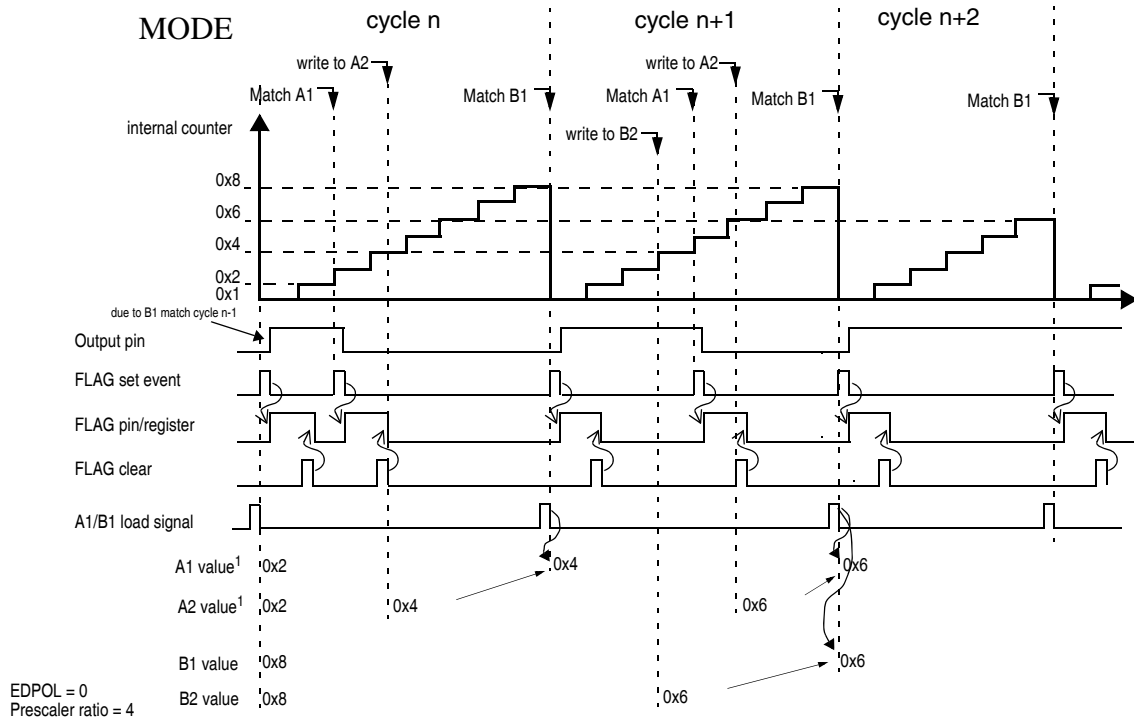


Figure 27-38. OPWFMB A1 and B1 registers update and flags

The FORCMA and FORCMB bits allow the software to force the output flip-flop to the level corresponding to a match on comparators A or B respectively. Similarly to a B1 match FORCMB sets the internal counter to 0x1. The FLAG bit is not set by the FORCMA or FORCMB bits being asserted.

Figure 27-39 describes the generation of 100% and 0% duty cycle signals. It is assumed EDPOL = 0 and the resultant prescaler value is 1. Initially A1 = 0x8 and B1 = 0x8. In this case, B1 match has precedence over A1 match, thus the output flip-flop is set to the complement of EDPOL bit. This cycle corresponds to a 100% duty cycle signal. The same output signal can be generated for any A1 value greater or equal to B1.

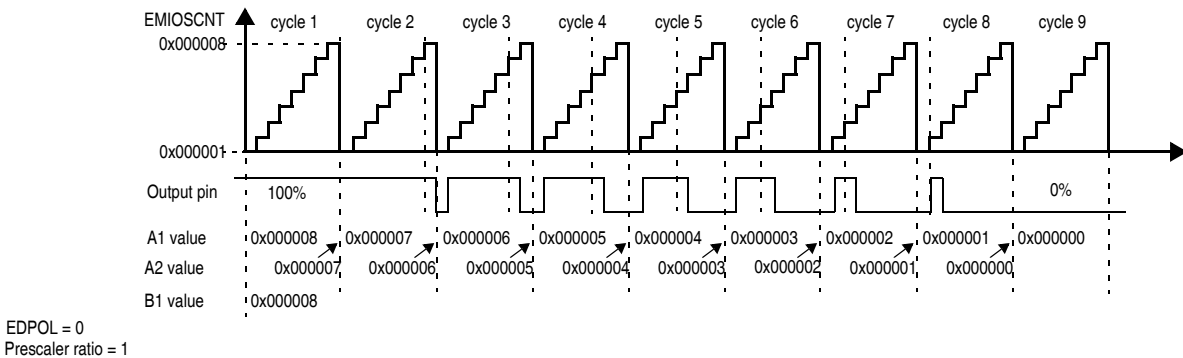


Figure 27-39. OPWFMB mode from 100% to 0% duty cycle

A 0% duty cycle signal is generated if A1 = 0x0 as shown in Figure 27-39 cycle 9. In this case B1 = 0x8 match from cycle 8 occurs at the same time as the A1 = 0x0 match from cycle 9. Please, refer to



Figure 27-37 for a description of the A1 and B1 match generation. In this case A1 match has precedence over B1 match and the output signal transitions to EDPOL.

#### 27.4.4.1.10 Center Aligned Output PWM Buffered with Dead Time (OPWMCB) mode

This operation mode generates a center aligned PWM with dead time insertion to the leading (MODE[0:6] = 10111b1) or trailing edge (MODE[0:6] = 10111b0). A1 and B1 registers are double buffered to allow smooth output signal generation when changing A2 or B2 registers values on the fly.

Bits BSL[0:1] select the time base. The time base selected for a channel configured to OPWMCB mode should be a channel configured to MCB Up/Down mode, as shown in Figure 27-33. It is recommended to start the MCB channel time base after the OPWMCB mode is entered in order to avoid missing A matches at the very first duty cycle.

Register A1 contains the ideal duty cycle for the PWM signal and is compared with the selected time base.

Register B1 contains the dead time value and is compared against the internal counter. For a leading edge dead time insertion, the output PWM duty cycle is equal to the difference between register A1 and register B1, and for a trailing edge dead time insertion, the output PWM duty cycle is equal to the sum of register A1 and register B1. Bit Mode[6] selects between trailing and leading dead time insertion, respectively.

#### NOTE

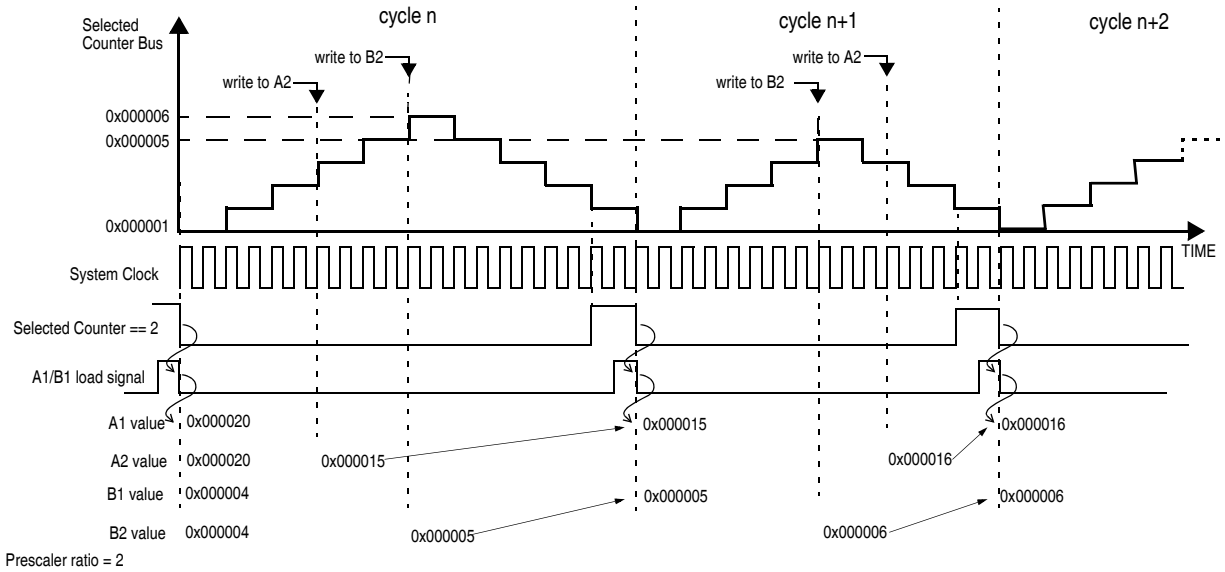
The internal counter runs in the internal prescaler ratio, while the selected time base may be running in a different prescaler ratio.

When OPWMCB mode is entered, coming out from GPIO mode, the output flip-flop is set to the complement of the EDPOL bit in the EMIOSC[n] register.

The following basic steps summarize proper OPWMCB startup, assuming the channels are initially in GPIO mode:

1. [global] Disable Global Prescaler;
2. [MCB channel] Disable Channel Prescaler;
3. [MCB channel] Write 0x1 at internal counter;
4. [MCB channel] Set A register;
5. [MCB channel] Set channel to MCB Up mode;
6. [MCB channel] Set prescaler ratio;
7. [MCB channel] Enable Channel Prescaler;
8. [OPWMCB channel] Disable Channel Prescaler;
9. [OPWMCB channel] Set A register;
10. [OPWMCB channel] Set B register;
11. [OPWMCB channel] Select time base input through BSL[1:0] bits;
12. [OPWMCB channel] Enter OPWMCB mode;
13. [OPWMCB channel] Set prescaler ratio;
14. [OPWMCB channel] Enable Channel Prescaler;
15. [global] Enable Global Prescaler.

Figure 27-40 describes the load of A1 and B1 registers that occurs when the selected counter bus transitions from 0x2 to 0x1. This event defines the cycle boundary. Note that values written to A2 or B2 within cycle *n* are loaded into A1 or B1 registers, respectively, and used to generate matches in cycle *n+1*.

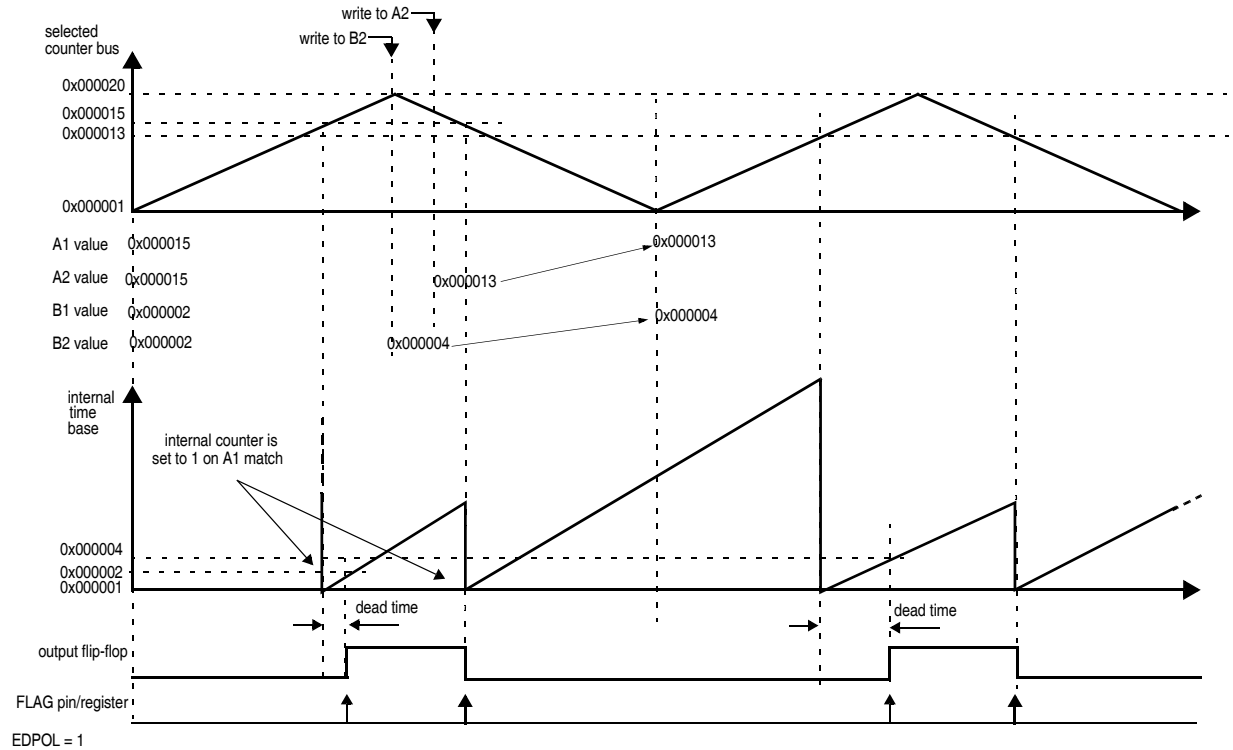


**Figure 27-40. OPWMCB A1 and B1 registers load**

Bit OU[n] of the EMIOSOUDIS register can be used to disable the A1 and B1 updates, thus allowing to synchronize the load on these registers with the load of A1 or B1 registers in others channels. Note that using the update disable bit A1 and B1 registers can be updated at the same counter cycle thus allowing to change both registers at the same time.

In this mode A1 matches always sets the internal counter to 0x1. When operating with leading edge dead time insertion the first A1 match sets the internal counter to 0x1. When a match occurs between register B1 and the internal time base, the output flip-flop is set to the value of the EDPOL bit. In the following match between register A1 and the selected time base, the output flip-flop is set to the complement of the EDPOL bit. This sequence repeats continuously. The internal counter should not reach 0x0 as consequence of a rollover. In order to avoid it the user should not write to the EMIOSB register a value greater than twice the difference between external count up limit and EMIOSA value.

Figure 27-41 shows two cycles of a Center Aligned PWM signal. Note that both A1 and B1 register values are changing within the same cycle, which allows to vary at the same time the duty cycle and dead time values.



**Figure 27-41. OPWMCB with lead dead time insertion**

When operating with trailing edge dead time insertion, the first match between A1 and the selected time base sets the output flip-flop to the value of the EDPOL bit and sets the internal counter to 0x1. In the second match between register A1 and the selected time base, the internal counter is set to 0x1 and B1 matches are enabled. When the match between register B1 and the selected time base occurs the output flip-flop is set to the complement of the EDPOL bit. This sequence repeats continuously.

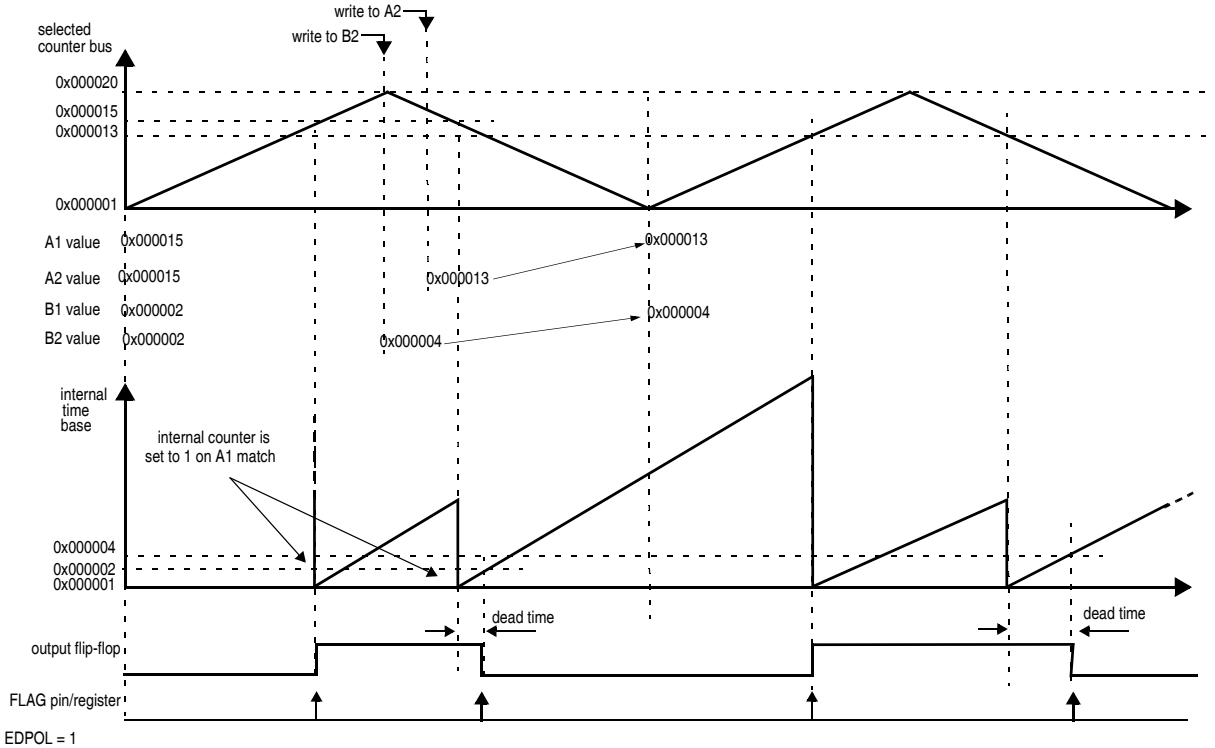


Figure 27-42. OPWMCB with trail dead time insertion

FLAG can be generated in the trailing edge of the output PWM signal when MODE[5] is cleared, or in both edges, when MODE[5] is set. If subsequent matches occur on comparators A and B, the PWM pulses continue to be generated, regardless of the state of the FLAG bit.

**NOTE**

In OPWMCB mode, FORCMA and FORCMB do not have the same behavior as a regular match. Instead, they force the output flip-flop to a constant value that depends upon the selected dead time insertion mode, lead or trail, and the value of the EDPOL bit.

FORCMA has different behaviors depending upon the selected dead time insertion mode, lead or trail. In lead dead time insertion FORCMA force a transition in the output flip-flop to the opposite of EDPOL. In trail dead time insertion the output flip-flop is forced to the value of EDPOL bit.

If bit FORCMB is set, the output flip-flop value depends upon the selected dead time insertion mode. In lead dead time insertion FORCMB forces the output flip-flop to transition to EDPOL bit value. In trail dead time insertion the output flip-flop is forced to the opposite of EDPOL bit value.

**NOTE**

FORCMA bit set does not set the internal time-base to 0x1 as a regular A1 match.

The FLAG bit is not set either in case of a FORCMA or FORCMB or even if both forces are issued at the same time.

**NOTE**

FORCMA and FORCMB have the same behavior even in Freeze or normal mode regarding the output pin transition.

When FORCMA is issued along with FORCMB the output flip-flop is set to the opposite of EDPOL bit value. This is equivalent of saying that FORCMA has precedence over FORCMB when lead dead time insertion is selected and FORCMB has precedence over FORCMA when trail dead time insertion is selected.

Duty cycle from 0% to 100% can be generated by setting appropriate values to A1 and B1 registers relatively to the period of the external time base. Setting  $A1 = 1$  generates a 100% duty cycle waveform. Assuming EDPOL is set to 1 and OPWMCB mode with trail dead time insertion, 100% duty cycle signals can be generated if B1 occurs at or after the cycle boundary (external counter = 1). If A1 is greater than the maximum value of the selected counter bus period, then a 0% duty cycle is produced, only if the pin starts the current cycle in the opposite of EDPOL value. In case of 100% duty cycle, the transition from EDPOL to the opposite of EDPOL may be obtained by forcing pin, using FORCMA or FORCMB, or both.

**NOTE**

If A1 is set to 0x1 at OPWMCB entry the 100% duty cycle may not be obtained in the very first PWM cycle due to the pin condition at mode entry.

Only values different than 0x0 are allowed to be written to A1 register. If 0x0 is loaded to A1 the results are unpredictable.

**NOTE**

A special case occurs when A1 is set to  $(\text{external counter bus period})/2$ , which is the maximum value of the external counter. In this case the output flip-flop is constantly set to the EDPOL bit value.

The internal channel logic prevents matches from one cycle to propagate to the next cycle. In trail dead time insertion B1 match from cycle  $n$  could eventually cross the cycle boundary and occur in cycle  $n+1$ . In this case B1 match is masked out and does not cause the output flip-flop to transition. Therefore matches in cycle  $n+1$  are not affected by the late B1 matches from cycle  $n$ .

Figure 27-43 shows a 100% duty cycle output signal generated by setting  $A1 = 4$  and  $B1 = 3$ . In this case the trailing edge is positioned at the boundary of cycle  $n+1$ , which is actually considered to belong to cycle  $n+2$  and therefore does not cause the output flip-flip to transition.

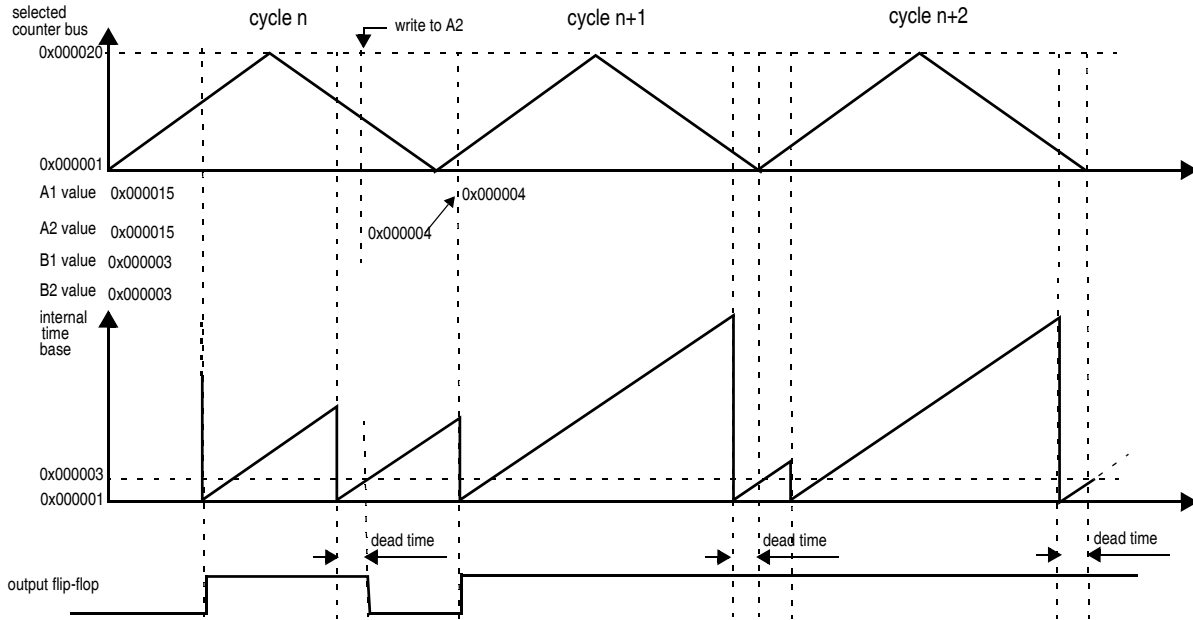


Figure 27-43. OPWMCB with 100% Duty Cycle (A1 = 4 and B1 = 3)

It is important to notice that, such as in OPWMB and OPWFMB modes, the match signal used to set or clear the channel output flip-flop is generated on the deassertion of the channel combinational comparator output signal, which compares the selected time base with A1 or B1 register values. Please refer to [Figure 27-36](#), which describes the delay from matches to output flip-flop transition in OPWFMB mode. The operation of OPWMCB mode is similar to OPWFMB regarding matches and output pin transition.

#### 27.4.4.1.11 Output Pulse Width Modulation Buffered (OPWMB) Mode

OPWMB mode (MODE[0:6] = 11000b0) is used to generate pulses with programmable leading and trailing edge placement. An external counter driven in MCB Up mode must be selected from one of the counter buses. A1 register value defines the first edge and B1 the second edge. The output signal polarity is defined by the EDPOL bit. If EDPOL is zero, a negative edge occurs when A1 matches the selected counter bus and a positive edge occurs when B1 matches the selected counter bus.

The A1 and B1 registers are double buffered and updated from A2 and B2, respectively, at the cycle boundary. The load operation is similar to the OPWFMB mode. Please refer to [Figure 27-38](#) for more information about A1 and B1 registers update.

FLAG can be generated at B1 matches, when MODE[5] is cleared, or in both A1 and B1 matches, when MODE[5] is set. If subsequent matches occur on comparators A and B, the PWM pulses continue to be generated, regardless of the state of the FLAG bit.

FORCMA and FORCMB bits allow the software to force the output flip-flop to the level corresponding to a match on A1 or B1 respectively. FLAG bit is not set by the FORCMA and FORCMB operations.

At OPWMB mode entry the output flip-flop is set to the value of the EDPOL bit in the EMIOSC[n] register.

Some rules applicable to the OPWMB mode are:

- B1 matches have precedence over A1 matches if they occur at the same time within the same counter cycle
- A1 = 0 match from cycle  $n$  has precedence over B1 match from cycle  $n - 1$
- A1 matches are masked out if they occur after B1 match within the same cycle
- Any value written to A2 or B2 on cycle  $n$  is loaded to A1 and B1 registers at the following cycle boundary (assuming OU[n] bit of EMIOSOUDIS register is not asserted). Thus the new values will be used for A1 and B1 matches in cycle  $n+1$

Figure 27-44 describes the operation of the OPWMB mode regarding A1 and B1 matches and the transition of the channel output pin. In this example EDPOL is set to 0.

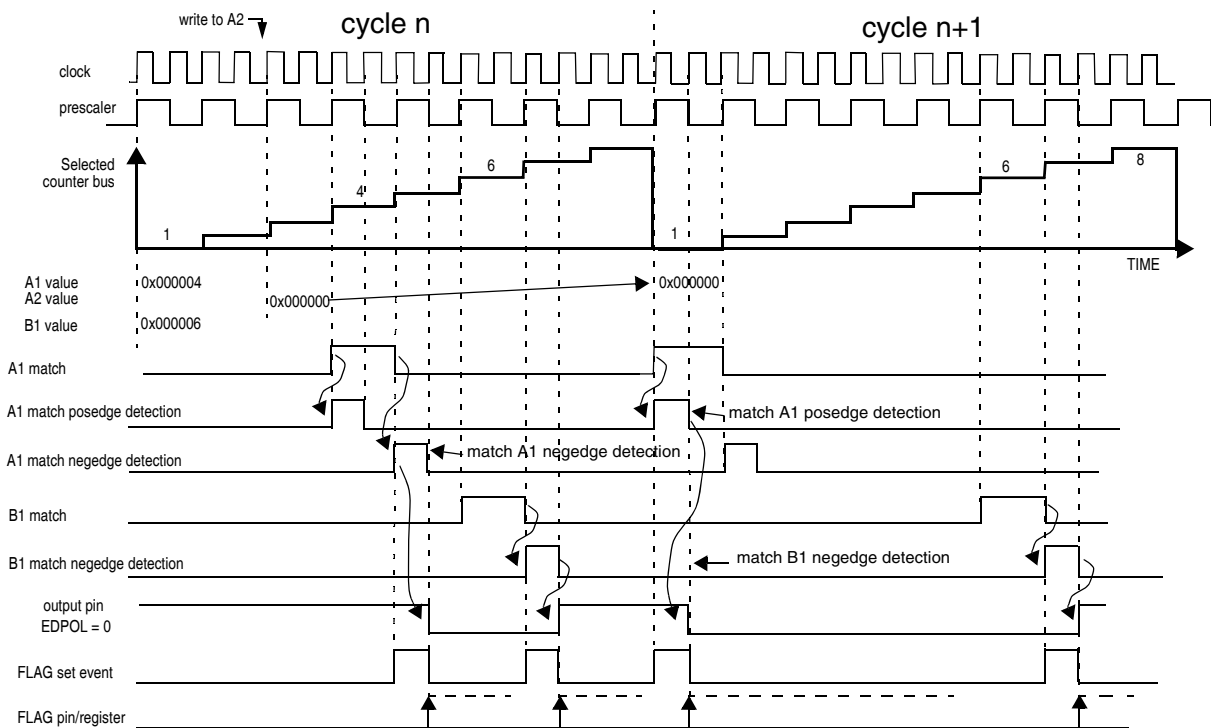


Figure 27-44. OPWMB mode matches and flags

Note that the output pin transitions are based on the negedges of the A1 and B1 match signals.

Figure 27-44 shows in cycle  $n+1$  the value of A1 register being set to 0. In this case the match posedge is used instead of the negedge to transition the output flip-flop.

Figure 27-45 describes the channel operation for 0% duty cycle. Note that the A1 match posedge signal occurs at the same time as the B1 = 0x8 negedge signal. In this case A1 match has precedence over B1 match, causing the output pin to remain at EDPOL bit value, thus generating a 0% duty cycle signal.

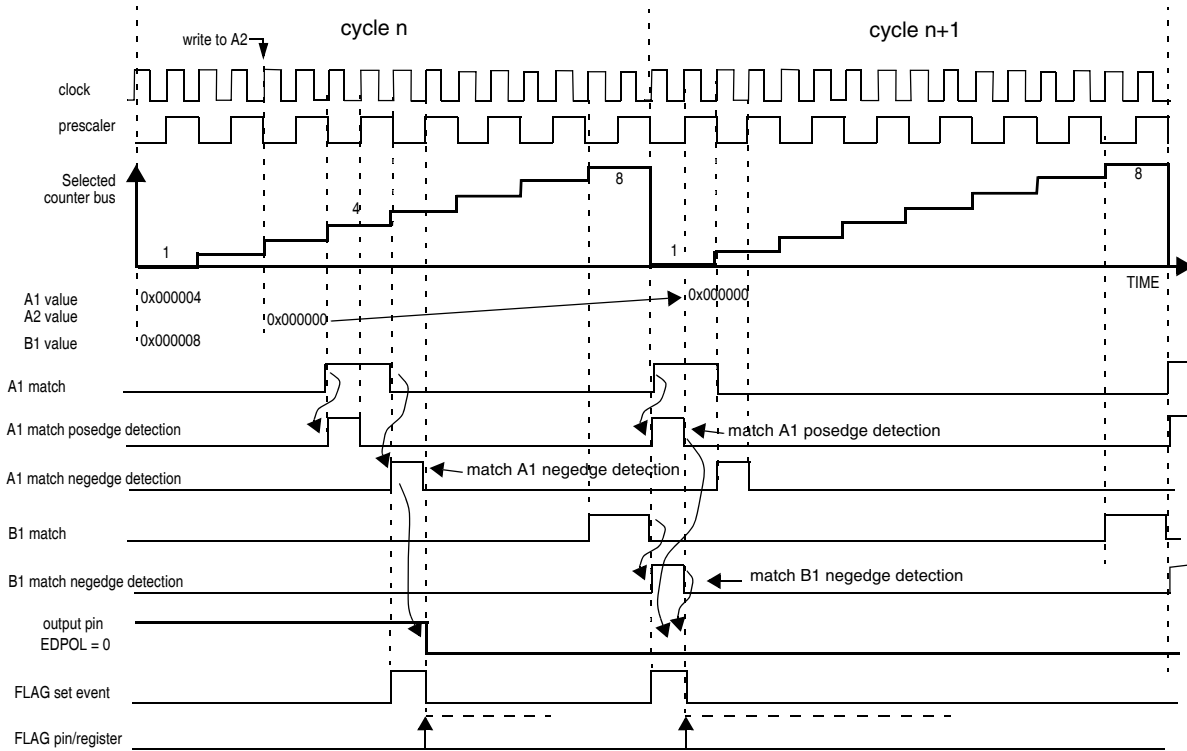


Figure 27-45. OPWMB mode with 0% duty cycle

Figure 27-46 shows a waveform changing from 100% to 0% duty cycle. EDPOL in this case is zero. In this example B1 is programmed to the same value as the period of the external selected time base.

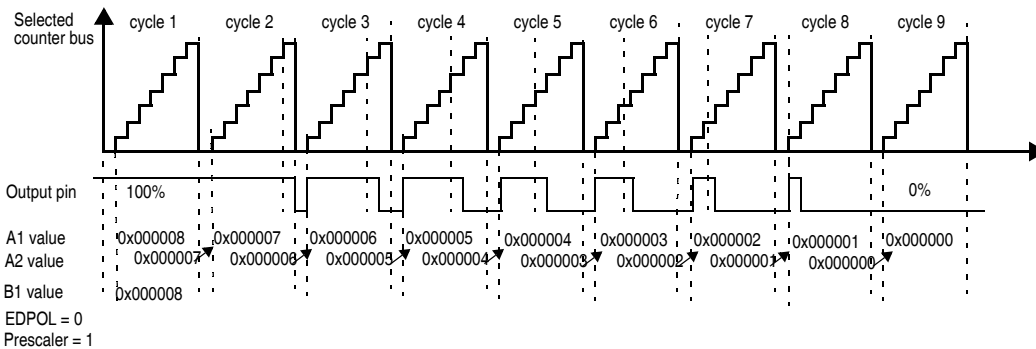


Figure 27-46. OPWMB mode from 100% to 0% duty cycle

In Figure 27-46 if B1 is set to a value lower than 0x8 it is not possible to achieve 0% duty cycle by only changing A1 register value. Since B1 matches have precedence over A1 matches the output pin transitions to the opposite of EDPOL bit at B1 match. Note also that if B1 is set to 0x9, for instance, B1 match does not occur, thus a 0% duty cycle signal is generated.



### 27.4.4.1.12 Output Pulse Width Modulation with Trigger (OPWMT) mode

OPWMT mode (MODE[0:6] = 0100110) is intended to support the generation of pulse width modulation signals where the period is not modified while the signal is being output, but where the duty cycle will be varied and must not create glitches. The mode is intended to be used in conjunction with other channels executing in the same mode and sharing a common timebase. It will support each channel with a fixed PWM leading edge position with respect to the other channels and the ability to generate a trigger signal at any point in the period that can be output from the module to initiate activity in other parts of the device such as starting ADC conversions.

An external counter driven in either MC Up or MCB Up mode must be selected from one of the counter buses.

Register A1 defines the leading edge of the PWM output pulse and as such the beginning of the PWM's period. This makes it possible to insure that the leading edge of multiple channels in OPWMT mode can occur at a specific time with respect to the other channels when using a shared timebase. This can allow the introduction of a fixed offset for each channel, which can be particularly useful in the generation of lighting PWM control signals where it is desirable that edges are not coincident with each other to help eliminate noise generation. The value of register A1 represents the shift of the PWM channel with respect to the selected timebase. A1 can be configured with any value within the range of the selected time base. Note that registers loaded with 0x0 will not produce matches if the timebase is driven by a channel in MCB mode.

A1 is not buffered as the shift of a PWM channel must not be modified while the PWM signal is being generated. In case A1 is modified it is immediately updated and one PWM pulse could be lost.

EMIOSB[n] address gives access to B2 register for write and B1 register for read. Register B1 defines the trailing edge of the PWM output pulse and as such the duty cycle of the PWM signal. To synchronize B1 update with the PWM signal and so ensure a correct output pulse generation the transfer from B2 to B1 is done at every match of register A1.

EMIOSOUDIS register affects transfers between B2 and B1 only.

In order to account for the shift in the leading edge of the waveform defined by register A1 it will be necessary that the trailing edge, held in register B1, can roll over into the next period. This means that a match against the B1 register should not have to be qualified by a match in the A1 register. The impact of this would mean that incorrectly setting register B1 to a value less than register A1 will result in the output being held over a cycle boundary until the B1 value is encountered.

This mode provides a buffered update of the trailing edge by updating register B1 with register B2 contents only at a match of register A1.

The value loaded in register A1 is compared with the value on the selected time base. When a match on comparator A1 occurs, the output flip-flop is set to the value of the EDPOL bit. When a match occurs on comparator B, the output flip-flop is set to the complement of the EDPOL bit.

Note that the output pin and flag transitions are based on the posedges of the A1, B1 and A2 match signals. Please, refer to [Figure 27-44](#) at [Section 27.4.4.1.11, Output Pulse Width Modulation Buffered \(OPWMB\) Mode](#), for details on match posedge.

Register A2 defines the generation of a trigger event within the PWM period. A2 should be configured with any value within the range of the selected time base. Otherwise, no trigger will be generated. A match on the comparator will generate the FLAG signal but it has no effect on the PWM output signal generation. The typical setup to obtain a trigger with FLAG is to enable DMA and to drive the channel's ipd\_done input high.

A2 is not buffered and therefore its update is immediate. If the channel is running when a change is made, this could cause either the loss of one trigger event or the generation of two trigger events within the same period. Register A2 can be accessed by reading or writing the eMIOS UC Alternate A Register (EMIOSALTA) at UC[n] base address +0x14.

FLAG signal is set only at match on the comparator with A2. A match on the comparator with A1 or B1 or B2 has no effect on FLAG.

At any time, the FORCMA and FORCMB bits allow the software to force the output flip-flop to the level corresponding to a match on A or B respectively. Any FORCMA and/or FORCMB has priority over any simultaneous match regarding to output pin transitions. Note that the load of B2 content on B1 register at an A match is not inhibited due to a simultaneous FORCMA/FORCMB assertion. If both FORCMA and FORCMB are asserted simultaneously the output pin goes to the opposite of EDPOL value such as if A1 and B1 registers had the same value. FORCMA assertion causes the transfer from register B2 to B1 such as a regular A match, regardless of FORCMB assertion.

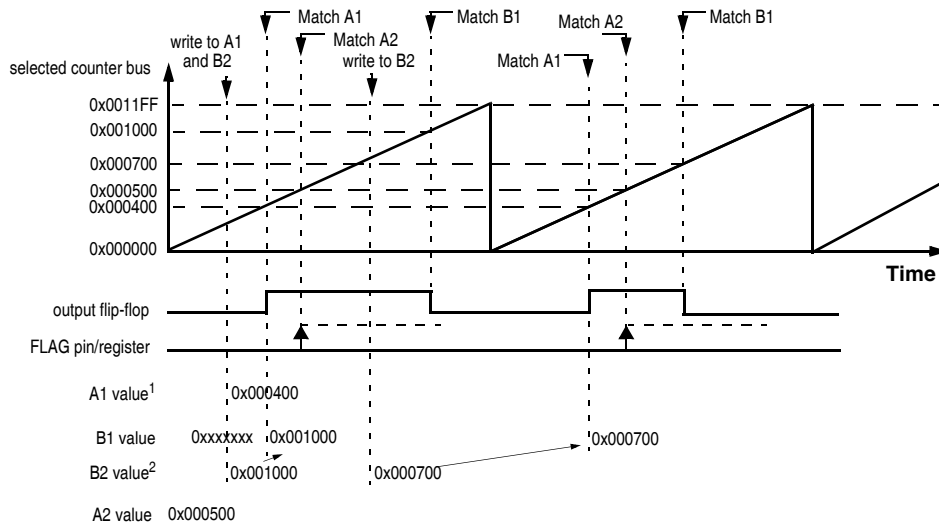
If subsequent matches occur on comparators A1 and B, the PWM pulses continue to be generated, regardless of the state of the FLAG bit.

At OPWMT mode entry the output flip-flop is set to the complement of the EDPOL bit in the EMIOSC[n] register.

In order to achieve 0% duty cycle both registers A1 and B must be set to the same value. When a simultaneous match on comparators A and B occur, the output flip-flop is set at every period to the complement value of EDPOL.

In order to achieve 100% duty cycle the register B1 must be set to a value greater than maximum value of the selected time base. As a consequence, if 100% duty cycle must be implemented, the maximum counter value for the time base is 0xFFFFE for a 16-bit counter. When a match on comparator A1 occurs the output flip-flop is set at every period to the value of EDPOL bit. The transfer from register B2 to B1 is still triggered by the match at comparator A.

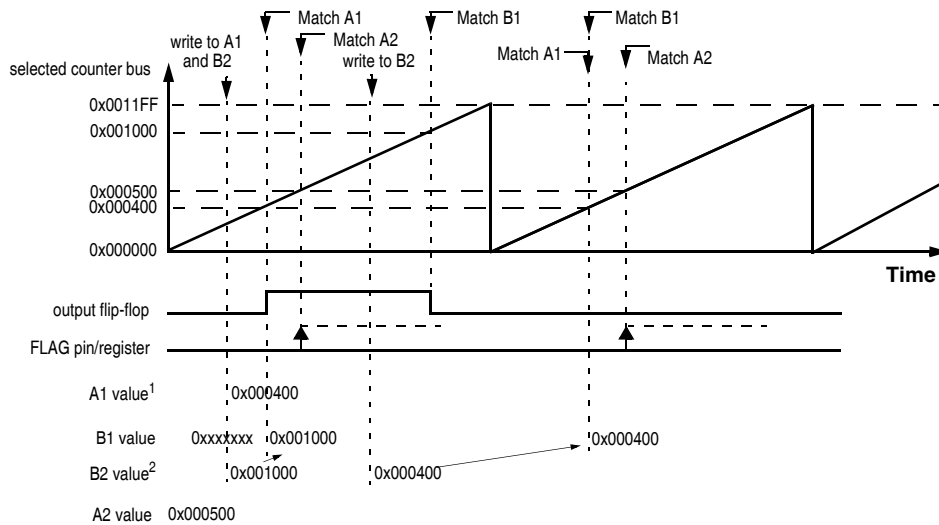
Figure 27-47 shows the Unified Channel running in OPWMT mode with Trigger Event Generation and duty cycle update on next period update.



Notes: 1. EMIOSA[n] = A1  
 Notes: 2. EMIOSB[n] = B2 for write, B1 for read

Figure 27-47. OPWMT example

Figure 27-48 shows the Unified Channel running in OPWMT mode with Trigger Event Generation and 0% duty.



Notes: 1. EMIOSA[n] = A1  
 Notes: 2. EMIOSB[n] = B2 for write, B1 for read

Figure 27-48. OPWMT with 0% Duty Cycle

Figure 27-49 shows the Unified Channel running in OPWMT mode with Trigger Event Generation and 100% duty cycle.

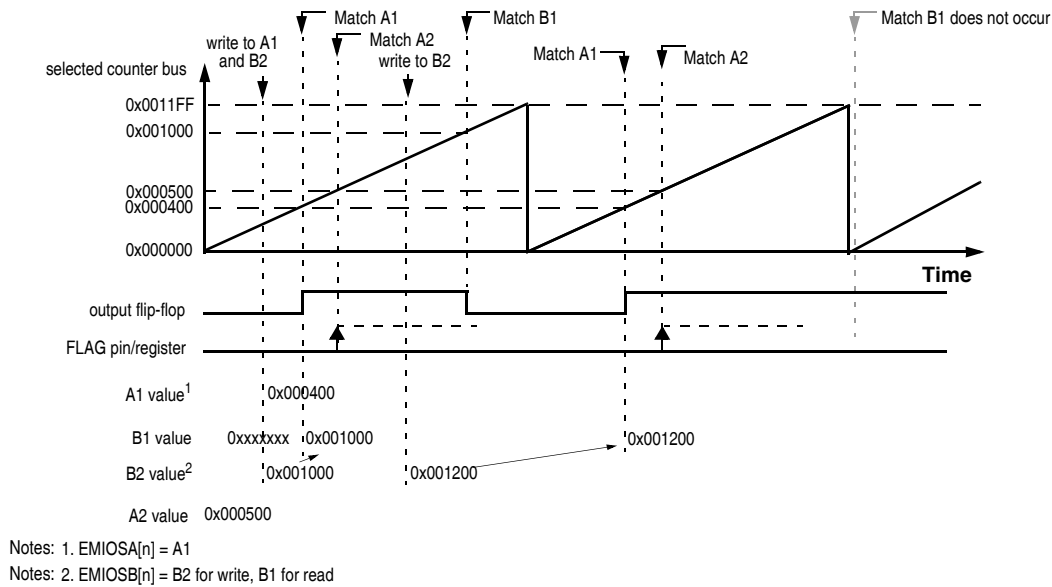


Figure 27-49. OPWMT with 100% duty cycle

### 27.4.4.1.2 Input Programmable Filter (IPF)

The IPF ensures that only valid input pin transitions are received by the Unified Channel edge detector. A block diagram of the IPF is shown in Figure 27-50.

The IPF is a 5-bit programmable up counter that is incremented by the selected clock source, according to bits IF[0:3] in EMIOSC[n] register.

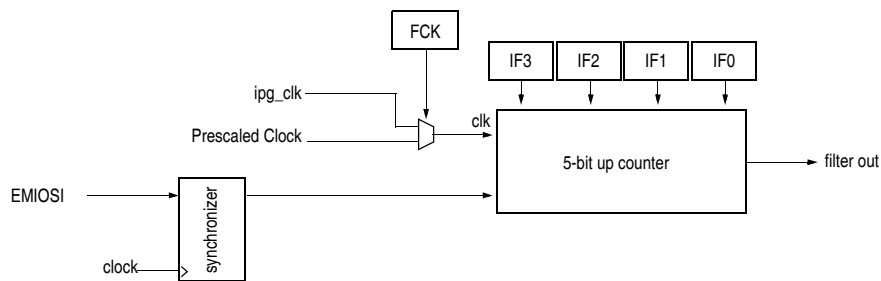
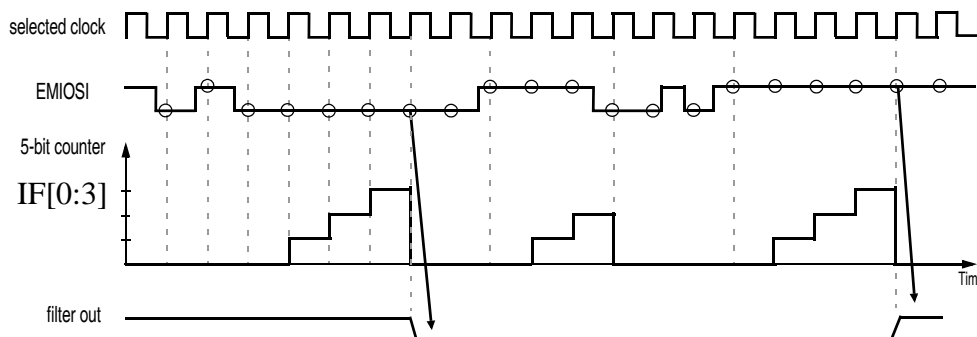


Figure 27-50. Input programmable filter submodule diagram

The input signal is synchronized by system clock. When a state change occurs in this signal, the 5-bit counter starts counting up. As long as the new state is stable on the pin, the counter remains incrementing. If a counter overflow occurs, the new pin value is validated. In this case, it is transmitted as a pulse edge to the edge detector. If the opposite edge appears on the pin before validation (overflow), the counter is reset. At the next pin transition, the counter starts counting again. Any pulse that is shorter than a full range of the masked counter is regarded as a glitch and it is not passed on to the edge detector. A timing diagram of the input filter is shown in Figure 27-51.



**Figure 27-51. Input programmable filter example**

The filter is not disabled during either freeze state or negated GTBE input.

#### 27.4.4.1.3 Clock Prescaler (CP)

The CP divides the GCP output signal to generate a clock enable for the internal counter of the Unified Channels. The GCP output signal is prescaled by the value defined in [Figure 27-18](#) according to the UCPRE[0:1] bits in EMIOSC[n] register. The prescaler is enabled by setting the UCPREN bit in the EMIOSC[n] and can be stopped at any time by clearing this bit, thereby stopping the internal counter in the Unified Channel.

In order to ensure safe working and avoid glitches the following steps must be performed whenever any update in the prescaling rate is desired:

1. Write 0 at both GPREN bit in EMIOSMCR register and UCPREN bit in EMIOSC[n] register, thus disabling prescalers.
2. Write the desired value for prescaling rate at UCPRE[0:1] bits in EMIOSC[n] register.
3. Enable channel prescaler by writing 1 at UCPREN bit in EMIOSC[n] register.
4. Enable global prescaler by writing 1 at GPREN bit in EMIOSMCR register.

The prescaler is not disabled during either freeze state or negated GTBE input.

#### 27.4.4.1.4 Effect of Freeze on the Unified Channel

When in debug mode, bit FRZ in the EMIOSMCR and bit FREN in the EMIOSC[n] register are both set, the internal counter, and the Unified Channel capture and compare functions are halted. The UC is frozen in its current state.

During freeze, all registers are accessible. When the Unified Channel is operating in an output mode, the force match functions remain available, allowing the software to force the output to the desired level.

Note that for input modes, any input events that may occur while the channel is frozen are ignored.

When exiting debug mode or freeze enable bit is cleared (FRZ in the EMIOSMCR or FREN in the EMIOSC[n] register) the channel actions resume, but may be inconsistent until channel enters GPIO mode again.

### 27.4.4.2 IP Bus Interface Unit (BIU)

The BIU provides the interface between the Internal Interface Bus (IIB) and the Peripheral Bus, allowing communication among all submodules and this IP interface.

The BIU allows 8-, 16-, and 32-bit access. They are performed over a 32-bit data bus in a single cycle clock.

#### 27.4.4.2.1 Effect of Freeze on the BIU

When the FRZ bit in the EMIOSMCR is set and the module is in debug mode, the operation of BIU is not affected.

### 27.4.4.3 Global Clock Prescaler Submodule (GCP)

The GCP divides the system clock to generate a clock for the CPs of the channels. The main clock signal is prescaled by the value defined in [Figure 27-12](#) according to bits GPRE[0:7] in the EMIOSMCR. The global prescaler is enabled by setting the GPREN bit in the EMIOSMCR and can be stopped at any time by clearing this bit, thereby stopping the internal counters in all the channels.

In order to ensure safe working and avoid glitches the following steps must be performed whenever any update in the prescaling rate is desired:

1. Write 0 at GPREN bit in EMIOSMCR, thus disabling global prescaler.
2. Write the desired value for prescaling rate at GPRE[0:7] bits in EMIOSMCR.
3. Enable global prescaler by writing 1 at GPREN bit in EMIOSMCR.

The prescaler is not disabled during either freeze state or negated GTBE input.

#### 27.4.4.3.1 Effect of Freeze on the GCP

When the FRZ bit in the EMIOSMCR is set and the module is in debug mode, the operation of GCP submodule is not affected, that is, there is no freeze function in this submodule.

## 27.4.5 Initialization/Application information

On resetting the eMIOS the Unified Channels enter GPIO input mode.

### 27.4.5.1 Considerations

Before changing an operating mode, the UC must be programmed to GPIO mode and EMIOSA[n] and EMIOSB[n] registers must be updated with the correct values for the next operating mode. Then the EMIOSC[n] register can be written with the new operating mode. If a UC is changed from one mode to another without performing this procedure, the first operation cycle of the selected time base can be random, that is, matches can occur in random time if the contents of EMIOSA[n] or EMIOSB[n] were not updated with the correct value before the time base matches the previous contents of EMIOSA[n] or EMIOSB[n].

When interrupts are enabled, the software must clear the FLAG bits before exiting the interrupt service routine.

### 27.4.5.2 Application information

Correlated output signals can be generated by all output operation modes. Bits OU[n] of the EMIOSOUDIS register can be used to control the update of these output signals.

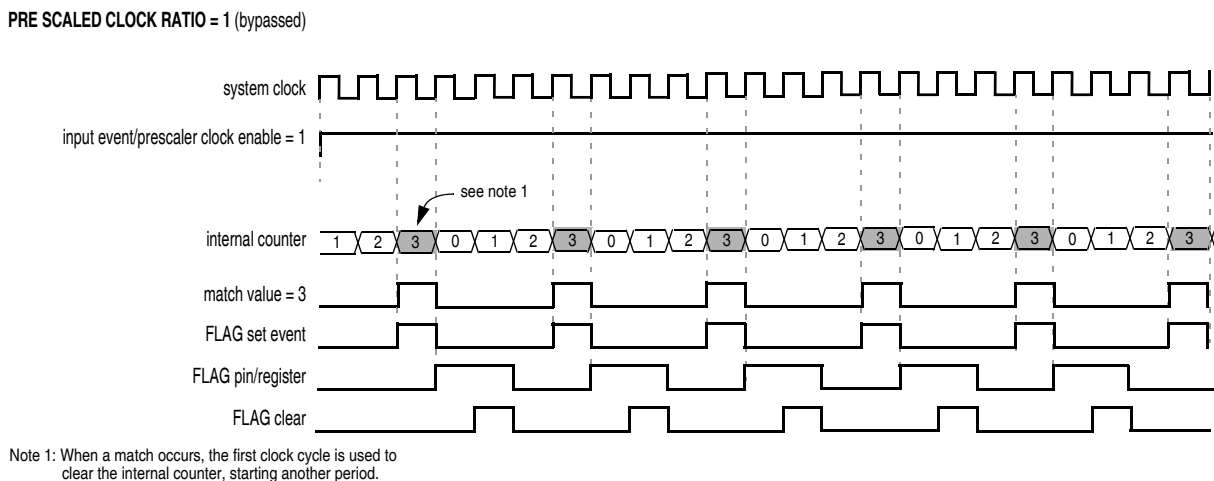
In order to guarantee that the internal counters of correlated channels are incremented in the same clock cycle, the internal prescalers must be set up before enabling the global prescaler. If the internal prescalers are set after enabling the global prescaler, the internal counters may increment in the same ratio, but at a different clock cycle.

#### 27.4.5.2.1 Time base generation

For MC with internal clock source operation modes, the internal counter rate can be modified by configuring the clock prescaler ratio. Figure 27-52 shows an example of a time base with prescaler ratio equal to one.

#### NOTE

MCB and OPWFMB modes have a different behavior.



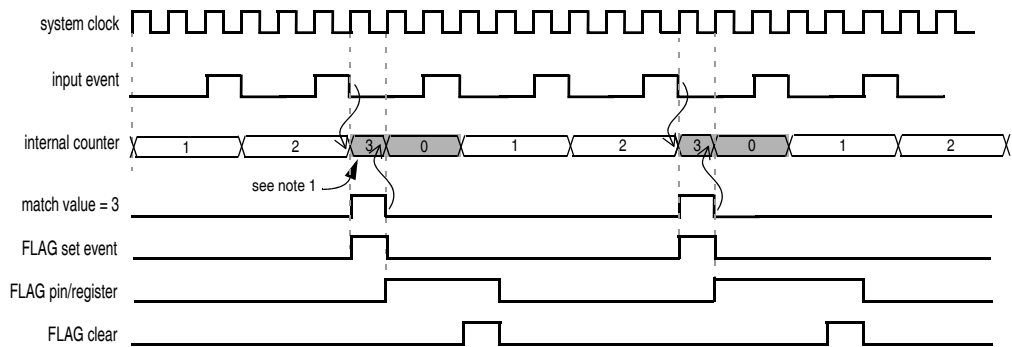
**Figure 27-52. Time base period when running in the fastest prescaler ratio**

If the prescaler ratio is greater than one or external clock is selected, the counter may behave in three different ways depending on the channel mode:

- If MC mode and Clear on Match Start and External Clock source are selected, the internal counter behaves as described in Figure 27-53.
- If MC mode and Clear on Match Start and Internal Clock source are selected, the internal counter behaves as described in Figure 27-54.
- If MC mode and Clear on Match End are selected, the internal counter behaves as described in Figure 27-55.

**NOTE**

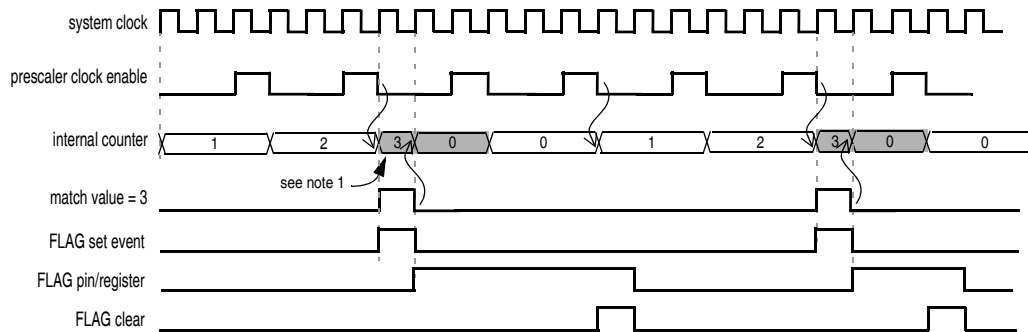
MCB and OPWFMB modes have a different behavior.



Note 1: When a match occurs, the first system clock cycle is used to clear the internal counter, and at the next edge of prescaler clock enable the counter will start counting.

**Figure 27-53. Time base generation with external clock and clear on match start**

PRESCALED CLOCK RATIO = 3



Note 1: When a match occurs, the first clock cycle is used to clear the internal counter, and only after a second edge of pre scaled clock the counter will start counting.

**Figure 27-54. Time base generation with internal clock and clear on match start**



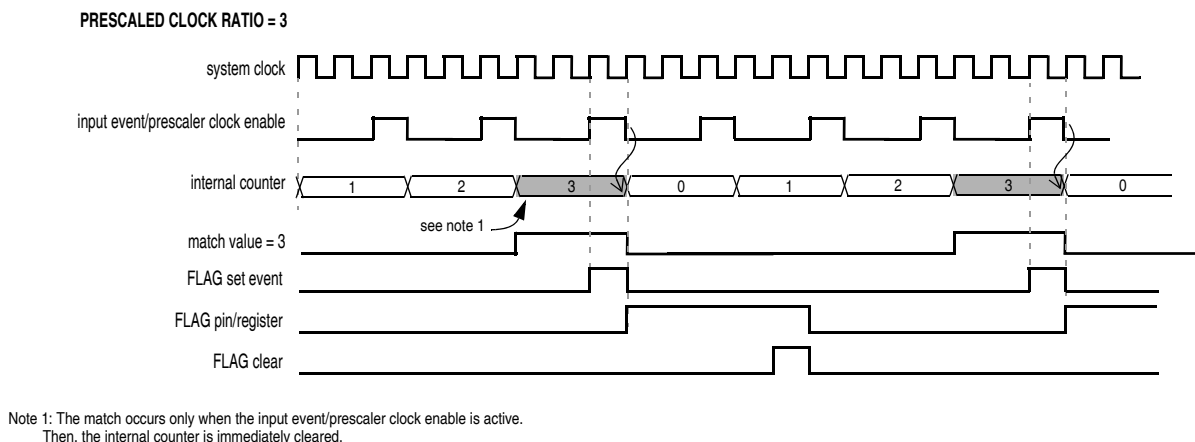


Figure 27-55. Time base generation with clear on match end

### 27.4.5.2.2 Coherent accesses

It is highly recommended that the software waits for a new FLAG set event before start reading EMIOSA[n] and EMIOSB[n] registers to get a new measurement. The FLAG indicates that new data has been captured and it is the only way to assure data coherency.

The FLAG set event can be detected by polling the FLAG bit or by enabling the interrupt request or DMA request or CTU trigger generation.

Reading the EMIOSA[n] register again in the same period of the last read of EMIOSB[n] register may lead to incoherent results. This will occur if the last read of EMIOSB[n] register occurred after a disabled B2 to B1 transfer.

### 27.4.5.2.3 Channel/Modes initialization

The following basic steps summarize basic output mode startup, assuming the channels are initially in GPIO mode:

1. [global] Disable Global Prescaler.
2. [timebase channel] Disable Channel Prescaler.
3. [timebase channel] Write initial value at internal counter.
4. [timebase channel] Set A/B register.
5. [timebase channel] Set channel to MC(B) Up mode.
6. [timebase channel] Set prescaler ratio.
7. [timebase channel] Enable Channel Prescaler.
8. [output channel] Disable Channel Prescaler.
9. [output channel] Set A/B register.
10. [output channel] Select timebase input through bits BSL[1:0].
11. [output channel] Enter output mode.

12. *[output channel]* Set prescaler ratio (same ratio as timebase channel).
13. *[output channel]* Enable Channel Prescaler.
14. *[global]* Enable Global Prescaler.
15. *[global]* Enable Global Time Base.

The timebase channel and the output channel may be the same for some applications such as in OPWFM(B) mode or whenever the output channel is intended to run the timebase itself.

The flags can be configured at any time.

## 27.5 Periodic Interrupt Timer (PIT)

### 27.5.1 Introduction

The PIT is an array of timers that can be used to raise interrupts and trigger DMA channels.

Figure 27-56 shows the PIT block diagram.

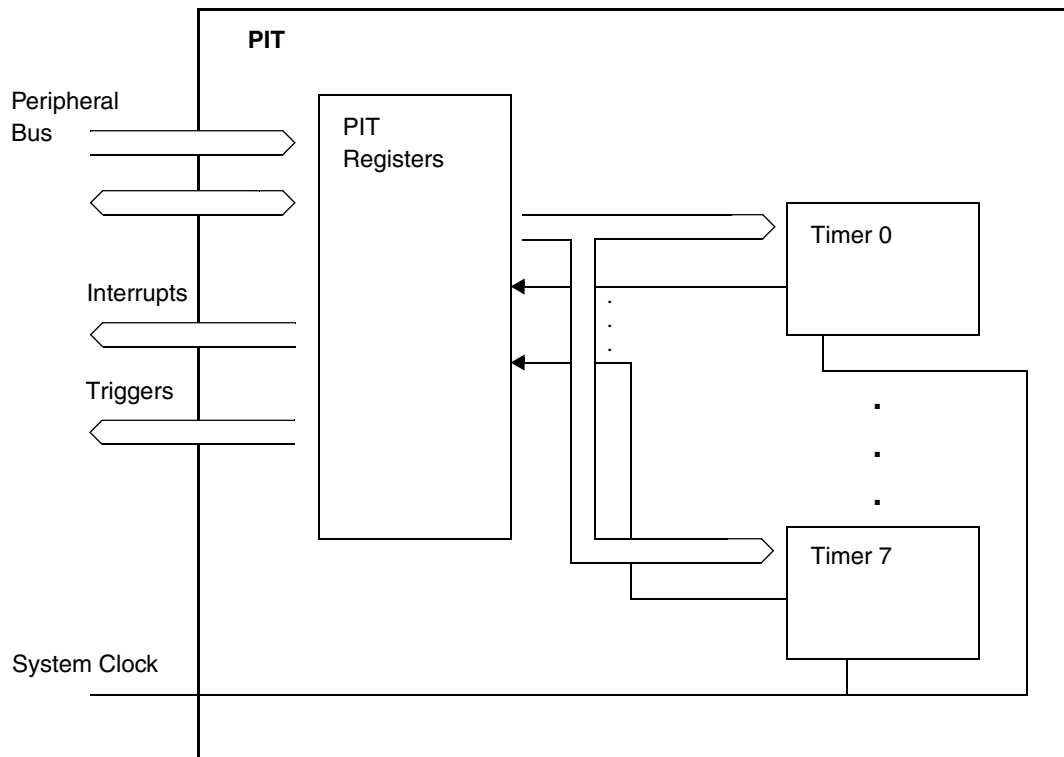


Figure 27-56. PIT block diagram

### 27.5.2 Features

The main features of this block are:

- Timers can generate DMA trigger pulses

- Timers can generate interrupts
- All interrupts are maskable
- Independent timeout periods for each timer

### 27.5.3 Signal description

The PIT module has no external pins.

### 27.5.4 Memory map and register description

This section provides a detailed description of all registers accessible in the PIT module.

#### 27.5.4.1 Memory map

[Table 27-23](#) gives an overview of the PIT registers. See the chip memory map for the PIT base address.

**Table 27-23. PIT memory map**

| Base address: 0xC3FF_0000 |                                      |                                 |
|---------------------------|--------------------------------------|---------------------------------|
| Address offset            | Use                                  | Location                        |
| 0x000                     | PIT Module Control Register (PITMCR) | <a href="#">on page 706</a>     |
| 0x004–0x0FC               | Reserved                             |                                 |
| 0x100–0x10C               | Timer Channel 0                      | See <a href="#">Table 27-24</a> |
| 0x110–0x11C               | Timer Channel 1                      | See <a href="#">Table 27-24</a> |
| 0x120–0x12C               | Timer Channel 2                      | See <a href="#">Table 27-24</a> |
| 0x130–0x13C               | Timer Channel 3                      | See <a href="#">Table 27-24</a> |
| 0x140–0x14C               | Timer Channel 4                      | See <a href="#">Table 27-24</a> |
| 0x150–0x15C               | Timer Channel 5                      | See <a href="#">Table 27-24</a> |
| 0x160–0x16C               | Timer Channel 6                      | See <a href="#">Table 27-24</a> |
| 0x170–0x17C               | Timer Channel 7                      | See <a href="#">Table 27-24</a> |

**Table 27-24. Timer channel *n***

| Address offset | Use                                 | Location                    |
|----------------|-------------------------------------|-----------------------------|
| channel + 0x00 | Timer Load Value Register (LDVAL)   | <a href="#">on page 706</a> |
| channel + 0x04 | Current Timer Value Register (CVAL) | <a href="#">on page 707</a> |
| channel + 0x08 | Timer Control Register (TCTRL)      | <a href="#">on page 708</a> |
| channel + 0x0C | Timer Flag Register (TFLG)          | <a href="#">on page 708</a> |

**NOTE**

Register Address = Base Address + Address Offset, where the Base Address is defined at the MCU level and the Address Offset is defined at the module level.

**NOTE**

Reserved registers will read as 0, writes will have no effect.

**27.5.4.2 PIT Module Control Register (PITMCR)**

This register controls whether the timer clocks should be enabled and whether the timers should run in debug mode.

Offset: 0x000 Access: Read/Write

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       |    |    |    |    |    |    |    |    |    |    |    |    |    |    |      |     |
|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|-----|
|       | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30   | 31  |
| R     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | MDIS | FRZ |
| W     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |      |     |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 1    | 0   |

**Figure 27-57. PIT Module Control Register (PITMCR)**

**Table 27-25. PITMCR field descriptions**

| Field | Description   |
|-------|---|
| MDIS  | Module Disable<br>This is used to disable the module clock. This bit should be enabled before any other setup is done.<br>0 Clock for PIT timers is enabled<br>1 Clock for PIT timers is disabled (default) |
| FRZ   | Freeze<br>Allows the timers to be stopped when the device enters debug mode.<br>0 = Timers continue to run in debug mode.<br>1 = Timers are stopped in debug mode.  |

**27.5.4.3 Timer Load Value Register (LDVAL)**

This register selects the timeout period for the timer interrupts.

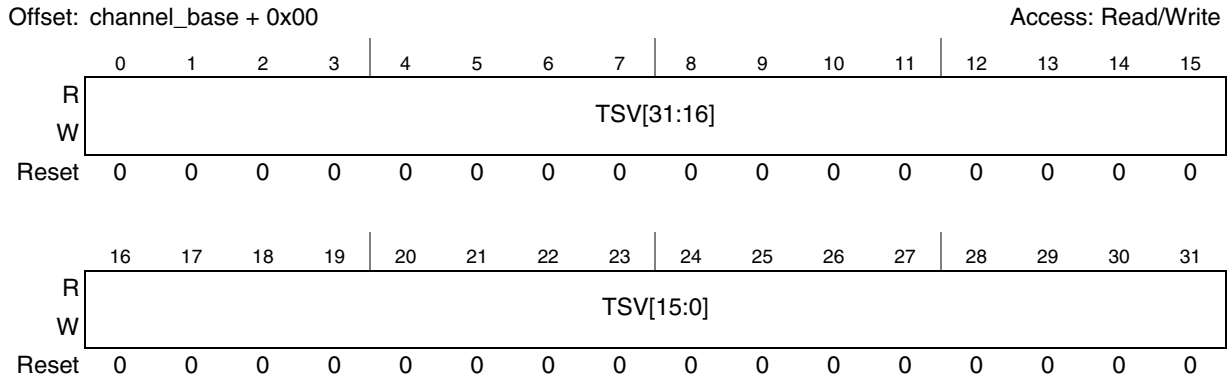


Figure 27-58. Timer Load Value Register (LDVAL)

Table 27-26. LDVAL field descriptions

| Field | Description   |
|-------|---|
| TSV   | Time Start Value<br>This field sets the timer start value. The timer counts down until it reaches 0, then it generates an interrupt and loads this register value again. Writing a new value to this register does not restart the timer, instead the value is loaded once the timer expires. To abort the current cycle and start a timer period with the new value, the timer must be disabled and enabled again (see <a href="#">Figure 27-63</a> ). |

#### 27.5.4.4 Current Timer Value Register (CVAL)

This register indicates the current timer position.

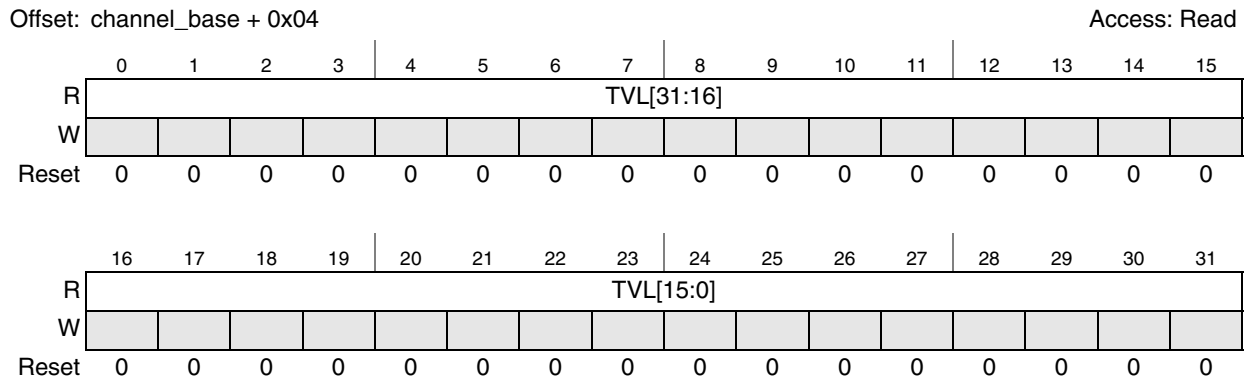


Figure 27-59. Current Timer Value Register (CVAL)

Table 27-27. CVAL field descriptions

| Field | Description  |
|-------|--|
| TVL   | Current Timer Value<br>This field represents the current timer value. Note that the timer uses a downcounter.<br><br><b>Note:</b> The timer values will be frozen in Debug mode if the FRZ bit is set in the PIT Module Control Register (see <a href="#">Figure 27-2</a> ). |

### 27.5.4.5 Timer Control Register (TCTRL)

This register contains the control bits for each timer.

Offset: channel\_base + 0x08 Access: Read/Write

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       |    |    |    |    |    |    |    |    |    |    |    |    |    |    |     |     |
|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|-----|
|       | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30  | 31  |
| R     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |     |     |
| W     |    |    |    |    |    |    |    |    |    |    |    |    |    |    | TIE | TEN |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 0   |

Figure 27-60. Timer Control Register (TCTRL)

Table 27-28. TCTRL field descriptions

| Field | Description   |
|-------|---|
| TIE   | Timer Interrupt Enable Bit<br>0 Interrupt requests from Timer x are disabled<br>1 Interrupt will be requested whenever TIF is set<br>When an interrupt is pending (TIF set), enabling the interrupt will immediately cause an interrupt event.<br>To avoid this, the associated TIF flag must be cleared first. |
| TEN   | Timer Enable Bit<br>0 Timer will be disabled<br>1 Timer will be active  |

### 27.5.4.6 Timer Flag Register (TFLG)

This register holds the PIT interrupt flags.

Offset: channel\_base + 0x0C Access: Read/Write

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |     |
|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|
|       | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31  |
| R     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | TIF |
| W     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | w1c |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   |

Figure 27-61. Timer Flag Register (TFLG)

Table 27-29. TFLG field descriptions

| Field | Description  |
|-------|--|
| TIF   | Time Interrupt Flag<br>TIF is set to 1 at the end of the timer period. This flag can be cleared only by writing it with a 1. Writing a 0 has no effect. If enabled (TIE = 1), TIF causes an interrupt request.<br>0 Timeout has not yet occurred<br>1 Timeout has occurred |

## 27.5.5 Functional description

### 27.5.5.1 General

This section gives detailed information on the internal operation of the module. Each timer can be used to generate trigger pulses as well as to generate interrupts, each interrupt will be available on a separate interrupt line.

#### 27.5.5.1.1 Timers

The timers generate triggers at periodic intervals, when enabled. They load their start values, as specified in their LDVAL registers, then count down until they reach 0. Then they load their respective start value again. Each time a timer reaches 0, it will generate a trigger pulse and set the interrupt flag.

All interrupts can be enabled or masked (by setting the TIE bits in the TCTRL registers). A new interrupt can be generated only after the previous one is cleared.

If desired, the current counter value of the timer can be read via the CVAL registers.

The counter period can be restarted, by first disabling, then enabling the timer with the TEN bit (see [Figure 27-62](#)).

The counter period of a running timer can be modified, by first disabling the timer, setting a new load value, and then enabling the timer again (see [Figure 27-63](#)).

It is also possible to change the counter period without restarting the timer by writing the LDVAL register with the new load value. This value will then be loaded after the next trigger event (see [Figure 27-64](#)).

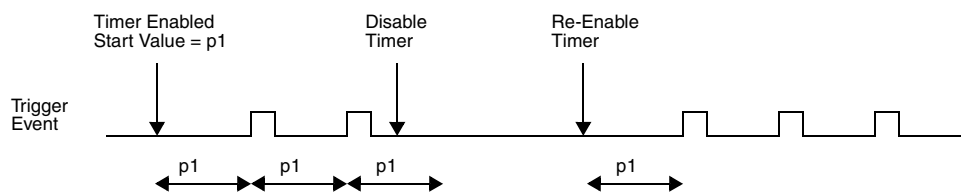


Figure 27-62. Stopping and starting a timer

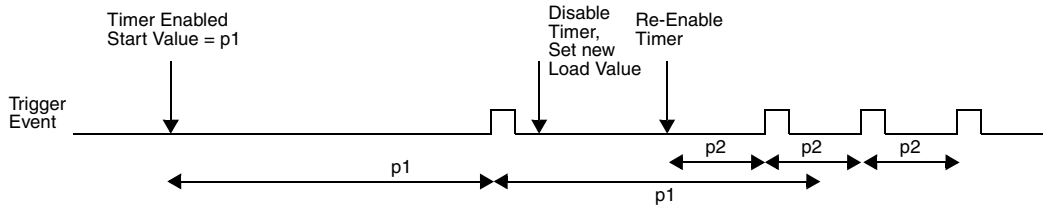


Figure 27-63. Modifying running timer period

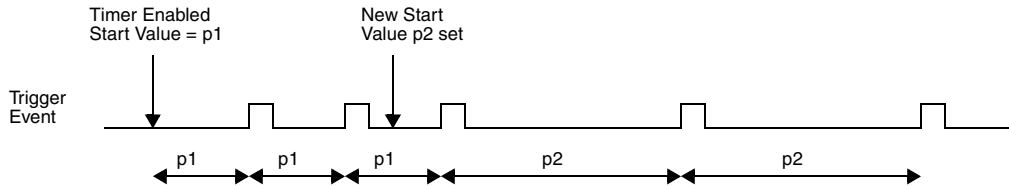


Figure 27-64. Dynamically setting a new load value

### 27.5.5.1.2 Debug mode

In Debug mode the timers will be frozen. This is intended to aid software development, allowing the developer to halt the processor, investigate the current state of the system (for example, the timer values) and then continue the operation.

### 27.5.5.2 Interrupts

All of the timers support interrupt generation. See [Chapter 18, Interrupt Controller \(INTC\)](#), for related vector addresses and priorities.

Timer interrupts can be disabled by setting the TIE bits to zero. The timer interrupt flags (TIF) are set to 1 when a timeout occurs on the associated timer, and are cleared to 0 by writing a 1 to that TIF bit.

## 27.5.6 Initialization and application information

### 27.5.6.1 Example configuration

In the example configuration:

- The PIT clock has a frequency of 50 MHz
- Timer 1 creates an interrupt every 5.12 ms
- Timer 3 creates a trigger event every 30 ms

First the PIT module needs to be activated by programming `PIT_MCR[MDIS] = 0`.

The 50 MHz clock frequency equates to a clock period of 20 ns. Timer 1 needs to trigger every  $5.12 \text{ ms} / 20 \text{ ns} = 256000$  cycles and Timer 3 every  $30 \text{ ms} / 20 \text{ ns} = 1500000$  cycles. The value for the LDVAL register trigger would be calculated as  $(\text{period} / \text{clock period}) - 1$ .



The LDVAL registers must be set as follows:

- LDVAL for Timer 1 is set to 0x0003E7FF
- LDVAL for Timer 3 is set to 0x0016E35F

The interrupt for Timer 1 is enabled by setting TIE in the TCTRL1 register. The timer is started by writing a 1 to bit TEN in the TCTRL1 register.

Timer 3 shall be used only for triggering. Therefore Timer 3 is started by writing a 1 to bit TEN in the TCTRL3 register; bit TIE stays at 0.

The following example code matches the described setup:

```
// turn on PIT
PIT_CTRL = 0x00;


// Timer 1
PIT_LDVAL1 = 0x0003E7FF; // setup timer 1 for 256000 cycles
PIT_TCTRL1 = TIE; // enable Timer 1 interrupts
PIT_TCTRL1 |= TEN; // start timer 1

// Timer 3
PIT_LDVAL3 = 0x0016E35F; // setup timer 3 for 1500000 cycles
PIT_TCTRL3 = TEN; // start timer 3
```

This page is intentionally left blank.

---

## —— ADC system ——



This page is intentionally left blank.

# Chapter 28

## Analog-to-Digital Converter (ADC)

### 28.1 Overview

#### 28.1.1 Device-specific features

- Two ADC modules, (ADC\_0 with 10-bit resolution and ADC\_1 with 12-bit resolution)
- $0-V_{DD}$  common mode conversion range
- Independent reference supplies for each ADC
- 53 single-ended input channels (depending on package type), expandable to 81 channels via external multiplexing
  - Internally multiplexed channels
    - 16 precision channels shared between 10-bit and 12-bit ADCs
    - Three standard channels shared between 10-bit and 12-bit ADCs
    - Five dedicated standard channels on 12-bit ADC
    - As many as 29 dedicated standard channels on 10-bit ADC
  - Externally multiplexed channels
    - Internal control to support generation of external analog multiplexer selection
    - Four internal channels optionally used to support externally multiplexed inputs, providing transparent control for additional ADC channels
    - Each of the four channels supports as many as eight externally multiplexed inputs
- Three independently configurable sample and conversion times for high precision channels, standard precision channels, and externally multiplexed channels
- Dedicated result registers available for every channel. Conversion information, such as mode of operation (normal, injected or CTU), is associated to data value.
- One Shot/Scan Modes
- Chain Injection Mode
- Conversion triggering sources:
  - Software
  - CTU
  - PIT channel 2 and 6 (for injected conversion)
- Conversion triggering support — internal conversion triggering from periodic interrupt timer (PIT) or timed I/O module (eMIOS)
- Power-down mode for analog portion of ADC
- Supports DMA transfer of results based on the end of conversion
- 6 + 3 analog watchdogs (6 on 10-bit ADC, 3 on 12-bit ADC) with interrupt capability for continuous hardware monitoring

### 28.1.2 Device-specific implementation

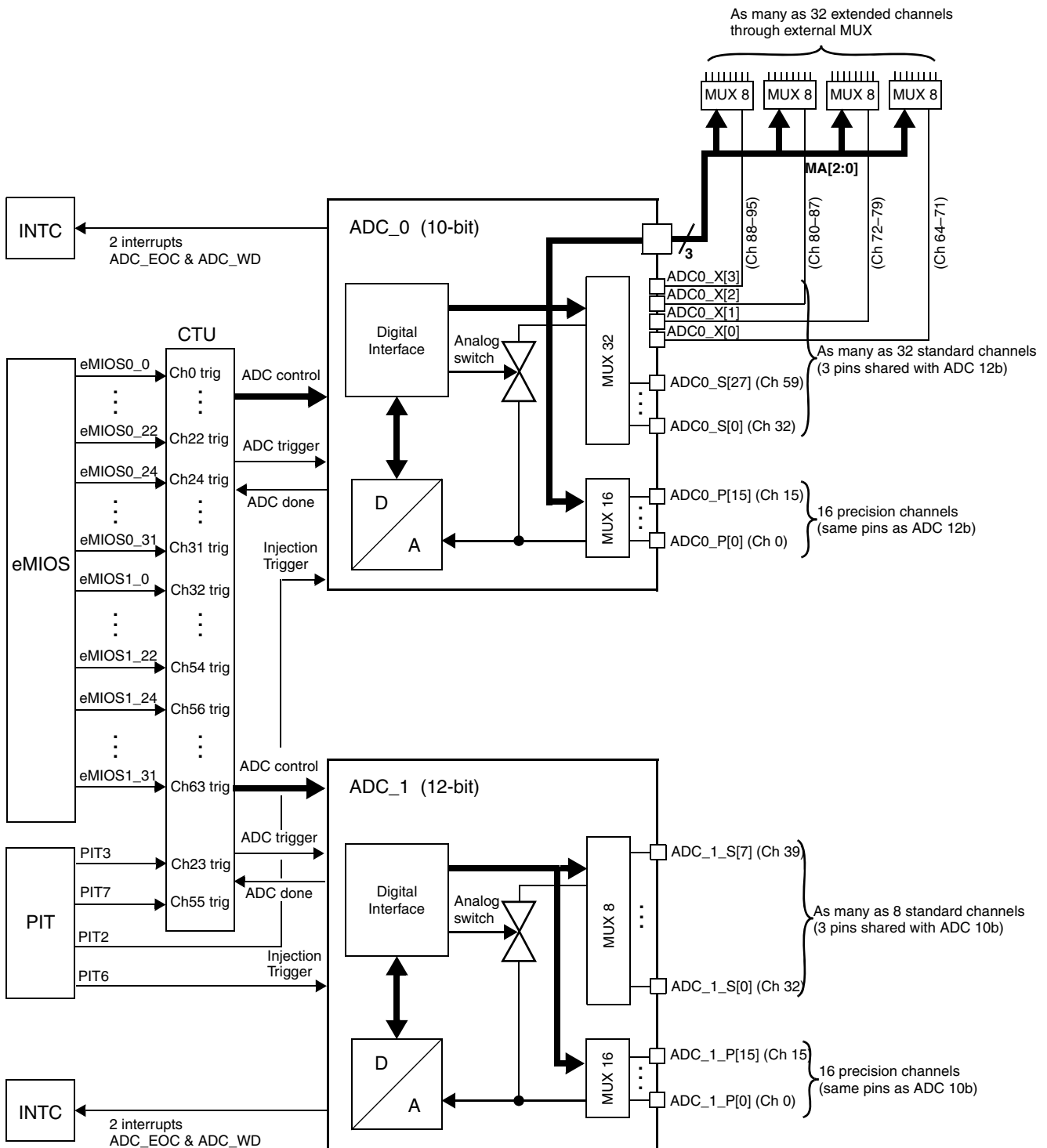


Figure 28-1. Implementation of ADC\_0 and ADC\_1

## 28.2 Introduction

The analog-to-digital converter (ADC) block provides accurate and fast conversions for a wide range of applications.

The ADC contains advanced features for normal or injected conversion. It provides support for eDMA (direct memory access) mode operation. A conversion can be triggered by software or hardware (Cross Triggering Unit or PIT).

There are three types of input channels:

- Internal precision, ADCx\_P[n] (internally multiplexed precision channels)
- Internal standard, ADCx\_S[n] (internally multiplexed standard channels)
- External ADCx\_X[n] (externally multiplexed standard channels)

The mask registers present within the ADC can be programmed to configure which channel has to be converted.

Three external decode signals MA[2:0] (multiplexer address) are provided for external channel selection and are available as alternate functions on GPIO.

The MA[0:2] are controlled by the ADC itself and are set automatically by the hardware.

A conversion timing register for configuring different sampling and conversion times is associated to each channel type.

Analog watchdogs allow continuous hardware monitoring.

## 28.3 Functional description

### 28.3.1 Analog channel conversion

Three conversion modes are available within the ADC:

- Normal conversion
- Injected conversion
- CTU triggered conversion

#### 28.3.1.1 Normal conversion

This is the normal conversion that the user programs by configuring the normal conversion mask registers (NCMR). Each channel can be individually enabled by setting 1 in the corresponding field of NCMR registers. Mask registers must be programmed before starting the conversion and cannot be changed until the conversion of all the selected channels ends (NSTART bit in the Main Status Register (MSR) is reset).

#### 28.3.1.2 Start of normal conversion

The conversion chain starts when the NSTART bit in the Main Configuration Register (MCR) is set.

The MSR[NSTART] status bit is automatically set when the normal conversion starts. At the same time the MCR[NSTART] bit is reset, allowing the software to program a new start of conversion. In that case the new requested conversion starts after the running conversion is completed.

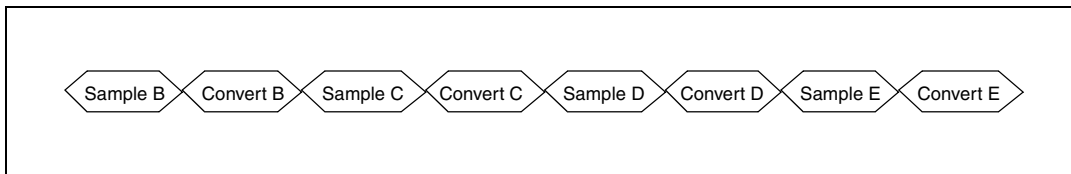
If the content of all the normal conversion mask registers is zero (that is, no channel is selected) the conversion operation is considered completed and the interrupt ECH (see [Chapter 18, Interrupt Controller \(INTC\)](#), for further details) is immediately issued after the start of conversion.

### 28.3.1.3 Normal conversion operating modes

Two operating modes are available for the normal conversion:

- One Shot
- Scan

To enter one of these modes, it is necessary to program the MCR[MODE] bit. The first phase of the conversion process involves sampling the analog channel and the next phase involves the conversion phase when the sampled analog value is converted to digital as shown in [Figure 28-2](#).



**Figure 28-2. Normal conversion flow**

In **One Shot Mode** (MODE = 0) a sequential conversion specified in the NCMR registers is performed only once. At the end of each conversion, the digital result of the conversion is stored in the corresponding data register.

#### Example 28-1. One Shot Mode (MODE = 0)

Channels A-B-C-D-E-F-G-H are present in the device where channels B-D-E are to be converted in the One Shot Mode. MODE = 0 is set for One Shot mode. Conversion starts from the channel B followed by conversion of channels D-E. At the end of conversion of channel E the scanning of channels stops.

The NSTART status bit in the MSR is automatically set when the Normal conversion starts. At the same time the MCR[NSTART] bit is reset, allowing the software to program a new start of conversion. In that case the new requested conversion starts after the running conversion is completed.

In **Scan Mode** (MODE = 1), a sequential conversion of N channels specified in the NCMR registers is continuously performed. As in the previous case, at the end of each conversion the digital result of the conversion is stored into the corresponding data register.

The MSR[NSTART] status bit is automatically set when the Normal conversion starts. Unlike One Shot Mode, the MCR[NSTART] bit is not reset. It can be reset by software when the user needs to stop scan mode. In that case, the ADC completes the current scan conversion and, after the last conversion, also resets the MSR[NSTART] bit.



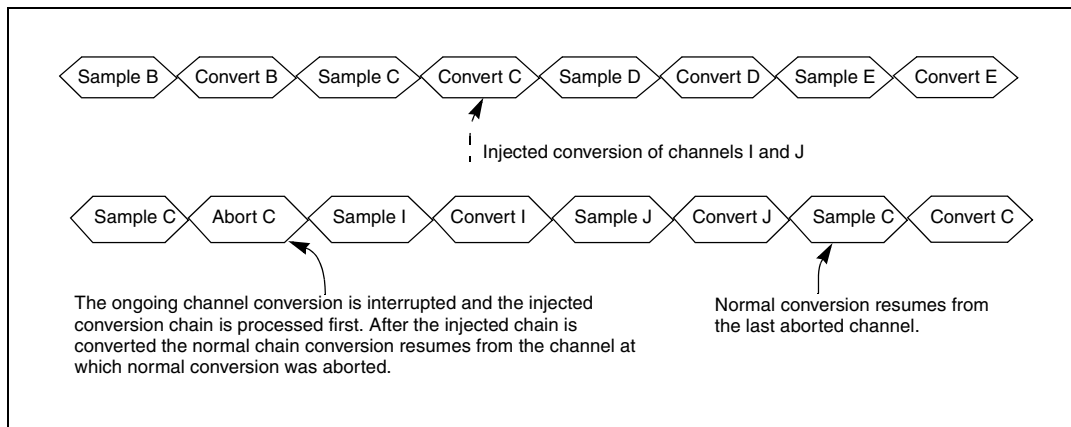
**Example 28-2. Scan Mode (MODE = 1)**

Channels A-B-C-D-E-F-G-H are present in the device where channels B-D-E are to be converted in the Scan Mode. MODE = 1 is set for Scan Mode. Conversion starts from the channel B followed by conversion of the channels D-E. At the end of conversion of channel E the scanning of channel B starts followed by conversion of the channels D-E. This sequence repeats itself till the MCR[NSTART] bit is cleared by software.

At the end of each conversion an End Of Conversion interrupt is issued (if enabled by the corresponding mask bit) and at the end of the conversion sequence an End Of Chain interrupt is issued (if enabled by the corresponding mask bit in the IMR register).

**28.3.1.4 Injected channel conversion**

A conversion chain can be injected into the ongoing Normal conversion by configuring the Injected Conversion Mask Registers (JCMR). As Normal conversion, each channel can be individually selected. This injected conversion (which can only occur in One Shot mode) interrupts the normal conversion (which can be in One Shot or Scan mode). When an injected conversion is inserted, ongoing normal channel conversion is aborted and the injected channel request is processed. After the last channel in the injected chain is converted, normal conversion resumes from the channel at which the normal conversion was aborted as shown in [Figure 28-3](#).



**Figure 28-3. Injected sample/conversion sequence**

The injected conversion can be started using two options:

- By software setting the MCR[JSTART]; the current conversion is suspended and the injected chain is converted. At the end of the chain, the JSTART bit in the MSR is reset and the normal chain conversion is resumed.
- By an internal trigger signal from the PIT when MCR[JTRGEN] is set; a programmed event (rising/falling edge depending on MCR[JEDGE]) on the signal coming from PIT starts the injected conversion by setting the MSR[JSTART]. At the end of the chain, the MSR[JSTART] is cleared and the normal conversion chain is resumed.

The MSR[JSTART] is automatically set when the Injected conversion starts. At the same time the MCR[JSTART] is reset, allowing the software to program a new start of conversion. In that case the new requested conversion starts after the running injected conversion is completed.

At the end of each injected conversion, an End Of Injected Conversion (JEOC) interrupt is issued (if enabled by the IMR[MSKJEOC]) and at the end of the sequence an End Of Injected Chain (JECH) interrupt is issued (if enabled by the IMR[MSKJEOC]).

If the content of all the injected conversion mask registers (JCMR) is zero (that is, no channel is selected) the JECH interrupt is immediately issued after the start of conversion.

### 28.3.1.5 Abort conversion

Two different abort functions are provided.

- The user can abort the ongoing conversion by setting the MCR[ABORT] bit. The current conversion is aborted and the conversion of the next channel of the chain is immediately started. In the case of an abort operation, the NSTART/JSTART bit remains set and the ABORT bit is reset after the conversion of the next channel starts. The EOC interrupt corresponding to the aborted channel is not generated. This behavior is true for normal or Injected conversion modes. If the last channel of a chain is aborted, the end of chain is reported generating an ECH interrupt.
- It is also possible to abort the current chain conversion by setting the MCR[ABORTCHAIN] bit. In that case the behavior of the ADC depends on the MODE bit. If scan mode is disabled, the NSTART bit is automatically reset together with the MCR[ABORTCHAIN] bit. Otherwise, if the scan mode is enabled, a new chain conversion is started. The EOC interrupt of the current aborted conversion is not generated but an ECH interrupt is generated to signal the end of the chain. When a chain conversion abort is requested (ABORTCHAIN bit is set) while an injected conversion is running over a suspended Normal conversion, both injected chain and Normal conversion chain are aborted (both the NSTART and JSTART bits are also reset).

## 28.3.2 Analog clock generator and conversion timings

The clock frequency can be selected by programming the MCR[ADCLKSEL]. When this bit is set to 1 the ADC clock has the same frequency as the peripheral set 3 clock. Otherwise, the ADC clock is half of the peripheral set 3 clock frequency. The ADCLKSEL bit can be written only in power-down mode.

When the internal divider is not enabled (ADCCLKSEL = 1), it is important that the associated clock divider in the clock generation module is 1. This is needed to ensure 50% clock duty cycle.

In all other cases, the ADC should use the clock divided by two internally.

## 28.3.3 ADC sampling and conversion timing

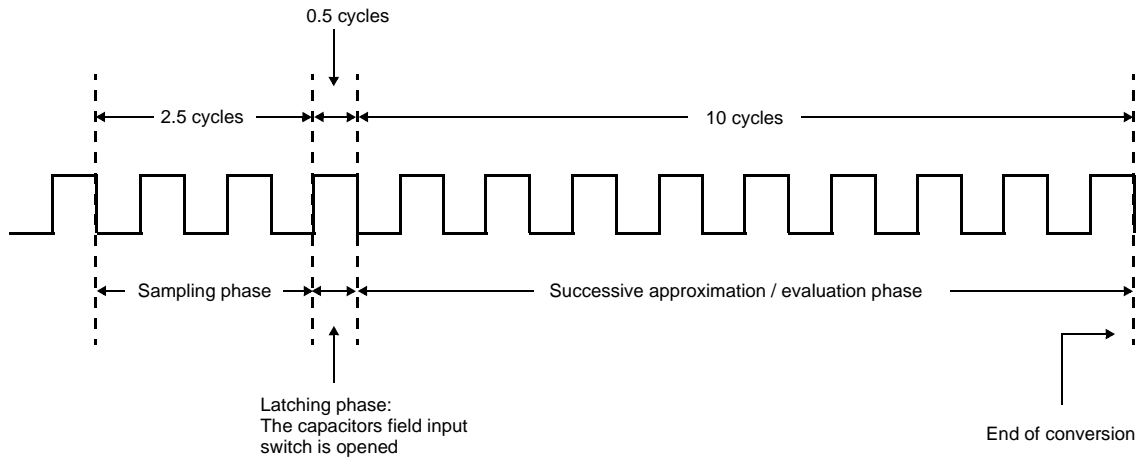
In order to support different loading and switching times, several different Conversion Timing registers (CTR) are present. There is one register per channel type. INPLATCH and INPCMP configurations are limited when the system clock frequency is greater than 20 MHz.

When a conversion is started, the ADC connects the internal sampling capacitor to the respective analog input pin, allowing the capacitance to charge up to the input voltage value. The time to load the capacitor is referred to as sampling time. After completion of the sampling phase, the evaluation phase starts and all the bits corresponding to the resolution of the ADC are estimated to provide the conversion result.

The conversion times are programmed via the bit fields of the CTR. Bit fields INPLATCH, INPCMP, and INPSAMP are used to define the total conversion duration ( $T_{\text{conv}}$ ) and in particular the partition between sampling phase duration ( $T_{\text{sample}}$ ) and total evaluation phase duration ( $T_{\text{eval}}$ ).

### 28.3.3.1 ADC\_0

Figure 28-4 represents the sampling and conversion sequence.



**Note:** Operating conditions — INPLATCH = 0, INPSAMP = 3, INPCMP = 1 and Fadc clk = 20 MHz

**Figure 28-4. Sampling and conversion timings**

The sampling phase duration is:

$$T_{\text{sample}} = (\text{INPSAMP} - \text{ndelay}) \cdot T_{\text{ck}}$$

$$\text{INPSAMP} \geq 3$$

where ndelay is equal to 0.5 if INPSAMP is less than or equal to 06h, otherwise it is 1. INPSAMP must be greater than or equal to 3 (hardware requirement).

The total evaluation phase duration is:

$$T_{\text{eval}} = 10 \cdot T_{\text{biteval}} = 10 \cdot (\text{INPCMP} \cdot T_{\text{ck}})$$

$$(\text{INPCMP} \geq 1) \quad \text{and} \quad (\text{INPLATCH} < \text{INPCMP})$$

INPCMP must be greater than or equal to 1 and INPLATCH must be less than INPCMP (hardware requirements).

The total conversion duration is (not including external multiplexing):

$$T_{\text{conv}} = T_{\text{sample}} + T_{\text{eval}} + (\text{ndelay} \cdot T_{\text{ck}})$$

The timings refer to the unit  $T_{\text{ck}}$ , where  $f_{\text{ck}} = (1/2 \times \text{ADC peripheral set clock})$ .

**Table 28-1. ADC sampling and conversion timing at 5 V / 3.3 V for ADC\_0**

| Clock (MHz) | T <sub>ck</sub> (μs) | INPSAMPLE <sup>1</sup> | Ndelay <sup>2</sup> | T <sub>sample</sub> <sup>3</sup> | T <sub>sample</sub> /T <sub>ck</sub> | INPCMP | T <sub>eval</sub> (μs) | INPLATCH | T <sub>conv</sub> (μs) | T <sub>conv</sub> /T <sub>ck</sub> |
|-------------|----------------------|------------------------|---------------------|----------------------------------|--------------------------------------|--------|------------------------|----------|------------------------|------------------------------------|
| 6           | 0.167                | 4                      | 0.5                 | 0.583                            | 3.500                                | 1      | 1.667                  | 0        | 2.333                  | 14.000                             |
| 7           | 0.143                | 4                      | 0.5                 | 0.500                            | 3.500                                | 1      | 1.429                  | 0        | 2.000                  | 14.000                             |
| 8           | 0.125                | 5                      | 0.5                 | 0.563                            | 4.500                                | 1      | 1.250                  | 0        | 1.875                  | 15.000                             |
| 16          | 0.063                | 9                      | 1                   | 0.500                            | 8.000                                | 1      | 0.625                  | 0        | 1.188                  | 19.000                             |
| 32          | 0.031                | 17                     | 1                   | 0.500                            | 16.000                               | 2      | 0.625                  | 1        | 1.156                  | 37.000                             |

<sup>1</sup> Where: INPSAMPLE ≥ 3

<sup>2</sup> Where: INPSAMP ≤ 6, N = 0.5; INPSAMP > 6, N = 1

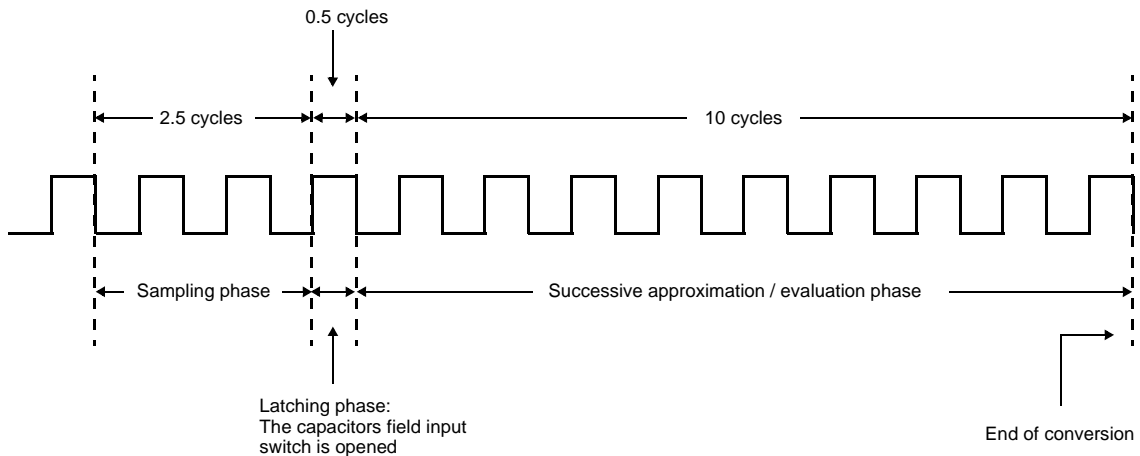
<sup>3</sup> Where: T<sub>sample</sub> = (INPSAMP-N)T<sub>ck</sub>; Must be ≥ 500 ns

**Table 28-2. Max/Min ADC\_clk frequency and related configuration settings at 5 V / 3.3 V for ADC\_0**

| INPCMP | INPLATCH | Max f <sub>ADC_clk</sub> | Min f <sub>ADC_clk</sub> |
|--------|----------|--------------------------|--------------------------|
| 00/01  | 0        | 20 + 4%                  | 6                        |
|        | 1        | —                        | —                        |
| 10     | 0        | —                        | —                        |
|        | 1        | 32+4%                    | 6                        |
| 11     | 0        | —                        | —                        |
|        | 1        | 32 + 4%                  | 9                        |

### 28.3.3.2 ADC\_1

Figure 28-5 represents the sampling and conversion sequence.



**Note:** Operating conditions — INPLATCH = 0, INPSAMP = 3, INPCMP = 1 and F<sub>adc clk</sub> = 20 MHz

**Figure 28-5. Sampling and conversion timings**

The sampling phase duration is:

$$T_{\text{sample}} = (\text{INPSAMP} - 1) \cdot T_{\text{ck}}$$

$$\text{INPSAMP} \geq 8$$

where  $n_{\text{delay}}$  is equal to 0.5 if INPSAMP is less than or equal to 06h, otherwise it is 1. INPSAMP must be greater than or equal to 8 (hardware requirement).

The total evaluation phase duration is:

$$T_{\text{eval}} = 12 \cdot T_{\text{biteval}}$$

Where:

$$T_{\text{biteval}} = \text{INPCMP} \cdot T_{\text{ck}} \quad [\text{if } \text{INPCMP} \geq 1]$$

$$T_{\text{biteval}} = 4 \cdot T_{\text{ck}} \quad [\text{if } \text{INPCMP} = 0]$$

The total conversion duration is (not including external multiplexing):

$$T_{\text{conv}} = T_{\text{sample}} + T_{\text{eval}} + T_{\text{ck}}$$

The timings refer to the unit  $T_{\text{ck}}$ , where  $f_{\text{ck}} = (1/2 \times \text{ADC peripheral set clock})$ .

**Table 28-3. ADC sampling and conversion timing at 5 V for ADC\_1**

| Clock (MHz) | $T_{\text{ck}}$ ( $\mu\text{s}$ ) | INPSAMPLE <sup>1</sup> | Ndelay <sup>2</sup> | $T_{\text{sample}}$ <sup>3</sup> | $T_{\text{sample}}/T_{\text{ck}}$ | INPCMP | $T_{\text{eval}}$ ( $\mu\text{s}$ ) | INPLATCH | $T_{\text{conv}}$ ( $\mu\text{s}$ ) | $T_{\text{conv}}/T_{\text{ck}}$ |
|-------------|-----------------------------------|------------------------|---------------------|----------------------------------|-----------------------------------|--------|-------------------------------------|----------|-------------------------------------|---------------------------------|
| 4           | 0.250                             | 8                      | 1                   | 1.750                            | 7.000                             | 1      | 3.000                               | 1        | 5.000                               | 20.000                          |
| 5           | 0.200                             | 8                      | 1                   | 1.400                            | 7.000                             | 1      | 2.400                               | 1        | 4.000                               | 20.000                          |
| 6           | 0.167                             | 8                      | 1                   | 1.167                            | 7.000                             | 1      | 2.000                               | 1        | 3.333                               | 20.000                          |
| 7           | 0.143                             | 8                      | 1                   | 1.000                            | 7.000                             | 1      | 1.714                               | 1        | 2.857                               | 20.000                          |
| 8           | 0.125                             | 8                      | 1                   | 0.875                            | 7.000                             | 1      | 1.500                               | 1        | 2.500                               | 20.000                          |
| 16          | 0.063                             | 9                      | 1                   | 0.500                            | 8.000                             | 2      | 1.500                               | 1        | 2.063                               | 33.000                          |
| 32          | 0.031                             | 17                     | 1                   | 0.500                            | 16.000                            | 0      | 1.500                               | 1        | 2.031                               | 65.000                          |

<sup>1</sup> Where: INPSAMPLE  $\geq 8$

<sup>2</sup> Where: INPSAMP  $\leq 6$ , N = 0.5; INPSAMP > 6, N = 1

<sup>3</sup> Where:  $T_{\text{sample}} = (\text{INPSAMP} - N)T_{\text{ck}}$ ; Must be  $\geq 500$  ns

Table 28-4. ADC sampling and conversion timing at 3.3 V for ADC\_1

| Clock (MHz) | T <sub>ck</sub> (μs) | INPSAMPLE <sup>1</sup> | Ndelay <sup>2</sup> | T <sub>sample</sub> <sup>3</sup> | T <sub>sample</sub> /T <sub>ck</sub> | INPCMP | T <sub>eval</sub> (μs) | INPLATCH | T <sub>conv</sub> (μs) | T <sub>conv</sub> /T <sub>ck</sub> |
|-------------|----------------------|------------------------|---------------------|----------------------------------|--------------------------------------|--------|------------------------|----------|------------------------|------------------------------------|
| 4           | 0.250                | 8                      | 1                   | 1.750                            | 7.000                                | 1      | 3.000                  | 1        | 5.000                  | 20.000                             |
| 5           | 0.200                | 8                      | 1                   | 1.400                            | 7.000                                | 1      | 2.400                  | 1        | 4.000                  | 20.000                             |
| 7           | 0.143                | 8                      | 1                   | 1.000                            | 7.000                                | 2      | 3.429                  | 1        | 4.571                  | 32.000                             |
| 8           | 0.125                | 8                      | 1                   | 0.875                            | 7.000                                | 2      | 3.000                  | 1        | 4.000                  | 32.000                             |
| 16          | 0.063                | 11                     | 1                   | 0.625                            | 10.000                               | 0      | 3.000                  | 1        | 3.688                  | 59.000                             |
| 20          | 0.050                | 13                     | 1                   | 0.600                            | 12.000                               | 0      | 2.400                  | 1        | 3.050                  | 61.000                             |

<sup>1</sup> Where: INPSAMPLE ≥ 8

<sup>2</sup> Where: INPSAMP ≤ 6, N = 0.5; INPSAMP > 6, N = 1

<sup>3</sup> Where: T<sub>sample</sub> = (INPSAMP - N)T<sub>ck</sub>; Must be ≥ 600 ns

Table 28-5. Max/Min ADC\_clk frequency and related configuration settings at 5 V for ADC\_1

| INPCMP | INPLATCH | Max f <sub>ADC_clk</sub> | Min f <sub>ADC_clk</sub> |
|--------|----------|--------------------------|--------------------------|
| 00     | 0        | 16 + 4%                  | 13.33                    |
|        | 1        | 32 + 4%                  | 13.33                    |
| 01     | 0/1      | 8 + 4%                   | 3.33                     |
| 10     | 0        | 8 + 4%                   | 6.67                     |
|        | 1        | 16 + 4%                  | 6.67                     |
| 11     | 0        | 16 + 4%                  | 10                       |
|        | 1        | 24 + 4%                  | 10                       |

Table 28-6. Max/Min ADC\_clk frequency and related configuration settings at 3.3 V for ADC\_1

| INPCMP | INPLATCH | Max f <sub>ADC_clk</sub> | Min f <sub>ADC_clk</sub> |
|--------|----------|--------------------------|--------------------------|
| 00     | 0        | Not allowed              | Not allowed              |
|        | 1        | 20 + 4%                  | 13.33                    |
| 01     | 0/1      | 5 + 4%                   | 3.33                     |
| 10     | 0        | Not allowed              | Not allowed              |
|        | 1        | 10 + 4%                  | 6.67                     |
| 11     | 0        | 10 + 4%                  | 10                       |
|        | 1        | 15 + 4%                  | 10                       |

## 28.3.4 ADC CTU (Cross Triggering Unit)

### 28.3.4.1 Overview

The ADC cross triggering unit (CTU) is added to enhance the injected conversion capability of the ADC. The CTU is triggered by multiple input events (eMIOS and PIT) and can be used to select the channels to be converted from the appropriate event configuration register. A single channel is converted for each request.

The CTU can be enabled by setting MCR[CTUEN].

The CTU and the ADC are synchronous with the peripheral set 3 clock in both cases.

### 28.3.4.2 CTU in trigger mode

In CTU trigger mode, normal and injected conversions triggered by the CPU are still enabled.

Once the CTU event configuration register (CTU\_EVTCFGR<sub>x</sub>) is configured and the corresponding trigger from the eMIOS or PIT is received, the conversion starts. The MSR[CTUSTART] is set automatically at this point and it is also automatically reset when the CTU triggered conversion is completed.

If an injected conversion (programmed by the user by setting the JSTART bit) is ongoing and CTU conversion is triggered, then the injected channel conversion chain is aborted and only the CTU triggered conversion proceeds. By aborting the injected conversion, the MSR[JSTART] is reset. That abort is signaled through the status bit MSR[JABORT].

If a normal conversion is ongoing and a CTU conversion is triggered, then any ongoing channel conversion is aborted and the CTU triggered conversion is processed. When it is finished, the normal conversion resumes from the channel at which the normal conversion was aborted.

If another CTU conversion is triggered before the end of the current CTU triggered conversion, the new request is discarded.

When a normal conversion is requested during CTU conversion (CTUSTART bit = 1), the normal conversion starts when CTU conversion is completed (CTUSTART = 0). Otherwise, when an Injected conversion is requested during CTU conversion, the injected conversion is discarded and the MCR[JSTART] is immediately reset.

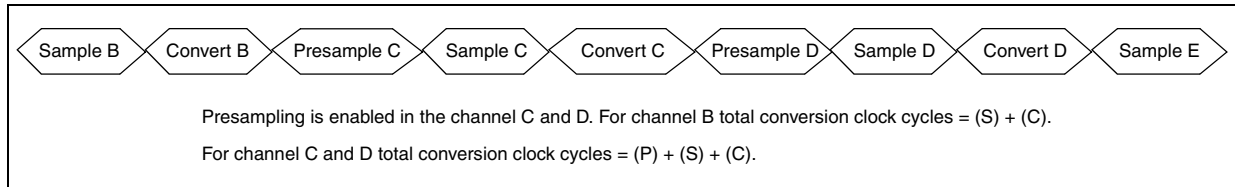
## 28.3.5 Presampling

### 28.3.5.1 Introduction

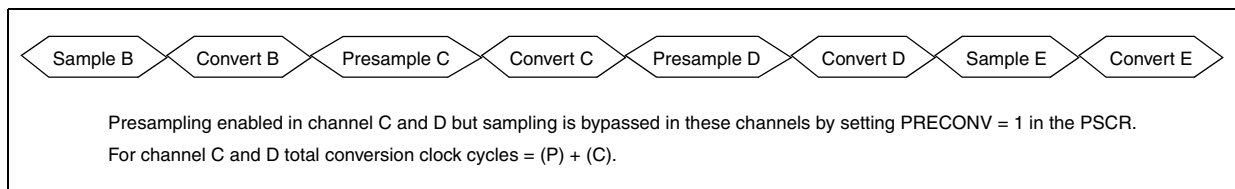
Presampling is used to precharge or discharge the ADC internal capacitor before it starts sampling of the analog input coming from the input pins. This is useful for resetting information regarding the last converted data or to have more accurate control of conversion speed. During presampling, the ADC samples the internally generated voltage.

Presampling can be enabled/disabled on a channel basis by setting the corresponding bits in the PSR registers.

After enabling the presampling for a channel, the normal sequence of operation will be Presampling + Sampling + Conversion for that channel. Sampling of the channel can be bypassed by setting the PRECONV bit in the PSCR. When sampling of a channel is bypassed, the sampled data of internal voltage in the presampling state is converted (Figure 28-6, Figure 28-7).



**Figure 28-6. Presampling sequence**



**Figure 28-7. Presampling sequence with PRECONV = 1**

### 28.3.5.2 Presampling channel enable signals

It is possible to select between two internally generated voltages V0 and V1 depending on the value of the PSCR[PREVAL] as shown in Table 28-7.

**Table 28-7. Presampling voltage selection based on PREVALx fields**

| PSCR[PREVALx] | Presampling voltage                           |
|---------------|---|
| 00            | $V0 = V_{SS\_HV\_ADC0}$ or $V_{SS\_HV\_ADC1}$ |
| 01            | $V1 = V_{DD\_HV\_ADC0}$ or $V_{DD\_HV\_ADC1}$ |
| 10            | Reserved                                      |
| 11            | Reserved                                      |

Three presampling value fields, one per channel type, in the PSCR make it possible to select different presampling values for each type.

## 28.3.6 Programmable analog watchdog

### 28.3.6.1 Introduction

The analog watchdogs are used for determining whether the result of a channel conversion lies within a given guarded area (as shown in Figure 28-8) specified by upper and lower threshold values named THR<sub>H</sub> and THR<sub>L</sub>, respectively.



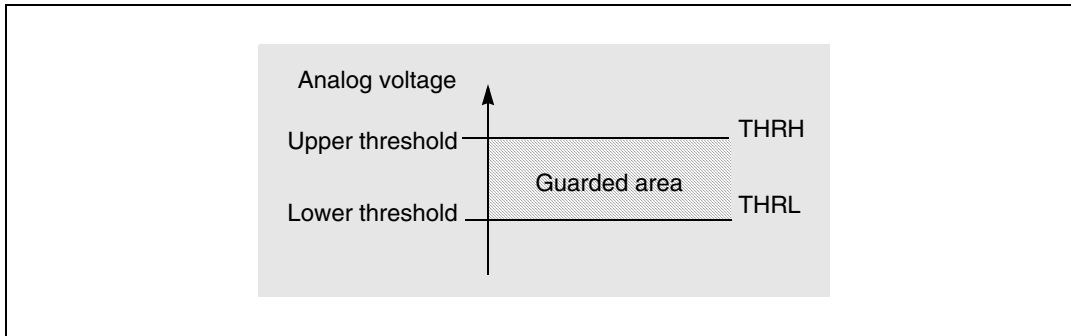


Figure 28-8. Guarded area

After the conversion of the selected channel, a comparison is performed between the converted value and the threshold values. If the converted value lies outside that guarded area then corresponding threshold violation interrupts are generated. The comparison result is stored as WTISR[WDGxH] and WTISR[WDGxL] as explained in Table 28-8. Depending on the mask bits WTIMR[MSKWDGxL] and WTIMR[MSKWDGxH], an interrupt is generated on threshold violation.

Table 28-8. Values of WDGxH and WDGxL fields

| WDGxH | WDGxL | Converted data               |
|-------|-------|------------------------------|
| 1     | 0     | Converted data > THRH        |
| 0     | 1     | Converted data < THRL        |
| 0     | 0     | THRL ≤ converted data ≤ THRH |

Each channel can be enabled independently from the CWENR registers and can select the watchdog threshold registers (THRHLRx) to be used by programming the CWSELR registers. The threshold registers selected by the CWSELR[WSEL\_CHx] provides the threshold values.

For example, if channel number 15 is to be monitored with the threshold values in THRHLR1, then CWSELR[WSEL\_CH15] is programmed to select THRHLR1 to provide the threshold values. The channel monitoring is enabled by setting the bit corresponding to channel 15 in the CWENR.

If a converted value for a particular channel lies outside the range specified by threshold values, then the corresponding bit is set in the Analog Watchdog Out of Range Register (AWORR).

A set of threshold registers (THRHLRx) can be linked to several ADC channels. The threshold values to be selected for a channel need be programmed only once in the CWSELRx.

#### NOTE

If the higher threshold for the analog watchdog is programmed lower than the lower threshold and the converted value is less than the lower threshold, then the WDGxL interrupt for the low threshold violation is set, else if the converted value is greater than the lower threshold (consequently also greater than the higher threshold) then the interrupt WDGxH for high threshold violation is set. Thus, the user should avoid that situation as it could lead to misinterpretation of the watchdog interrupts.

### 28.3.7 DMA functionality

A DMA request can be programmed after the conversion of every channel by setting the respective masking bit in the DMAR registers. The DMAR masking registers must be programmed before starting any conversion. There is one DMAR per channel type and each ADC module has one DMA request associated with it.

The DMA transfers can be enabled using the DMAEN bit of DMAE register. When the DCLR bit of DMAE register is set then the DMA request is cleared on the reading of the register for which DMA transfer has been enabled.

### 28.3.8 Interrupts

The ADC generates the following two maskable interrupt signals:

- ADC\_EOC interrupt requests
  - EOC (end of conversion)
  - ECH (end of chain)
  - JEOC (end of injected conversion)
  - JECH (end of injected chain)
  - EOCTU (end of CTU conversion)
- WDGxL and WDGxH (watchdog threshold) interrupt requests

Interrupts are generated during the conversion process to signal events such as End Of Conversion as explained in register description for CEOCFR[0..2]. Two registers named CEOCFR[0..2] (Channel Pending Registers) and IMR (Interrupt Mask Register) are provided in order to check and enable the interrupt request to INT module.

Interrupts can be individually enabled on a channel by channel basis by programming the CIMR (Channel Interrupt Mask Register).

Several CEOCFR[0..2] are also provided in order to signal which of the channels' measurement has been completed.

The analog watchdog interrupts are handled by two registers, WTISR (Watchdog Threshold Interrupt Status Register) and WTIMR (Watchdog Threshold Interrupt Mask Register), in order to check and enable the interrupt request to the INTC module. The Watchdog interrupt source sets two pending bits WDGxH and WDGxL in the WTISR for each of the channels being monitored.

The CEOCFR[0..2] contains the interrupt pending request status. If the user wants to clear a particular interrupt event status, then writing a 1 to the corresponding status bit clears the pending interrupt flag (at this write operation all the other bits of the CEOCFR[0..2] must be maintained at 0).

### 28.3.9 External decode signals delay

The ADC provides several external decode signals to select which external channel has to be converted. In order to take into account the control switching time of the external analog multiplexer, a Decode

Signals Delay register (DSDR) is provided. The delay between the decoding signal selection and the actual start of conversion can be programmed by writing the field DSD[0:11].

After having selected the channel to be converted, the MA[0:2] control lines are automatically reset. For instance, in the event of normal scan conversion on ANP[0] followed by ANX[0,7] (ADC ch 71) all the MA[0:2] bits are set and subsequently reset.

### 28.3.10 Power-down mode

The analog part of the ADC can be put in low power mode by setting the MCR[PWDN]. After releasing the reset signal the ADC analog module is kept in power-down mode by default, so this state must be exited before starting any operation by resetting the appropriate bit in the MCR.

The power-down mode can be requested at any time by setting the MCR[PWDN]. If a conversion is ongoing, the ADC must complete the conversion before entering the power down mode. In fact, the ADC enters power-down mode only after completing the ongoing conversion. Otherwise, the ongoing operation should be aborted manually by resetting the NSTART bit and using the ABORTCHAIN bit.

MSR[ADCSTATUS] bit is set only when ADC enters power-down mode.

After the power-down phase is completed the process ongoing before the power-down phase must be restarted manually by setting the appropriate MCR[START] bit.

Resetting MCR[PWDN] bit and setting MCR[NSTART] or MCR[JSTART] bit during the same cycle is forbidden.

If a CTU trigger pulse is received during power-down, it is discarded.

If the CTU is enabled and the CSR[CTUSTART] bit is 1, then the MCR[PWDN] bit cannot be set.

When CTU trigger mode is enabled, the application has to wait for the end of conversion (CTUSTART bit automatically reset).

### 28.3.11 Auto-clock-off mode

To reduce power consumption during the IDLE mode of operation (without going into power-down mode), an auto-clock-off feature can be enabled by setting the MCR[ACKO] bit. When enabled, the analog clock is automatically switched off when no operation is ongoing, that is, no conversion is programmed by the user.

#### NOTE

The auto-clock-off feature cannot operate when the digital interface runs at the same rate as the analog interface. This means that when  $MCR.ADCCLKSEL = 1$ , the analog clock will not shut down in IDLE mode.

## 28.4 Register descriptions

### 28.4.1 Introduction

MPC5606BK has two ADCs (10-bit ADC\_0 and 12-bit ADC\_1) and each has specific registers.

Table 28-9 lists the ADC\_0 registers with their address offsets and reset values.

**Table 28-9. 10-bit ADC\_0 digital registers**

| Base address: 0xFFE0_0000 |  | Location                    |
|---------------------------|--|-----------------------------|
| Address offset            | Register name  |                             |
| 0x0000                    | Main Configuration Register (MCR)                    | <a href="#">on page 737</a> |
| 0x0004                    | Main Status Register (MSR)                           | <a href="#">on page 739</a> |
| 0x0008–0x000F             | Reserved   | —                           |
| 0x0010                    | Interrupt Status Register (ISR)                      | <a href="#">on page 740</a> |
| 0x0014                    | Channel Pending Register (CEOCFR0)                   | <a href="#">on page 741</a> |
| 0x0018                    | Channel Pending Register (CEOCFR1)                   | <a href="#">on page 741</a> |
| 0x001C                    | Channel Pending Register (CEOCFR2)                   | <a href="#">on page 741</a> |
| 0x0020                    | Interrupt Mask Register (IMR)                        | <a href="#">on page 743</a> |
| 0x0024                    | Channel Interrupt Mask Register (CIMR0)              | <a href="#">on page 744</a> |
| 0x0028                    | Channel Interrupt Mask Register (CIMR1)              | <a href="#">on page 744</a> |
| 0x002C                    | Channel Interrupt Mask Register (CIMR2)              | <a href="#">on page 744</a> |
| 0x0030                    | Watchdog Threshold Interrupt Status Register (WTISR) | <a href="#">on page 746</a> |
| 0x0034                    | Watchdog Threshold Interrupt Mask Register (WTIMR)   | <a href="#">on page 747</a> |
| 0x0038–0x003F             | Reserved   | —                           |
| 0x0040                    | DMA Enable Register (DMAE)                           | <a href="#">on page 748</a> |
| 0x0044                    | DMA Channel Select Register 0 ()                     | <a href="#">on page 749</a> |
| 0x0048                    | DMA Channel Select Register 1 (DMAR1)                | <a href="#">on page 749</a> |
| 0x004C                    | DMA Channel Select Register 2 (DMAR2)                | <a href="#">on page 749</a> |
| 0x0050–0x005F             | Reserved   | —                           |
| 0x0060                    | Threshold Register 0 (THRHLR0)                       | <a href="#">on page 752</a> |
| 0x0064                    | Threshold Register 1 (THRHLR1)                       | <a href="#">on page 752</a> |
| 0x0068                    | Threshold Register 2 (THRHLR2)                       | <a href="#">on page 752</a> |
| 0x006C                    | Threshold Register 3 (THRHLR3)                       | <a href="#">on page 752</a> |
| 0x0070–0x007F             | Reserved   | —                           |
| 0x0080                    | Presampling Control Register (PSCR)                  | <a href="#">on page 753</a> |
| 0x0084                    | Presampling Register 0 (PSR0)                        | <a href="#">on page 753</a> |

Table 28-9. 10-bit ADC\_0 digital registers (continued)

| Base address: 0xFFE0_0000 |   | Location    |
|---------------------------|---|-------------|
| Address offset            | Register name                               |             |
| 0x0088                    | Presampling Register 1 (PSR1)               | on page 753 |
| 0x008C                    | Presampling Register 2 (PSR2)               | on page 753 |
| 0x0090–0x0093             | Reserved                                    | —           |
| 0x0094                    | Conversion Timing Register 0 (CTR0)         | on page 756 |
| 0x0098                    | Conversion Timing Register 1 (CTR1)         | on page 756 |
| 0x009C                    | Conversion Timing Register 2 (CTR2)         | on page 756 |
| 0x00A0–0x00A3             | Reserved                                    | —           |
| 0x00A4                    | Normal Conversion Mask Register 0 (NCMR0)   | on page 757 |
| 0x00A8                    | Normal Conversion Mask Register 1 (NCMR1)   | on page 757 |
| 0x00AC                    | Normal Conversion Mask Register 2 (NCMR2)   | on page 757 |
| 0x00B0–0x00B3             | Reserved                                    | —           |
| 0x00B4                    | Injected Conversion Mask Register 0 (JCMR0) | on page 760 |
| 0x00B8                    | Injected Conversion Mask Register 1 (JCMR1) | on page 760 |
| 0x00BC                    | Injected Conversion Mask Register 2 (JCMR2) | on page 760 |
| 0x00C0–0x00C3             | Reserved                                    | —           |
| 0x00C4                    | Decode Signals Delay Register (DSDR)        | on page 761 |
| 0x00C8                    | Power-down Exit Delay Register (PDED R)     | on page 762 |
| 0x00CC–0x00FF             | Reserved                                    | —           |
| 0x0100                    | Channel 0 Data Register (CDR0)              | on page 763 |
| 0x0104                    | Channel 1 Data Register (CDR1)              | on page 763 |
| 0x0108                    | Channel 2 Data Register (CDR2)              | on page 763 |
| 0x010C                    | Channel 3 Data Register (CDR3)              | on page 763 |
| 0x0110                    | Channel 4 Data Register (CDR4)              | on page 763 |
| 0x0114                    | Channel 5 Data Register (CDR5)              | on page 763 |
| 0x0118                    | Channel 6 Data Register (CDR6)              | on page 763 |
| 0x011C                    | Channel 7 Data Register (CDR7)              | on page 763 |
| 0x0120                    | Channel 8 Data Register (CDR8)              | on page 763 |
| 0x0124                    | Channel 9 Data Register (CDR9)              | on page 763 |
| 0x0128                    | Channel 10 Data Register (CDR10)            | on page 763 |
| 0x012C                    | Channel 11 Data Register (CDR11)            | on page 763 |
| 0x0130                    | Channel 12 Data Register (CDR12)            | on page 763 |

Table 28-9. 10-bit ADC\_0 digital registers (continued)

| Base address: 0xFFE0_0000 |                                  | Location    |
|---------------------------|----------------------------------|-------------|
| Address offset            | Register name                    |             |
| 0x0134                    | Channel 13 Data Register (CDR13) | on page 763 |
| 0x0138                    | Channel 14 Data Register (CDR14) | on page 763 |
| 0x013C                    | Channel 15 Data Register (CDR15) | on page 763 |
| 0x0140–0x017F             | Reserved                         | —           |
| 0x0180                    | Channel 32 Data Register (CDR32) | on page 763 |
| 0x0184                    | Channel 33 Data Register (CDR33) | on page 763 |
| 0x0188                    | Channel 34 Data Register (CDR34) | on page 763 |
| 0x018C                    | Channel 35 Data Register (CDR35) | on page 763 |
| 0x0190                    | Channel 36 Data Register (CDR36) | on page 763 |
| 0x0194                    | Channel 37 Data Register (CDR37) | on page 763 |
| 0x0198                    | Channel 38 Data Register (CDR38) | on page 763 |
| 0x019C                    | Channel 39 Data Register (CDR39) | on page 763 |
| 0x01A0                    | Channel 40 Data Register (CDR40) | on page 763 |
| 0x01A4                    | Channel 41 Data Register (CDR41) | on page 763 |
| 0x01A8                    | Channel 42 Data Register (CDR42) | on page 763 |
| 0x01AC                    | Channel 43 Data Register (CDR43) | on page 763 |
| 0x01B0                    | Channel 44 Data Register (CDR44) | on page 763 |
| 0x01B4                    | Channel 45 Data Register (CDR45) | on page 763 |
| 0x01B8                    | Channel 46 Data Register (CDR46) | on page 763 |
| 0x01BC                    | Channel 47 Data Register (CDR47) | on page 763 |
| 0x01C0                    | Channel 48 Data Register (CDR48) | on page 763 |
| 0x01C4                    | Channel 49 Data Register (CDR49) | on page 763 |
| 0x01C8                    | Channel 50 Data Register (CDR50) | on page 763 |
| 0x01CC                    | Channel 51 Data Register (CDR51) | on page 763 |
| 0x01D0                    | Channel 52 Data Register (CDR52) | on page 763 |
| 0x01D4                    | Channel 53 Data Register (CDR53) | on page 763 |
| 0x01D8                    | Channel 54 Data Register (CDR54) | on page 763 |
| 0x01DC                    | Channel 55 Data Register (CDR55) | on page 763 |
| 0x01E0                    | Channel 56 Data Register (CDR56) | on page 763 |
| 0x01E4                    | Channel 57 Data Register (CDR57) | on page 763 |
| 0x01E8                    | Channel 58 Data Register (CDR58) | on page 763 |

Table 28-9. 10-bit ADC\_0 digital registers (continued)

| Base address: 0xFFE0_0000 |                                  | Location                    |
|---------------------------|----------------------------------|-----------------------------|
| Address offset            | Register name                    |                             |
| 0x01EC                    | Channel 59 Data Register (CDR59) | <a href="#">on page 763</a> |
| 0x01F0–0x01FF             | Reserved                         | —                           |
| 0x0200                    | Channel 64 Data Register (CDR64) | <a href="#">on page 763</a> |
| 0x0204                    | Channel 65 Data Register (CDR65) | <a href="#">on page 763</a> |
| 0x0208                    | Channel 66 Data Register (CDR66) | <a href="#">on page 763</a> |
| 0x020C                    | Channel 67 Data Register (CDR67) | <a href="#">on page 763</a> |
| 0x0210                    | Channel 68 Data Register (CDR68) | <a href="#">on page 763</a> |
| 0x0214                    | Channel 69 Data Register (CDR69) | <a href="#">on page 763</a> |
| 0x0218                    | Channel 70 Data Register (CDR70) | <a href="#">on page 763</a> |
| 0x021C                    | Channel 71 Data Register (CDR71) | <a href="#">on page 763</a> |
| 0x0220                    | Channel 72 Data Register (CDR72) | <a href="#">on page 763</a> |
| 0x0224                    | Channel 73 Data Register (CDR73) | <a href="#">on page 763</a> |
| 0x0228                    | Channel 74 Data Register (CDR74) | <a href="#">on page 763</a> |
| 0x022C                    | Channel 75 Data Register (CDR75) | <a href="#">on page 763</a> |
| 0x0230                    | Channel 76 Data Register (CDR76) | <a href="#">on page 763</a> |
| 0x0234                    | Channel 77 Data Register (CDR77) | <a href="#">on page 763</a> |
| 0x0238                    | Channel 78 Data Register (CDR78) | <a href="#">on page 763</a> |
| 0x023C                    | Channel 79 Data Register (CDR79) | <a href="#">on page 763</a> |
| 0x0240                    | Channel 80 Data Register (CDR80) | <a href="#">on page 763</a> |
| 0x0244                    | Channel 81 Data Register (CDR81) | <a href="#">on page 763</a> |
| 0x0248                    | Channel 82 Data Register (CDR82) | <a href="#">on page 763</a> |
| 0x024C                    | Channel 83 Data Register (CDR83) | <a href="#">on page 763</a> |
| 0x0250                    | Channel 84 Data Register (CDR84) | <a href="#">on page 763</a> |
| 0x0254                    | Channel 85 Data Register (CDR85) | <a href="#">on page 763</a> |
| 0x0258                    | Channel 86 Data Register (CDR86) | <a href="#">on page 763</a> |
| 0x025C                    | Channel 87 Data Register (CDR87) | <a href="#">on page 763</a> |
| 0x0260                    | Channel 88 Data Register (CDR88) | <a href="#">on page 763</a> |
| 0x0264                    | Channel 89 Data Register (CDR89) | <a href="#">on page 763</a> |
| 0x0268                    | Channel 90 Data Register (CDR90) | <a href="#">on page 763</a> |
| 0x026C                    | Channel 91 Data Register (CDR91) | <a href="#">on page 763</a> |
| 0x0270                    | Channel 92 Data Register (CDR92) | <a href="#">on page 763</a> |

Table 28-9. 10-bit ADC\_0 digital registers (continued)

| Base address: 0xFFE0_0000 |   | Location    |
|---------------------------|---|-------------|
| Address offset            | Register name                                     |             |
| 0x0274                    | Channel 93 Data Register (CDR93)                  | on page 763 |
| 0x0278                    | Channel 94 Data Register (CDR94)                  | on page 763 |
| 0x027C                    | Channel 95 Data Register (CDR95)                  | on page 763 |
| 0x0280                    | Threshold Register 4 (THRHLR4)                    | on page 763 |
| 0x0284                    | Threshold Register 5 (THRHLR5)                    | on page 763 |
| 0x0288–0x02AF             | Reserved  | —           |
| 0x02B0                    | Channel Watchdog Selection Register 0 (CWSELR0)   | on page 765 |
| 0x02B4                    | Channel Watchdog Selection Register 1 (CWSELR1)   | on page 765 |
| 0x02B8–0x02BF             | Reserved  | —           |
| 0x02C0                    | Channel Watchdog Selection Register 4 (CWSELR4)   | on page 765 |
| 0x02C4                    | Channel Watchdog Selection Register 5 (CWSELR5)   | on page 765 |
| 0x02C8                    | Channel Watchdog Selection Register 6 (CWSELR6)   | on page 765 |
| 0x02CC                    | Channel Watchdog Selection Register 7 (CWSELR7)   | on page 765 |
| 0x02D0                    | Channel Watchdog Selection Register 8 (CWSELR8)   | on page 765 |
| 0x02D4                    | Channel Watchdog Selection Register 9 (CWSELR9)   | on page 765 |
| 0x02D8                    | Channel Watchdog Selection Register 10 (CWSELR10) | on page 765 |
| 0x02DC                    | Channel Watchdog Selection Register 11 (CWSELR11) | on page 765 |
| 0x02E0                    | Channel Watchdog Enable Register 0 (CWENR0)       | on page 773 |
| 0x02E4                    | Channel Watchdog Enable Register 1 (CWENR1)       | on page 773 |
| 0x02E8                    | Channel Watchdog Enable Register 2 (CWENR2)       | on page 773 |
| 0x02EC–0x02EF             | Reserved  | —           |
| 0x02F0                    | Analog Watchdog Out of Range register 0 (AWORR0)  | on page 775 |
| 0x02F4                    | Analog Watchdog Out of Range register 1 (AWORR1)  | on page 775 |
| 0x02F8                    | Analog Watchdog Out of Range register 2 (AWORR2)  | on page 775 |
| 0x2FC–0x02FF              | Reserved  | —           |

Table 28-10 lists the ADC\_1 registers with their address offsets and reset values.



Table 28-10. 12-bit ADC\_1 digital registers

| Base address: 0xFFE0_4000 |  | Location                    |
|---------------------------|--|-----------------------------|
| Address offset            | Register name  |                             |
| 0x0000                    | Main Configuration Register (MCR)                    | <a href="#">on page 737</a> |
| 0x0004                    | Main Status Register (MSR)                           | <a href="#">on page 739</a> |
| 0x0008–0x000F             | Reserved   | —                           |
| 0x0010                    | Interrupt Status Register (ISR)                      | <a href="#">on page 740</a> |
| 0x0014                    | Channel Pending Register (CEOCFR0)                   | <a href="#">on page 741</a> |
| 0x0018                    | Channel Pending Register (CEOCFR1)                   | <a href="#">on page 741</a> |
| 0x001C                    | Reserved   | —                           |
| 0x0020                    | Interrupt Mask Register (IMR)                        | <a href="#">on page 743</a> |
| 0x0024                    | Channel Interrupt Mask Register 0 (CIMR0)            | <a href="#">on page 744</a> |
| 0x0028                    | Channel Interrupt Mask Register 1 (CIMR1)            | <a href="#">on page 744</a> |
| 0x002C                    | Reserved   | —                           |
| 0x0030                    | Watchdog Threshold Interrupt Status Register (WTISR) | <a href="#">on page 746</a> |
| 0x0034                    | Watchdog Threshold Interrupt Mask Register (WTIMR)   | <a href="#">on page 747</a> |
| 0x0038–0x003F             | Reserved   | —                           |
| 0x0040                    | DMA Enable Register (DMAE)                           | <a href="#">on page 748</a> |
| 0x0044                    | DMA Channel Select Register 0 (DMAR0)                | <a href="#">on page 748</a> |
| 0x0048                    | DMA Channel Select Register 1 (DMAR1)                | <a href="#">on page 748</a> |
| 0x004C                    | Reserved   | —                           |
| 0x0060                    | Threshold Register 0 (THRHLR0)                       | <a href="#">on page 752</a> |
| 0x0064                    | Threshold Register 1 (THRHLR1)                       | <a href="#">on page 752</a> |
| 0x0068                    | Threshold Register 2 (THRHLR2)                       | <a href="#">on page 752</a> |
| 0x006C–0x007F             | Reserved   | —                           |
| 0x0080                    | Presampling Control Register (PSCR)                  | <a href="#">on page 753</a> |
| 0x0084                    | Presampling Register 0 (PSR0)                        | <a href="#">on page 753</a> |
| 0x0088                    | Presampling Register 1 (PSR1)                        | <a href="#">on page 753</a> |
| 0x008C–0x0093             | Reserved   | —                           |
| 0x0094                    | Conversion Timing Register 0 (CTR0)                  | <a href="#">on page 756</a> |
| 0x0098                    | Conversion Timing Register 1 (CTR1)                  | <a href="#">on page 756</a> |
| 0x009C–0x00A3             | Reserved   | —                           |
| 0x00A4                    | Normal Conversion Mask Register 0 (NCMR0)            | <a href="#">on page 757</a> |
| 0x00A8                    | Normal Conversion Mask Register 1 (NCMR1)            | <a href="#">on page 757</a> |

Table 28-10. 12-bit ADC\_1 digital registers (continued)

| Base address: 0xFFE0_4000 |   | Location    |
|---------------------------|---|-------------|
| Address offset            | Register name                               |             |
| 0x00A8–0x00AC             | Reserved                                    | —           |
| 0x00B4                    | Injected Conversion Mask Register 0 (JCMR0) | on page 760 |
| 0x00B8                    | Injected Conversion Mask Register 1 (JCMR1) |             |
| 0x00BC–0x00C7             | Reserved                                    | —           |
| 0x00C8                    | Power-down Exit Delay Register (PDEDL)      | on page 762 |
| 0x00CC–0x00FF             | Reserved                                    | —           |
| 0x0100                    | Channel 0 Data Register (CDR0)              | on page 763 |
| 0x0104                    | Channel 1 Data Register (CDR1)              | on page 763 |
| 0x0108                    | Channel 2 Data Register (CDR2)              | on page 763 |
| 0x010C                    | Channel 3 Data Register (CDR3)              | on page 763 |
| 0x0110                    | Channel 4 Data Register (CDR4)              | on page 763 |
| 0x0114                    | Channel 5 Data Register (CDR5)              | on page 763 |
| 0x0118                    | Channel 6 Data Register (CDR6)              | on page 763 |
| 0x011C                    | Channel 7 Data Register (CDR7)              | on page 763 |
| 0x0120                    | Channel 8 Data Register (CDR8)              | on page 763 |
| 0x0124                    | Channel 9 Data Register (CDR9)              | on page 763 |
| 0x0128                    | Channel 10 Data Register (CDR10)            | on page 763 |
| 0x012C                    | Channel 11 Data Register (CDR11)            | on page 763 |
| 0x0130                    | Channel 12 Data Register (CDR12)            | on page 763 |
| 0x0134                    | Channel 13 Data Register (CDR13)            | on page 763 |
| 0x0138                    | Channel 14 Data Register (CDR14)            | on page 763 |
| 0x013C                    | Channel 15 Data Register (CDR15)            | on page 763 |
| 0x0140–0x017F             | Reserved                                    | —           |
| 0x0180                    | Channel 32 Data Register (CDR32)            | on page 763 |
| 0x0184                    | Channel 33 Data Register (CDR33)            | on page 763 |
| 0x0188                    | Channel 34 Data Register (CDR34)            | on page 763 |
| 0x018C                    | Channel 35 Data Register (CDR35)            | on page 763 |
| 0x0190                    | Channel 36 Data Register (CDR36)            | on page 763 |
| 0x0194                    | Channel 37 Data Register (CDR37)            | on page 763 |
| 0x0198                    | Channel 38 Data Register (CDR38)            | on page 763 |
| 0x019C                    | Channel 39 Data Register (CDR39)            | on page 763 |

Table 28-10. 12-bit ADC\_1 digital registers (continued)

| Base address: 0xFFE0_4000 |  | Location    |
|---------------------------|--|-------------|
| Address offset            | Register name                                    |             |
| 0x01A0–0x02AF             | Reserved   | —           |
| 0x02B0                    | Channel Watchdog Selection Register 0 (CWSELR0)  | on page 765 |
| 0x02B4                    | Channel Watchdog Selection Register 1 (CWSELR1)  | on page 765 |
| 0x02B8–0x02BF             | Reserved   | —           |
| 0x02C0                    | Channel Watchdog Selection Register 4 (CWSELR4)  | on page 765 |
| 0x02C4–0x02DF             | Reserved   | —           |
| 0x02E0                    | Channel Watchdog Enable Register 0 (CWENR0)      | on page 773 |
| 0x02E4                    | Channel Watchdog Enable Register 1 (CWENR1)      | on page 773 |
| 0x02F0                    | Analog Watchdog Out of Range register 0 (AWORR0) | on page 775 |
| 0x02F4                    | Analog Watchdog Out of Range register 1 (AWORR1) | on page 775 |
| 0x02F8–0x02FF             | Reserved   | —           |

## 28.4.2 Control logic registers

### 28.4.2.1 Main Configuration Register (MCR)

The Main Configuration Register (MCR) provides configuration settings for the ADC.

Address: Base + 0x0000

Access: User read/write

|       |       |        |      |    |    |    |    |           |             |        |       |        |    |    |       |      |
|-------|-------|--------|------|----|----|----|----|-----------|-------------|--------|-------|--------|----|----|-------|------|
|       | 0     | 1      | 2    | 3  | 4  | 5  | 6  | 7         | 8           | 9      | 10    | 11     | 12 | 13 | 14    | 15   |
| R     | OWREN | WLSIDE | MODE | 0  | 0  | 0  | 0  | NSTART    | 0           | JTRGEN | JEDGE | JSTART | 0  | 0  | CTUEN | 0    |
| W     |       |        |      |    |    |    |    |           |             |        |       |        |    |    |       |      |
| Reset | 0     | 0      | 0    | 0  | 0  | 0  | 0  | 0         | 0           | 0      | 0     | 0      | 0  | 0  | 0     | 0    |
|       | 16    | 17     | 18   | 19 | 20 | 21 | 22 | 23        | 24          | 25     | 26    | 27     | 28 | 29 | 30    | 31   |
| R     | 0     | 0      | 0    | 0  | 0  | 0  | 0  | ADCLK SEL | ABORT CHAIN | ABORT  | ACKO  | 0      | 0  | 0  | 0     |      |
| W     |       |        |      |    |    |    |    |           |             |        |       |        |    |    |       | PWDN |
| Reset | 0     | 0      | 0    | 0  | 0  | 0  | 0  | 0         | 0           | 0      | 0     | 0      | 0  | 0  | 0     | 1    |

Figure 28-9. Main Configuration Register (MCR)

Table 28-11. MCR field descriptions

| Field      | Description   |
|------------|---|
| OWREN      | <p>Overwrite enable</p> <p>This bit enables or disables the functionality to overwrite unread converted data.</p> <p>0 Prevents overwrite of unread converted data; new result is discarded</p> <p>1 Enables converted data to be overwritten by a new conversion</p>   |
| WLSIDE     | <p>Write left/right-aligned</p> <p>0 The conversion data is written right-aligned.</p> <p>1 Data is left-aligned (from 15 to (15 – resolution + 1)).</p> <p>The WLSIDE bit affects all the CDR registers simultaneously. See <a href="#">Figure 28-48</a> and <a href="#">Figure 28-48</a>.</p>   |
| MODE       | <p>One Shot/Scan</p> <p>0 One Shot Mode—Configures the normal conversion of one chain.</p> <p>1 Scan Mode—Configures continuous chain conversion mode; when the programmed chain conversion is finished it restarts immediately.</p>  |
| NSTART     | <p>Normal Start conversion</p> <p>Setting this bit starts the chain or scan conversion. Resetting this bit during scan mode causes the current chain conversion to finish, then stops the operation.</p> <p>This bit stays high while the conversion is ongoing (or pending during injection mode).</p> <p>0 Causes the current chain conversion to finish and stops the operation</p> <p>1 Starts the chain or scan conversion</p> |
| JTRGEN     | <p>Injection external trigger enable</p> <p>0 External trigger disabled for channel injection</p> <p>1 External trigger enabled for channel injection</p>   |
| JEDGE      | <p>Injection trigger edge selection</p> <p>Edge selection for external trigger, if JTRGEN = 1.</p> <p>0 Selects falling edge for the external trigger</p> <p>1 Selects rising edge for the external trigger</p>   |
| JSTART     | <p>Injection start</p> <p>Setting this bit will start the configured injected analog channels to be converted by software. Resetting this bit has no effect, as the injected chain conversion cannot be interrupted.</p>  |
| CTUEN      | <p>Cross trigger unit conversion enable</p> <p>0 CTU triggered conversion disabled</p> <p>1 CTU triggered conversion enabled</p>  |
| ADCLKSEL   | <p>Analog clock select</p> <p>This bit can only be written when ADC in Power-Down mode</p> <p>0 ADC clock frequency is half Peripheral Set Clock frequency</p> <p>1 ADC clock frequency is equal to Peripheral Set Clock frequency</p>  |
| ABORTCHAIN | <p>Abort Chain</p> <p>When this bit is set, the ongoing Chain Conversion is aborted. This bit is reset by hardware as soon as a new conversion is requested.</p> <p>0 Conversion is not affected</p> <p>1 Aborts the ongoing chain conversion</p>   |
| ABORT      | <p>Abort Conversion</p> <p>When this bit is set, the ongoing conversion is aborted and a new conversion is invoked. This bit is reset by hardware as soon as a new conversion is invoked. If it is set during a scan chain, only the ongoing conversion is aborted and the next conversion is performed as planned.</p> <p>0 Conversion is not affected</p> <p>1 Aborts the ongoing conversion</p>                                  |

Table 28-11. MCR field descriptions (continued)

| Field | Description  |
|-------|--|
| ACKO  | Auto-clock-off enable<br>If set, this bit enables the Auto clock off feature.<br>0 Auto clock off disabled<br>1 Auto clock off enabled   |
| PWDN  | Power-down enable<br>When this bit is set, the analog module is requested to enter Power Down mode. When ADC status is PWDN, resetting this bit starts ADC transition to IDLE mode.<br>0 ADC is in normal mode<br>1 ADC has been requested to power down |

### 28.4.2.2 Main Status Register (MSR)

The Main Status Register (MSR) provides status bits for the ADC.

Address: Base + 0x0004

Access: User read-only

|       |        |    |    |    |    |    |    |        |        |    |    |        |    |    |           |          |
|-------|--------|----|----|----|----|----|----|--------|--------|----|----|--------|----|----|-----------|----------|
|       | 0      | 1  | 2  | 3  | 4  | 5  | 6  | 7      | 8      | 9  | 10 | 11     | 12 | 13 | 14        | 15       |
| R     | 0      | 0  | 0  | 0  | 0  | 0  | 0  | NSTART | JABORT | 0  | 0  | JSTART | 0  | 0  | 0         | CTUSTART |
| W     |        |    |    |    |    |    |    |        |        |    |    |        |    |    |           |          |
| Reset | 0      | 0  | 0  | 0  | 0  | 0  | 0  | 0      | 0      | 0  | 0  | 0      | 0  | 0  | 0         | 0        |
|       | 16     | 17 | 18 | 19 | 20 | 21 | 22 | 23     | 24     | 25 | 26 | 27     | 28 | 29 | 30        | 31       |
| R     | CHADDR |    |    |    |    |    |    |        | 0      | 0  | 0  | ACK0   | 0  | 0  | ADCSTATUS |          |
| W     |        |    |    |    |    |    |    |        |        |    |    |        |    |    |           |          |
| Reset | 0      | 0  | 0  | 0  | 0  | 0  | 0  | 0      | 0      | 0  | 0  | 0      | 0  | 0  | 0         | 1        |

Figure 28-10. Main Status Register (MSR)

Table 28-12. MSR field descriptions

| Field    | Description  |
|----------|--|
| NSTART   | This status bit is used to signal that a Normal conversion is ongoing.   |
| JABORT   | This status bit is used to signal that an Injected conversion has been aborted. This bit is reset when a new injected conversion starts. |
| JSTART   | This status bit is used to signal that an Injected conversion is ongoing.  |
| CTUSTART | This status bit is used to signal that a CTU conversion is ongoing.  |
| CHADDR   | Current conversion channel address<br>This status field indicates current conversion channel address.                                    |
| ACKO     | Auto-clock-off enable<br>This status bit is used to signal if the Auto-clock-off feature is on.  |

**Table 28-12. MSR field descriptions (continued)**

| Field     | Description   |
|-----------|---|
| ADCSTATUS | The value of this parameter depends on ADC status:<br>000 IDLE — The ADC is powered up but idle.<br>001 Power-down — The ADC is powered down.<br>010 Wait state — The ADC is waiting for an external multiplexer, This only occurs when the DSDR register is non-zero.<br>011 Reserved<br>100 Sample — The ADC is sampling the analog signal.<br>101 Reserved<br>110 Conversion — The ADC is converting the sampled signal.<br>111 Reserved |

**NOTE**

MSR[JSTART] is automatically set when the injected conversion starts. At the same time MCR[JSTART] is reset, allowing the software to program a new start of conversion.

The JCMR registers do not change their values.

### 28.4.3 Interrupt registers

#### 28.4.3.1 Interrupt Status Register (ISR)

The Interrupt Status Register (ISR) contains interrupt status bits for the ADC.

Address: Base + 0x0010 Access: User read/write

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       |    |    |    |    |    |    |    |    |    |    |    |       |      |      |     |     |
|-------|----|----|----|----|----|----|----|----|----|----|----|-------|------|------|-----|-----|
|       | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27    | 28   | 29   | 30  | 31  |
| R     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | EOCTU | JEOC | JECH | EOC | ECH |
| W     |    |    |    |    |    |    |    |    |    |    |    | w1c   | w1c  | w1c  | w1c | w1c |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0     | 0    | 0    | 0   | 0   |

**Figure 28-11. Interrupt Status Register (ISR)**

**Table 28-13. ISR field descriptions**

| Field | Description  |
|-------|--|
| EOCTU | End of CTU Conversion interrupt flag<br>When this bit is set, an EOCTU interrupt has occurred. |

**Table 28-13. ISR field descriptions (continued)**

| Field | Description   |
|-------|---|
| JEOC  | End of Injected Channel Conversion interrupt flag<br>When this bit is set, a JEOC interrupt has occurred. |
| JECH  | End of Injected Chain Conversion interrupt flag<br>When this bit is set, a JECH interrupt has occurred.   |
| EOC   | End of Channel Conversion interrupt flag<br>When this bit is set, an EOC interrupt has occurred.          |
| ECH   | End of Chain Conversion interrupt flag<br>When this bit is set, an ECH interrupt has occurred.            |

### 28.4.3.2 Channel Pending Registers (CEOCFR[0..2])

Table 28-14 shows the available channels.

**Table 28-14. CEOCFR[0..2] register description**

| ADC   | Register | Description  |
|-------|----------|--|
| ADC_0 | CEOCFR0  | End of conversion pending interrupt for channel 0 to 15 (precision channels)             |
| ADC_0 | CEOCFR1  | End of conversion pending interrupt for channel 32 to 59 (standard channels)             |
| ADC_0 | CEOCFR2  | End of conversion pending interrupt for channel 64 to 95 (external multiplexed channels) |
| ADC_1 | CEOCFR0  | End of conversion pending interrupt for channel 0 to 15 (precision channels)             |
| ADC_1 | CEOCFR1  | End of conversion pending interrupt for channel 32 to 39 (standard channels)             |

Address: Base + 0x0014

Access: User read/write

|       |          |          |          |          |          |          |         |         |         |         |         |         |         |         |         |         |
|-------|----------|----------|----------|----------|----------|----------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|
|       | 0        | 1        | 2        | 3        | 4        | 5        | 6       | 7       | 8       | 9       | 10      | 11      | 12      | 13      | 14      | 15      |
| R     | 0        | 0        | 0        | 0        | 0        | 0        | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       |
| W     |          |          |          |          |          |          |         |         |         |         |         |         |         |         |         |         |
| Reset | 0        | 0        | 0        | 0        | 0        | 0        | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       |
|       | 16       | 17       | 18       | 19       | 20       | 21       | 22      | 23      | 24      | 25      | 26      | 27      | 28      | 29      | 30      | 31      |
| R     | EOC_CH15 | EOC_CH14 | EOC_CH13 | EOC_CH12 | EOC_CH11 | EOC_CH10 | EOC_CH9 | EOC_CH8 | EOC_CH7 | EOC_CH6 | EOC_CH5 | EOC_CH4 | EOC_CH3 | EOC_CH2 | EOC_CH1 | EOC_CH0 |
| W     | w1c      | w1c      | w1c      | w1c      | w1c      | w1c      | w1c     | w1c     | w1c     | w1c     | w1c     | w1c     | w1c     | w1c     | w1c     | w1c     |
| Reset | 0        | 0        | 0        | 0        | 0        | 0        | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       |

**Figure 28-12. Channel Pending Register 0 (CEOCFR0)**

Address: Base + 0x0018

Access: User read/write

|       |   |   |   |   |          |          |          |          |          |          |          |          |          |          |          |          |
|-------|---|---|---|---|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|
|       | 0 | 1 | 2 | 3 | 4        | 5        | 6        | 7        | 8        | 9        | 10       | 11       | 12       | 13       | 14       | 15       |
| R     | 0 | 0 | 0 | 0 | EOC_CH59 | EOC_CH58 | EOC_CH57 | EOC_CH56 | EOC_CH55 | EOC_CH54 | EOC_CH53 | EOC_CH52 | EOC_CH51 | EOC_CH50 | EOC_CH49 | EOC_CH48 |
| W     |   |   |   |   | w1c      | w1c      | w1c      | w1c      | w1c      | w1c      | w1c      | w1c      | w1c      | w1c      | w1c      | w1c      |
| Reset | 0 | 0 | 0 | 0 | 0        | 0        | 0        | 0        | 0        | 0        | 0        | 0        | 0        | 0        | 0        | 0        |

|       |          |          |          |          |          |          |          |          |          |          |          |          |          |          |          |          |
|-------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|
|       | 16       | 17       | 18       | 19       | 20       | 21       | 22       | 23       | 24       | 25       | 26       | 27       | 28       | 29       | 30       | 31       |
| R     | EOC_CH47 | EOC_CH46 | EOC_CH43 | EOC_CH44 | EOC_CH43 | EOC_CH42 | EOC_CH41 | EOC_CH40 | EOC_CH39 | EOC_CH38 | EOC_CH37 | EOC_CH36 | EOC_CH35 | EOC_CH34 | EOC_CH33 | EOC_CH32 |
| W     | w1c      | w1c      | w1c      | w1c      | w1c      | w1c      | w1c      | w1c      | w1c      | w1c      | w1c      | w1c      | w1c      | w1c      | w1c      | w1c      |
| Reset | 0        | 0        | 0        | 0        | 0        | 0        | 0        | 0        | 0        | 0        | 0        | 0        | 0        | 0        | 0        | 0        |

Figure 28-13. Channel Pending Register 1 (CEOFR1) – ADC\_0

Address: Base + 0x0018

Access: User read/write

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       |    |    |    |    |    |    |    |    |          |          |          |          |          |          |          |          |
|-------|----|----|----|----|----|----|----|----|----------|----------|----------|----------|----------|----------|----------|----------|
|       | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24       | 25       | 26       | 27       | 28       | 29       | 30       | 31       |
| R     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | EOC_CH39 | EOC_CH38 | EOC_CH37 | EOC_CH36 | EOC_CH35 | EOC_CH34 | EOC_CH33 | EOC_CH32 |
| W     |    |    |    |    |    |    |    |    | w1c      | w1c      | w1c      | w1c      | w1c      | w1c      | w1c      | w1c      |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0        | 0        | 0        | 0        | 0        | 0        | 0        | 0        |

Figure 28-14. Channel Pending Register 1 (CEOFR1) – ADC\_1



Address: Base + 0x001C

Access: User read/write

|       |          |          |          |          |          |          |          |          |          |          |          |          |          |          |          |          |
|-------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|
|       | 0        | 1        | 2        | 3        | 4        | 5        | 6        | 7        | 8        | 9        | 10       | 11       | 12       | 13       | 14       | 15       |
| R     | EOC_CH95 | EOC_CH94 | EOC_CH93 | EOC_CH92 | EOC_CH91 | EOC_CH90 | EOC_CH89 | EOC_CH88 | EOC_CH87 | EOC_CH86 | EOC_CH85 | EOC_CH84 | EOC_CH83 | EOC_CH82 | EOC_CH81 | EOC_CH80 |
| W     | w1c      | w1c      | w1c      | w1c      | w1c      | w1c      | w1c      | w1c      | w1c      | w1c      | w1c      | w1c      | w1c      | w1c      | w1c      | w1c      |
| Reset | 0        | 0        | 0        | 0        | 0        | 0        | 0        | 0        | 0        | 0        | 0        | 0        | 0        | 0        | 0        | 0        |

|       |          |          |          |          |          |          |          |          |          |          |          |          |          |          |          |          |
|-------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|
|       | 16       | 17       | 18       | 19       | 20       | 21       | 22       | 23       | 24       | 25       | 26       | 27       | 28       | 29       | 30       | 31       |
| R     | EOC_CH79 | EOC_CH78 | EOC_CH77 | EOC_CH76 | EOC_CH75 | EOC_CH74 | EOC_CH73 | EOC_CH72 | EOC_CH71 | EOC_CH70 | EOC_CH69 | EOC_CH68 | EOC_CH67 | EOC_CH66 | EOC_CH65 | EOC_CH64 |
| W     | w1c      | w1c      | w1c      | w1c      | w1c      | w1c      | w1c      | w1c      | w1c      | w1c      | w1c      | w1c      | w1c      | w1c      | w1c      | w1c      |
| Reset | 0        | 0        | 0        | 0        | 0        | 0        | 0        | 0        | 0        | 0        | 0        | 0        | 0        | 0        | 0        | 0        |

Figure 28-15. Channel Pending Register 2 (CEOCFR2)

Table 28-15. CEOCFR field descriptions

| Field   | Description                                      |
|---------|--|
| EOC_CHn | When set, the measure of channel n is completed. |

**NOTE**

CEOCFR2 is not implemented on ADC\_1.

**28.4.3.3 Interrupt Mask Register (IMR)**

The Interrupt Mask Register (IMR) contains the interrupt enable bits for the ADC.

Address: Base + 0x0020

Access: User read/write

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       |    |    |    |    |    |    |    |    |    |    |    |    |              |             |             |            |            |
|-------|----|----|----|----|----|----|----|----|----|----|----|----|--------------|-------------|-------------|------------|------------|
|       | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28           | 29          | 30          | 31         |            |
| R     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | MSKE<br>OCTU | MSK<br>JEOC | MSK<br>JECH | MSK<br>EOC | MSK<br>ECH |
| W     |    |    |    |    |    |    |    |    |    |    |    |    |              |             |             |            |            |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0            | 0           | 0           | 0          |            |

Figure 28-16. Interrupt Mask Register (IMR)

**Table 28-16. Interrupt Mask Register (IMR) field descriptions**

| Field    | Description  |
|----------|--|
| MSKEOCTU | Mask for end of CTU conversion (EOCTU) interrupt<br>When set, the EOCTU interrupt is enabled.            |
| MSKJEOC  | Mask for end of injected channel conversion (JEOC) interrupt<br>When set, the JEOC interrupt is enabled. |
| MSKJECH  | Mask for end of injected chain conversion (JECH) interrupt<br>When set, the JECH interrupt is enabled.   |
| MSKEOC   | Mask for end of channel conversion (EOC) interrupt<br>When set, the EOC interrupt is enabled.            |
| MSKECH   | Mask for end of chain conversion (ECH) interrupt<br>When set, the ECH interrupt is enabled.              |

### 28.4.3.4 Channel Interrupt Mask Register (CIMR[0..2])

Table 28-17 shows the available channels.

| ADC   | Register | Description   |
|-------|----------|---|
| ADC_0 | CIMR0    | Enable bit for channel 0 to 15 (precision channels)             |
| ADC_0 | CIMR1    | Enable bit for channel 32 to 59 (standard channels)             |
| ADC_0 | CIMR2    | Enable bit for channel 64 to 95 (external multiplexed channels) |
| ADC_1 | CIMR0    | Enable bit for channel 0 to 15 (precision channels)             |
| ADC_1 | CIMR1    | Enable bit for channel 32 to 39 (standard channels)             |

**Table 28-17. CIMR[0..2] register description**

Address: Base + 0x0024

Access: User read/write

|       |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|-------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
|       | 0   | 1   | 2   | 3   | 4   | 5   | 6   | 7   | 8   | 9   | 10  | 11  | 12  | 13  | 14  | 15  |
| R     | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |
| W     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| Reset | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |
|       | 16  | 17  | 18  | 19  | 20  | 21  | 22  | 23  | 24  | 25  | 26  | 27  | 28  | 29  | 30  | 31  |
| R     | CIM | CIM | CIM | CIM | CIM | CIM | CIM | CIM | CIM | CIM | CIM | CIM | CIM | CIM | CIM | CIM |
| W     | 15  | 14  | 13  | 12  | 11  | 10  | 9   | 8   | 7   | 6   | 5   | 4   | 3   | 2   | 1   | 0   |
| Reset | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |

**Figure 28-17. Channel Interrupt Mask Register 0 (CIMR0)**

Address: Base + 0x0028

Access: User read/write

|       |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|-------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
|       | 0   | 1   | 2   | 3   | 4   | 5   | 6   | 7   | 8   | 9   | 10  | 11  | 12  | 13  | 14  | 15  |
| R     | 0   | 0   | 0   | 0   | CIM | CIM | CIM | CIM | CIM | CIM | CIM | CIM | CIM | CIM | CIM | CIM |
| W     |     |     |     |     | 59  | 58  | 57  | 56  | 55  | 54  | 53  | 52  | 51  | 50  | 49  | 48  |
| Reset | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |
|       | 16  | 17  | 18  | 19  | 20  | 21  | 22  | 23  | 24  | 25  | 26  | 27  | 28  | 29  | 30  | 31  |
| R     | CIM | CIM | CIM | CIM | CIM | CIM | CIM | CIM | CIM | CIM | CIM | CIM | CIM | CIM | CIM | CIM |
| W     | 47  | 46  | 43  | 44  | 43  | 42  | 41  | 40  | 39  | 38  | 37  | 36  | 35  | 34  | 33  | 32  |
| Reset | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |

Figure 28-18. Channel Interrupt Mask Register 1 (CIMR1) for ADC\_0

Address: Base + 0x0028

Access: User read/write

|       |    |    |    |    |    |    |    |    |     |     |     |     |     |     |     |     |
|-------|----|----|----|----|----|----|----|----|-----|-----|-----|-----|-----|-----|-----|-----|
|       | 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8   | 9   | 10  | 11  | 12  | 13  | 14  | 15  |
| R     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |
| W     |    |    |    |    |    |    |    |    |     |     |     |     |     |     |     |     |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |
|       | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24  | 25  | 26  | 27  | 28  | 29  | 30  | 31  |
| R     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | CIM | CIM | CIM | CIM | CIM | CIM | CIM | CIM |
| W     |    |    |    |    |    |    |    |    | 39  | 38  | 37  | 36  | 35  | 34  | 33  | 32  |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |

Figure 28-19. Channel Interrupt Mask Register 1 (CIMR1) for ADC\_1

Address: Base + 0x002C

Access: User read/write

|       |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|-------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
|       | 0   | 1   | 2   | 3   | 4   | 5   | 6   | 7   | 8   | 9   | 10  | 11  | 12  | 13  | 14  | 15  |
| R     | CIM | CIM | CIM | CIM | CIM | CIM | CIM | CIM | CIM | CIM | CIM | CIM | CIM | CIM | CIM | CIM |
| W     | 95  | 94  | 93  | 92  | 91  | 90  | 89  | 88  | 87  | 86  | 85  | 84  | 83  | 82  | 81  | 80  |
| Reset | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |
|       | 16  | 17  | 18  | 19  | 20  | 21  | 22  | 23  | 24  | 25  | 26  | 27  | 28  | 29  | 30  | 31  |
| R     | CIM | CIM | CIM | CIM | CIM | CIM | CIM | CIM | CIM | CIM | CIM | CIM | CIM | CIM | CIM | CIM |
| W     | 79  | 78  | 77  | 76  | 75  | 74  | 73  | 72  | 71  | 70  | 69  | 68  | 67  | 66  | 65  | 64  |
| Reset | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |

Figure 28-20. Channel Interrupt Mask Register 2 (CIMR2)

**Table 28-18. CIMR field descriptions**

| Field | Description  |
|-------|--|
| CIMn  | Interrupt enable<br>When set (CIMn = 1), interrupt for channel n is enabled. |

### 28.4.3.5 Watchdog Threshold Interrupt Status Register (WTISR)

For ADC\_0 (10-bit)

Address: Base + 0x0030

Access: User read/write

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       |    |    |    |    |           |           |           |           |           |           |           |           |           |           |           |           |
|-------|----|----|----|----|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|
|       | 16 | 17 | 18 | 19 | 20        | 21        | 22        | 23        | 24        | 25        | 26        | 27        | 28        | 29        | 30        | 31        |
| R     | 0  | 0  | 0  | 0  | WDG<br>5H | WDG<br>5L | WDG<br>4H | WDG<br>4L | WDG<br>3H | WDG<br>3L | WDG<br>2H | WDG<br>2L | WDG<br>1H | WDG<br>1L | WDG<br>0H | WDG<br>0L |
| W     |    |    |    |    | w1c       | w1c       | w1c       | w1c       | w1c       | w1c       | w1c       | w1c       | w1c       | w1c       | w1c       | w1c       |
| Reset | 0  | 0  | 0  | 0  | 0         | 0         | 0         | 0         | 0         | 0         | 0         | 0         | 0         | 0         | 0         | 0         |

**Figure 28-21. ADC\_0 Watchdog Threshold Interrupt Status Register (WTISR)**

**Table 28-19. ADC\_0 WTISR field descriptions**

| Field | Description   |
|-------|---|
| WDGxH | This corresponds to the interrupt generated on the converted value being higher than the programmed higher threshold (for[x = 0..5]). |
| WDGxL | This corresponds to the interrupt generated on the converted value being lower than the programmed lower threshold (for[x = 0..5]).   |

For ADC\_1 (12-bit)

Address: Base + 0x0030

Access: User read/write

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       |    |    |    |    |    |    |    |    |    |    |           |           |           |           |           |           |
|-------|----|----|----|----|----|----|----|----|----|----|-----------|-----------|-----------|-----------|-----------|-----------|
|       | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26        | 27        | 28        | 29        | 30        | 31        |
| R     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | WDG<br>2H | WDG<br>2L | WDG<br>1H | WDG<br>1L | WDG<br>0H | WDG<br>0L |
| W     |    |    |    |    |    |    |    |    |    |    | w1c       | w1c       | w1c       | w1c       | w1c       | w1c       |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0         | 0         | 0         | 0         | 0         | 0         |

Figure 28-22. ADC\_1 Watchdog Threshold Interrupt Status Register (WTISR)

Table 28-20. ADC\_1 WTISR field descriptions

| Field | Description  |
|-------|--|
| WDGxH | This corresponds to the interrupt generated on the converted value being higher than the programmed higher threshold (for [x = 0..2]). |
| WDGxL | This corresponds to the interrupt generated on the converted value being lower than the programmed lower threshold (for [x = 0..2]).   |

### 28.4.3.6 Watchdog Threshold Interrupt Mask Register (WTIMR)

For ADC\_0 (10-bit):

Address: Base + 0x0034

Access: User read/write

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       |    |    |    |    |                  |                  |                  |                  |                  |                  |                  |                  |                  |                  |                  |                  |
|-------|----|----|----|----|------------------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|
|       | 16 | 17 | 18 | 19 | 20               | 21               | 22               | 23               | 24               | 25               | 26               | 27               | 28               | 29               | 30               | 31               |
| R     | 0  | 0  | 0  | 0  | MSK<br>WDG<br>5H | MSK<br>WDG<br>5L | MSK<br>WDG<br>4H | MSK<br>WDG<br>4L | MSK<br>WDG<br>3H | MSK<br>WDG<br>3L | MSK<br>WDG<br>2H | MSK<br>WDG<br>2L | MSK<br>WDG<br>1H | MSK<br>WDG<br>1L | MSK<br>WDG<br>0H | MSK<br>WDG<br>0L |
| W     |    |    |    |    |                  |                  |                  |                  |                  |                  |                  |                  |                  |                  |                  |                  |
| Reset | 0  | 0  | 0  | 0  | 0                | 0                | 0                | 0                | 0                | 0                | 0                | 0                | 0                | 0                | 0                | 0                |

Figure 28-23. ADC\_0 Watchdog Threshold Interrupt Mask Register (WTIMR)

**Table 28-21. ADC\_0 WTIMR field descriptions**

| Field  | Description  |
|--------|--|
| 2x + 1 | MSKWDGxH [x = 0..5]<br>This corresponds to the mask bit for the interrupt generated on the converted value being higher than the programmed higher threshold. When set the interrupt is enabled. |
| 2x     | MSKWDGxL [x = 0..5]his corresponds to the mask bit for the interrupt generated on the converted value being lower than the programmed lower threshold. When set the interrupt is enabled.        |

For ADC\_1 (12-bit)

Address: Base + 0x0034

Access: User read/write

|       |    |    |    |    |    |    |    |    |    |    |                  |                  |                  |                  |                  |                  |
|-------|----|----|----|----|----|----|----|----|----|----|------------------|------------------|------------------|------------------|------------------|------------------|
|       | 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10               | 11               | 12               | 13               | 14               | 15               |
| R     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0                | 0                | 0                | 0                | 0                | 0                |
| W     |    |    |    |    |    |    |    |    |    |    |                  |                  |                  |                  |                  |                  |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0                | 0                | 0                | 0                | 0                | 0                |
|       | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26               | 27               | 28               | 29               | 30               | 31               |
| R     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | MSK<br>WDG<br>2H | MSK<br>WDG<br>2L | MSK<br>WDG<br>1H | MSK<br>WDG<br>1L | MSK<br>WDG<br>0H | MSK<br>WDG<br>0L |
| W     |    |    |    |    |    |    |    |    |    |    |                  |                  |                  |                  |                  |                  |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0                | 0                | 0                | 0                | 0                | 0                |

**Figure 28-24. ADC\_1 Watchdog Threshold Interrupt Mask Register (WTIMR)**

**Table 28-22. ADC\_1 WTIMR field descriptions**

| Field    | Description  |
|----------|--|
| MSKWDGxH | This corresponds to the mask bit for the interrupt generated on the converted value being higher than the programmed higher threshold (for [x = 0..2]). When set the interrupt is enabled. |
| MSKWDGxL | This corresponds to the mask bit for the interrupt generated on the converted value being lower than the programmed lower threshold (for [x = 0..2]). When set the interrupt is enabled.   |

## 28.4.4 DMA registers

### 28.4.4.1 DMA Enable Register (DMAE)

The DMA Enable (DMAE) register sets up the DMA for use with the ADC.

Address: Base + 0x0040

Access: User read/write

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       |    |    |    |    |    |    |    |    |    |    |    |    |    |    |      |       |
|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|-------|
|       | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30   | 31    |
| R     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | DCLR | DMAEN |
| W     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |      |       |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0    | 0     |

Figure 28-25. DMA Enable Register (DMAE)

Table 28-23. DMAE field descriptions

| Field | Description  |
|-------|--|
| DCLR  | DMA clear sequence enable<br>0 DMA request cleared by Acknowledge from DMA controller<br>1 DMA request cleared on read of data registers |
| DMAEN | DMA global enable<br>0 DMA feature disabled<br>1 DMA feature enabled   |

### 28.4.4.2 DMA Channel Select Register (DMAR[0..2])

Table 28-24 shows the available channels.

Table 28-24. DMAR[0..2] register description

| ADC   | Register | Description  |
|-------|----------|--|
| ADC_0 | DMAR0    | Enable bits for channel 0 to 15 (precision channels)             |
| ADC_0 | DMAR1    | Enable bits for channel 32 to 59 (standard channels)             |
| ADC_0 | DMAR2    | Enable bits for channel 64 to 95 (external multiplexed channels) |
| ADC_1 | DMAR0    | Enable bits for channel 0 to 15 (precision channels)             |
| ADC_1 | DMAR1    | Enable bit for channel 32 to 39 (standard channels)              |

Address: Base + 0x0044

Access: User read/write

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|-------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
|       | 16  | 17  | 18  | 19  | 20  | 21  | 22  | 23  | 24  | 25  | 26  | 27  | 28  | 29  | 30  | 31  |
| R     | DMA | DMA | DMA | DMA | DMA | DMA | DMA | DMA | DMA | DMA | DMA | DMA | DMA | DMA | DMA | DMA |
| W     | 15  | 14  | 13  | 12  | 11  | 10  | 9   | 8   | 7   | 6   | 5   | 4   | 3   | 2   | 1   | 0   |
| Reset | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |

Figure 28-26. DMA Channel Select Register 0 (DMAR0)

Address: Base + 0x0048

Access: User read/write

|       |   |   |   |   |     |     |     |     |     |     |     |     |     |     |     |     |
|-------|---|---|---|---|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
|       | 0 | 1 | 2 | 3 | 4   | 5   | 6   | 7   | 8   | 9   | 10  | 11  | 12  | 13  | 14  | 15  |
| R     | 0 | 0 | 0 | 0 | DMA | DMA | DMA | DMA | DMA | DMA | DMA | DMA | DMA | DMA | DMA | DMA |
| W     |   |   |   |   | 59  | 58  | 57  | 56  | 55  | 54  | 53  | 52  | 51  | 50  | 49  | 48  |
| Reset | 0 | 0 | 0 | 0 | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |

|       |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|-------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
|       | 16  | 17  | 18  | 19  | 20  | 21  | 22  | 23  | 24  | 25  | 26  | 27  | 28  | 29  | 30  | 31  |
| R     | DMA | DMA | DMA | DMA | DMA | DMA | DMA | DMA | DMA | DMA | DMA | DMA | DMA | DMA | DMA | DMA |
| W     | 47  | 46  | 43  | 44  | 43  | 42  | 41  | 40  | 39  | 38  | 37  | 36  | 35  | 34  | 33  | 32  |
| Reset | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |

Figure 28-27. DMA Channel Select Register 1 (DMAR1) for ADC\_0

Address: Base + 0x0048

Access: User read/write

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       |    |    |    |    |    |    |    |    |     |     |     |     |     |     |     |     |
|-------|----|----|----|----|----|----|----|----|-----|-----|-----|-----|-----|-----|-----|-----|
|       | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24  | 25  | 26  | 27  | 28  | 29  | 30  | 31  |
| R     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | DMA | DMA | DMA | DMA | DMA | DMA | DMA | DMA |
| W     |    |    |    |    |    |    |    |    | 39  | 38  | 37  | 36  | 35  | 34  | 33  | 32  |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |

Figure 28-28. DMA Channel Select Register 1 (DMAR1) for ADC\_1



Address: Base + 0x004C

Access: User read/write

|       |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|-------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
|       | 0   | 1   | 2   | 3   | 4   | 5   | 6   | 7   | 8   | 9   | 10  | 11  | 12  | 13  | 14  | 15  |
| R     | DMA | DMA | DMA | DMA | DMA | DMA | DMA | DMA | DMA | DMA | DMA | DMA | DMA | DMA | DMA | DMA |
| W     | 95  | 94  | 93  | 92  | 91  | 90  | 89  | 88  | 87  | 86  | 85  | 84  | 83  | 82  | 81  | 80  |
| Reset | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |
|       | 16  | 17  | 18  | 19  | 20  | 21  | 22  | 23  | 24  | 25  | 26  | 27  | 28  | 29  | 30  | 31  |
| R     | DMA | DMA | DMA | DMA | DMA | DMA | DMA | DMA | DMA | DMA | DMA | DMA | DMA | DMA | DMA | DMA |
| W     | 79  | 78  | 77  | 76  | 75  | 74  | 73  | 72  | 71  | 70  | 69  | 68  | 67  | 66  | 65  | 64  |
| Reset | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |

Figure 28-29. DMA Channel Select Register 2 (DMAR2)

Table 28-25. DMARx field descriptions

| Field   | Description  |
|---------|--|
| DMA $n$ | DMA enable<br>When set (DMA $n$ = 1), channel $n$ is enabled to transfer data in DMA mode. |

## 28.4.5 Threshold registers

### 28.4.5.1 Threshold Register (THRHLR)

For ADC\_0 (10-bit): THRHLR[0..5]

Address: Base + 0x0060 (THRHLR0)      Base + 0x006C (THRHLR3)  
 Base + 0x0064 (THRHLR1)      Base + 0x0280 (THRHLR4)      Access: User read/write  
 Base + 0x0068 (THRHLR2)      Base + 0x0284 (THRHLR5)

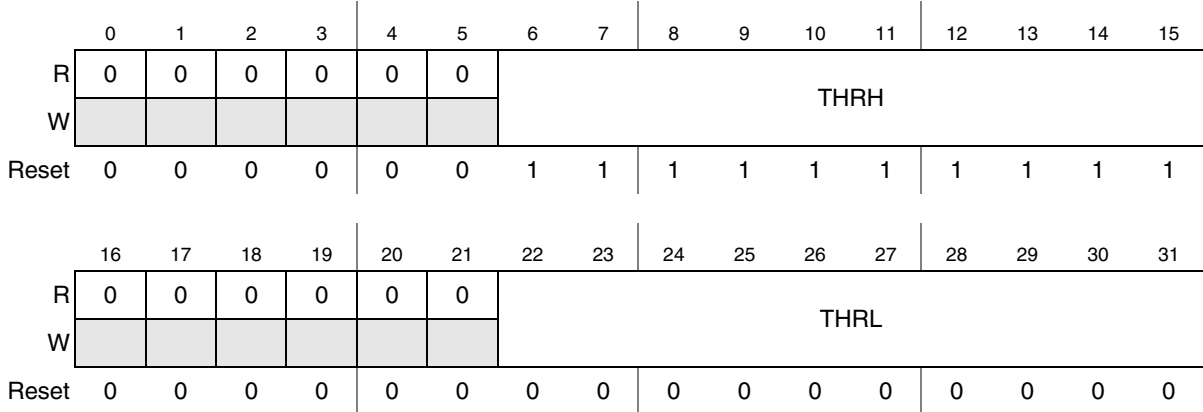


Figure 28-30. ADC\_0 Threshold Register THRHLR[0..5]

Table 28-26. ADC\_0 THRHLR field descriptions

| Field | Description                                 |
|-------|---|
| THRH  | High threshold value for channel <i>n</i> . |
| THRL  | Low threshold value for channel <i>n</i> .  |

For ADC\_1 (12-bit): THRHLR[0..2]

Address: Base + 0x0060 (THRHLR0)      Access: User read/write  
 Base + 0x0064 (THRHLR1)  
 Base + 0x0068 (THRHLR2)

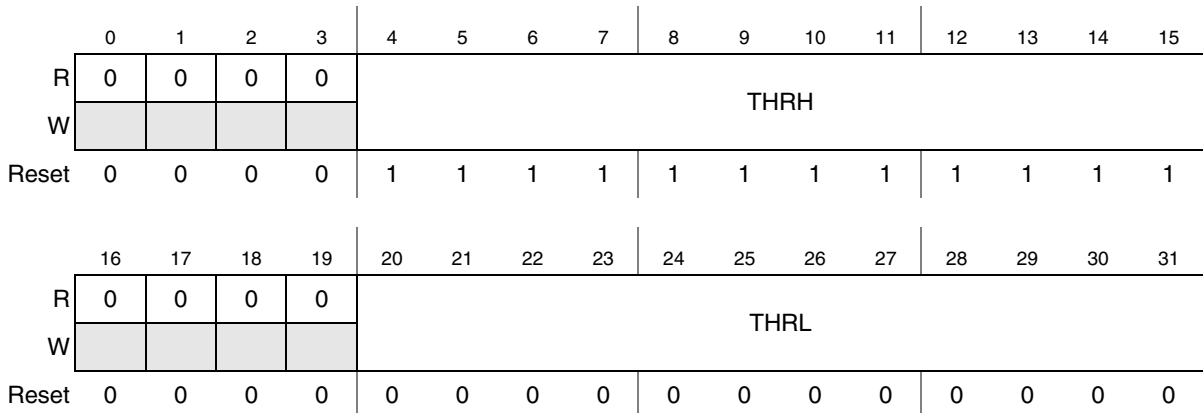


Figure 28-31. ADC\_1 Threshold Register THRHLR[0..2]

Table 28-27. ADC\_1 THRHLR field descriptions

| Field | Description                                 |
|-------|---|
| THRH  | High threshold value for channel <i>n</i> . |
| THRL  | Low threshold value for channel <i>n</i> .  |

## 28.4.6 Presampling registers

### 28.4.6.1 Presampling Control Register (PSCR)

Address: Base + 0x0080

Access: User read/write

|       |    |    |    |    |    |    |    |    |    |         |    |         |    |         |    |          |
|-------|----|----|----|----|----|----|----|----|----|---------|----|---------|----|---------|----|----------|
|       | 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9       | 10 | 11      | 12 | 13      | 14 | 15       |
| R     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0       | 0  | 0       | 0  | 0       | 0  | 0        |
| W     |    |    |    |    |    |    |    |    |    |         |    |         |    |         |    |          |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0       | 0  | 0       | 0  | 0       | 0  | 0        |
|       | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25      | 26 | 27      | 28 | 29      | 30 | 31       |
| R     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | PREVAL2 |    | PREVAL1 |    | PREVAL0 |    | PRE CONV |
| W     |    |    |    |    |    |    |    |    |    |         |    |         |    |         |    |          |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0       | 0  | 0       | 0  | 0       | 0  | 0        |

Figure 28-32. Presampling Control Register (PSCR)

Table 28-28. PSCR field descriptions

| Field   | Description   |
|---------|---|
| PREVAL2 | Internal voltage selection for presampling<br>Selects analog input voltage for presampling from the available two internal voltages (external multiplexed channels). See <a href="#">Table 28-7</a> . |
| PREVAL1 | Internal voltage selection for presampling<br>Selects analog input voltage for presampling from the available two internal voltages (standard channels). See <a href="#">Table 28-7</a> .             |
| PREVAL0 | Internal voltage selection for presampling<br>Selects analog input voltage for presampling from the available two internal voltages (precision channels). See <a href="#">Table 28-7</a> .            |
| PRECONV | Convert presampled value<br>If bit PRECONV is set, presampling is followed by the conversion. Sampling will be bypassed and conversion of presampled data will be done.                               |

### 28.4.6.2 Presampling Register (PSR[0..2])

[Table 28-29](#) shows the available channels.

**Table 28-29. PSR[0..2] register description**

| ADC   | Register | Description   |
|-------|----------|---|
| ADC_0 | PSR0     | Enable bits of presampling for channel 0 to 15 (precision channels)             |
| ADC_0 | PSR1     | Enable bits of presampling for channel 32 to 59 (standard channels)             |
| ADC_0 | PSR2     | Enable bits of presampling for channel 64 to 95 (external multiplexed channels) |
| ADC_1 | PSR0     | Enable bits of presampling for channel 0 to 15 (precision channels)             |
| ADC_1 | PSR1     | Enable bit for channel 32 to 39 (standard channels)                             |

Address: Base + 0x0084

Access: User read/write

|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       | 16   | 17   | 18   | 19   | 20   | 21   | 22   | 23   | 24   | 25   | 26   | 27   | 28   | 29   | 30   | 31   |
|-------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|
| R     | PRES | PRES | PRES | PRES | PRES | PRES | PRES | PRES | PRES | PRES | PRES | PRES | PRES | PRES | PRES | PRES |
| W     | 15   | 14   | 13   | 12   | 11   | 10   | 9    | 8    | 7    | 6    | 5    | 4    | 3    | 2    | 1    | 0    |
| Reset | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    |

**Figure 28-33. Presampling Register 0 (PSR0)**

Address: Base + 0x0088

Access: User read/write

|       | 0 | 1 | 2 | 3 | 4    | 5    | 6    | 7    | 8    | 9    | 10   | 11   | 12   | 13   | 14   | 15   |
|-------|---|---|---|---|------|------|------|------|------|------|------|------|------|------|------|------|
| R     | 0 | 0 | 0 | 0 | PRES | PRES | PRES | PRES | PRES | PRES | PRES | PRES | PRES | PRES | PRES | PRES |
| W     |   |   |   |   | 59   | 58   | 57   | 56   | 55   | 54   | 53   | 52   | 51   | 50   | 49   | 48   |
| Reset | 0 | 0 | 0 | 0 | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    |

|       | 16   | 17   | 18   | 19   | 20   | 21   | 22   | 23   | 24   | 25   | 26   | 27   | 28   | 29   | 30   | 31   |
|-------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|
| R     | PRES | PRES | PRES | PRES | PRES | PRES | PRES | PRES | PRES | PRES | PRES | PRES | PRES | PRES | PRES | PRES |
| W     | 47   | 46   | 43   | 44   | 43   | 42   | 41   | 40   | 39   | 38   | 37   | 36   | 35   | 34   | 33   | 32   |
| Reset | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    |

**Figure 28-34. Presampling Register 1 (PSR1) for ADC\_0**

Address: Base + 0x0088

Access: User read/write

|       |    |    |    |    |    |    |    |    |      |      |      |      |      |      |      |      |
|-------|----|----|----|----|----|----|----|----|------|------|------|------|------|------|------|------|
|       | 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8    | 9    | 10   | 11   | 12   | 13   | 14   | 15   |
| R     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    |
| W     |    |    |    |    |    |    |    |    |      |      |      |      |      |      |      |      |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    |
|       | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24   | 25   | 26   | 27   | 28   | 29   | 30   | 31   |
| R     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | PRES | PRES | PRES | PRES | PRES | PRES | PRES | PRES |
| W     |    |    |    |    |    |    |    |    | 39   | 38   | 37   | 36   | 35   | 34   | 33   | 32   |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    |

Figure 28-35. Presampling Register 1 (PSR1) for ADC\_1

Address: Base + 0x008C

Access: User read/write

|       |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
|-------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|
|       | 0    | 1    | 2    | 3    | 4    | 5    | 6    | 7    | 8    | 9    | 10   | 11   | 12   | 13   | 14   | 15   |
| R     | PRES | PRES | PRES | PRES | PRES | PRES | PRES | PRES | PRES | PRES | PRES | PRES | PRES | PRES | PRES | PRES |
| W     | 95   | 94   | 93   | 92   | 91   | 90   | 89   | 88   | 87   | 86   | 85   | 84   | 83   | 82   | 81   | 80   |
| Reset | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    |
|       | 16   | 17   | 18   | 19   | 20   | 21   | 22   | 23   | 24   | 25   | 26   | 27   | 28   | 29   | 30   | 31   |
| R     | PRES | PRES | PRES | PRES | PRES | PRES | PRES | PRES | PRES | PRES | PRES | PRES | PRES | PRES | PRES | PRES |
| W     | 79   | 78   | 77   | 76   | 75   | 74   | 73   | 72   | 71   | 70   | 69   | 68   | 67   | 66   | 65   | 64   |
| Reset | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    |

Figure 28-36. Presampling Register 2 (PSR2)

Table 28-30. PSR field descriptions

| Field    | Description   |
|----------|---|
| PRES $n$ | Presampling enable<br>When set (PRES $n$ = 1), presampling is enabled for channel $n$ . |

## 28.4.7 Conversion timing registers CTR[0..2]

Table 28-31 shows the available channels.

**Table 28-31. CTR[0..2] register description**

| ADC   | Register | Description   |
|-------|----------|---|
| ADC_0 | CTR0     | Associated to internal precision channels (from 0 to 15)    |
| ADC_0 | CTR1     | Associated to internal standard channels (from 32 to 59)    |
| ADC_0 | CTR2     | Associated to external multiplexed channels (from 64 to 95) |
| ADC_1 | CTR0     | Associated to internal precision channels (from 0 to 15)    |
| ADC_1 | CTR1     | Associated to internal standard channel 32 to 39            |

Address: Base + 0x0094 (CTR0)  
 Base + 0x0098 (CTR1)  
 Base + 0x009C (CTR2)

Access: User read/write

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       |          |    |                       |    |    |        |    |    |         |    |    |    |    |    |    |    |
|-------|----------|----|-----------------------|----|----|--------|----|----|---------|----|----|----|----|----|----|----|
|       | 16       | 17 | 18                    | 19 | 20 | 21     | 22 | 23 | 24      | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| R     | INPLATCH | 0  | OFFSHIFT <sup>1</sup> |    | 0  | INPCMP |    | 0  | INPSAMP |    |    |    |    |    |    |    |
| W     |          |    |                       |    |    |        |    |    |         |    |    |    |    |    |    |    |
| Reset | 0        | 0  | 0                     | 0  | 0  | 0      | 1  | 0  | 0       | 0  | 0  | 0  | 0  | 0  | 1  | 1  |

**Figure 28-37. Conversion timing registers CTR[0..2]**

<sup>1</sup> Available only on CTR0

**Table 28-32. CTR field descriptions**

| Field    | Description   |
|----------|---|
| INPLATCH | Configuration bit for latching phase duration   |
| OFFSHIFT | Configuration for offset shift characteristic<br>00 No shift (that is the transition between codes 000h and 001h) is reached when the $A_{VIN}$ (analog input voltage) is equal to 1 LSB.<br>01 Transition between code 000h and 001h is reached when the $A_{VIN}$ is equal to 1/2 LSB<br>10 Transition between code 00h and 001h is reached when the $A_{VIN}$ is equal to 0<br>11 Not used<br><b>Note:</b> Available only on CTR0. |
| INPCMP   | Configuration bits for comparison phase duration  |
| INPSAMP  | Configuration bits for sampling phase duration  |

## 28.4.8 Mask registers

### 28.4.8.1 Introduction

These registers are used to program which of the 96 input channels must be converted during Normal and Injected conversion.

### 28.4.8.2 Normal Conversion Mask Registers (NCMR[0..2])

Table 28-33 shows the available channels.

Table 28-33. NCMR[0..2] register description

| ADC   | Register | Description   |
|-------|----------|---|
| ADC_0 | NCMR0    | Enable bits of normal sampling for channel 0 to 15 (precision channels)             |
| ADC_0 | NCMR1    | Enable bits of normal sampling for channel 32 to 59 (standard channels)             |
| ADC_0 | NCMR2    | Enable bits of normal sampling for channel 64 to 95 (external multiplexed channels) |
| ADC_1 | NCMR0    | Enable bits of normal sampling for channel 0 to 15 (precision channels)             |
| ADC_1 | NCMR1    | Enable bit of normal sampling channel 32 to 39 (standard channels)                  |

Address: Base + 0x00A4

Access: User read/write

|       |      |      |      |      |      |      |     |     |     |     |     |     |     |     |     |     |
|-------|------|------|------|------|------|------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
|       | 0    | 1    | 2    | 3    | 4    | 5    | 6   | 7   | 8   | 9   | 10  | 11  | 12  | 13  | 14  | 15  |
| R     | 0    | 0    | 0    | 0    | 0    | 0    | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |
| W     |      |      |      |      |      |      |     |     |     |     |     |     |     |     |     |     |
| Reset | 0    | 0    | 0    | 0    | 0    | 0    | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |
|       | 16   | 17   | 18   | 19   | 20   | 21   | 22  | 23  | 24  | 25  | 26  | 27  | 28  | 29  | 30  | 31  |
| R     | CH15 | CH14 | CH13 | CH12 | CH11 | CH10 | CH9 | CH8 | CH7 | CH6 | CH5 | CH4 | CH3 | CH2 | CH1 | CH0 |
| W     |      |      |      |      |      |      |     |     |     |     |     |     |     |     |     |     |
| Reset | 0    | 0    | 0    | 0    | 0    | 0    | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |

Figure 28-38. Normal Conversion Mask Register 0 (NCMR0)

Address: Base + 0x00A8

Access: User read/write

|       |   |   |   |   |      |      |      |      |      |      |      |      |      |      |      |      |
|-------|---|---|---|---|------|------|------|------|------|------|------|------|------|------|------|------|
|       | 0 | 1 | 2 | 3 | 4    | 5    | 6    | 7    | 8    | 9    | 10   | 11   | 12   | 13   | 14   | 15   |
| R     | 0 | 0 | 0 | 0 | CH59 | CH58 | CH57 | CH56 | CH55 | CH54 | CH53 | CH52 | CH51 | CH50 | CH49 | CH48 |
| W     |   |   |   |   |      |      |      |      |      |      |      |      |      |      |      |      |
| Reset | 0 | 0 | 0 | 0 | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    |

|       |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
|-------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|
|       | 16   | 17   | 18   | 19   | 20   | 21   | 22   | 23   | 24   | 25   | 26   | 27   | 28   | 29   | 30   | 31   |
| R     | CH47 | CH46 | CH45 | CH44 | CH43 | CH42 | CH41 | CH40 | CH39 | CH38 | CH37 | CH36 | CH35 | CH34 | CH33 | CH32 |
| W     |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
| Reset | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    |

Figure 28-39. Normal Conversion Mask Register 1 (NCMR1) for ADC\_0

Address: Base + 0x00A8

Access: User read/write

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       |    |    |    |    |    |    |    |    |      |      |      |      |      |      |      |      |
|-------|----|----|----|----|----|----|----|----|------|------|------|------|------|------|------|------|
|       | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24   | 25   | 26   | 27   | 28   | 29   | 30   | 31   |
| R     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | CH39 | CH38 | CH37 | CH36 | CH35 | CH34 | CH33 | CH32 |
| W     |    |    |    |    |    |    |    |    |      |      |      |      |      |      |      |      |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    |

Figure 28-40. Normal Conversion Mask Register 1 (NCMR1) for ADC\_1

Address: Base + 0x00AC

Access: User read/write

|       |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
|-------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|
|       | 0    | 1    | 2    | 3    | 4    | 5    | 6    | 7    | 8    | 9    | 10   | 11   | 12   | 13   | 14   | 15   |
| R     | CH95 | CH94 | CH93 | CH92 | CH91 | CH90 | CH89 | CH88 | CH87 | CH86 | CH85 | CH84 | CH83 | CH82 | CH81 | CH80 |
| W     |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
| Reset | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    |

|       |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
|-------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|
|       | 16   | 17   | 18   | 19   | 20   | 21   | 22   | 23   | 24   | 25   | 26   | 27   | 28   | 29   | 30   | 31   |
| R     | CH79 | CH78 | CH77 | CH76 | CH75 | CH74 | CH73 | CH72 | CH71 | CH70 | CH69 | CH68 | CH67 | CH66 | CH65 | CH64 |
| W     |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
| Reset | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    |

Figure 28-41. Normal Conversion Mask Register 2 (NCMR2)



Table 28-34. NCMR field descriptions

| Field  | Description  |
|--------|--|
| CH $n$ | Sampling enable<br>When set, sampling is enabled for channel $n$ . |

**NOTE**

The implicit channel conversion priority in the case in which all channels are selected is the following: ADC $n$ \_P[0: $x$ ], ADC $n$ \_S[0: $y$ ], ADC $n$ \_X[0: $z$ ].

The channels always start with 0, the lowest index.

### 28.4.8.3 Injected Conversion Mask Registers (JCMR[0..2])

Table 28-35 shows the available channels.

**Table 28-35. JCMR[0..2] register description**

| ADC   | Register | Description   |
|-------|----------|---|
| ADC_0 | JCMR0    | Enable bits of injected sampling for channel 0 to 15 (precision channels)             |
| ADC_0 | JCMR1    | Enable bits of injected sampling for channel 32 to 59 (standard channels)             |
| ADC_0 | JCMR2    | Enable bits of injected sampling for channel 64 to 95 (external multiplexed channels) |
| ADC_1 | JCMR0    | Enable bits of injected sampling for channel 0 to 15 (precision channels)             |
| ADC_1 | JCMR1    | Enable bit of injected sampling for channel 32 to 39 (standard channels)              |

Address: Base + 0x00B4

Access: User read/write

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       |      |      |      |      |      |      |     |     |     |     |     |     |     |     |     |     |
|-------|------|------|------|------|------|------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
|       | 16   | 17   | 18   | 19   | 20   | 21   | 22  | 23  | 24  | 25  | 26  | 27  | 28  | 29  | 30  | 31  |
| R     | CH15 | CH14 | CH13 | CH12 | CH11 | CH10 | CH9 | CH8 | CH7 | CH6 | CH5 | CH4 | CH3 | CH2 | CH1 | CH0 |
| W     |      |      |      |      |      |      |     |     |     |     |     |     |     |     |     |     |
| Reset | 0    | 0    | 0    | 0    | 0    | 0    | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |

**Figure 28-42. Injected Conversion Mask Register 0 (JCMR0)**

Address: Base + 0x00B8

Access: User read/write

|       |   |   |   |   |      |      |      |      |      |      |      |      |      |      |      |      |
|-------|---|---|---|---|------|------|------|------|------|------|------|------|------|------|------|------|
|       | 0 | 1 | 2 | 3 | 4    | 5    | 6    | 7    | 8    | 9    | 10   | 11   | 12   | 13   | 14   | 15   |
| R     | 0 | 0 | 0 | 0 | CH59 | CH58 | CH57 | CH56 | CH55 | CH54 | CH53 | CH52 | CH51 | CH50 | CH49 | CH48 |
| W     |   |   |   |   |      |      |      |      |      |      |      |      |      |      |      |      |
| Reset | 0 | 0 | 0 | 0 | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    |

|       |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
|-------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|
|       | 16   | 17   | 18   | 19   | 20   | 21   | 22   | 23   | 24   | 25   | 26   | 27   | 28   | 29   | 30   | 31   |
| R     | CH47 | CH46 | CH45 | CH44 | CH43 | CH42 | CH41 | CH40 | CH39 | CH38 | CH37 | CH36 | CH35 | CH34 | CH33 | CH32 |
| W     |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
| Reset | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    |

**Figure 28-43. Injected Conversion Mask Register 1 (JCMR1) for ADC\_0**

Address: Base + 0x00B8

Access: User read/write

|       |    |    |    |    |    |    |    |    |      |      |      |      |      |      |      |      |
|-------|----|----|----|----|----|----|----|----|------|------|------|------|------|------|------|------|
|       | 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8    | 9    | 10   | 11   | 12   | 13   | 14   | 15   |
| R     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    |
| W     |    |    |    |    |    |    |    |    |      |      |      |      |      |      |      |      |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    |
|       | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24   | 25   | 26   | 27   | 28   | 29   | 30   | 31   |
| R     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | CH39 | CH38 | CH37 | CH36 | CH35 | CH34 | CH33 | CH32 |
| W     |    |    |    |    |    |    |    |    |      |      |      |      |      |      |      |      |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    |

Figure 28-44. Injected Conversion Mask Register 1 (JCMR1) for ADC\_1

Address: Base + 0x00BC

Access: User read/write

|       |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
|-------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|
|       | 0    | 1    | 2    | 3    | 4    | 5    | 6    | 7    | 8    | 9    | 10   | 11   | 12   | 13   | 14   | 15   |
| R     | CH95 | CH94 | CH93 | CH92 | CH91 | CH90 | CH89 | CH88 | CH87 | CH86 | CH85 | CH84 | CH83 | CH82 | CH81 | CH80 |
| W     |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
| Reset | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    |
|       | 16   | 17   | 18   | 19   | 20   | 21   | 22   | 23   | 24   | 25   | 26   | 27   | 28   | 29   | 30   | 31   |
| R     | CH79 | CH78 | CH77 | CH76 | CH75 | CH74 | CH73 | CH72 | CH71 | CH70 | CH69 | CH68 | CH67 | CH66 | CH65 | CH64 |
| W     |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
| Reset | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    |

Figure 28-45. Injected Conversion Mask Register 2 (JCMR2)

Table 28-36. JCMR field descriptions

| Field  | Description  |
|--------|--|
| CH $n$ | Sampling enable<br>When set, sampling is enabled for channel $n$ . |

## 28.4.9 Delay registers

### 28.4.9.1 Decode Signals Delay Register (DSDR)

The Decode Signals Delay Register (DSDR) is implemented only on ADC\_0.

Address: Base + 0x00C4 Access: User read/write

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       |    |    |    |    |     |    |    |    |    |    |    |    |    |    |    |    |
|-------|----|----|----|----|-----|----|----|----|----|----|----|----|----|----|----|----|
|       | 16 | 17 | 18 | 19 | 20  | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| R     | 0  | 0  | 0  | 0  | DSD |    |    |    |    |    |    |    |    |    |    |    |
| W     |    |    |    |    |     |    |    |    |    |    |    |    |    |    |    |    |
| Reset | 0  | 0  | 0  | 0  | 0   | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |

Figure 28-46. Decode Signals Delay Register (DSDR)

Table 28-37. DSDR field descriptions

| Field | Description  |
|-------|--|
| DSD   | Delay between the external decode signals and the start of the sampling phase<br>It is used to take into account the settling time of the external multiplexer.<br>The decode signal delay is calculated as: $DSD \times 1/\text{frequency of ADC clock}$ .<br><b>Note:</b> When $ADC\ clock = Peripheral\ Clock/2$ the DSD has to be incremented by 2 to see an additional ADC clock cycle delay on the decode signal.<br>For example:<br>DSD = 0; 0 ADC clock cycle delay<br>DSD = 2; 1 ADC clock cycle delay<br>DSD = 4; 2 ADC clock cycles delay |

### 28.4.9.2 Power-down Exit Delay Register (PDEDR)

Address: Base + 0x00C8 Access: User read/write

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       |    |    |    |    |    |    |    |    |      |    |    |    |    |    |    |    |
|-------|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|
|       | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24   | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| R     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | PDED |    |    |    |    |    |    |    |
| W     |    |    |    |    |    |    |    |    |      |    |    |    |    |    |    |    |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0    | 0  | 0  | 0  | 0  | 0  | 0  | 0  |

Figure 28-47. Power-down Exit Delay Register (PDEDR)

**Table 28-38. PDED field descriptions**

| Field | Description  |
|-------|--|
| PDED  | Delay between the power-down bit reset and the start of conversion. The delay is to allow time for the ADC power supply to settle before commencing conversions.<br>The power down delay is calculated as: $PDED \times 1/\text{frequency of ADC clock}$ . |

## 28.4.10 Data registers

### 28.4.10.1 Introduction

ADC conversion results are stored in data registers. There is one register per channel.

### 28.4.10.2 Channel Data Register (CDR[0..95])

Table 28-39 shows the available channels.

| ADC   | Registers    | Description   |
|-------|--------------|---|
| ADC_0 | CDR0[0..15]  | Enable bits of injected sampling for channel 0 to 15 (precision channels)             |
| ADC_0 | CDR0[32..59] | Enable bits of injected sampling for channel 32 to 59 (standard channels)             |
| ADC_0 | CDR0[64..95] | Enable bits of injected sampling for channel 64 to 95 (external multiplexed channels) |
| ADC_1 | CDR1[0..15]  | Enable bits of injected sampling for channel 0 to 15 (precision channels)             |
| ADC_1 | CDR1[32..59] | Enable bits of injected sampling for channel 32 to 39 (standard channels)             |

**Table 28-39. CDR[0..95] register description**

Each data register also gives information regarding the corresponding result as described below.

For ADC\_0 10-bit:

Address: See [Table 28-9](#)

Access: User read-only

|       |                                 |    |    |    |    |    |                                 |    |    |    |    |    |           |           |        |    |  |  |  |
|-------|---------------------------------|----|----|----|----|----|---------------------------------|----|----|----|----|----|-----------|-----------|--------|----|--|--|--|
|       | 0                               | 1  | 2  | 3  | 4  | 5  | 6                               | 7  | 8  | 9  | 10 | 11 | 12        | 13        | 14     | 15 |  |  |  |
| R     | 0                               | 0  | 0  | 0  | 0  | 0  | 0                               | 0  | 0  | 0  | 0  | 0  | VA<br>LID | OVER<br>W | RESULT |    |  |  |  |
| W     | [Greyed out]                    |    |    |    |    |    |                                 |    |    |    |    |    |           |           |        |    |  |  |  |
| Reset | 0                               | 0  | 0  | 0  | 0  | 0  | 0                               | 0  | 0  | 0  | 0  | 0  | 0         | 0         | 0      | 0  |  |  |  |
|       | 16                              | 17 | 18 | 19 | 20 | 21 | 22                              | 23 | 24 | 25 | 26 | 27 | 28        | 29        | 30     | 31 |  |  |  |
| R     | 0                               | 0  | 0  | 0  | 0  | 0  | CDATA[0:9]<br>(MCR[WLSIDE] = 0) |    |    |    |    |    |           |           |        |    |  |  |  |
| W     | [Greyed out]                    |    |    |    |    |    |                                 |    |    |    |    |    |           |           |        |    |  |  |  |
| Reset | 0                               | 0  | 0  | 0  | 0  | 0  | 0                               | 0  | 0  | 0  | 0  | 0  | 0         | 0         | 0      | 0  |  |  |  |
|       | 16                              | 17 | 18 | 19 | 20 | 21 | 22                              | 23 | 24 | 25 | 26 | 27 | 28        | 29        | 30     | 31 |  |  |  |
| R     | CDATA[0:9]<br>(MCR[WLSIDE] = 1) |    |    |    |    |    |                                 |    |    |    |    | 0  | 0         | 0         | 0      |    |  |  |  |
| W     | [Greyed out]                    |    |    |    |    |    |                                 |    |    |    |    |    |           |           |        |    |  |  |  |
| Reset | 0                               | 0  | 0  | 0  | 0  | 0  | 0                               | 0  | 0  | 0  | 0  | 0  | 0         | 0         | 0      | 0  |  |  |  |

Figure 28-48. Channel Data Register (CDR[0..95])

For ADC\_1 12-bit:

Address: See [Table 28-10](#)

Access: User read-only

|       |                                  |    |    |    |                                  |    |    |    |    |    |    |    |           |           |                 |    |  |
|-------|----------------------------------|----|----|----|----------------------------------|----|----|----|----|----|----|----|-----------|-----------|-----------------|----|--|
|       | 0                                | 1  | 2  | 3  | 4                                | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12        | 13        | 14              | 15 |  |
| R     | 0                                | 0  | 0  | 0  | 0                                | 0  | 0  | 0  | 0  | 0  | 0  | 0  | VA<br>LID | OVER<br>W | RESULT<br>[0:1] |    |  |
| W     | [Greyed out]                     |    |    |    |                                  |    |    |    |    |    |    |    |           |           |                 |    |  |
| Reset | 0                                | 0  | 0  | 0  | 0                                | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0         | 0         | 0               | 0  |  |
|       | 16                               | 17 | 18 | 19 | 20                               | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28        | 29        | 30              | 31 |  |
| R     | 0                                | 0  | 0  | 0  | CDATA[0:11]<br>(MCR[WLSIDE] = 0) |    |    |    |    |    |    |    |           |           |                 |    |  |
| W     | [Greyed out]                     |    |    |    |                                  |    |    |    |    |    |    |    |           |           |                 |    |  |
| Reset | 0                                | 0  | 0  | 0  | 0                                | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0         | 0         | 0               | 0  |  |
|       | 16                               | 17 | 18 | 19 | 20                               | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28        | 29        | 30              | 31 |  |
| R     | CDATA[0:11]<br>(MCR[WLSIDE] = 1) |    |    |    |                                  |    |    |    |    |    |    | 0  | 0         | 0         | 0               |    |  |
| W     | [Greyed out]                     |    |    |    |                                  |    |    |    |    |    |    |    |           |           |                 |    |  |
| Reset | 0                                | 0  | 0  | 0  | 0                                | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0         | 0         | 0               | 0  |  |

Figure 28-49. Channel Data Register (CDR[0..95])

Table 28-40. CDR field descriptions

| Field  | Description  |
|--------|--|
| VALID  | Used to notify when the data is valid (a new value has been written). It is automatically cleared when data is read.   |
| OVERW  | <p>Overwrite data</p> <p>This bit signals that the previous converted data has been overwritten by a new conversion. This functionality depends on the value of MCR[OWREN]:</p> <ul style="list-style-type: none"> <li>– When OWREN = 0, then OVERW is frozen to 0 and CDATA field is protected against being overwritten until being read.</li> <li>– When OWREN = 1, then OVERW flags the CDATA field overwrite status.</li> </ul> <p>0 Converted data has not been overwritten<br/>1 Previous converted data has been overwritten before having been read</p> |
| RESULT | <p>This bit reflects the mode of conversion for the corresponding channel.</p> <p>00 Data is a result of Normal conversion mode<br/>01 Data is a result of Injected conversion mode<br/>10 Data is a result of CTU conversion mode<br/>11 Reserved</p>   |
| CDATA  | Channel 0-95 converted data. Depending on the value of the MCR[WLSIDE] bit, the position of this field can be changed as shown in <a href="#">Figure 28-48</a> and <a href="#">Figure 28-48</a> .  |

## 28.4.11 Watchdog register

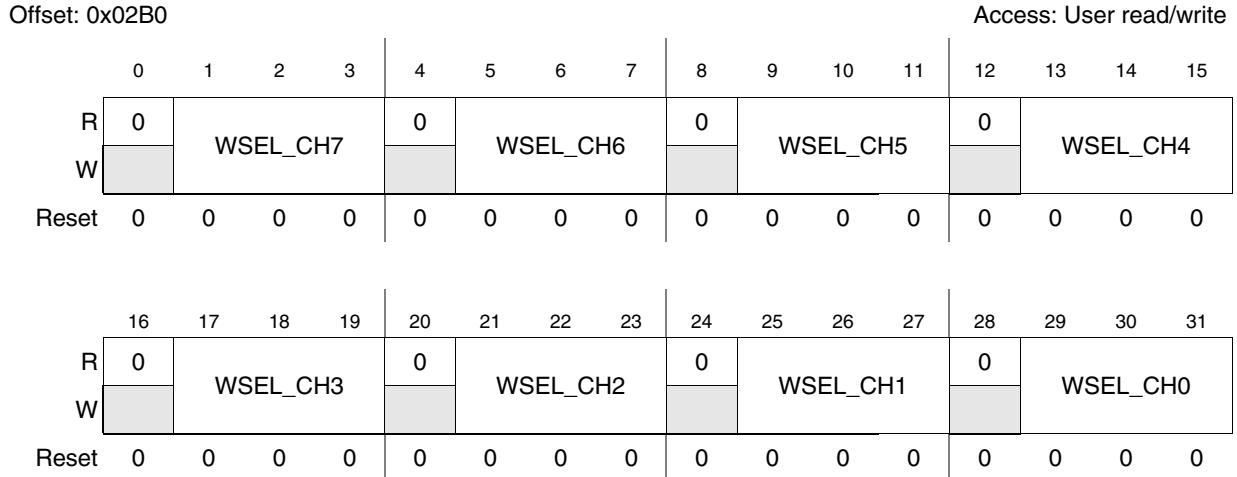
### 28.4.11.1 Channel Watchdog Select Register (CWSELR[0..11])

In Channel Watchdog Select Register (CWSELR[0..11]) the field WSEL\_CH $n$ [3:0] = selects the threshold register that provides the values to be used for upper and lower bounds for channel  $n$ .

[Table 28-41](#) shows the available channels.

Table 28-41. CWSELR[0..11] register description

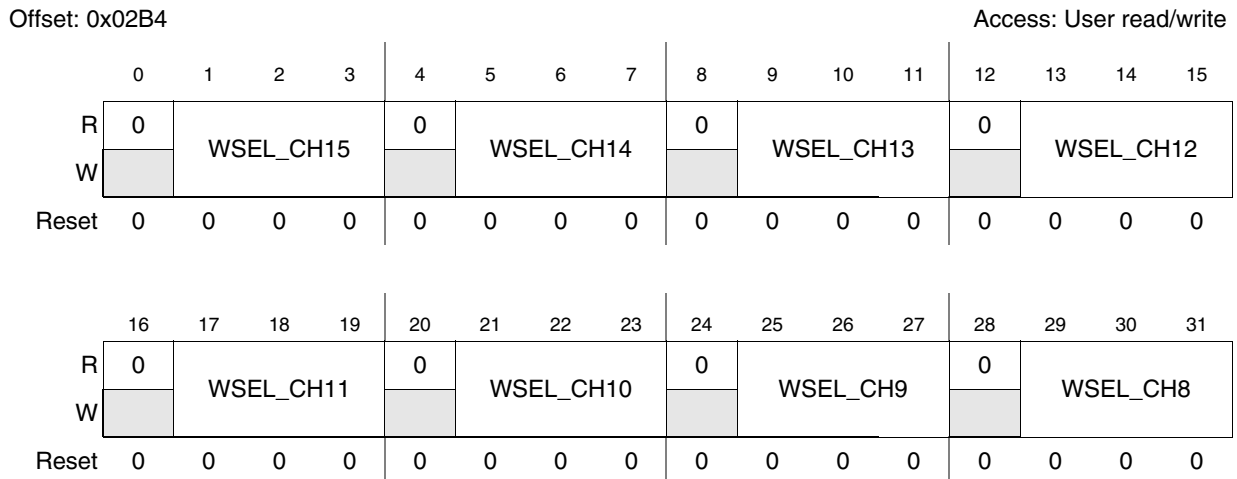
| ADC   | Register      | Description   |
|-------|---------------|---|
| ADC_0 | CWSELR[0..1]  | Channel watchdog select register for channel 0 to 15 (precision channels)             |
| ADC_0 | CWSELR[2..3]  | Not implemented   |
| ADC_0 | CWSELR[4..7]  | Channel watchdog select register for channel 32 to 59 (standard channels)             |
| ADC_0 | CWSELR[8..11] | Channel watchdog select register for channel 64 to 95 (external multiplexed channels) |
| ADC_1 | CWSELR[0..1]  | Channel watchdog select register for channel 0 to 15 (precision channels)             |
| ADC_1 | CWSELR[2..3]  | Not implemented   |
| ADC_1 | CWSELR[4]     | Channel watchdog select register for channel 32 to 39 (standard channels)             |
| ADC_1 | CWSELR[5..11] | Not implemented   |



**Figure 28-50. Channel Watchdog Select Register 0 (CWSELR0) – ADC\_0**

**Table 28-42. CWSELR field descriptions – ADC\_0**

| Field       | Description   |
|-------------|---|
| WSEL_CH $n$ | Channel Watchdog select for channel $n$<br>000 THRHLR0 register is selected<br>001 THRHLR1 register is selected<br>010 THRHLR2 register is selected<br>011 THRHLR3 register is selected<br>100 THRHLR4 register is selected<br>101 THRHLR5 register is selected<br>110 Reserved<br>111 Reserved |



**Figure 28-51. Channel Watchdog Select Register 1 (CWSELR1) – ADC\_0**



Table 28-43. CWSELR1 field descriptions – ADC\_0

| Field    | Description      |
|----------|------------------|
| WSEL_CHn | See Table 28-42. |

Offset: 0x02C0

Access: User read/write

|       | 0 | 1         | 2 | 3 | 4 | 5         | 6 | 7 | 8 | 9         | 10 | 11 | 12 | 13        | 14 | 15 |
|-------|---|-----------|---|---|---|-----------|---|---|---|-----------|----|----|----|-----------|----|----|
| R     | 0 | WSEL_CH39 |   |   | 0 | WSEL_CH38 |   |   | 0 | WSEL_CH37 |    |    | 0  | WSEL_CH36 |    |    |
| W     |   |           |   |   |   |           |   |   |   |           |    |    |    |           |    |    |
| Reset | 0 | 0         | 0 | 0 | 0 | 0         | 0 | 0 | 0 | 0         | 0  | 0  | 0  | 0         | 0  | 0  |

|       | 16 | 17        | 18 | 19 | 20 | 21        | 22 | 23 | 24 | 25        | 26 | 27 | 28 | 29        | 30 | 31 |
|-------|----|-----------|----|----|----|-----------|----|----|----|-----------|----|----|----|-----------|----|----|
| R     | 0  | WSEL_CH35 |    |    | 0  | WSEL_CH34 |    |    | 0  | WSEL_CH33 |    |    | 0  | WSEL_CH32 |    |    |
| W     |    |           |    |    |    |           |    |    |    |           |    |    |    |           |    |    |
| Reset | 0  | 0         | 0  | 0  | 0  | 0         | 0  | 0  | 0  | 0         | 0  | 0  | 0  | 0         | 0  | 0  |

Figure 28-52. Channel Watchdog Select Register 4 (CWSELR4) – ADC\_0

Table 28-44. CWSELR4 field descriptions – ADC\_0

| Field    | Description      |
|----------|------------------|
| WSEL_CHn | See Table 28-42. |

Offset: 0x02C4

Access: User read/write

|       | 0 | 1         | 2 | 3 | 4 | 5         | 6 | 7 | 8 | 9         | 10 | 11 | 12 | 13        | 14 | 15 |
|-------|---|-----------|---|---|---|-----------|---|---|---|-----------|----|----|----|-----------|----|----|
| R     | 0 | WSEL_CH47 |   |   | 0 | WSEL_CH46 |   |   | 0 | WSEL_CH45 |    |    | 0  | WSEL_CH44 |    |    |
| W     |   |           |   |   |   |           |   |   |   |           |    |    |    |           |    |    |
| Reset | 0 | 0         | 0 | 0 | 0 | 0         | 0 | 0 | 0 | 0         | 0  | 0  | 0  | 0         | 0  | 0  |

|       | 16 | 17        | 18 | 19 | 20 | 21        | 22 | 23 | 24 | 25        | 26 | 27 | 28 | 29        | 30 | 31 |
|-------|----|-----------|----|----|----|-----------|----|----|----|-----------|----|----|----|-----------|----|----|
| R     | 0  | WSEL_CH43 |    |    | 0  | WSEL_CH42 |    |    | 0  | WSEL_CH41 |    |    | 0  | WSEL_CH40 |    |    |
| W     |    |           |    |    |    |           |    |    |    |           |    |    |    |           |    |    |
| Reset | 0  | 0         | 0  | 0  | 0  | 0         | 0  | 0  | 0  | 0         | 0  | 0  | 0  | 0         | 0  | 0  |

Figure 28-53. Channel Watchdog Select Register 5 (CWSELR5) – ADC\_0

**Table 28-45. CWSELR5 field descriptions – ADC\_0**

| Field    | Description                       |
|----------|-----------------------------------|
| WSEL_CHn | See <a href="#">Table 28-42</a> . |

Offset: 0x02C8

Access: User read/write

|       |   |           |   |   |   |           |   |   |   |           |    |    |    |           |    |    |
|-------|---|-----------|---|---|---|-----------|---|---|---|-----------|----|----|----|-----------|----|----|
|       | 0 | 1         | 2 | 3 | 4 | 5         | 6 | 7 | 8 | 9         | 10 | 11 | 12 | 13        | 14 | 15 |
| R     | 0 | WSEL_CH55 |   |   | 0 | WSEL_CH54 |   |   | 0 | WSEL_CH53 |    |    | 0  | WSEL_CH52 |    |    |
| W     |   |           |   |   |   |           |   |   |   |           |    |    |    |           |    |    |
| Reset | 0 | 0         | 0 | 0 | 0 | 0         | 0 | 0 | 0 | 0         | 0  | 0  | 0  | 0         | 0  | 0  |

|       |    |           |    |    |    |           |    |    |    |           |    |    |    |           |    |    |
|-------|----|-----------|----|----|----|-----------|----|----|----|-----------|----|----|----|-----------|----|----|
|       | 16 | 17        | 18 | 19 | 20 | 21        | 22 | 23 | 24 | 25        | 26 | 27 | 28 | 29        | 30 | 31 |
| R     | 0  | WSEL_CH51 |    |    | 0  | WSEL_CH50 |    |    | 0  | WSEL_CH49 |    |    | 0  | WSEL_CH48 |    |    |
| W     |    |           |    |    |    |           |    |    |    |           |    |    |    |           |    |    |
| Reset | 0  | 0         | 0  | 0  | 0  | 0         | 0  | 0  | 0  | 0         | 0  | 0  | 0  | 0         | 0  | 0  |

**Figure 28-54. Channel Watchdog Select Register 6 (CWSELR6) – ADC\_0**

**Table 28-46. CWSELR6 field descriptions – ADC\_0**

| Field    | Description                       |
|----------|-----------------------------------|
| WSEL_CHn | See <a href="#">Table 28-42</a> . |

Offset: 0x02CC

Access: User read/write

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       |    |           |    |    |    |           |    |    |    |           |    |    |    |           |    |    |
|-------|----|-----------|----|----|----|-----------|----|----|----|-----------|----|----|----|-----------|----|----|
|       | 16 | 17        | 18 | 19 | 20 | 21        | 22 | 23 | 24 | 25        | 26 | 27 | 28 | 29        | 30 | 31 |
| R     | 0  | WSEL_CH59 |    |    | 0  | WSEL_CH58 |    |    | 0  | WSEL_CH57 |    |    | 0  | WSEL_CH56 |    |    |
| W     |    |           |    |    |    |           |    |    |    |           |    |    |    |           |    |    |
| Reset | 0  | 0         | 0  | 0  | 0  | 0         | 0  | 0  | 0  | 0         | 0  | 0  | 0  | 0         | 0  | 0  |

**Figure 28-55. Channel Watchdog Select Register 7 (CWSELR7) – ADC\_0**

Table 28-47. CWSELR7 field descriptions – ADC\_0

| Field    | Description      |
|----------|------------------|
| WSEL_CHn | See Table 28-42. |

Offset: 0x02D0

Access: User read/write

|       | 0 | 1         | 2 | 3 | 4 | 5         | 6 | 7 | 8 | 9         | 10 | 11 | 12 | 13        | 14 | 15 |
|-------|---|-----------|---|---|---|-----------|---|---|---|-----------|----|----|----|-----------|----|----|
| R     | 0 | WSEL_CH71 |   |   | 0 | WSEL_CH70 |   |   | 0 | WSEL_CH69 |    |    | 0  | WSEL_CH68 |    |    |
| W     |   |           |   |   |   |           |   |   |   |           |    |    |    |           |    |    |
| Reset | 0 | 0         | 0 | 0 | 0 | 0         | 0 | 0 | 0 | 0         | 0  | 0  | 0  | 0         | 0  | 0  |

|       | 16 | 17        | 18 | 19 | 20 | 21        | 22 | 23 | 24 | 25        | 26 | 27 | 28 | 29        | 30 | 31 |
|-------|----|-----------|----|----|----|-----------|----|----|----|-----------|----|----|----|-----------|----|----|
| R     | 0  | WSEL_CH67 |    |    | 0  | WSEL_CH66 |    |    | 0  | WSEL_CH65 |    |    | 0  | WSEL_CH64 |    |    |
| W     |    |           |    |    |    |           |    |    |    |           |    |    |    |           |    |    |
| Reset | 0  | 0         | 0  | 0  | 0  | 0         | 0  | 0  | 0  | 0         | 0  | 0  | 0  | 0         | 0  | 0  |

Figure 28-56. Channel Watchdog Select Register 8 (CWSELR8) – ADC\_0

Table 28-48. CWSELR8 field descriptions – ADC\_0

| Field    | Description      |
|----------|------------------|
| WSEL_CHn | See Table 28-42. |

Offset: 0x02D4

Access: User read/write

|       | 0 | 1         | 2 | 3 | 4 | 5         | 6 | 7 | 8 | 9         | 10 | 11 | 12 | 13        | 14 | 15 |
|-------|---|-----------|---|---|---|-----------|---|---|---|-----------|----|----|----|-----------|----|----|
| R     | 0 | WSEL_CH79 |   |   | 0 | WSEL_CH78 |   |   | 0 | WSEL_CH77 |    |    | 0  | WSEL_CH76 |    |    |
| W     |   |           |   |   |   |           |   |   |   |           |    |    |    |           |    |    |
| Reset | 0 | 0         | 0 | 0 | 0 | 0         | 0 | 0 | 0 | 0         | 0  | 0  | 0  | 0         | 0  | 0  |

|       | 16 | 17        | 18 | 19 | 20 | 21        | 22 | 23 | 24 | 25        | 26 | 27 | 28 | 29        | 30 | 31 |
|-------|----|-----------|----|----|----|-----------|----|----|----|-----------|----|----|----|-----------|----|----|
| R     | 0  | WSEL_CH75 |    |    | 0  | WSEL_CH74 |    |    | 0  | WSEL_CH73 |    |    | 0  | WSEL_CH72 |    |    |
| W     |    |           |    |    |    |           |    |    |    |           |    |    |    |           |    |    |
| Reset | 0  | 0         | 0  | 0  | 0  | 0         | 0  | 0  | 0  | 0         | 0  | 0  | 0  | 0         | 0  | 0  |

Figure 28-57. Channel Watchdog Select Register 9 (CWSELR9) – ADC\_0

**Table 28-49. CWSELR9 field descriptions – ADC\_0**

| Field    | Description                       |
|----------|-----------------------------------|
| WSEL_CHn | See <a href="#">Table 28-42</a> . |

Offset: 0x02D8

Access: User read/write

|       |   |           |   |   |   |           |   |   |   |           |    |    |    |           |    |    |
|-------|---|-----------|---|---|---|-----------|---|---|---|-----------|----|----|----|-----------|----|----|
|       | 0 | 1         | 2 | 3 | 4 | 5         | 6 | 7 | 8 | 9         | 10 | 11 | 12 | 13        | 14 | 15 |
| R     | 0 | WSEL_CH87 |   |   | 0 | WSEL_CH86 |   |   | 0 | WSEL_CH85 |    |    | 0  | WSEL_CH84 |    |    |
| W     |   |           |   |   |   |           |   |   |   |           |    |    |    |           |    |    |
| Reset | 0 | 0         | 0 | 0 | 0 | 0         | 0 | 0 | 0 | 0         | 0  | 0  | 0  | 0         | 0  | 0  |

|       |    |           |    |    |    |           |    |    |    |           |    |    |    |           |    |    |
|-------|----|-----------|----|----|----|-----------|----|----|----|-----------|----|----|----|-----------|----|----|
|       | 16 | 17        | 18 | 19 | 20 | 21        | 22 | 23 | 24 | 25        | 26 | 27 | 28 | 29        | 30 | 31 |
| R     | 0  | WSEL_CH83 |    |    | 0  | WSEL_CH82 |    |    | 0  | WSEL_CH81 |    |    | 0  | WSEL_CH80 |    |    |
| W     |    |           |    |    |    |           |    |    |    |           |    |    |    |           |    |    |
| Reset | 0  | 0         | 0  | 0  | 0  | 0         | 0  | 0  | 0  | 0         | 0  | 0  | 0  | 0         | 0  | 0  |

**Figure 28-58. Channel Watchdog Select Register 10 (CWSELR10) – ADC\_0**

**Table 28-50. CWSELR10 field descriptions – ADC\_0**

| Field    | Description                       |
|----------|-----------------------------------|
| WSEL_CHn | See <a href="#">Table 28-42</a> . |

Offset: 0x02DC

Access: User read/write

|       |   |           |   |   |   |           |   |   |   |           |    |    |    |           |    |    |
|-------|---|-----------|---|---|---|-----------|---|---|---|-----------|----|----|----|-----------|----|----|
|       | 0 | 1         | 2 | 3 | 4 | 5         | 6 | 7 | 8 | 9         | 10 | 11 | 12 | 13        | 14 | 15 |
| R     | 0 | WSEL_CH95 |   |   | 0 | WSEL_CH94 |   |   | 0 | WSEL_CH93 |    |    | 0  | WSEL_CH92 |    |    |
| W     |   |           |   |   |   |           |   |   |   |           |    |    |    |           |    |    |
| Reset | 0 | 0         | 0 | 0 | 0 | 0         | 0 | 0 | 0 | 0         | 0  | 0  | 0  | 0         | 0  | 0  |

|       |    |           |    |    |    |           |    |    |    |           |    |    |    |           |    |    |
|-------|----|-----------|----|----|----|-----------|----|----|----|-----------|----|----|----|-----------|----|----|
|       | 16 | 17        | 18 | 19 | 20 | 21        | 22 | 23 | 24 | 25        | 26 | 27 | 28 | 29        | 30 | 31 |
| R     | 0  | WSEL_CH91 |    |    | 0  | WSEL_CH90 |    |    | 0  | WSEL_CH89 |    |    | 0  | WSEL_CH88 |    |    |
| W     |    |           |    |    |    |           |    |    |    |           |    |    |    |           |    |    |
| Reset | 0  | 0         | 0  | 0  | 0  | 0         | 0  | 0  | 0  | 0         | 0  | 0  | 0  | 0         | 0  | 0  |

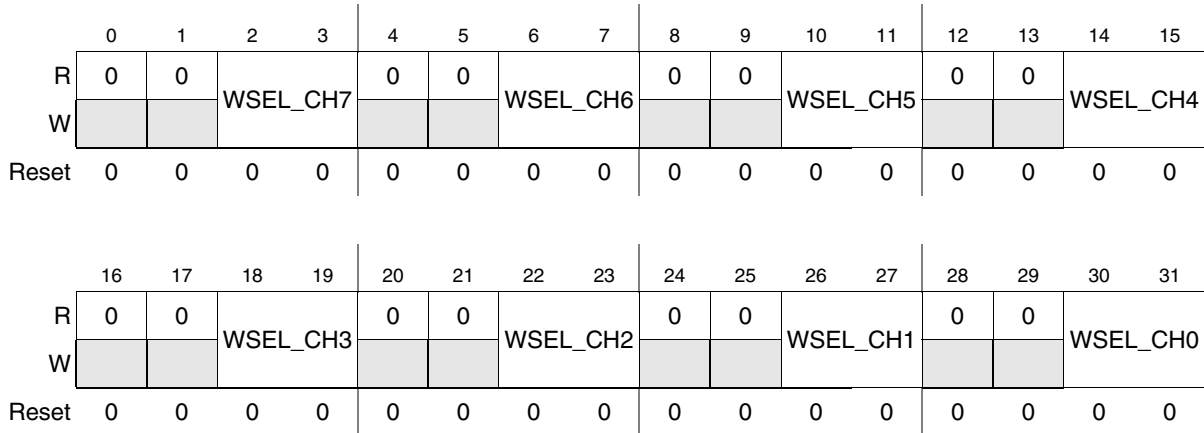
**Figure 28-59. Channel Watchdog Select Register 11 (CWSELR11) – ADC\_0**

**Table 28-51. CWSELR11 field descriptions – ADC\_0**

| Field    | Description                       |
|----------|-----------------------------------|
| WSEL_CHn | See <a href="#">Table 28-42</a> . |

Offset: 0x02B0

Access: User read/write



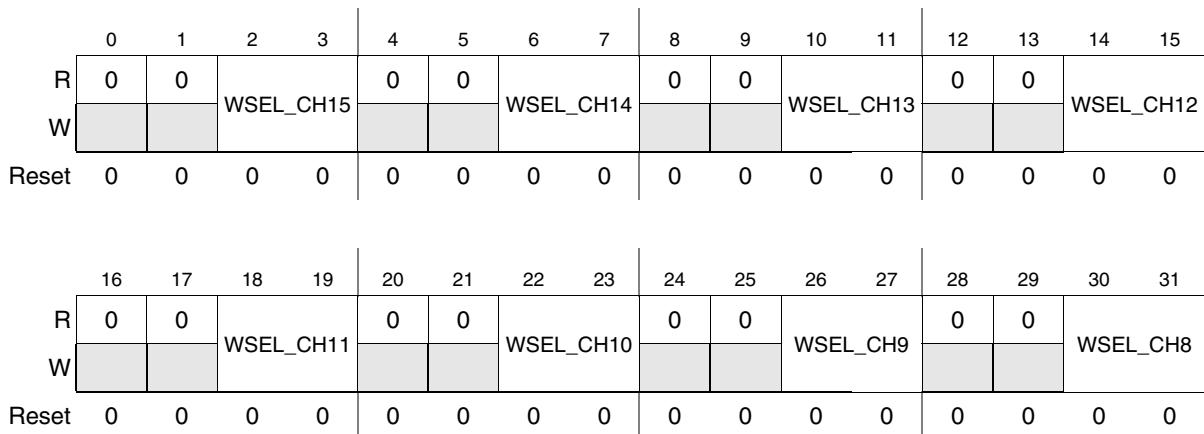
**Figure 28-60. Channel Watchdog Select Register 0 (CWSELR0) – ADC\_1**

**Table 28-52. CWSELR0 field descriptions – ADC\_1**

| Field    | Description   |
|----------|---|
| WSEL_CHn | Channel Watchdog select for channel n<br>00 THRHLR0 register is selected<br>01 THRHLR1 register is selected<br>10 THRHLR2 register is selected<br>11 Reserved |

Offset: 0x02B4

Access: User read/write



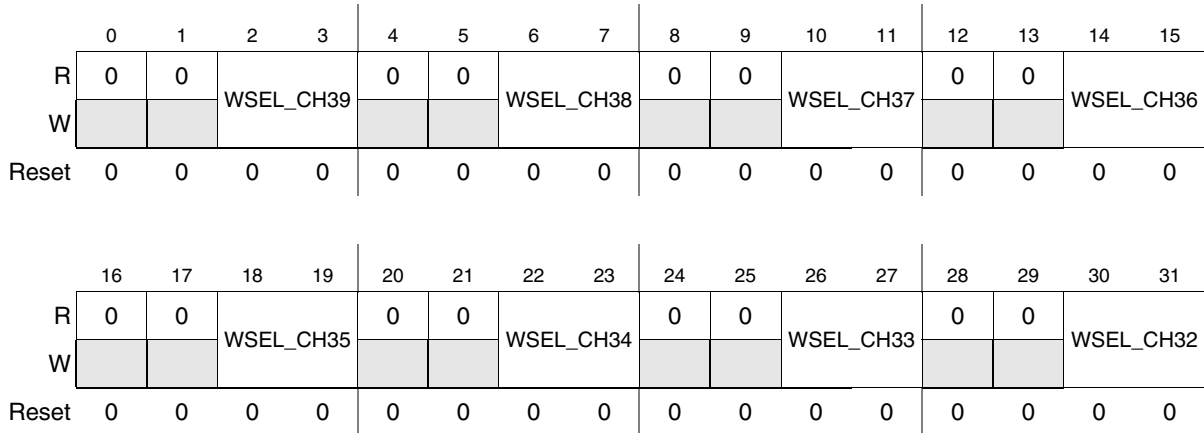
**Figure 28-61. Channel Watchdog Select Register 1 (CWSELR1) – ADC\_1**

**Table 28-53. CWSELR1 field descriptions – ADC\_1**

| Field    | Description                       |
|----------|-----------------------------------|
| WSEL_CHn | See <a href="#">Table 28-52</a> . |

Offset: 0x02C0

Access: User read/write



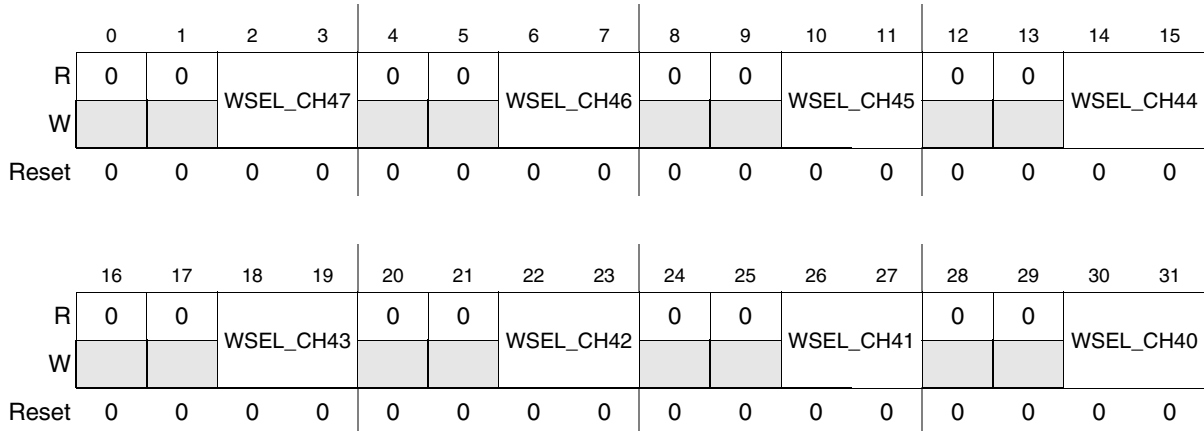
**Figure 28-62. Channel Watchdog Select Register 4 (CWSELR4) – ADC\_1**

**Table 28-54. CWSELR4 field descriptions – ADC\_1**

| Field    | Description                       |
|----------|-----------------------------------|
| WSEL_CHn | See <a href="#">Table 28-52</a> . |

Offset: 0x02C4

Access: User read/write



**Figure 28-63. Channel Watchdog Select Register 5 (CWSELR5) – ADC\_1**

Table 28-55. CWSELR5 field descriptions – ADC\_1

| Field    | Description      |
|----------|------------------|
| WSEL_CHn | See Table 28-52. |

### 28.4.11.2 Channel Watchdog Enable Register (CWENRx, x = [0..2])

Table 28-56 shows the available channels.

| ADC   | Register | Description   |
|-------|----------|---|
| ADC_0 | CWENR0   | Watchdog enable bits for channel 0 to 15 (precision channels)             |
| ADC_0 | CWENR1   | Watchdog enable bits for channel 32 to 59 (standard channels)             |
| ADC_0 | CWENR2   | Watchdog enable bits for channel 64 to 95 (external multiplexed channels) |
| ADC_1 | CWENR0   | Watchdog enable bits for channel 0 to 15 (precision channels)             |
| ADC_1 | CWENR1   | Watchdog enable bits for channel 32 to 39 (standard channels)             |

Table 28-56. CWENR[0..2] register description

Address: Base + 0x02E0

Access: User read/write

|       |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
|-------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|
|       | 0    | 1    | 2    | 3    | 4    | 5    | 6    | 7    | 8    | 9    | 10   | 11   | 12   | 13   | 14   | 15   |
| R     | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    |
| W     |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
| Reset | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    |
|       | 16   | 17   | 18   | 19   | 20   | 21   | 22   | 23   | 24   | 25   | 26   | 27   | 28   | 29   | 30   | 31   |
| R     | CWEN | CWEN | CWEN | CWEN | CWEN | CWEN | CWEN | CWEN | CWEN | CWEN | CWEN | CWEN | CWEN | CWEN | CWEN | CWEN |
| W     | 15   | 14   | 13   | 12   | 11   | 10   | 9    | 8    | 7    | 6    | 5    | 4    | 3    | 2    | 1    | 0    |
| Reset | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    |

Figure 28-64. Channel Watchdog Enable Register 0 (CWENR0)

Address: Base + 0x02E4

Access: User read/write

|       |   |   |   |   |      |      |      |      |      |      |      |      |      |      |      |      |
|-------|---|---|---|---|------|------|------|------|------|------|------|------|------|------|------|------|
|       | 0 | 1 | 2 | 3 | 4    | 5    | 6    | 7    | 8    | 9    | 10   | 11   | 12   | 13   | 14   | 15   |
| R     | 0 | 0 | 0 | 0 | CWEN | CWEN | CWEN | CWEN | CWEN | CWEN | CWEN | CWEN | CWEN | CWEN | CWEN | CWEN |
| W     |   |   |   |   | 59   | 58   | 57   | 56   | 55   | 54   | 53   | 52   | 51   | 50   | 49   | 48   |
| Reset | 0 | 0 | 0 | 0 | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    |

|       |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
|-------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|
|       | 16   | 17   | 18   | 19   | 20   | 21   | 22   | 23   | 24   | 25   | 26   | 27   | 28   | 29   | 30   | 31   |
| R     | CWEN | CWEN | CWEN | CWEN | CWEN | CWEN | CWEN | CWEN | CWEN | CWEN | CWEN | CWEN | CWEN | CWEN | CWEN | CWEN |
| W     | 47   | 46   | 43   | 44   | 43   | 42   | 41   | 40   | 39   | 38   | 37   | 36   | 35   | 34   | 33   | 32   |
| Reset | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    |

Figure 28-65. Channel Watchdog Enable Register 1 (CWENR1)

Address: Base + 0x02E4

Access: User read/write

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       |    |    |    |    |    |    |    |    |      |      |      |      |      |      |      |      |
|-------|----|----|----|----|----|----|----|----|------|------|------|------|------|------|------|------|
|       | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24   | 25   | 26   | 27   | 28   | 29   | 30   | 31   |
| R     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | CWEN | CWEN | CWEN | CWEN | CWEN | CWEN | CWEN | CWEN |
| W     |    |    |    |    |    |    |    |    | 39   | 38   | 37   | 36   | 35   | 34   | 33   | 32   |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    |

Figure 28-66. Channel Watchdog Enable Register 1 (CWENR1) – ADC\_1

Address: Base + 0x02E08

Access: User read/write

|       |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
|-------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|
|       | 0    | 1    | 2    | 3    | 4    | 5    | 6    | 7    | 8    | 9    | 10   | 11   | 12   | 13   | 14   | 15   |
| R     | CWEN | CWEN | CWEN | CWEN | CWEN | CWEN | CWEN | CWEN | CWEN | CWEN | CWEN | CWEN | CWEN | CWEN | CWEN | CWEN |
| W     | 95   | 94   | 93   | 92   | 91   | 90   | 89   | 88   | 87   | 86   | 85   | 84   | 83   | 82   | 81   | 80   |
| Reset | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    |

|       |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |
|-------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|
|       | 16   | 17   | 18   | 19   | 20   | 21   | 22   | 23   | 24   | 25   | 26   | 27   | 28   | 29   | 30   | 31   |
| R     | CWEN | CWEN | CWEN | CWEN | CWEN | CWEN | CWEN | CWEN | CWEN | CWEN | CWEN | CWEN | CWEN | CWEN | CWEN | CWEN |
| W     | 79   | 78   | 77   | 76   | 75   | 74   | 73   | 72   | 71   | 70   | 69   | 68   | 67   | 66   | 65   | 64   |
| Reset | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    | 0    |

Figure 28-67. Channel Watchdog Enable Register 2 (CWENR2)



Table 28-57. CWENRx field descriptions

| Field    | Description   |
|----------|---|
| CWEN $n$ | Channel Watchdog enable<br>When set (CWEN $n$ = 1), watchdog feature is enabled for channel $n$ . |

### 28.4.11.3 Analog Watchdog Out of Range Register (AWORRx, x = [0..2])

Address: Base + 0x02F0

Access: User read/write

|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       | 16        | 17        | 18        | 19        | 20        | 21        | 22       | 23       | 24       | 25       | 26       | 27       | 28       | 29       | 30       | 31       |
|-------|-----------|-----------|-----------|-----------|-----------|-----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|
| R     | AWOR_CH15 | AWOR_CH14 | AWOR_CH13 | AWOR_CH12 | AWOR_CH11 | AWOR_CH10 | AWOR_CH9 | AWOR_CH8 | AWOR_CH7 | AWOR_CH6 | AWOR_CH5 | AWOR_CH4 | AWOR_CH3 | AWOR_CH2 | AWOR_CH1 | AWOR_CH0 |
| W     |           |           |           |           |           |           |          |          |          |          |          |          |          |          |          |          |
| Reset | 0         | 0         | 0         | 0         | 0         | 0         | 0        | 0        | 0        | 0        | 0        | 0        | 0        | 0        | 0        | 0        |

Figure 28-68. Analog Watchdog Out of Range Register 0 (AWORR0)

Address: Base + 0x02F4

Access: User read/write

|       | 0 | 1 | 2 | 3 | 4         | 5         | 6         | 7         | 8         | 9         | 10        | 11        | 12        | 13        | 14        | 15        |
|-------|---|---|---|---|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|
| R     | 0 | 0 | 0 | 0 | AWOR_CH59 | AWOR_CH58 | AWOR_CH57 | AWOR_CH56 | AWOR_CH55 | AWOR_CH54 | AWOR_CH53 | AWOR_CH52 | AWOR_CH51 | AWOR_CH50 | AWOR_CH49 | AWOR_CH48 |
| W     |   |   |   |   | w1c       | w1c       | w1c       | w1c       | w1c       | w1c       | w1c       | w1c       | w1c       | w1c       | w1c       | w1c       |
| Reset | 0 | 0 | 0 | 0 | 0         | 0         | 0         | 0         | 0         | 0         | 0         | 0         | 0         | 0         | 0         | 0         |

|       | 16        | 17        | 18        | 19        | 20        | 21        | 22        | 23        | 24        | 25        | 26        | 27        | 28        | 29        | 30        | 31        |
|-------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|
| R     | AWOR_CH47 | AWOR_CH46 | AWOR_CH43 | AWOR_CH44 | AWOR_CH43 | AWOR_CH42 | AWOR_CH41 | AWOR_CH40 | AWOR_CH39 | AWOR_CH38 | AWOR_CH37 | AWOR_CH36 | AWOR_CH35 | AWOR_CH34 | AWOR_CH33 | AWOR_CH32 |
| W     | w1c       | w1c       | w1c       | w1c       | w1c       | w1c       | w1c       | w1c       | w1c       | w1c       | w1c       | w1c       | w1c       | w1c       | w1c       | w1c       |
| Reset | 0         | 0         | 0         | 0         | 0         | 0         | 0         | 0         | 0         | 0         | 0         | 0         | 0         | 0         | 0         | 0         |

Figure 28-69. Analog Watchdog Out of Range Register 1 (AWORR1) – ADC\_0

Address: Base + 0x02F4

Access: User read/write

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       |    |    |    |    |    |    |    |    |           |           |           |           |           |           |           |           |
|-------|----|----|----|----|----|----|----|----|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|
|       | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24        | 25        | 26        | 27        | 28        | 29        | 30        | 31        |
| R     |    |    |    |    |    |    |    |    | AWOR_CH39 | AWOR_CH38 | AWOR_CH37 | AWOR_CH36 | AWOR_CH35 | AWOR_CH34 | AWOR_CH33 | AWOR_CH32 |
| W     |    |    |    |    |    |    |    |    | w1c       | w1c       | w1c       | w1c       | w1c       | w1c       | w1c       | w1c       |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0         | 0         | 0         | 0         | 0         | 0         | 0         | 0         |

Figure 28-70. Analog Watchdog Out of Range Register 1 (AWORR1) – ADC\_1

Address: Base + 0x02F8

Access: User read/write

|       |           |           |           |           |           |           |           |           |           |           |           |           |           |           |           |           |
|-------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|
|       | 0         | 1         | 2         | 3         | 4         | 5         | 6         | 7         | 8         | 9         | 10        | 11        | 12        | 13        | 14        | 15        |
| R     | AWOR_CH95 | AWOR_CH94 | AWOR_CH93 | AWOR_CH92 | AWOR_CH91 | AWOR_CH90 | AWOR_CH89 | AWOR_CH88 | AWOR_CH87 | AWOR_CH86 | AWOR_CH85 | AWOR_CH84 | AWOR_CH83 | AWOR_CH82 | AWOR_CH81 | AWOR_CH80 |
| W     | w1c       | w1c       | w1c       | w1c       | w1c       | w1c       | w1c       | w1c       | w1c       | w1c       | w1c       | w1c       | w1c       | w1c       | w1c       | w1c       |
| Reset | 0         | 0         | 0         | 0         | 0         | 0         | 0         | 0         | 0         | 0         | 0         | 0         | 0         | 0         | 0         | 0         |

|       |           |           |           |           |           |           |           |           |           |           |           |           |           |           |           |           |
|-------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|
|       | 16        | 17        | 18        | 19        | 20        | 21        | 22        | 23        | 24        | 25        | 26        | 27        | 28        | 29        | 30        | 31        |
| R     | AWOR_CH79 | AWOR_CH78 | AWOR_CH77 | AWOR_CH76 | AWOR_CH75 | AWOR_CH74 | AWOR_CH73 | AWOR_CH72 | AWOR_CH71 | AWOR_CH70 | AWOR_CH69 | AWOR_CH68 | AWOR_CH67 | AWOR_CH66 | AWOR_CH65 | AWOR_CH64 |
| W     | w1c       | w1c       | w1c       | w1c       | w1c       | w1c       | w1c       | w1c       | w1c       | w1c       | w1c       | w1c       | w1c       | w1c       | w1c       | w1c       |
| Reset | 0         | 0         | 0         | 0         | 0         | 0         | 0         | 0         | 0         | 0         | 0         | 0         | 0         | 0         | 0         | 0         |

Figure 28-71. Analog Watchdog Out of Range Register 2 (AWORR2)

Table 28-58. AWORRx field descriptions

| Field     | Description  |
|-----------|--|
| AWORR_CHn | When set, indicates channel n converted data is out of range |

# Chapter 29

## Cross Triggering Unit (CTU)

### 29.1 Introduction

The Cross Triggering Unit (CTU) allows synchronizing an ADC conversion with a timer event from eMIOS (every mode that can generate a DMA request can trigger CTU) or PIT. To select which ADC channel is converted on a particular timer event, the CTU provides the ADC with a 7-bit channel number. This channel number can be configured for each timer channel event by the application.

### 29.2 Main features

- Single cycle delayed trigger output. The trigger output is a combination of 64 (generic value) input flags/events connected to different timers in the system.
- One event configuration register dedicated to each timer event to define the corresponding ADC channel
- Acknowledgment signal to eMIOS/PIT for clearing the flag
- Synchronization with ADC to avoid collision

### 29.3 Block diagram

The CTU block diagram is shown in [Figure 29-1](#).

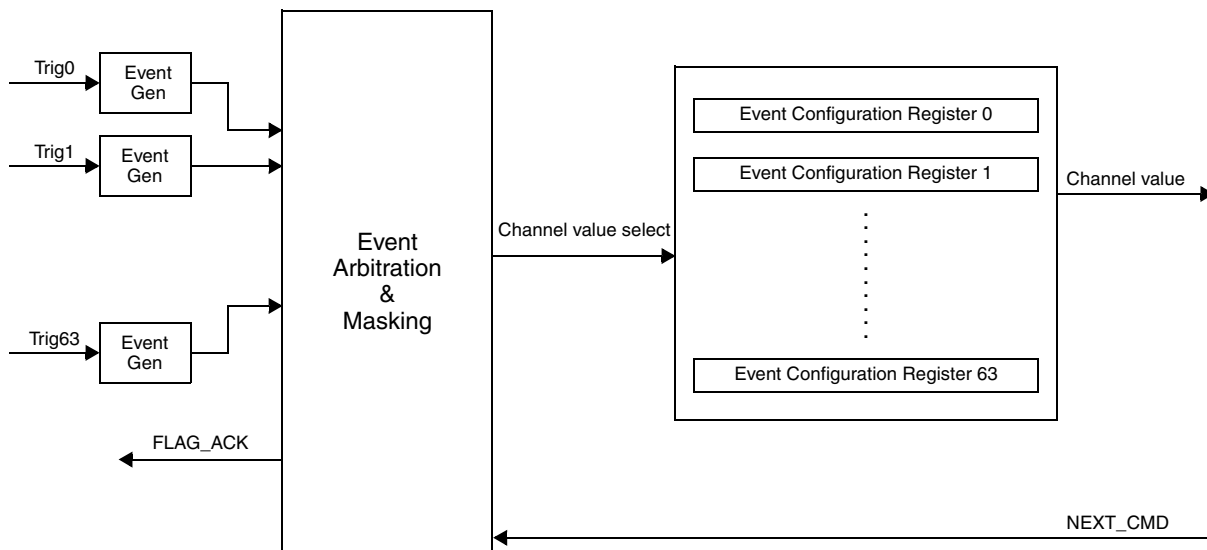


Figure 29-1. Cross Triggering Unit block diagram

### 29.4 Memory map and register descriptions

The CTU registers are listed in [Table 29-1](#). Every register can have 32-bit access. The base address of the CTU is 0xFFE6\_4000.

Table 29-1. CTU memory map

| Base address: 0xFFE6_4000 |  |                             |
|---------------------------|--|-----------------------------|
| Address offset            | Register   | Location                    |
| 0x000–0x02F               | Reserved   |                             |
| 0x030–0x12C               | Event Configuration Registers 0..63 (CTU_EVTCFGR0..63) | <a href="#">on page 780</a> |

### 29.4.1 Event Configuration Registers (CTU\_EVTCFGRx) (x = 0...63)

Offsets: 0x030–0x12C

Access: Read/write

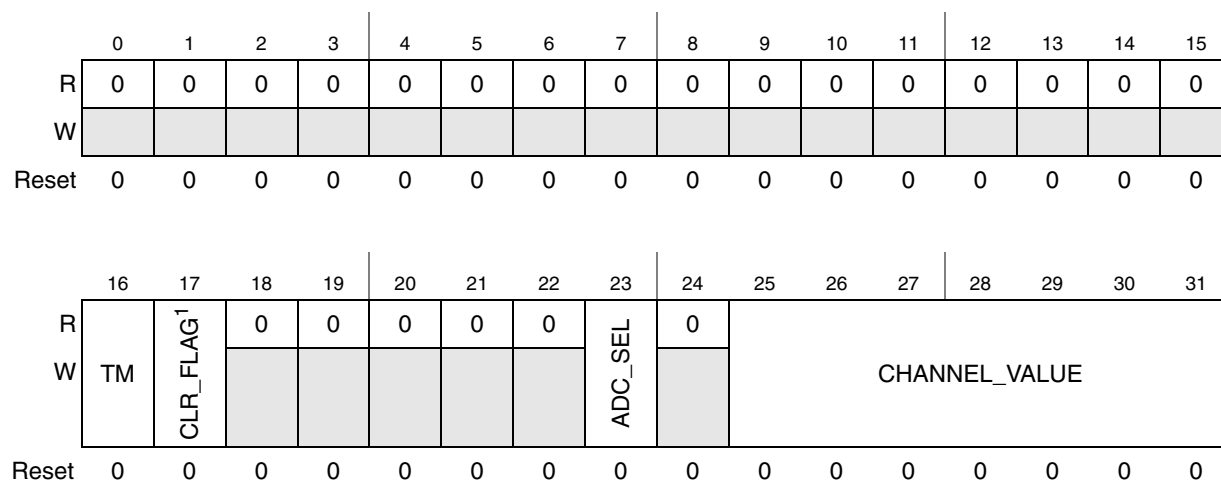


Figure 29-2. Event Configuration Registers (CTU\_EVTCFGRx) (x = 0...63)

<sup>1</sup> This bit implementation is generic based and implemented only for inputs mapped to PIT event flags.

Table 29-2. CTU\_EVTCFGRx field descriptions

| Field         | Description   |
|---------------|---|
| TM            | Trigger Mask<br>0: Trigger masked<br>1: Trigger enabled   |
| CLR_FLAG      | To provide flag_ack through software<br>1: Flag_ack is forced to 1 for the particular event<br>0: Flag_ack is dependent on flag servicing |
| ADC_SEL       | This bit selects the ADC number.<br>0: 10-bit ADC0 is selected<br>1: 12-bit ADC1 is selected  |
| CHANNEL_VALUE | These bits provide the ADC channel number to be converted. Valid values are 0b0 to 0b1011111 (decimal 95).                                |

These registers contain the ADC channel number to be converted when the timer event occurs. The CLR\_FLAG is used to clear the respective timer event flag by software (this applies only to the PIT as the eMIOS flags are automatically cleared by the CTU).

The CLR\_FLAG bit has to be used cautiously as setting this bit may result in a loss of events.

The event input can be masked by writing 0 to bit TM of the CTU\_EVTCFGR register. Writing 1 to bit TM enables the CTU triggering and automatically disables the DMA connection for the corresponding eMIOS channel.

### NOTE

The CTU tracks issued conversion requests to the ADC. When the ADC is being triggered by the CTU and there is a need to shut down the ADC, the ADC must be allowed to complete conversions before being shut down.

This ensures that the CTU is notified of completion; if the ADC is shut down while performing a CTU-triggered conversion, the CTU is not notified and will not be able to trigger further conversions until the device is reset.

## 29.5 Functional description

This peripheral is used to synchronize ADC conversions with timer events (from eMIOS or PIT). When a timer event occurs, the CTU triggers an ADC conversion providing the ADC channel number to be converted. In case concurrent events occur the priority is managed according to the index of the timer event. The trigger output is a single cycle pulse used to trigger ADC conversion of the channel number provided by the CTU.

Each trigger input from the CTU is connected to the Event Trigger signal of an eMIOS channel. The assignment between eMIOS outputs and CTU trigger inputs is defined in [Table 29-3](#).

**Table 29-3. Trigger source**

| CTU trigger No. | Module  | Source     |
|-----------------|---------|------------|
| 0               | eMIOS 0 | Channel_0  |
| 1               | eMIOS 0 | Channel_1  |
| 2               | eMIOS 0 | Channel_2  |
| 3               | eMIOS 0 | Channel_3  |
| 4               | eMIOS 0 | Channel_4  |
| 5               | eMIOS 0 | Channel_5  |
| 6               | eMIOS 0 | Channel_6  |
| 7               | eMIOS 0 | Channel_7  |
| 8               | eMIOS 0 | Channel_8  |
| 9               | eMIOS 0 | Channel_9  |
| 10              | eMIOS 0 | Channel_10 |
| 11              | eMIOS 0 | Channel_11 |
| 12              | eMIOS 0 | Channel_12 |
| 13              | eMIOS 0 | Channel_13 |
| 14              | eMIOS 0 | Channel_14 |

Table 29-3. Trigger source (continued)

| CTU trigger No. | Module  | Source     |
|-----------------|---------|------------|
| 15              | eMIOS 0 | Channel_15 |
| 16              | eMIOS 0 | Channel_16 |
| 17              | eMIOS 0 | Channel_17 |
| 18              | eMIOS 0 | Channel_18 |
| 19              | eMIOS 0 | Channel_19 |
| 20              | eMIOS 0 | Channel_20 |
| 21              | eMIOS 0 | Channel_21 |
| 22              | eMIOS 0 | Channel_22 |
| 23              | PIT     | PIT_3      |
| 24              | eMIOS 0 | Channel_24 |
| 25              | eMIOS 0 | Channel_25 |
| 26              | eMIOS 0 | Channel_26 |
| 27              | eMIOS 0 | Channel_27 |
| 28              | eMIOS 0 | Channel_28 |
| 29              | eMIOS 0 | Channel_29 |
| 30              | eMIOS 0 | Channel_30 |
| 31              | eMIOS 0 | Channel_31 |
| 32              | eMIOS 1 | Channel_0  |
| 33              | eMIOS 1 | Channel_1  |
| 34              | eMIOS 1 | Channel_2  |
| 35              | eMIOS 1 | Channel_3  |
| 36              | eMIOS 1 | Channel_4  |
| 37              | eMIOS 1 | Channel_5  |
| 38              | eMIOS 1 | Channel_6  |
| 39              | eMIOS 1 | Channel_7  |
| 40              | eMIOS 1 | Channel_8  |
| 41              | eMIOS 1 | Channel_9  |
| 42              | eMIOS 1 | Channel_10 |
| 43              | eMIOS 1 | Channel_11 |
| 44              | eMIOS 1 | Channel_12 |
| 45              | eMIOS 1 | Channel_13 |
| 46              | eMIOS 1 | Channel_14 |
| 47              | eMIOS 1 | Channel_15 |

Table 29-3. Trigger source (continued)

| CTU trigger No. | Module  | Source     |
|-----------------|---------|------------|
| 48              | eMIOS 1 | Channel_16 |
| 49              | eMIOS 1 | Channel_17 |
| 50              | eMIOS 1 | Channel_18 |
| 51              | eMIOS 1 | Channel_19 |
| 52              | eMIOS 1 | Channel_20 |
| 53              | eMIOS 1 | Channel_21 |
| 54              | eMIOS 1 | Channel_22 |
| 55              | PIT     | PIT_7      |
| 56              | eMIOS 1 | Channel_24 |
| 57              | eMIOS 1 | Channel_25 |
| 58              | eMIOS 1 | Channel_26 |
| 59              | eMIOS 1 | Channel_27 |
| 60              | eMIOS 1 | Channel_28 |
| 61              | eMIOS 1 | Channel_29 |
| 62              | eMIOS 1 | Channel_30 |
| 63              | eMIOS 1 | Channel_31 |

Each event has a dedicated configuration register (CTU\_EVTCFGR). These registers store a channel number that is used to communicate which channel needs to be converted.

In case several events are pending for ADC request, the priority is managed according to the timer event index. The lowest index has the highest priority. Once an event has been serviced (conversion requested to ADC) the eMIOS flag is cleared by the CTU and next prior event is handled.

The acknowledgment signal can be forced to 1 by setting the CLR\_FLAG bit of the CTU\_EVTCFGR register. These bits are implemented for only those input flags to which PIT flags are connected. Providing these bits offers the option of clearing PIT flags by software.

### 29.5.1 Channel value

The channel value stored in an event configuration register is demultiplexed to 7 bits and then provided to the ADC.

The mapping of the channel number value to the corresponding ADC channel is provided in [Table 29-3](#).

Table 29-4. CTU-to-ADC channel assignment

| 10-bit ADC                  |                           |                              | 12-bit ADC                  |                           |                              |
|-----------------------------|---------------------------|------------------------------|-----------------------------|---------------------------|------------------------------|
| 10-bit ADC_0<br>signal name | 10-bit ADC_0<br>channel # | Channel # in<br>CTU_EVTFCGRx | 12-bit ADC_1<br>signal name | 12-bit ADC_1<br>channel # | Channel # in<br>CTU_EVTFCGRx |
| ADC0_P[0]                   | CH0                       | 0                            | ADC1_P[0]                   | CH0                       | 0                            |
| ADC0_P[1]                   | CH1                       | 1                            | ADC1_P[1]                   | CH1                       | 1                            |
| ADC0_P[2]                   | CH2                       | 2                            | ADC1_P[2]                   | CH2                       | 2                            |
| ADC0_P[3]                   | CH3                       | 3                            | ADC1_P[3]                   | CH3                       | 3                            |
| ADC0_P[4]                   | CH4                       | 4                            | ADC1_P[4]                   | CH4                       | 4                            |
| ADC0_P[5]                   | CH5                       | 5                            | ADC1_P[5]                   | CH5                       | 5                            |
| ADC0_P[6]                   | CH6                       | 6                            | ADC1_P[6]                   | CH6                       | 6                            |
| ADC0_P[7]                   | CH7                       | 7                            | ADC1_P[7]                   | CH7                       | 7                            |
| ADC0_P[8]                   | CH8                       | 8                            | ADC1_P[8]                   | CH8                       | 8                            |
| ADC0_P[9]                   | CH9                       | 9                            | ADC1_P[9]                   | CH9                       | 9                            |
| ADC0_P[10]                  | CH10                      | 10                           | ADC1_P[10]                  | CH10                      | 10                           |
| ADC0_P[11]                  | CH11                      | 11                           | ADC1_P[11]                  | CH11                      | 11                           |
| ADC0_P[12]                  | CH12                      | 12                           | ADC1_P[12]                  | CH12                      | 12                           |
| ADC0_P[13]                  | CH13                      | 13                           | ADC1_P[13]                  | CH13                      | 13                           |
| ADC0_P[14]                  | CH14                      | 14                           | ADC1_P[14]                  | CH14                      | 14                           |
| ADC0_P[15]                  | CH15                      | 15                           | ADC1_P[15]                  | CH15                      | 15                           |
| ADC0_S[0]                   | CH32                      | 32                           | ADC1_S[0]                   | CH32                      | 32                           |
| ADC0_S[1]                   | CH33                      | 33                           | ADC1_S[1]                   | CH33                      | 33                           |
| ADC0_S[2]                   | CH34                      | 34                           | ADC1_S[2]                   | CH34                      | 34                           |
| ADC0_S[3]                   | CH35                      | 35                           | ADC1_S[3]                   | CH35                      | 35                           |
| ADC0_S[4]                   | CH36                      | 36                           | ADC1_S[4]                   | CH36                      | 36                           |
| ADC0_S[5]                   | CH37                      | 37                           | ADC1_S[5]                   | CH37                      | 37                           |
| ADC0_S[6]                   | CH38                      | 38                           | ADC1_S[6]                   | CH38                      | 38                           |
| ADC0_S[7]                   | CH39                      | 39                           | ADC1_S[7]                   | CH39                      | 39                           |
| ADC0_S[8]                   | CH40                      | 40                           |                             |                           |                              |
| ADC0_S[9]                   | CH41                      | 41                           |                             |                           |                              |
| ADC0_S[10]                  | CH42                      | 42                           |                             |                           |                              |
| ADC0_S[11]                  | CH43                      | 43                           |                             |                           |                              |
| ADC0_S[12]                  | CH44                      | 44                           |                             |                           |                              |
| ADC0_S[13]                  | CH45                      | 45                           |                             |                           |                              |



Table 29-4. CTU-to-ADC channel assignment (continued)


| 10-bit ADC                  |                           |                              | 12-bit ADC                  |                           |                              |
|-----------------------------|---------------------------|------------------------------|-----------------------------|---------------------------|------------------------------|
| 10-bit ADC_0<br>signal name | 10-bit ADC_0<br>channel # | Channel # in<br>CTU_EVTFCGRx | 12-bit ADC_1<br>signal name | 12-bit ADC_1<br>channel # | Channel # in<br>CTU_EVTFCGRx |
| ADC0_S[14]                  | CH46                      | 46                           |                             |                           |                              |
| ADC0_S[15]                  | CH47                      | 47                           |                             |                           |                              |
| ADC0_S[16]                  | CH48                      | 48                           |                             |                           |                              |
| ADC0_S[17]                  | CH49                      | 49                           |                             |                           |                              |
| ADC0_S[18]                  | CH50                      | 50                           |                             |                           |                              |
| ADC0_S[19]                  | CH51                      | 51                           |                             |                           |                              |
| ADC0_S[20]                  | CH52                      | 52                           |                             |                           |                              |
| ADC0_S[21]                  | CH53                      | 53                           |                             |                           |                              |
| ADC0_S[22]                  | CH54                      | 54                           |                             |                           |                              |
| ADC0_S[23]                  | CH55                      | 55                           |                             |                           |                              |
| ADC0_S[24]                  | CH56                      | 56                           |                             |                           |                              |
| ADC0_S[25]                  | CH57                      | 57                           |                             |                           |                              |
| ADC0_S[26]                  | CH58                      | 58                           |                             |                           |                              |
| ADC0_S[27]                  | CH59                      | 59                           |                             |                           |                              |
| ADC0_X[0]                   | CH64 : CH71               | 64:71                        |                             |                           |                              |
| ADC0_X[1]                   | CH72 : CH79               | 72:79                        |                             |                           |                              |
| ADC0_X[2]                   | CH80 : CH87               | 80:87                        |                             |                           |                              |
| ADC0_X[3]                   | CH88 : CH95               | 88:95                        |                             |                           |                              |

CTU channel mapping should be taken into consideration when programming an event configuration register. For example, if the channel value of any event configuration register is programmed to 16, it will actually correspond to ADC channel 32 and conversion will occur for this channel.

This page is intentionally left blank.

---

# Memory



This page is intentionally left blank.

# Chapter 30

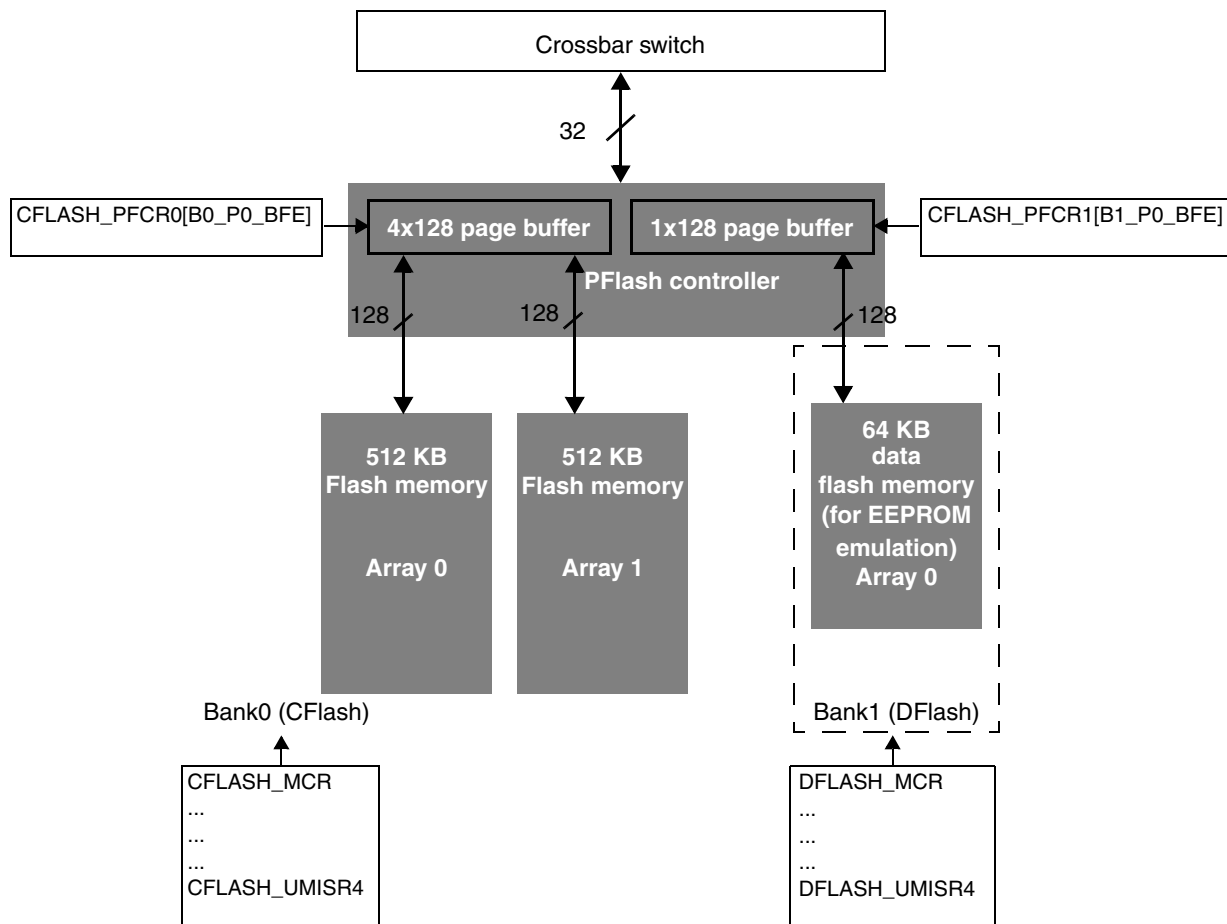
## Flash Memory

### 30.1 Introduction

The flash memory comprises a platform flash memory controller (PFlash) interface and the following flash memory arrays:

- Two arrays of 512 KB for code (CFlash)
- One array of 64 KB for data (DFlash)

The flash memory architecture of this device is illustrated in [Figure 30-1](#).



**Figure 30-1. Flash memory architecture**

The primary function of the flash memory module is to serve as electrically programmable and erasable nonvolatile memory.

Nonvolatile memory may be used for instruction and/or data storage.

The module is a nonvolatile solid-state silicon memory device consisting of:

- Blocks (also called sectors) of single transistor storage elements
- An electrical means for selectively adding (programming) and removing (erasing) charge from these elements
- A means of selectively sensing (reading) the charge stored in these elements

The flash memory module is arranged as two functional units:

- The flash memory core
- The memory interface

The flash memory core is composed of arrayed nonvolatile storage elements, sense amplifiers, row decoders, column decoders, and charge pumps. The arrayed storage elements in the flash memory core are subdivided into physically separate units referred to as blocks (or sectors).

The memory interface contains the registers and logic that control the operation of the flash memory core. The memory interface is also the interface between the flash memory module and a platform flash memory controller. It contains the ECC logic and redundancy logic.

A platform flash memory controller connects the flash memory module to a system bus, and contains all system level customization required for the device application.

## 30.2 Main features

**Table 30-1. Flash memory features**

| Feature   | CFlash | DFlash |
|---|--------|--------|
| High read parallelism (128 bits)  | Yes    |        |
| Error Correction Code (SEC-DED) to enhance data retention                 | Yes    |        |
| Double word program (64 bits)   | Yes    |        |
| Sector erase  | Yes    |        |
| Single bank—Read-While-Write (RWW)  | No     |        |
| Erase suspend   | Yes    |        |
| Program suspend   | No     |        |
| Software programmable program/erase protection to avoid unwanted writings | Yes    |        |
| Censored mode against piracy  | Yes    |        |
| Shadow sector available   | Yes    | No     |
| One-Time Programmable (OTP) area in Test Flash block                      | Yes    |        |
| Boot sectors  | Yes    | No     |

## 30.3 Block diagram

The flash memory module contains one Matrix Module, composed of a single bank (Bank 0) normally used for code storage. RWW operations are not possible.

Modify operations are managed by an embedded Flash Memory Program/Erase Controller (FPEC). Commands to the FPEC are given through a user registers interface.

The read data bus is 128 bits wide, while the flash memory registers are on a separate bus 32 bits wide addressed in the user memory map.

The high voltages needed for program/erase operations are generated internally.

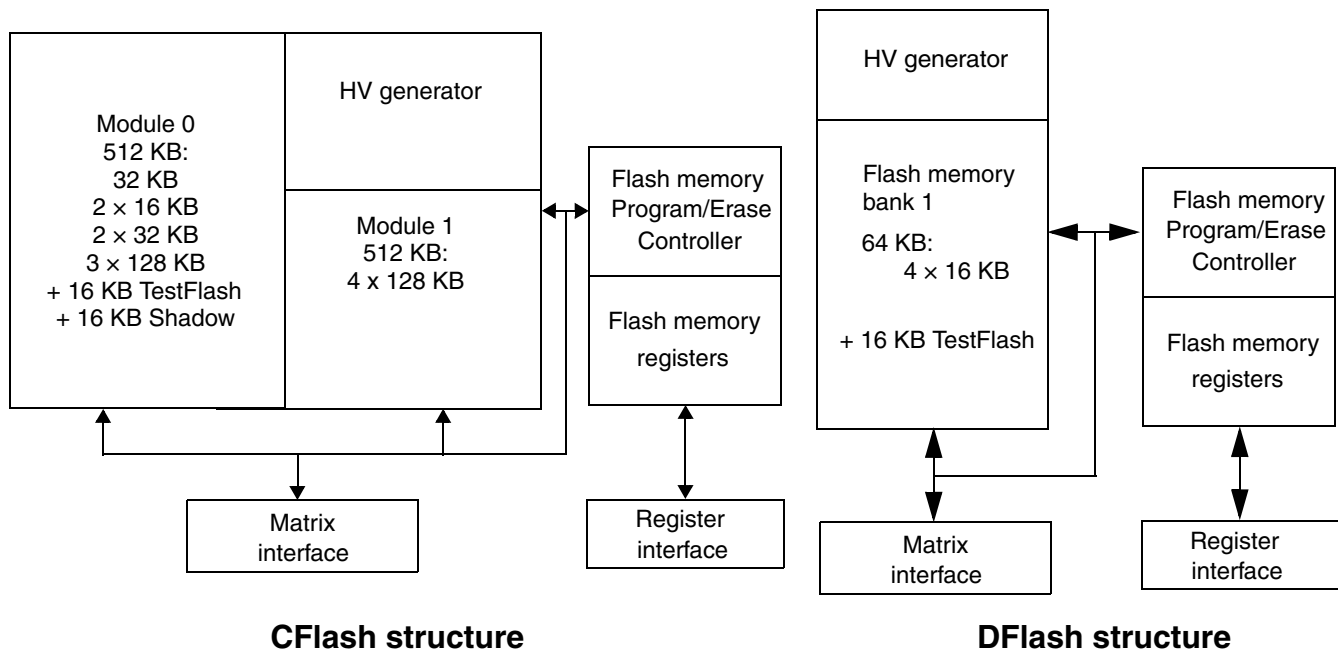


Figure 30-2. CFlash and DFlash module structures

## 30.4 Functional description

### 30.4.1 Module structure

The flash memory module is addressable by double word (64 bits) for program, and page (128 bits) for read. Reads to the flash memory always return 128 bits, although read page buffering may be done in the platform flash memory controller.

Each read of the flash memory module retrieves a page, or four consecutive words (128 bits) of information. The address for each word retrieved within a page differs from the other addresses in the page only by address bits (3:2).

The flash memory module supports fault tolerance through Error Correction Code (ECC) or error detection, or both. The ECC implemented within the flash memory module will correct single bit failures and detect double bit failures.

The flash memory module uses an embedded hardware algorithm implemented in the memory interface to program and erase the flash memory core.

The embedded hardware algorithm includes control logic that works with software block enables and software lock mechanisms to guard against accidental program/erase.

The hardware algorithm performs the steps necessary to ensure that the storage elements are programmed and erased with sufficient margin to guarantee data integrity and reliability.

In the flash memory module, logic levels are defined as follows:

- A programmed bit reads as logic level 0 (or low).
- An erased bit reads as logic level 1 (or high).

Program and erase of the flash memory module requires multiple system clock cycles to complete.

The erase sequence may be suspended.

The program and erase sequences may be aborted.

## 30.4.2 Flash memory module sectorization

### 30.4.2.1 CFlash module sectorization

The CFlash module supports 1 MB of user memory, plus 16 KB of test memory (a portion of which is One-Time Programmable by the user). An extra 16 KB sector is available as Shadow space usable for user option bits and censorship settings.

The module is composed of a single bank (Bank 0): Read-While-Write is not supported.

Bank 0 of the module is divided in 14 sectors including a reserved sector, named TestFlash, in which some One-Time Programmable (OTP) user data are stored, as well as a Shadow Sector in which user erasable configuration values can be stored.

The matrix module sectorization is shown in [Table 30-2](#).

**Table 30-2. CFlash module sectorization**

| CFlash |        |        | Sector address range  | Sector size | Address space | Address space locking register |            |
|--------|--------|--------|-----------------------|-------------|---------------|--------------------------------|------------|
| Bank   | Sector | Module |                       |             |               | CFLASH_LML                     | CFLASH_HBL |
| 0      | 0      | 0      | 0x00000000–0x00007FFF | 32 KB       | Low           | LLK0                           |            |
| 0      | 1      | 0      | 0x00008000–0x0000BFFF | 16 KB       | Low           | LLK1                           |            |
| 0      | 2      | 0      | 0x0000C000–0x0000FFFF | 16 KB       | Low           | LLK2                           |            |
| 0      | 3      | 0      | 0x00010000–0x00017FFF | 32 KB       | Low           | LLK3                           |            |
| 0      | 4      | 0      | 0x00018000–0x0001FFFF | 32 KB       | Low           | LLK4                           |            |
| 0      | 5      | 0      | 0x00020000–0x0003FFFF | 128 KB      | Low           | LLK5                           |            |
| 0      | 6      | 0      | 0x00040000–0x0005FFFF | 128 KB      | Mid           | MLK0                           |            |
| 0      | 7      | 0      | 0x00060000–0x0007FFFF | 128 KB      | Mid           | MLK1                           |            |



Table 30-2. CFlash module sectorization (continued)

| CFlash |        |        | Sector address range   | Sector size | Address space | Address space locking register |            |
|--------|--------|--------|------------------------|-------------|---------------|--------------------------------|------------|
| Bank   | Sector | Module |                        |             |               | CFLASH_LML                     | CFLASH_HBL |
| 0      | 8      | 1      | 0x00080000–0x0009FFFF  | 128 KB      | High          |                                | HLK0       |
| 0      | 9      | 1      | 0x000A0000–0x000BFFFF  | 128 KB      | High          |                                | HLK1       |
| 0      | 10     | 1      | 0x000C0000–0x000DFFFF  | 128 KB      | High          |                                | HLK2       |
| 0      | 11     | 1      | 0x000E0000–0x000FFFFFF | 128 KB      | High          |                                | HLK3       |
| 0      | Shadow | 0      | 0x00200000–0x00203FFF  | 16 KB       | Shadow        | TSLK                           |            |
| 0      | Test   | 0      | 0x00400000–0x00403FFF  | 16 KB       | Test          | TSLK                           |            |

The division into blocks of the flash memory module is also used to implement independent erase/program protection. A software mechanism is provided to independently lock/unlock each block in low and mid address space against program and erase.

### 30.4.2.2 DFlash module sectorization

The DFlash module supports 64 KB of user memory, plus 16 KB of test memory (a portion of which is One-Time Programmable by the user).

The module is composed of a single bank (Bank 0): Read-While-Write is not supported.

Bank 0 of the 80 KB module is divided in four sectors. Bank 0 also contains a reserved sector named TestFlash in which some One-Time Programmable user data are stored.

The sectorization of the 80 KB matrix module is shown in [Table 30-3](#).

Table 30-3. DFlash module sectorization

| Bank | Sector | Addresses             | Size (KB) | Address space | DFLASH_LML field for locking the address space |
|------|--------|-----------------------|-----------|---------------|--|
| 0    | 0      | 0x00800000–0x00803FFF | 16        | Low           | LLK0   |
|      | 1      | 0x00804000–0x00807FFF |           |               | LLK1   |
|      | 2      | 0x00808000–0x0080BFFF |           |               | LLK2   |
|      | 3      | 0x0080C000–0x0080FFFF |           |               | LLK3   |
|      | Test   | 0x00C00000–0x00C03FFF |           | Test          | TSLK   |

The flash memory module is divided into blocks also to implement independent erase/program protection. A software mechanism is provided to independently lock/unlock each block in low and mid address space against program and erase.

### 30.4.3 TestFlash block

A TestFlash block is available in both the CFlash and DFlash modules. The TestFlash block exists outside the normal address space and is programmed and read independently of the other blocks. The independent

TestFlash block is included to also support systems that require nonvolatile memory for security or to store system initialization information, or both.

A section of the TestFlash is reserved to store the nonvolatile information related to redundancy, configuration, and protection.

The ECC is also applied to TestFlash.

The structure of the TestFlash sector is detailed in [Table 30-4](#) and [Table 30-5](#).

**Table 30-4. CFlash TestFlash structure**

| Name         | Description   | Addresses         | Size       |
|--------------|---|-------------------|------------|
| —            | User OTP area   | 0x400000–0x401FFF | 8192 bytes |
| —            | Reserved  | 0x402000–0x403CFF | 7424 bytes |
| —            | User OTP area   | 0x403D00–0x403DE7 | 232 bytes  |
| CFLASH_NVLML | CFlash Nonvolatile Low/Mid Address Space Block Locking Register           | 0x403DE8–0x403DEF | 8 bytes    |
| CFLASH_NVHBL | CFlash Nonvolatile High Address Space Block Locking Register              | 0x403DF0–0x403DF7 | 8 bytes    |
| CFLASH_NVSL  | CFlash Nonvolatile Secondary Low/Mid Address Space Block Locking Register | 0x403DF8–0x403DFF | 8 bytes    |
| —            | User OTP area   | 0x403E00–0x403EFF | 256 bytes  |
| —            | Reserved  | 0x403F00–0x403FFF | 256 bytes  |

**Table 30-5. DFlash TestFlash structure**

| Name         | Description   | Addresses         | Size       |
|--------------|---|-------------------|------------|
| —            | User OTP area   | 0xC00000–0xC01FFF | 8192 bytes |
| —            | Reserved  | 0xC02000–0xC03CFF | 7424 bytes |
| —            | User OTP area   | 0xC03D00–0xC03DE7 | 232 bytes  |
| DFLASH_NVLML | DFlash Nonvolatile Low/Mid Address Space Block Locking Register           | 0xC03DE8–0xC03DEF | 8 bytes    |
| —            | Reserved  | 0xC03DF0–0xC03DF7 | 8 bytes    |
| DFLASH_NVSL  | DFlash Nonvolatile Secondary Low/Mid Address Space Block Locking Register | 0xC03DF8–0xC03DFF | 8 bytes    |
| —            | User OTP area   | 0xC03E00–0xC03EFF | 256 bytes  |
| —            | Reserved  | 0xC03F00–0xC03FFF | 256 bytes  |

Erase of the TestFlash block is always locked.

Programming of the TestFlash block has similar restrictions as the array in terms of how ECC is calculated. Only one programming operation is allowed per 64-bit ECC segment.

The first 8 KB of TestFlash block may be used for user defined functions (possibly to store serial numbers, other configuration words or factory process codes). Locations of the TestFlash other than the first 8 KB of OTP area cannot be programmed by the user application.

### 30.4.4 Shadow sector

The shadow sector is only present in the CFlash module.

User mode program and erase of the shadow sector are enabled only when CFLASH\_MCR[PEAS] is high.

The shadow sector may be locked/unlocked against program or erase by using the CFLASH\_LML[TSLK] and CFLASH\_SLL[STSLK] fields.

Programming of the shadow sector has similar restrictions as the array in terms of how ECC is calculated. Only one programming operation is allowed per 64-bit ECC segment between erases.

Erase of the shadow sector is done similarly to a sector erase.

The shadow sector contains specified data that are needed for user features.

The user area of shadow sector may be used for user defined functions (possibly to store boot code, other configuration words or factory process codes).

The structure of the shadow sector is detailed in [Table 30-6](#).

**Table 30-6. Shadow sector structure**

| Name     | Description  | Addresses         | Size (bytes) |
|----------|--|-------------------|--------------|
| —        | User area  | 0x200000–0x203DCF | 15824        |
| —        | Reserved   | 0x203DD0–0x203DD7 | 8            |
| NVPWD0–1 | Nonvolatile Private Censorship PassWord 0–1 registers        | 0x203DD8–0x203DDF | 8            |
| NVSCC0–1 | Nonvolatile System Censorship Control 0–1 registers          | 0x203DE0–0x203DE7 | 8            |
| —        | Reserved   | 0x203DE8–0x203DFF | 24           |
| NVPFAPR  | Nonvolatile Platform Flash Memory Access Protection Register | 0x203E00–0x203E07 | 8            |
| —        | Reserved   | 0x203E08–0x203E17 | 16           |
| NVUSRO   | Nonvolatile User Options register                            | 0x203E18–0x203E1F | 8            |
| —        | Reserved   | 0x203E20–0x203FFF | 480          |

### 30.4.5 User mode operation

In User mode, the flash memory module may be read and written (register writes and interlock writes), programmed, or erased.

The default state of the flash memory module is read.

The main, shadow and test address space can be read only in the read state.

The majority of CFlash and DFlash memory-mapped registers can be read even when the CFlash or DFlash is in power-down or low-power mode. The exceptions are as follows:

- CFlash
  - UT0[MRE, MRV, AIS, DSI0:7]
  - UT1
  - UT2
- DFlash
  - UT0[MRE, MRV, AIS, DSI0:7]
  - UT1
  - UT2

The flash memory module enters the read state on reset.

The module is in the read state under two sets of conditions:

- The read state is active when the module is enabled (User mode read).
- The read state is active when the ERS and ESUS fields in the corresponding MCR (CFLASH\_MCR or DFLASH\_MCR) are 1 and the PGM field is 0 (erase suspend).

Flash memory core reads return 128 bits (1 page = 2 double words).

Register reads return 32 bits (1 Word).

Flash memory core reads are done through the platform flash memory controller.

Register reads to unmapped register address space will return all 0s.

Register writes to unmapped register address space will have no effect.

Attempted array reads to invalid locations will result in indeterminate data. Invalid locations occur when blocks that do not exist in non  $2^n$  array sizes are addressed.

Attempted interlock writes to invalid locations will result in an interlock occurring, but attempts to program these blocks will not occur since they are forced to be locked. Erase will occur to selected and unlocked blocks even if the interlock write is to an invalid location.

Simultaneous Read cycle on the Flash Matrix and Read/Write cycles on the registers are possible. On the contrary, registers read/write accesses simultaneous to a Flash Matrix interlock write are forbidden.

### 30.4.6 Reset

A reset is the highest priority operation for the flash memory module and terminates all other operations.

The flash memory module uses reset to initialize register and status bits to their default reset values. If the flash memory module is executing a program or erase operation (PGM = 1 or ERS = 1 in CFLASH\_MCR or DFLASH\_MCR) and a reset is issued, the operation will be suddenly terminated and the module will disable the high voltage logic without damage to the high voltage circuits. Reset terminates all operations and forces the flash memory module into User mode ready to receive accesses. Reset and power-off must not be used as a systematic way to terminate a program or erase operation.

After reset is negated, read register access may be done, although it should be noted that registers that require updating from shadow information, or other inputs, may not read updated values until the DONE field (in CFLASH\_MCR or DFLASH\_MCR) transitions. The DONE field may be polled to determine if the flash memory module has transitioned out of reset. Notice that the registers cannot be written until the DONE field is high.

### 30.4.7 Power-down mode

All flash memory DC current sources can be turned off in power-down mode, so that all power dissipation is due only to leakage in this mode. Flash memory power-down mode can be selected at `ME_<mode>_MC`.

Reads from or writes to the module are not possible in power-down mode.

When enabled the flash memory module returns to its pre-disable state in all cases unless in the process of executing an erase high voltage operation at the time of disable.

If the flash memory module is disabled during an erase operation, MCR[ESUS] bit is programmed to 1. The user may resume the erase operation at the time the module is enabled by programming MCR[ESUS] = 0. MCR[EHV] must be high to resume the erase operation.

If the flash memory module is disabled during a program operation, the operation will in any case be completed and the power-down mode will be entered only after the programming ends.

The user should realize that, if the flash memory module is put in power-down mode and the interrupt vectors remain mapped in the flash memory address space, the flash memory module will greatly increase the interrupt response time by adding several wait-states.

It is forbidden to enter in low power mode when the power-down mode is active.

### 30.4.8 Low power mode

The low power mode turns off most of the DC current sources within the flash memory module. Flash memory low power mode can be selected at `ME_<mode>_MC`.

The module (flash memory core and registers) is not accessible for read or write once it enters low power mode.

Wake-up time from low power mode is faster than wake-up time from power-down mode.

When exiting from low power mode the flash memory module returns to its pre-sleep state in all cases unless it is executing an erase high voltage operation at the time low power mode is entered.

If the flash memory module enters low power mode during an erase operation, MCR[ESUS] is programmed to 1. The user may resume the erase operation at the time the module exits low power mode by programming MCR[ESUS] = 0. MCR[EHV] must be high to resume the erase operation.

If the flash memory module enters low power mode during a program operation, the operation will be in any case completed and the low power mode will be entered only after the programming end.

It is forbidden to enter power-down mode when the low power mode is active.

## 30.5 Register description

The CFlash and DFlash modules have respective sets of memory mapped registers. The CFlash register mapping is shown in [Table 30-7](#). The DFlash register mapping is shown in [Table 30-8](#).

**Table 30-7. CFlash registers**

| Address offset | Register   | Location                    |
|----------------|--|-----------------------------|
| 0x0000         | CFlash Module Configuration Register (CFLASH_MCR)                          | <a href="#">on page 799</a> |
| 0x0004         | CFlash Low/Mid Address Space Block Locking Register (CFLASH_LML)           | <a href="#">on page 805</a> |
| 0x0008         | CFlash High Address Space Block Locking Register (CFLASH_HBL)              | <a href="#">on page 809</a> |
| 0x000C         | CFlash Secondary Low/Mid Address Space Block Locking Register (CFLASH_SLL) | <a href="#">on page 812</a> |
| 0x0010         | CFlash Low/Mid Address Space Block Select Register (CFLASH_LMS)            | <a href="#">on page 818</a> |
| 0x0014         | CFlash High Address Space Block Select Register (CFLASH_HBS)               | <a href="#">on page 819</a> |
| 0x0018         | CFlash Address Register (CFLASH_ADR)                                       | <a href="#">on page 820</a> |
| 0x0028–0x0038  | Reserved   |                             |
| 0x003C         | CFlash User Test 0 register (CFLASH_UT0)                                   | <a href="#">on page 821</a> |
| 0x0040         | CFlash User Test 1 register (CFLASH_UT1)                                   | <a href="#">on page 823</a> |
| 0x0044         | CFlash User Test 2 register (CFLASH_UT2)                                   | <a href="#">on page 823</a> |
| 0x0048         | CFlash User Multiple Input Signature Register 0 (CFLASH_UMISR0)            | <a href="#">on page 824</a> |
| 0x004C         | CFlash User Multiple Input Signature Register 1 (CFLASH_UMISR1)            | <a href="#">on page 825</a> |
| 0x0050         | CFlash User Multiple Input Signature Register 2 (CFLASH_UMISR2)            | <a href="#">on page 826</a> |
| 0x0054         | CFlash User Multiple Input Signature Register 3 (CFLASH_UMISR3)            | <a href="#">on page 827</a> |
| 0x0058         | CFlash User Multiple Input Signature Register 4 (CFLASH_UMISR4)            | <a href="#">on page 828</a> |

**Table 30-8. DFlash registers**

| Address offset | Register name  | Location                    |
|----------------|--|-----------------------------|
| 0x0000         | DFlash Module Configuration Register (DFLASH_MCR)                | <a href="#">on page 834</a> |
| 0x0004         | DFlash Low/Mid Address Space Block Locking Register (DFLASH_LML) | <a href="#">on page 839</a> |
| 0x0008         | Reserved   | —                           |

Table 30-8. DFlash registers (continued)

| Address offset | Register name  | Location                    |
|----------------|--|-----------------------------|
| 0x000C         | DFlash Secondary Low/Mid Address Space Block Locking Register (DFLASH_SLL) | <a href="#">on page 843</a> |
| 0x0010         | DFlash Low/Mid Address Space Block Select Register (DFLASH_LMS)            | <a href="#">on page 847</a> |
| 0x0014         | Reserved   | —                           |
| 0x0018         | DFlash Address Register (DFLASH_ADR)                                       | <a href="#">on page 847</a> |
| 0x001C–0x003B  | Reserved   | —                           |
| 0x003C         | DFlash User Test 0 register (DFLASH_UT0)                                   | <a href="#">on page 848</a> |
| 0x0040         | DFlash User Test 1 register (DFLASH_UT1)                                   | <a href="#">on page 851</a> |
| 0x0044         | DFlash User Test 2 register (DFLASH_UT2)                                   | <a href="#">on page 851</a> |
| 0x0048         | DFlash User Multiple Input Signature Register 0 (DFLASH_UMISR0)            | <a href="#">on page 852</a> |
| 0x004C         | DFlash User Multiple Input Signature Register 1 (DFLASH_UMISR1)            | <a href="#">on page 853</a> |
| 0x0050         | DFlash User Multiple Input Signature Register 2 (DFLASH_UMISR2)            | <a href="#">on page 854</a> |
| 0x0054         | DFlash User Multiple Input Signature Register 3 (DFLASH_UMISR3)            | <a href="#">on page 855</a> |
| 0x0058         | DFlash User Multiple Input Signature Register 4 (DFLASH_UMISR4)            | <a href="#">on page 856</a> |

In the following some nonvolatile registers are described. Please notice that such entities are not Flip-Flops, but locations of TestFlash or Shadow sectors with a special meaning.

During the flash memory initialization phase, the FPEC reads these nonvolatile registers and updates the corresponding volatile registers. When the FPEC detects ECC double errors in these special locations, it behaves in the following way:

- In case of a failing system locations (configurations, device options, redundancy, embedded firmware), the initialization phase is interrupted and a Fatal Error is flagged.
- In case of failing user locations (protections, censorship, platform flash memory controller, ...), the volatile registers are filled with all 1s and the flash memory initialization ends setting low the PEG bit of the corresponding MCR (CFLASH\_MCR or DFLASH\_MCR).

## 30.5.1 CFlash register description

### 30.5.1.1 CFlash Module Configuration Register (CFLASH\_MCR)

The CFlash Module Configuration Register is used to enable and monitor all modify operations of the flash memory module.

Offset: 0x0000

Access: Read/write

|       |     |   |   |   |   |      |   |   |     |   |    |    |    |    |     |    |
|-------|-----|---|---|---|---|------|---|---|-----|---|----|----|----|----|-----|----|
|       | 0   | 1 | 2 | 3 | 4 | 5    | 6 | 7 | 8   | 9 | 10 | 11 | 12 | 13 | 14  | 15 |
| R     | EDC | 0 | 0 | 0 | 0 | SIZE |   | 0 | LAS |   |    | 0  | 0  | 0  | MAS |    |
| W     | w1c |   |   |   |   |      |   |   |     |   |    |    |    |    |     |    |
| Reset | 0   | 0 | 0 | 0 | 0 | 0    | 1 | 1 | 0   | 0 | 1  | 0  | 0  | 0  | 0   | 0  |

|       |     |     |    |    |      |      |     |    |    |    |    |     |      |     |      |     |
|-------|-----|-----|----|----|------|------|-----|----|----|----|----|-----|------|-----|------|-----|
|       | 16  | 17  | 18 | 19 | 20   | 21   | 22  | 23 | 24 | 25 | 26 | 27  | 28   | 29  | 30   | 31  |
| R     | EER | RWE | 0  | 0  | PEAS | DONE | PEG | 0  | 0  | 0  | 0  | PGM | PSUS | ERS | ESUS | EHV |
| W     | w1c | w1c |    |    |      |      |     |    |    |    |    |     |      |     |      |     |
| Reset | 0   | 0   | 0  | 0  | 0    | 1    | 1   | 0  | 0  | 0  | 0  | 0   | 0    | 0   | 0    | 0   |

Figure 30-3. CFlash Module Configuration Register (CFLASH\_MCR)

Table 30-9. CFLASH\_MCR field descriptions

| Field | Description   |
|-------|---|
| EDC   | <p><i>ECC Data Correction</i></p> <p>EDC provides information on previous reads. If an ECC single error detection and correction occurred, the EDC bit is set to 1. This bit must then be cleared, or a reset must occur before this bit will return to a 0 state. This bit may not be set to 1 by the user.</p> <p>In the event of an ECC double error detection, this bit will not be set.</p> <p>If EDC is not set, or remains 0, this indicates that all previous reads (from the last reset, or clearing of EDC) were not corrected through ECC.</p> <p>0 Reads are occurring normally.<br/>1 An ECC single error occurred and was corrected during a previous read.</p> |
| SIZE  | <p><i>Array Space Size</i></p> <p>The value of SIZE field is dependent upon the size of the flash memory module. See <a href="#">Table 30-10</a>.</p>   |
| LAS   | <p><i>Low Address Space</i></p> <p>The value of the LAS field corresponds to the configuration of the low address space. See <a href="#">Table 30-11</a>.</p>   |
| MAS   | <p><i>Mid Address Space</i></p> <p>The value of the MAS field corresponds to the configuration of the mid address space. See <a href="#">Table 30-12</a>.</p>   |
| EER   | <p><i>ECC Event Error</i></p> <p>EER provides information on previous reads. If an ECC double error detection occurred, the EER bit is set to 1.</p> <p>This bit must then be cleared, or a reset must occur before this bit will return to a 0 state.</p> <p>This bit may not be set to 1 by the user.</p> <p>In the event of an ECC single error detection and correction, this bit will not be set.</p> <p>If EER is not set, or remains 0, this indicates that all previous reads (from the last reset, or clearing of EER) were correct.</p> <p>0 Reads are occurring normally.<br/>1 An ECC double error occurred during a previous read.</p>                           |



Table 30-9. CFLASH\_MCR field descriptions (continued)

| Field | Description   |
|-------|---|
| RWE   | <p><i>Read-While-Write Event Error</i></p> <p>RWE provides information on previous reads when a Modify operation is on going. If a RWW Error occurs, the RWE bit is set to 1. Read-While-Write Error means that a read access to the flash memory Matrix has occurred while the FPEC was performing a program or erase operation or an array integrity check.</p> <p>This bit must then be cleared, or a reset must occur before this bit will return to a 0 state. This bit may not be set to 1 by the user.</p> <p>If RWE is not set, or remains 0, this indicates that all previous RWW reads (from the last reset, or clearing of RWE) were correct.</p> <p>0 Reads are occurring normally.<br/>1 A RWW Error occurred during a previous read.</p>  |
| PEAS  | <p><i>Program/Erase Access Space</i></p> <p>PEAS indicates which space is valid for program and erase operations: main array space or shadow/test space.</p> <p>PEAS = 0 indicates that the main address space is active for all flash memory module program and erase operations.</p> <p>PEAS = 1 indicates that the test or shadow address space is active for program and erase. The value in PEAS is captured and held with the first interlock write done for Modify operations. The value of PEAS is retained between sampling events (that is, subsequent first interlock writes).</p> <p>0 Shadow/Test address space is disabled for program/erase and main address space enabled.<br/>1 Shadow/Test address space is enabled for program/erase and main address space disabled.</p>  |
| DONE  | <p><i>Modify Operation Done</i></p> <p>DONE indicates if the flash memory module is performing a high voltage operation. DONE is set to 1 on termination of the flash memory module reset.</p> <p>DONE is cleared to 0 just after a 0-to-1 transition of EHV, which initiates a high voltage operation, or after resuming a suspended operation.</p> <p>DONE is set to 1 at the end of program and erase high voltage sequences.</p> <p>DONE is set to 1 (within <math>t_{PABT}</math> or <math>t_{EABT}</math>, equal to P/E Abort Latency) after a 1-to-0 transition of EHV, which aborts a high voltage program/erase operation.</p> <p>DONE is set to 1 (within <math>t_{ESUS}</math>, time equals to Erase Suspend Latency) after a 0-to-1 transition of ESUS, which suspends an erase operation.</p> <p>0 Flash memory is executing a high voltage operation.<br/>1 Flash memory is not executing a high voltage operation.</p> |

Table 30-9. CFLASH\_MCR field descriptions (continued)

| Field | Description   |
|-------|---|
| PEG   | <p><i>Program/Erase Good</i></p> <p>The PEG bit indicates the completion status of the last flash memory program or erase sequence for which high voltage operations were initiated. The value of PEG is updated automatically during the program and erase high voltage operations. Aborting a program/erase high voltage operation will cause PEG to be cleared to 0, indicating the sequence failed. PEG is set to 1 when the flash memory module is reset, unless a flash memory initialization error has been detected. The value of PEG is valid only when PGM=1 and/or ERS=1 and after DONE transitions from 0-to-1 due to an abort or the completion of a program/erase operation. PEG is valid until PGM/ERS makes a 1-to-0 transition or EHV makes a 0-to-1 transition. The value in PEG is not valid after a 0-to-1 transition of DONE caused by ESUS being set to logic 1. If program or erase are attempted on blocks that are locked, the response will be PEG=1, indicating that the operation was successful, and the content of the block were properly protected from the program or erase operation. If a program operation tries to program at 1 bits that are at 0, the program operation is correctly executed on the new bits to be programmed at 0, but PEG is cleared, indicating that the requested operation has failed. In array integrity check or margin read, PEG is set to 1 when the operation is completed, regardless the occurrence of any error. The presence of errors can be detected only comparing checksum value stored in UMIRS0-1. Aborting an array integrity check or a margin read operation will cause PEG to be cleared to 0, indicating the sequence failed.</p> <p>0 Program or erase operation failed or program, erase, array integrity check, or margin mode aborted.</p> <p>1 Program or erase operation successful or array integrity check or margin mode completed.</p> |
| PGM   | <p><i>Program</i></p> <p>PGM is used to set up the flash memory module for a program operation.</p> <p>A 0-to-1 transition of PGM initiates a program sequence.</p> <p>A 1-to-0 transition of PGM ends the program sequence.</p> <p>PGM can be set only under User mode read (ERS is low and UT0[AIE] is low).</p> <p>PGM can be cleared by the user only when EHV is low and DONE is high.</p> <p>PGM is cleared on reset.</p> <p>0 Flash memory is not executing a program sequence.</p> <p>1 Flash memory is executing a program sequence.</p>   |
| PSUS  | <p><i>Program Suspend</i></p> <p>Write this bit has no effect, but the written data can be read back.</p>   |
| ERS   | <p><i>Erase</i></p> <p>ERS is used to set up the flash memory module for an erase operation.</p> <p>A 0-to-1 transition of ERS initiates an erase sequence.</p> <p>A 1-to-0 transition of ERS ends the erase sequence.</p> <p>ERS can be set only under User mode read (PGM is low and UT0[AIE] is low).</p> <p>ERS can be cleared by the user only when ESUS and EHV are low and DONE is high.</p> <p>ERS is cleared on reset.</p> <p>0 Flash memory is not executing an erase sequence.</p> <p>1 Flash memory is executing an erase sequence.</p>   |

**Table 30-9. CFLASH\_MCR field descriptions (continued)**

| Field | Description  |
|-------|--|
| ESUS  | <p><i>Erase Suspend</i></p> <p>ESUS is used to indicate that the flash memory module is in erase suspend or in the process of entering a Suspend state. The flash memory module is in erase suspend when ESUS = 1 and DONE = 1.</p> <p>ESUS can be set high only when ERS and EHV are high and PGM is low.</p> <p>A 0-to-1 transition of ESUS starts the sequence that sets DONE and places the flash memory in erase suspend. The flash memory module enters Suspend within <math>t_{ESUS}</math> of this transition.</p> <p>ESUS can be cleared only when DONE and EHV are high and PGM is low.</p> <p>A 1-to-0 transition of ESUS with EHV = 1 starts the sequence that clears DONE and returns the module to erase.</p> <p>The flash memory module cannot exit erase suspend and clear DONE while EHV is low.</p> <p>ESUS is cleared on reset.</p> <p>0 Erase sequence is not suspended.<br/>1 Erase sequence is suspended.</p>  |
| EHV   | <p><i>Enable High Voltage</i></p> <p>The EHV bit enables the flash memory module for a high voltage program/erase operation. EHV is cleared on reset.</p> <p>EHV must be set after an interlock write to start a program/erase sequence. EHV may be set under one of the following conditions:</p> <ul style="list-style-type: none"> <li>• Erase (ERS = 1, ESUS = 0, UT0[AIE] = 0)</li> <li>• Program (ERS = 0, ESUS = 0, PGM = 1, UT0[AIE] = 0)</li> </ul> <p>In normal operation, a 1-to-0 transition of EHV with DONE high and ESUS low terminates the current program/erase high voltage operation.</p> <p>When an operation is aborted, there is a 1-to-0 transition of EHV with DONE low and the eventual Suspend bit low. An abort causes the value of PEG to be cleared, indicating a failing program/erase; address locations being operated on by the aborted operation contain indeterminate data after an abort. A suspended operation cannot be aborted.</p> <p>Aborting a high voltage operation will leave the flash memory module addresses in an indeterminate data state. This may be recovered by executing an erase on the affected blocks.</p> <p>EHV may be written during Suspend. EHV must be high to exit Suspend. EHV may not be written after ESUS is set and before DONE transitions high. EHV may not be cleared after ESUS is cleared and before DONE transitions low.</p> <p>0 Flash memory is not enabled to perform an high voltage operation.<br/>1 Flash memory is enabled to perform an high voltage operation.</p> |

**Table 30-10. Array space size**

| SIZE | Array space size   |
|------|--------------------|
| 000  | 128 KB             |
| 001  | 256 KB             |
| 010  | 512 KB             |
| 011  | 1024 KB            |
| 100  | 1536 KB            |
| 101  | Reserved (2048 KB) |

**Table 30-10. Array space size (continued)**

| SIZE | Array space size |
|------|------------------|
| 110  | 64 KB            |
| 111  | Reserved         |

**Table 30-11. Low address space configuration**

| LAS | Low address space sectorization        |
|-----|--|
| 000 | Reserved                               |
| 001 | Reserved                               |
| 010 | 32 KB + 2 × 16 KB + 2 × 32 KB + 128 KB |
| 011 | Reserved                               |
| 100 | Reserved                               |
| 101 | Reserved                               |
| 110 | 4 × 16 KB                              |
| 111 | Reserved                               |

**Table 30-12. Mid address space configuration**

| MAS | Mid address space sectorization |
|-----|---------------------------------|
| 0   | 2 × 128 KB or 0 KB              |
| 1   | Reserved                        |

A number of CFLASH\_MCR bits are protected against write when another bit, or set of bits, is in a specific state. These write locks are covered on a bit by bit basis in the preceding description, but those locks do not consider the effects of trying to write two or more bits simultaneously.

The flash memory module does not allow the user to write bits simultaneously, which would put the device into an illegal state. This is implemented through a priority mechanism among the bits. The bit changing priorities are detailed in [Table 30-13](#).

**Table 30-13. CFLASH\_MCR bits set/clear priority levels**

| Priority level | CFLASH_MCR bits |
|----------------|-----------------|
| 1              | ERS             |
| 2              | PGM             |
| 3              | EHV             |
| 4              | ESUS            |

If the user attempts to write two or more CFLASH\_MCR bits simultaneously then only the bit with the lowest priority level is written.

If Stall/Abort-While-Write is enabled and an erase operation is started on one sector while fetching code from another, then the following sequence is executed:

1. CPU is stalled when flash is unavailable.
2. PEG flag set (stall case) or reset (abort case).
3. Interrupt triggered if enabled.

If Stall/Abort-While-Write is used, then application software should ignore the setting of the RWE flag. The RWE flag should be cleared after each HV operation.

If Stall/Abort-While-Write is not used the application software should handle RWE error. See [Section 30.8.10, Read-while-write functionality](#).

### 30.5.1.2 CFlash Low/Mid Address Space Block Locking Register (CFLASH\_LML)

The CFlash Low/Mid Address Space Block Locking register provides a means to protect blocks from being modified. These bits, along with bits in the CFLASH\_SLL register, determine if the block is locked from program or erase. An OR of CFLASH\_LML and CFLASH\_SLL determines the final lock status.

Offset: 0x0004

Access: Read/write

|       | 0   | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11   | 12 | 13 | 14 | 15  |
|-------|---|---|---|---|---|---|---|---|---|---|----|------|----|----|----|-----|
| R     | LME   | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  |      | 0  | 0  |    |     |
| W     |   |   |   |   |   |   |   |   |   |   |    | TSLK |    |    |    | MLK |
| Reset | Defined by CFLASH_NVLML at CFlash Test sector address 0x403DE8. This location is user OTP (One-Time Programmable). The CFLASH_NVLML register influences only the R/W bits of the CFLASH_LML register. |   |   |   |   |   |   |   |   |   |    |      |    |    |    |     |

|       | 16  | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26  | 27 | 28 | 29 | 30 | 31 |
|-------|---|----|----|----|----|----|----|----|----|----|-----|----|----|----|----|----|
| R     | 0   | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | LLK |    |    |    |    |    |
| W     |   |    |    |    |    |    |    |    |    |    |     |    |    |    |    |    |
| Reset | Defined by CFLASH_NVLML at CFlash Test sector address 0x403DE8. This location is user OTP (One-Time Programmable). The CFLASH_NVLML register influences only the R/W bits of the CFLASH_LML register. |    |    |    |    |    |    |    |    |    |     |    |    |    |    |    |

**Figure 30-4. CFlash Low/Mid Address Space Block Locking Register (CFLASH\_LML)**

Table 30-14. CFLASH\_LML field descriptions

| Field | Description  |
|-------|--|
| LME   | <p><i>Low/Mid Address Space Block Enable</i></p> <p>This bit is used to enable the Lock registers (TSLK, MLK1-0 and LLK15-0) to be set or cleared by registers writes.</p> <p>This bit is a status bit only. The method to set this bit is to write a password, and if the password matches, the LME bit will be set to reflect the status of enabled, and is enabled until a reset operation occurs. For LME the password 0xA1A11111 must be written to the CFLASH_LML register.</p> <p>0 Low Address Locks are disabled: TSLK, MLK1-0 and LLK15-0 cannot be written.<br/> 1 Low Address Locks are enabled: TSLK, MLK1-0 and LLK15-0 can be written.</p>  |
| TSLK  | <p><i>Test/Shadow Address Space Block Lock</i></p> <p>This bit is used to lock the block of Test and Shadow address space from program and erase (erase is any case forbidden for Test block).</p> <p>A value of 1 in the TSLK register signifies that the Test/shadow sector is locked for program and erase.</p> <p>A value of 0 in the TSLK register signifies that the Test/shadow sector is available to receive program and erase pulses.</p> <p>The TSLK register is not writable once an interlock write is completed until CFLASH_MCR[DONE] is set at the completion of the requested operation. Likewise, the TSLK register is not writable if a high voltage operation is suspended.</p> <p>Upon reset, information from the TestFlash block is loaded into the TSLK register. The TSLK bit may be written as a register. Reset will cause the bit to go back to its TestFlash block value. The default value of the TSLK bit (assuming erased fuses) would be locked.</p> <p>TSLK is not writable unless LME is high.</p> <p>0 Test/Shadow address space block is unlocked and can be modified (also if CFLASH_SLL[STSLK] = 0).<br/> 1 Test/Shadow address space block is locked and cannot be modified.</p> |

Table 30-14. CFLASH\_LML field descriptions (continued)

| Field | Description  |
|-------|--|
| MLK   | <p><i>Mid Address Space Block Lock</i></p> <p>This field is used to lock the blocks of Mid address space from program and erase. MLK is related to sectors B0F7-6, respectively.</p> <p>A value of 1 in a bit of the MLK field signifies that the corresponding block is locked for program and erase.</p> <p>A value of 0 in a bit of the MLK field signifies that the corresponding block is available to receive program and erase pulses.</p> <p>The MLK field is not writable after an interlock write is completed until CFLASH_MCR[DONE] is set at the completion of the requested operation. Likewise, the MLK field is not writable if a high voltage operation is suspended.</p> <p>Upon reset, information from the TestFlash block is loaded into the MLK field. The MLK field may be written as a register. Reset will cause the bits to go back to their TestFlash block value. The default value of the MLK field (assuming erased fuses) would be locked.</p> <p>In the event that blocks are not present (due to configuration or total memory size), the MLK field will default to locked, and will not be writable. The reset value will always be 1 (independent of the TestFlash block), and register writes will have no effect.</p> <p>MLK is not writable unless LME is high.</p> <p>0 Mid address space block is unlocked and can be modified (also if CFLASH_SLL[SMLK] = 0).</p> <p>1 Mid address space block is locked and cannot be modified.</p>  |
| LLK   | <p><i>Low Address Space Block Lock</i></p> <p>This field is used to lock the blocks of low address space from program and erase. LLK[5:0] are related to sectors B0F5-0, respectively. LLK[15:6] are not used for this memory cut.</p> <p>A value of 1 in a bit of the LLK field signifies that the corresponding block is locked for program and erase.</p> <p>A value of 0 in a bit of the LLK field signifies that the corresponding block is available to receive program and erase pulses.</p> <p>The LLK field is not writable after an interlock write is completed until CFLASH_MCR[DONE] is set at the completion of the requested operation. Likewise, the LLK field is not writable if a high voltage operation is suspended.</p> <p>Upon reset, information from the TestFlash block is loaded into the LLK field. The LLK field may be written as a register. Reset will cause the field to go back to its TestFlash block value. The default value of the LLK field (assuming erased fuses) would be locked.</p> <p>In the event that blocks are not present (due to configuration or total memory size), the LLK field will default to locked, and will not be writable. The reset value will always be 1 (independent of the TestFlash block), and register writes will have no effect.</p> <p>Bits LLK[15:6] are read-only and locked at 1.</p> <p>LLK is not writable unless LME is high.</p> <p>0 Low address space block is unlocked and can be modified (also if CFLASH_SLL[SLK] = 0).</p> <p>1 Low address space block is locked and cannot be modified.</p> |

### 30.5.1.2.1 CFlash Nonvolatile Low/Mid Address Space Block Locking Register (CFLASH\_NVLML)

The CFLASH\_LML register has a related CFlash Nonvolatile Low/Mid Address Space Block Locking register located in TestFlash that contains the default reset value for CFLASH\_LML. During the reset phase of the flash memory module, the CFLASH\_NVLML register content is read and loaded into the CFLASH\_LML.

The CFLASH\_NVLMML register is a 64-bit register, of which the 32 most significant bits 63:32 are “don’t care” and are used to manage ECC codes.

Offset: 0x403DE8

Access: Read/write

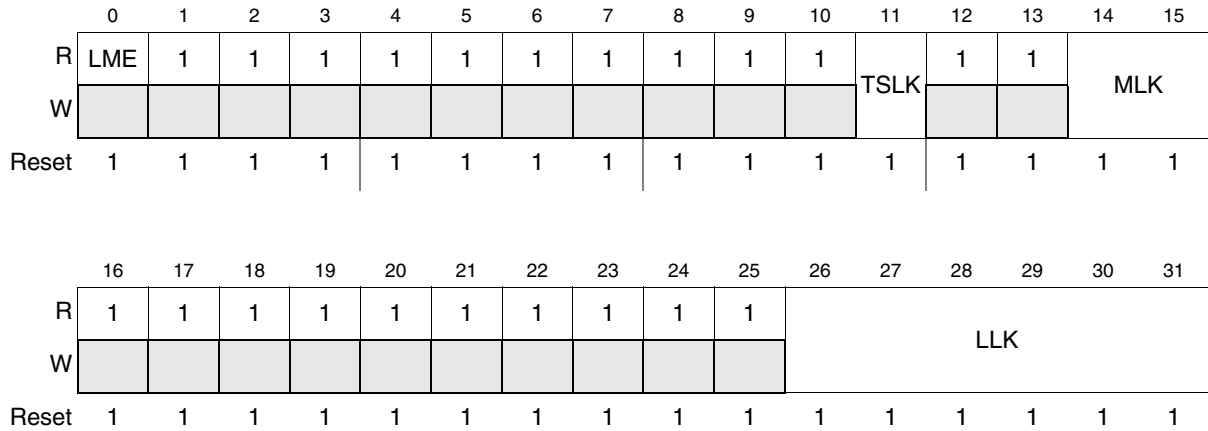


Figure 30-5. CFlash Nonvolatile Low/Mid address space block Locking register (CFLASH\_NVLMML)

Table 30-15. CFLASH\_NVLMML field descriptions

| Field | Description  |
|-------|--|
| LME   | <p><i>Low/Mid Address Space Block Enable</i></p> <p>This bit is used to enable the Lock registers (TSLK, MLK1-0 and LLK15-0) to be set or cleared by registers writes.</p> <p>This bit is a status bit only. The method to set this bit is to write a password, and if the password matches, the LME bit will be set to reflect the status of enabled, and is enabled until a reset operation occurs. For LME the password 0xA1A11111 must be written to the CFLASH_LML register.</p> <p>0 Low Address Locks are disabled: TSLK, MLK1-0 and LLK15-0 cannot be written.<br/>                     1 Low Address Locks are enabled: TSLK, MLK1-0 and LLK15-0 can be written.</p>  |
| TSLK  | <p><i>Test/Shadow Address Space Block Lock</i></p> <p>This bit is used to lock the block of Test and Shadow address space from program and erase (erase is any case forbidden for Test block).</p> <p>A value of 1 in the TSLK register signifies that the Test/shadow sector is locked for program and erase.</p> <p>A value of 0 in the TSLK register signifies that the Test/shadow sector is available to receive program and erase pulses.</p> <p>The TSLK register is not writable once an interlock write is completed until CFLASH_MCR[DONE] is set at the completion of the requested operation. Likewise, the TSLK register is not writable if a high voltage operation is suspended.</p> <p>Upon reset, information from the TestFlash block is loaded into the TSLK register. The TSLK bit may be written as a register. Reset will cause the bit to go back to its TestFlash block value. The default value of the TSLK bit (assuming erased fuses) would be locked.</p> <p>TSLK is not writable unless LME is high.</p> <p>0 Test/Shadow address space block is unlocked and can be modified (also if CFLASH_SLL[STSLK] = 0).<br/>                     1 Test/Shadow Address space block is locked and cannot be modified.</p> |



Table 30-15. CFLASH\_NVLMML field descriptions (continued)

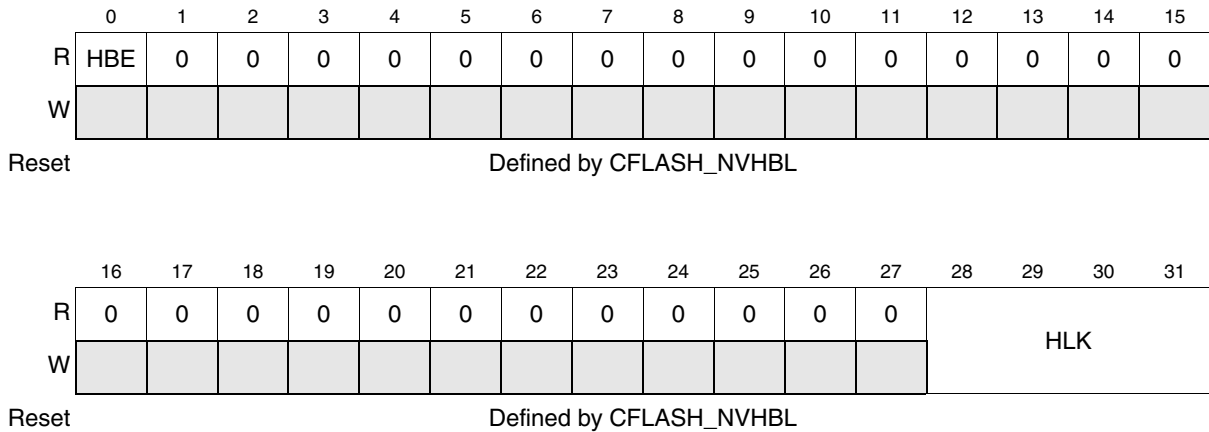
| Field | Description  |
|-------|--|
| MLK   | <p><i>Mid Address Space Block Lock</i></p> <p>These bits are used to lock the blocks of mid address space from program and erase. MLK[1:0] are related to sectors B0F7-6, respectively.</p> <p>A value of 1 in a bit of the MLK register signifies that the corresponding block is locked for program and erase.</p> <p>A value of 0 in a bit of the MLK register signifies that the corresponding block is available to receive program and erase pulses.</p> <p>The MLK register is not writable once an interlock write is completed until CFLASH_MCR[DONE] is set at the completion of the requested operation. Likewise, the MLK register is not writable if a high voltage operation is suspended.</p> <p>Upon reset, information from the TestFlash block is loaded into the MLK registers. The MLK bits may be written as a register. Reset will cause the bits to go back to their TestFlash block value. The default value of the MLK bits (assuming erased fuses) would be locked.</p> <p>In the event that blocks are not present (due to configuration or total memory size), the MLK bits will default to locked, and will not be writable. The reset value will always be 1 (independent of the TestFlash block), and register writes will have no effect.</p> <p>MLK is not writable unless LME is high.</p> <p>0 Mid address space block is unlocked and can be modified (also if CFLASH_SLL[SMLK] = 0).</p> <p>1 Mid address space block is locked and cannot be modified.</p>   |
| LLK   | <p><i>Low Address Space Block Lock</i></p> <p>These bits are used to lock the blocks of Low address space from program and erase. LLK[5:0] are related to sectors B0F5-0, respectively. LLK[15:6] are not used for this memory cut.</p> <p>A value of 1 in a bit of the LLK register signifies that the corresponding block is locked for program and erase.</p> <p>A value of 0 in a bit of the LLK register signifies that the corresponding block is available to receive program and erase pulses.</p> <p>The LLK register is not writable once an interlock write is completed until CFLASH_MCR[DONE] is set at the completion of the requested operation. Likewise, the LLK register is not writable if a high voltage operation is suspended.</p> <p>Upon reset, information from the TestFlash block is loaded into the LLK registers. The LLK bits may be written as a register. Reset will cause the bits to go back to their TestFlash block value. The default value of the LLK bits (assuming erased fuses) would be locked.</p> <p>In the event that blocks are not present (due to configuration or total memory size), the LLK bits will default to locked, and will not be writable. The reset value will always be 1 (independent of the TestFlash block), and register writes will have no effect.</p> <p>Bits LLK[15:6] are read-only and locked at 1.</p> <p>LLK is not writable unless LME is high.</p> <p>0 Low address space block is unlocked and can be modified (also if CFLASH_SLL[SLK] = 0).</p> <p>1 Low address space block is locked and cannot be modified.</p> |

### 30.5.1.3 CFlash High Address Space Block Locking Register (CFLASH\_HBL)

The CFLASH\_HBL register provides a means to protect blocks from being modified.

Offset: 0x0008

Access: Read/write



**Figure 30-6. CFlash Nonvolatile High Address Space Block Locking Register (CFLASH\_HBL)**

**Table 30-16. CFLASH\_HBL field descriptions**

| Field | Description   |
|-------|---|
| HBE   | <p><i>High Address Space Block Enable</i></p> <p>This bit is used to enable the Lock registers (HLK5-0) to be set or cleared by registers writes. This bit is a status bit only. The method to set this bit is to write a password, and if the password matches, the HBE bit will be set to reflect the status of enabled, and is enabled until a reset operation occurs. For HBE the password 0xB2B22222 must be written to the HBL register.</p> <p>0 High address locks are disabled: HLK5-0 cannot be written.<br/>                     1 High address locks are enabled: HLK5-0 can be written.</p>  |
| HLK   | <p><i>High Address Space Block Lock</i></p> <p>These bits are used to lock the blocks of High address space from program and erase. HLK11-8 are not used for this memory cut.</p> <p>A value of 1 in a bit of the HLK register signifies that the corresponding block is locked for program and erase.</p> <p>A value of 0 in a bit of the HLK register signifies that the corresponding block is available to receive program and erase pulses.</p> <p>The HLK register is not writable once an interlock write is completed until CFLASH_MCR[DONE] is set at the completion of the requested operation. Likewise, the HLK register is not writable if a high voltage operation is suspended.</p> <p>Upon reset, information from the TestFlash block is loaded into the HLK registers. The HLK bits may be written as a register. Reset will cause the bits to go back to their TestFlash block value. The default value of the HLK bits (assuming erased fuses) would be locked.</p> <p>In the event that blocks are not present (due to configuration or total memory size), the HLK bits will default to locked, and will not be writable. The reset value will always be 1 (independent of the TestFlash block), and register writes will have no effect.</p> <p>In the 544 KB Flash module bits HLK11-8 are read-only and locked at 1.</p> <p>HLK is not writable unless HBE is high.</p> <p>0 High address space block is unlocked and can be modified.<br/>                     1 High address space block is locked and cannot be modified.</p> |

### 30.5.1.3.1 CFlash Nonvolatile High Address Space Block Locking Register (CFLASH\_NVHBL)

The CFLASH\_HBL register has a related Nonvolatile High Address Space Block Locking register located in TestFlash that contains the default reset value for CFLASH\_HBL. During the reset phase of the Flash module, the CFLASH\_NVHBL register content is read and loaded into the CFLASH\_HBL.

The CFLASH\_NVHBL register is a 64-bit register, of which the 32 most significant bits 63:32 are “don’t care” and eventually used to manage ECC codes.

Offset: 0x403DF0

Access: Read/write

|       | 0   | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|-------|-----|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
| R     | HBE | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |     |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0   | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28  | 29 | 30 | 31 |
|-------|----|----|----|----|----|----|----|----|----|----|----|----|-----|----|----|----|
| R     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | HLK |    |    |    |
| W     |    |    |    |    |    |    |    |    |    |    |    |    |     |    |    |    |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | X   | X  | X  | X  |

Figure 30-7. CFlash Nonvolatile High Address Space Block Locking Register (CFLASH\_NVHBL)

Table 30-17. CFLASH\_NVHBL field descriptions

| Field | Description   |
|-------|---|
| HBE   | <p><i>High Address Space Block Enable</i></p> <p>This bit is used to enable the Lock registers (HLK5-0) to be set or cleared by registers writes. This bit is a status bit only. The method to set this bit is to write a password, and if the password matches, the HBE bit will be set to reflect the status of enabled, and is enabled until a reset operation occurs. For HBE the password 0xB2B22222 must be written to the HBL register.</p> <p>0 High Address Locks are disabled: HLK5-0 cannot be written.<br/> 1 High Address Locks are enabled: HLK5-0 can be written.</p>  |
| HLK   | <p><i>High Address Space Block Lock</i></p> <p>These bits are used to lock the blocks of high address space from program and erase. HLK11-8 are not used for this memory cut.</p> <p>A value of 1 in a bit of the HLK register signifies that the corresponding block is locked for program and erase.</p> <p>A value of 0 in a bit of the HLK register signifies that the corresponding block is available to receive program and erase pulses.</p> <p>The HLK register is not writable once an interlock write is completed until CFLASH_MCR[DONE] is set at the completion of the requested operation. Likewise, the HLK register is not writable if a high voltage operation is suspended.</p> <p>Upon reset, information from the TestFlash block is loaded into the HLK registers. The HLK bits may be written as a register. Reset will cause the bits to go back to their TestFlash block value. The default value of the HLK bits (assuming erased fuses) would be locked.</p> <p>In the event that blocks are not present (due to configuration or total memory size), the HLK bits will default to locked, and will not be writable. The reset value will always be 1 (independent of the TestFlash block), and register writes will have no effect.</p> <p>In the 544 KB Flash module bits HLK11-8 are read-only and locked at 1.</p> <p>HLK is not writable unless HBE is high.</p> <p>0 High address space block is unlocked and can be modified.<br/> 1 High address space block is locked and cannot be modified.</p> |

#### 30.5.1.4 CFlash Secondary Low/Mid Address Space Block Locking Register (CFLASH\_SLL)

The CFlash Secondary Low/Mid Address Space Block Locking Register provides an alternative means to protect blocks from being modified. These bits, along with bits in the CFLASH\_LML register, determine if the block is locked from program or erase. An OR of CFLASH\_LML and CFLASH\_SLL determines the final lock status.

Offset: 0x000C

Access: Read/write

|   |     |   |   |   |   |   |   |   |   |   |    |       |    |    |     |    |
|---|-----|---|---|---|---|---|---|---|---|---|----|-------|----|----|-----|----|
|   | 0   | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11    | 12 | 13 | 14  | 15 |
| R | SLE | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | STSLK | 0  | 0  | SMK |    |
| W |     |   |   |   |   |   |   |   |   |   |    |       |    |    |     |    |

Reset Defined by CFLASH\_NVSLI at CFlash Test sector address 0x403DF8. This location is user OTP (One-Time Programmable). The CFLASH\_NVSLI register influences only the R/W bits of the CFLASH\_SLI register.

|   |    |    |    |    |    |    |    |    |    |    |     |    |    |    |    |    |
|---|----|----|----|----|----|----|----|----|----|----|-----|----|----|----|----|----|
|   | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26  | 27 | 28 | 29 | 30 | 31 |
| R | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | SLK |    |    |    |    |    |
| W |    |    |    |    |    |    |    |    |    |    |     |    |    |    |    |    |

Reset Defined by CFLASH\_NVSLI at CFlash Test sector address 0x403DF8. This location is user OTP (One-Time Programmable). The CFLASH\_NVSLI register influences only the R/W bits of the CFLASH\_SLI register.

**Figure 30-8. CFlash Secondary Low/mid address space block Locking Register (CFLASH\_SLI)**

**Table 30-18. CFLASH\_SLI field descriptions**

| Field | Description  |
|-------|--|
| SLE   | <p><i>Secondary Low/Mid Address Space Block Enable</i></p> <p>This bit is used to enable the Lock registers (STSLK, SMK1-0 and SLK15-0) to be set or cleared by registers writes.</p> <p>This bit is a status bit only. The method to set this bit is to write a password, and if the password matches, the SLE bit will be set to reflect the status of enabled, and is enabled until a reset operation occurs. For SLE the password 0xC3C33333 must be written to the CFLASH_SLI register.</p> <p>0 Secondary Low/Mid Address Locks are disabled: STSLK, SMK1-0 and SLK15-0 cannot be written.</p> <p>1 Secondary Low/Mid Address Locks are enabled: STSLK, SMK1-0 and SLK15-0 can be written.</p>   |
| STSLK | <p><i>Secondary Test/Shadow Address Space Block Lock</i></p> <p>This bit is used as an alternate means to lock the block of Test and Shadow address space from program and erase (erase is any case forbidden for Test block).</p> <p>A value of 1 in the STSLK register signifies that the Test/shadow sector is locked for program and erase.</p> <p>A value of 0 in the STSLK register signifies that the Test/shadow sector is available to receive program and erase pulses.</p> <p>The STSLK register is not writable once an interlock write is completed until CFLASH_MCR[DONE] is set at the completion of the requested operation. Likewise, the STSLK register is not writable if a high voltage operation is suspended.</p> <p>Upon reset, information from the TestFlash block is loaded into the STSLK register. The STSLK bit may be written as a register. Reset will cause the bit to go back to its TestFlash block value. The default value of the STSLK bit (assuming erased fuses) would be locked. STSLK is not writable unless SLE is high.</p> <p>0 Test/Shadow address space block is unlocked and can be modified (also if CFLASH_LML[TSLK] = 0).</p> <p>1 Test/Shadow address space block is locked and cannot be modified.</p> |

Table 30-18. CFLASH\_SLL field descriptions (continued)

| Field | Description   |
|-------|---|
| SMK   | <p><i>Secondary Mid Address Space Block Lock</i></p> <p>These bits are used as an alternate means to lock the blocks of mid address space from program and erase.</p> <p>SMK[1:0] are related to sectors B0F7-6, respectively.</p> <p>A value of 1 in a bit of the SMK register signifies that the corresponding block is locked for program and erase.</p> <p>A value of 0 in a bit of the SMK register signifies that the corresponding block is available to receive program and erase pulses.</p> <p>The SMK register is not writable once an interlock write is completed until CFLASH_MCR[DONE] is set at the completion of the requested operation. Likewise, the SMK register is not writable if a high voltage operation is suspended.</p> <p>Upon reset, information from the TestFlash block is loaded into the SMK registers. The SMK bits may be written as a register. Reset will cause the bits to go back to their TestFlash block value. The default value of the SMK bits (assuming erased fuses) would be locked.</p> <p>In the event that blocks are not present (due to configuration or total memory size), the SMK bits will default to locked, and will not be writable. The reset value will always be 1 (independent of the TestFlash block), and register writes will have no effect.</p> <p>SMK is not writable unless SLE is high.</p> <p>0 Mid address space block is unlocked and can be modified (also if CFLASH_LML[MLK] = 0).</p> <p>1 Mid address space block is locked and cannot be modified.</p>  |
| SLK   | <p><i>Secondary Low Address Space Block Lock</i></p> <p>These bits are used as an alternate means to lock the blocks of low address space from program and erase.</p> <p>SLK[5:0] are related to sectors B0F5-0, respectively. SLK[15:6] are not used for this memory cut.</p> <p>A value of 1 in a bit of the SLK register signifies that the corresponding block is locked for program and erase.</p> <p>A value of 0 in a bit of the SLK register signifies that the corresponding block is available to receive program and erase pulses.</p> <p>The SLK register is not writable once an interlock write is completed until CFLASH_MCR[DONE] is set at the completion of the requested operation. Likewise, the SLK register is not writable if a high voltage operation is suspended.</p> <p>Upon reset, information from the TestFlash block is loaded into the SLK registers. The SLK bits may be written as a register. Reset will cause the bits to go back to their TestFlash block value. The default value of the SLK bits (assuming erased fuses) would be locked.</p> <p>In the event that blocks are not present (due to configuration or total memory size), the SLK bits will default to locked, and will not be writable. The reset value will always be 1 (independent of the TestFlash block), and register writes will have no effect.</p> <p>Bits SLK[15:6] are read-only and locked at 1.</p> <p>SLK is not writable unless SLE is high.</p> <p>0 Low address space block is unlocked and can be modified (also if CFLASH_LML[LLK] = 0).</p> <p>1 Low address space block is locked and cannot be modified.</p> |

### 30.5.1.4.1 CFlash Nonvolatile Secondary Low/Mid Address Space Block Locking Register (CFLASH\_NVSLL)

The CFLASH\_SLL register has a related Nonvolatile Secondary Low/Mid Address Space Block Locking register located in TestFlash that contains the default reset value for SLL. During the reset phase of the flash memory module, the CFLASH\_NVSLL register content is read and loaded into the CFLASH\_SLL.

The CFLASH\_NVSLL register is a 64-bit register, of which the 32 most significant bits 63:32 are “don’t care” and are used to manage ECC codes.

Offset: 0x403DF8

Access: Read/write

|       | 0   | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11    | 12 | 13 | 14  | 15 |
|-------|-----|---|---|---|---|---|---|---|---|---|----|-------|----|----|-----|----|
| R     | SLE | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1  | STSLK | 1  | 1  | SMK |    |
| W     |     |   |   |   |   |   |   |   |   |   |    |       |    |    |     |    |
| Reset | 1   | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1  | 1     | 1  | 1  | 1   | 1  |

|       | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26  | 27 | 28 | 29 | 30 | 31 |
|-------|----|----|----|----|----|----|----|----|----|----|-----|----|----|----|----|----|
| R     | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | SLK |    |    |    |    |    |
| W     |    |    |    |    |    |    |    |    |    |    |     |    |    |    |    |    |
| Reset | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1   | 1  | 1  | 1  | 1  | 1  |

**Figure 30-9. CFlash Nonvolatile Secondary Low/mid address space block Locking register (CFLASH\_NVSLL)**

Table 30-19. CFLASH\_NVSLI field descriptions

| Field | Description  |
|-------|--|
| SLE   | <p><i>Secondary Low/Mid Address Space Block Enable</i></p> <p>This bit is used to enable the Lock registers (STSLK, SMK1-0 and SLK15-0) to be set or cleared by registers writes.</p> <p>This bit is a status bit only. The method to set this bit is to write a password, and if the password matches, the SLE bit will be set to reflect the status of enabled, and is enabled until a reset operation occurs. For SLE the password 0xC3C33333 must be written to the CFLASH_SLL register.</p> <p>0 Secondary Low/Mid Address Locks are disabled: STSLK, SMK1-0 and SLK15-0 cannot be written.</p> <p>1 Secondary Low/Mid Address Locks are enabled: STSLK, SMK1-0 and SLK15-0 can be written.</p>   |
| STSLK | <p><i>Secondary Test/Shadow Address Space Block Lock</i></p> <p>This bit is used as an alternate means to lock the block of Test and Shadow address space from program and erase (erase is any case forbidden for Test block).</p> <p>A value of 1 in the STSLK register signifies that the Test/shadow sector is locked for program and erase.</p> <p>A value of 0 in the STSLK register signifies that the Test/shadow sector is available to receive program and erase pulses.</p> <p>The STSLK register is not writable once an interlock write is completed until CFLASH_MCR[DONE] is set at the completion of the requested operation. Likewise, the STSLK register is not writable if a high voltage operation is suspended.</p> <p>Upon reset, information from the TestFlash block is loaded into the STSLK register. The STSLK bit may be written as a register. Reset will cause the bit to go back to its TestFlash block value. The default value of the STSLK bit (assuming erased fuses) would be locked. STSLK is not writable unless SLE is high.</p> <p>0 Test/Shadow address space block is unlocked and can be modified (also if CFLASH_LML[TSLK] = 0).</p> <p>1 Test/Shadow address space block is locked and cannot be modified.</p> |



Table 30-19. CFLASH\_NVSL field descriptions (continued)

| Field | Description   |
|-------|---|
| SMK   | <p><i>Secondary Mid Address Space Block Lock</i></p> <p>These bits are used as an alternate means to lock the blocks of mid address space from program and erase.</p> <p>SMK[1:0] are related to sectors B0F7-6, respectively.</p> <p>A value of 1 in a bit of the SMK register signifies that the corresponding block is locked for program and erase.</p> <p>A value of 0 in a bit of the SMK register signifies that the corresponding block is available to receive program and erase pulses.</p> <p>The SMK register is not writable once an interlock write is completed until CFLASH_MCR[<i>DONE</i>] is set at the completion of the requested operation. Likewise, the SMK register is not writable if a high voltage operation is suspended.</p> <p>Upon reset, information from the TestFlash block is loaded into the SMK registers. The SMK bits may be written as a register. Reset will cause the bits to go back to their TestFlash block value. The default value of the SMK bits (assuming erased fuses) would be locked.</p> <p>In the event that blocks are not present (due to configuration or total memory size), the SMK bits will default to locked, and will not be writable. The reset value will always be 1 (independent of the TestFlash block), and register writes will have no effect.</p> <p>SMK is not writable unless SLE is high.</p> <p>0 Mid address space block is unlocked and can be modified (also if CFLASH_LML[<i>MLK</i>] = 0).</p> <p>1 Mid address space block is locked and cannot be modified.</p>  |
| SLK   | <p><i>Secondary Low Address Space Block Lock</i></p> <p>These bits are used as an alternate means to lock the blocks of low address space from program and erase.</p> <p>SLK[5:0] are related to sectors B0F5-0, respectively. SLK[15:6] are not used for this memory cut.</p> <p>A value of 1 in a bit of the SLK register signifies that the corresponding block is locked for program and erase.</p> <p>A value of 0 in a bit of the SLK register signifies that the corresponding block is available to receive program and erase pulses.</p> <p>The SLK register is not writable once an interlock write is completed until CFLASH_MCR[<i>DONE</i>] is set at the completion of the requested operation. Likewise, the SLK register is not writable if a high voltage operation is suspended.</p> <p>Upon reset, information from the TestFlash block is loaded into the SLK registers. The SLK bits may be written as a register. Reset will cause the bits to go back to their TestFlash block value. The default value of the SLK bits (assuming erased fuses) would be locked.</p> <p>In the event that blocks are not present (due to configuration or total memory size), the SLK bits will default to locked, and will not be writable. The reset value will always be 1 (independent of the TestFlash block), and register writes will have no effect.</p> <p>Bits SLK[15:6] are read-only and locked at 1.</p> <p>SLK is not writable unless SLE is high.</p> <p>0 Low address space block is unlocked and can be modified (also if CFLASH_LML[<i>LLK</i>] = 0).</p> <p>1 Low address space block is locked and cannot be modified.</p> |

### 30.5.1.5 CFlash Low/Mid Address Space Block Select Register (CFLASH\_LMS)

Offset: 0x00010

Access: Read/write

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |     |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|-----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14  | 15 |
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | MSL |    |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |     |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0   | 0  |

|       |    |    |    |    |    |    |    |    |    |    |     |    |    |    |    |    |
|-------|----|----|----|----|----|----|----|----|----|----|-----|----|----|----|----|----|
|       | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26  | 27 | 28 | 29 | 30 | 31 |
| R     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | LSL |    |    |    |    |    |
| W     |    |    |    |    |    |    |    |    |    |    |     |    |    |    |    |    |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 0  | 0  | 0  | 0  | 0  |

**Figure 30-10. CFlash Low/Mid address space block Select register (CFLASH\_LMS)**

The CFLASH\_LMS register provides a means to select blocks to be operated on during erase.

**Table 30-20. CFLASH\_LMS field descriptions**

| Field | Description  |
|-------|--|
| MSL   | <p><i>Mid Address Space Block Select</i></p> <p>A value of 1 in the select register signifies that the block is selected for erase. A value of 0 in the select register signifies that the block is not selected for erase. The reset value for the select register is 0, or not selected.</p> <p>MSL[1:0] are related to sectors B0F7-6, respectively.</p> <p>The blocks must be selected (or deselected) before doing an erase interlock write as part of the erase sequence. The select register is not writable once an interlock write is completed or if a high voltage operation is suspended.</p> <p>In the event that blocks are not present (due to configuration or total memory size), the corresponding MSL bits will default to not selected, and will not be writable. The reset value will always be 0, and register writes will have no effect.</p> <p>0 Mid address space block is not selected for erase.<br/>1 Mid address space block is selected for erase.</p>  |
| LSL   | <p><i>Low Address Space Block Select</i></p> <p>A value of 1 in the select register signifies that the block is selected for erase. A value of 0 in the select register signifies that the block is not selected for erase. The reset value for the select register is 0, or not selected.</p> <p>LSL[5:0] are related to sectors B0F5-0, respectively. LSL[15:6] are not used for this memory cut.</p> <p>The blocks must be selected (or deselected) before doing an erase interlock write as part of the erase sequence. The select register is not writable once an interlock write is completed or if a high voltage operation is suspended.</p> <p>In the event that blocks are not present (due to configuration or total memory size), the corresponding LSL bits will default to not selected, and will not be writable. The reset value will always be 0, and register writes will have no effect.</p> <p>Bits LSL[15:6] are read-only and locked at 0.</p> <p>0 Low address space block is not selected for erase.<br/>1 Low address space block is selected for erase.</p> |

### 30.5.1.6 CFlash High Address Space Block Select Register (CFLASH\_HBS)

The CFLASH\_HBS register provides a means to select blocks to be operated on during erase.

Offset: 0x00014

Access: Read/write

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       |    |    |    |    |    |    |    |    |    |    |    |    |     |    |    |    |
|-------|----|----|----|----|----|----|----|----|----|----|----|----|-----|----|----|----|
|       | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28  | 29 | 30 | 31 |
| R     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | HSL |    |    |    |
| W     |    |    |    |    |    |    |    |    |    |    |    |    | HSL |    |    |    |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 0  | 0  | 0  |

Figure 30-11. CFlash High Address Space Block Select Register (CFLASH\_HBS)

Table 30-21. CFLASH\_HBS field descriptions

| Field | Description  |
|-------|--|
| HSL   | <p><i>High Address Space Block Select</i></p> <p>A value of 1 in the select register signifies that the block is selected for erase. A value of 0 in the select register signifies that the block is not selected for erase. The reset value for the select register is 0, or not selected.</p> <p>HSL11-8 are not used for this memory cut.</p> <p>The blocks must be selected (or deselected) before doing an erase interlock write as part of the erase sequence. The select register is not writable once an interlock write is completed or if a high voltage operation is suspended.</p> <p>In the event that blocks are not present (due to configuration or total memory size), the corresponding HSL bits will default to not selected, and will not be writable. The reset value will always be 0, and register writes will have no effect.</p> <p>In the 544 KB Flash module, bits HSL11-8 are read-only and locked at 0.</p> <p>0 High address space block is not selected for erase.<br/> 1 High address space block is selected for erase.</p> |

### 30.5.1.7 CFlash Address Register (CFLASH\_ADR)

Offset: 0x00018

Access: Read

|       |   |   |   |   |   |   |   |   |   |      |      |      |      |      |      |      |
|-------|---|---|---|---|---|---|---|---|---|------|------|------|------|------|------|------|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9    | 10   | 11   | 12   | 13   | 14   | 15   |
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | AD22 | AD21 | AD20 | AD19 | AD18 | AD17 | AD16 |
| W     |   |   |   |   |   |   |   |   |   |      |      |      |      |      |      |      |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0    | 0    | 0    | 0    | 0    | 0    | 0    |

|       |      |      |      |      |      |      |     |     |     |     |     |     |     |    |    |    |
|-------|------|------|------|------|------|------|-----|-----|-----|-----|-----|-----|-----|----|----|----|
|       | 16   | 17   | 18   | 19   | 20   | 21   | 22  | 23  | 24  | 25  | 26  | 27  | 28  | 29 | 30 | 31 |
| R     | AD15 | AD14 | AD13 | AD12 | AD11 | AD10 | AD9 | AD8 | AD7 | AD6 | AD5 | AD4 | AD3 | 0  | 0  | 0  |
| W     |      |      |      |      |      |      |     |     |     |     |     |     |     |    |    |    |
| Reset | 1    | 1    | 0    | 0    | 0    | 0    | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0  | 0  | 0  |

**Figure 30-12. CFlash Address Register (CFLASH\_ADR)**

The CFLASH\_ADR provides the first failing address in the event module failures (ECC or FPEC) occur or the first address at which an ECC single error correction occurs.

**Table 30-22. CFLASH\_ADR field descriptions**

| Field | Description   |
|-------|---|
| AD    | <p><i>Address 22-3 (Read Only)</i></p> <p>The Address Register provides the first failing address in the event of ECC error (CFLASH_MCR[EER] = 1) or the first failing address in the event of RWW error (CFLASH_MCR[RWE] = 1), or the address of a failure that may have occurred in a FPEC operation (CFLASH_MCR[PEG] = 0). The Address Register also provides the first address at which an ECC single error correction occurs (CFLASH_MCR[EDC] = 1).</p> <p>The ECC double error detection takes the highest priority, followed by the FPEC error and the ECC single error correction. When accessed CFLASH_ADR will provide the address related to the first event occurred with the highest priority. The priorities between these four possible events is summarized in <a href="#">Table 30-23</a>.</p> <p>This address is always a double word address that selects 64 bits.</p> <p>In case of a simultaneous ECC double error detection on both double words of the same page, bit AD3 will output 0. The same is valid for a simultaneous ECC single error correction on both double words of the same page.</p> |

**Table 30-23. CFLASH\_ADR content: priority list**

| Priority level | Error flag          | CFLASH_ADR content                           |
|----------------|---------------------|--|
| 1              | CFLASH_MCR[EER] = 1 | Address of first ECC double error            |
| 2              | CFLASH_MCR[RWE] = 1 | Address of first RWW error                   |
| 3              | CFLASH_MCR[PEG] = 0 | Address of first FPEC error                  |
| 4              | CFLASH_MCR[EDC] = 1 | Address of first ECC single error correction |

### 30.5.1.8 CFlash User Test 0 register (CFLASH\_UT0)

The User Test Registers provide the user with the ability to test features on the flash memory module. The User Test 0 Register allows to control the way in which the flash memory content check is done.

Bits MRE, MRV, AIS, EIE, and DSI[7:0] of the User Test 0 Register are not accessible whenever CFLASH\_MCR[DONE] or UT0[AID] are low: reading returns indeterminate data while writing has no effect.

Offset: 0x0003C

Access: Read/write

|       |     |   |   |   |   |   |   |   |     |   |    |    |    |    |    |    |
|-------|-----|---|---|---|---|---|---|---|-----|---|----|----|----|----|----|----|
|       | 0   | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8   | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| R     | UTE | 0 | 0 | 0 | 0 | 0 | 0 | 0 | DSI |   |    |    |    |    |    |    |
| W     | w1c |   |   |   |   |   |   |   |     |   |    |    |    |    |    |    |
| Reset | 0   | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0   | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       |    |    |    |    |    |    |    |    |    |    |     |     |     |     |     |     |
|-------|----|----|----|----|----|----|----|----|----|----|-----|-----|-----|-----|-----|-----|
|       | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26  | 27  | 28  | 29  | 30  | 31  |
| R     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | X  | MRE | MRV | EIE | AIS | AIE | AID |
| W     |    |    |    |    |    |    |    |    |    |    |     |     |     |     |     |     |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 0   | 0   | 0   | 0   | 1   |

Figure 30-13. CFlash User Test 0 register (CFLASH\_UT0)

Table 30-24. CFLASH\_UT0 field descriptions

| Field | Description  |
|-------|--|
| UTE   | <i>User Test Enable</i><br>This status bit gives indication when User Test is enabled. All bits in CFLASH_UT0-2 and CFLASH_UMISR0-4 are locked when this bit is 0.<br>The method to set this bit is to provide a password, and if the password matches, the UTE bit is set to reflect the status of enabled, and is enabled until it is cleared by a register write. For UTE the password 0xF9F99999 must be written to the CFLASH_UT0 register. |
| DSI   | <i>Data Syndrome Input</i><br>These bits represent the input of Syndrome bits of ECC logic used in the ECC logic check. Bits DSI[7:0] correspond to the 8 syndrome bits on a double word.<br>These bits are not accessible whenever CFLASH_MCR[DONE] or CFLASH_UT0[AID] are low: reading returns indeterminate data while writing has no effect.<br>0 The syndrome bit is forced at 0.<br>1 The syndrome bit is forced at 1.                     |
| X     | <i>Reserved</i><br>This bit can be written and its value can be read back, but there is no function associated. This bit is not accessible whenever CFLASH_MCR[DONE] or CFLASH_UT0[AID] are low: reading returns indeterminate data while writing has no effect.   |

Table 30-24. CFLASH\_UT0 field descriptions (continued)

| Field | Description  |
|-------|--|
| MRE   | <p><i>Margin Read Enable</i></p> <p>MRE enables margin reads to be done. This bit, combined with MRV, enables regular User mode reads.</p> <p>to be replaced by margin reads inside the array integrity check sequences. Margin reads are only active during array integrity checks; normal User reads are not affected by MRE. This bit is not accessible whenever CFLASH_MCR[DONE] or CFLASH_UT0[AID] are low: reading returns indeterminate data while writing has no effect.</p> <p>0 Margin reads are not enabled<br/>1 Margin reads are enabled.</p>   |
| MRV   | <p><i>Margin Read Value</i></p> <p>If MRE is high, MRV selects the margin level that is being checked. Margin can be checked to an erased level (MRV = 1) or to a programmed level (MRV = 0).</p> <p>This bit is not accessible whenever CFLASH_MCR[DONE] or CFLASH_UT0[AID] are low: reading returns indeterminate data while writing has no effect.</p> <p>0 Zeros (programmed) margin reads are requested (if MRE = 1).<br/>1 Ones (erased) margin reads are requested (if MRE = 1).</p>  |
| EIE   | <p><i>ECC Data Input Enable</i></p> <p>EIE enables the ECC logic check operation to be done. This bit is not accessible whenever CFLASH_MCR[DONE] or CFLASH_UT0[AID] are low: reading returns indeterminate data while writing has no effect.</p> <p>0 ECC logic check is not enabled.<br/>1 ECC logic check is enabled.</p>   |
| AIS   | <p><i>Array Integrity Sequence</i></p> <p>AIS determines the address sequence to be used during array integrity checks or margin read. The default sequence (AIS=0) is meant to replicate sequences normal user code follows, and thoroughly checks the read propagation paths. This sequence is proprietary. The alternative sequence (AIS=1) is just logically sequential. It should be noted that the time to run a sequential sequence is significantly shorter than the time to run the proprietary sequence. The usage of proprietary sequence is forbidden in margin read. This bit is not accessible whenever CFLASH_MCR[DONE] or CFLASH_UT0[AID] are low: reading returns indeterminate data while writing has no effect.</p> <p>0 Array integrity sequence is proprietary sequence.<br/>1 Array integrity or margin read sequence is sequential.</p> |
| AIE   | <p><i>Array Integrity Enable</i></p> <p>AIE set to 1 starts the array integrity check done on all selected and unlocked blocks. The pattern is selected by AIS, and the MISR (CFLASH_UMISR0-4) can be checked after the operation is complete, to determine if a correct signature is obtained.</p> <p>AIE can be set only if CFLASH_MCR[ERS], CFLASH_MCR[PGM], and CFLASH_MCR[EHV] are all low.</p> <p>0 Array integrity checks, margin read, and ECC logic checks are not enabled.<br/>1 Array integrity checks, margin read, and ECC logic checks are enabled.</p>  |
| AID   | <p><i>Array Integrity Done</i></p> <p>AID will be cleared upon an array integrity check being enabled (to signify the operation is on-going).</p> <p>Once completed, AID will be set to indicate that the array integrity check is complete. At this time the MISR (CFLASH_UMISR0-4) can be checked.</p> <p>0 Array integrity check is on-going.<br/>1 Array integrity check is done.</p>  |

### 30.5.1.9 CFlash User Test 1 register (CFLASH\_UT1)

The CFLASH\_UT1 register allows to enable the checks on the ECC logic related to the 32 LSB of the double word.

The User Test 1 Register is not accessible whenever CFLASH\_MCR[DONE] or CFLASH\_UT0[AID] are low: reading returns indeterminate data while writing has no effect.

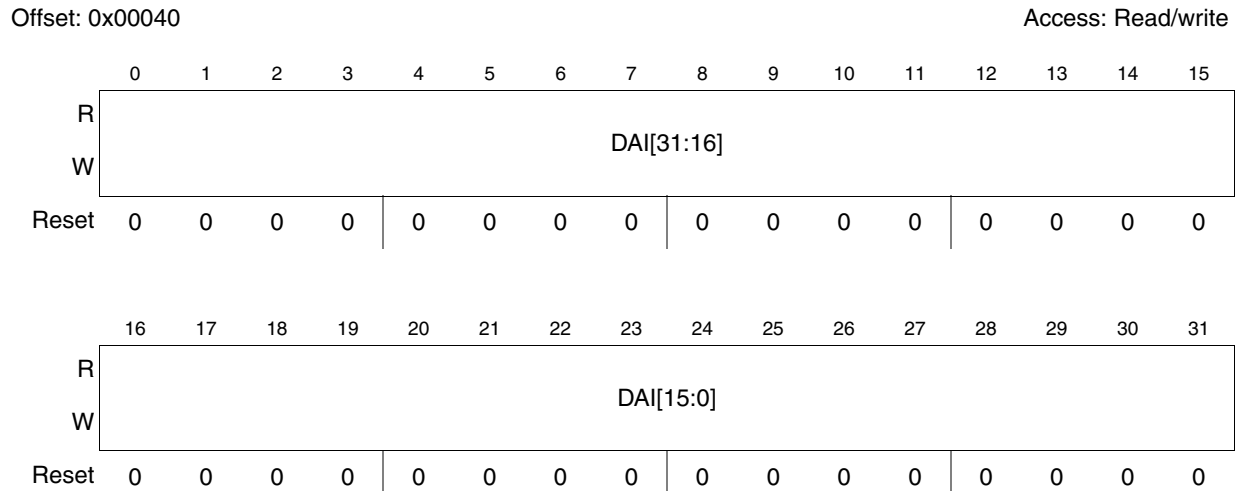


Figure 30-14. CFlash User Test 1 register (CFLASH\_UT1)

Table 30-25. CFLASH\_UT1 field descriptions

| Field     | Description   |
|-----------|---|
| DAI[31:0] | <i>Data Array Input, bits 31–0</i><br>These bits represent the input of even word of ECC logic used in the ECC logic check. Bits DAI[31:00] correspond to the 32 array bits representing Word 0 within the double word.<br>0 The array bit is forced at 0.<br>1 The array bit is forced at 1. |

### 30.5.1.10 CFlash User Test 2 register (CFLASH\_UT2)

The CFLASH\_UT2 register allows to enable the checks on the ECC logic related to the 32 MSB of the double word.

The User Test 2 Register is not accessible whenever CFLASH\_MCR[DONE] or CFLASH\_UT0[AID] are low: reading returns indeterminate data while writing has no effect.

Offset: 0x00044

Access: Read/write

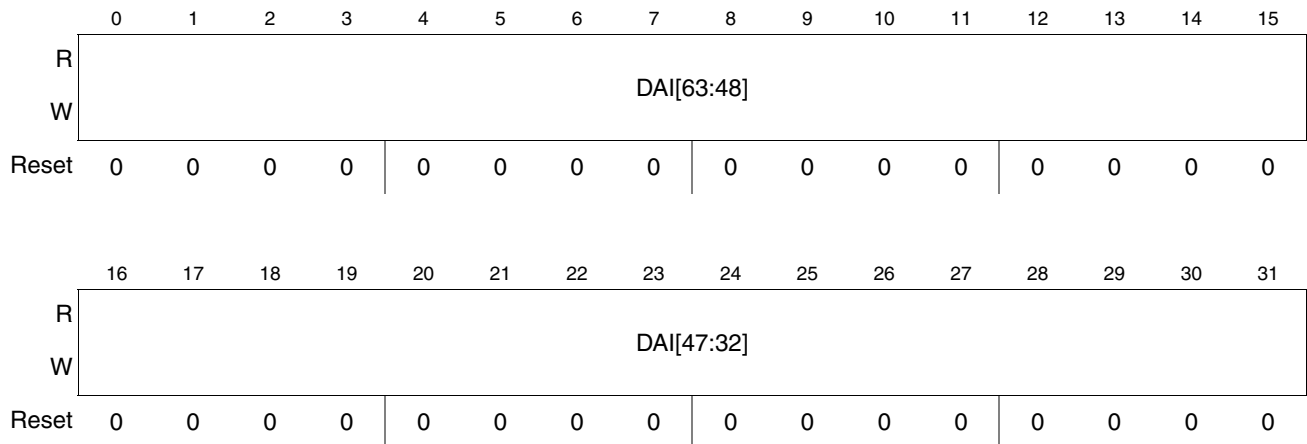


Figure 30-15. CFlash User Test 2 register (CFLASH\_UT2)

Table 30-26. CFLASH\_UT2 field descriptions

| Field      | Description   |
|------------|---|
| DAI[63:32] | <p><i>Data Array Input, bits 63–32</i></p> <p>These bits represent the input of odd word of ECC logic used in the ECC logic check. Bits DAI[63:32] correspond to the 32 array bits representing Word 1 within the double word.</p> <p>0 The array bit is forced at 0.<br/>1 The array bit is forced at 1.</p> |

### 30.5.1.11 CFlash User Multiple Input Signature Register 0 (CFLASH\_UMISR0)

The CFLASH\_UMISR0 register provides a mean to evaluate the array integrity.

The User Multiple Input Signature Register 0 represents the bits 31:0 of the whole 144 bits word (2 double words including ECC).

The CFLASH\_UMISR0 Register is not accessible whenever CFLASH\_MCR[DONE] or CFLASH\_UT0[AID] are low: reading returns indeterminate data while writing has no effect.



Offset: 0x00048

Access: Read/write

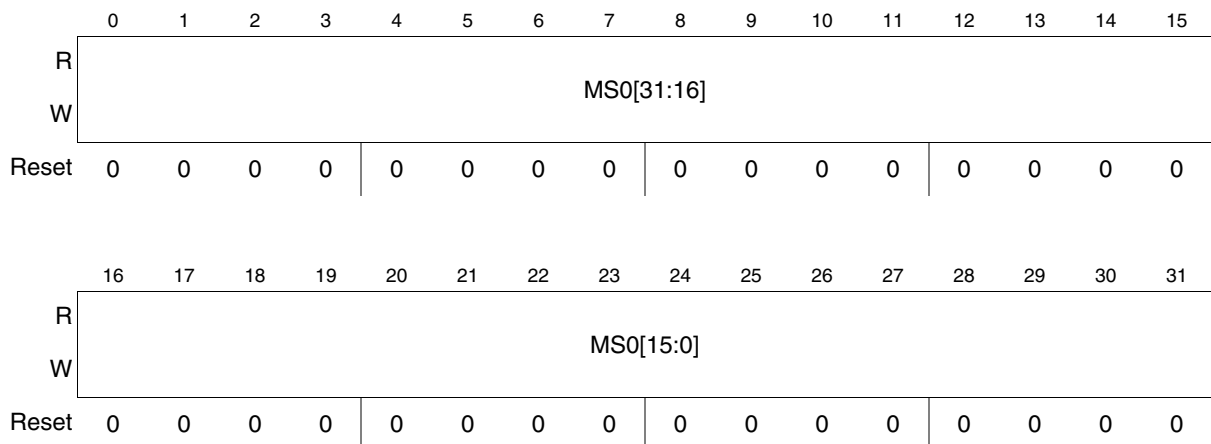


Figure 30-16. CFlash User Multiple Input Signature Register 0 (CFLASH\_UMISR0)

Table 30-27. CFLASH\_UMISR0 field descriptions

| Field     | Description   |
|-----------|---|
| MS0[31:0] | <p><i>Multiple Input Signature, bits 31–0</i></p> <p>These bits represent the MISR value obtained accumulating the bits 31:0 of all the pages read from the flash memory.</p> <p>The MS can be seeded to any value by writing the CFLASH_UMISR0 register.</p> |

### 30.5.1.12 CFlash User Multiple Input Signature Register 1 (CFLASH\_UMISR1)

The CFLASH\_UMISR1 provides a means to evaluate the array integrity.

The CFLASH\_UMISR1 represents the bits 63:32 of the whole 144 bits word (2 double words including ECC).

The CFLASH\_UMISR1 is not accessible whenever CFLASH\_MCR[DONE] or CFLASH\_UT0[AID] are low: reading returns indeterminate data while writing has no effect.

Offset: 0x0004C

Access: Read/write

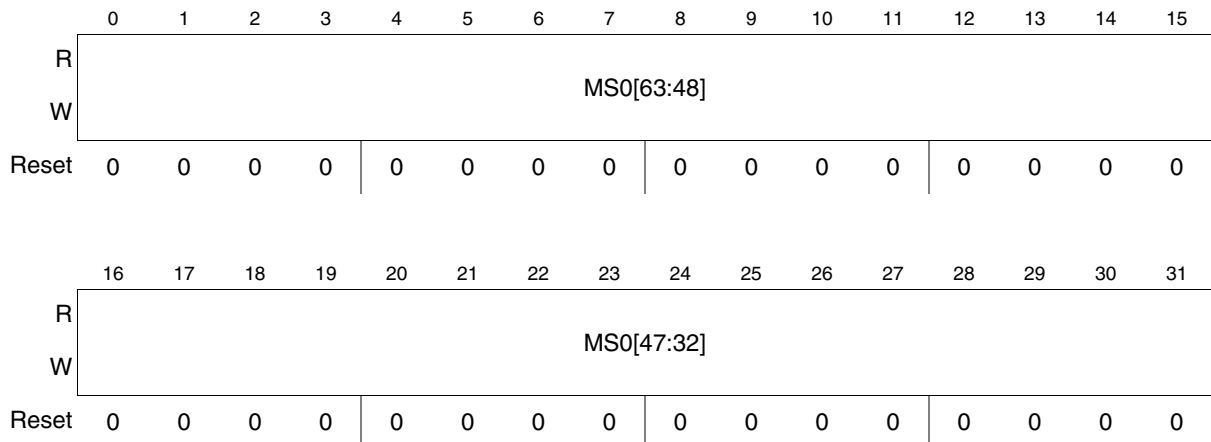


Figure 30-17. CFlash User Multiple Input Signature Register 1 (CFLASH\_UMISR1)

Table 30-28. CFLASH\_UMISR1 field descriptions

| Field      | Description   |
|------------|---|
| MS0[63:32] | <i>Multiple Input Signature, bits 63–32</i><br>These bits represent the MISR value obtained accumulating the bits 63:32 of all the pages read from the flash memory.<br>The MS can be seeded to any value by writing the CFLASH_UMISR1. |

### 30.5.1.13 CFlash User Multiple Input Signature Register 2 (CFLASH\_UMISR2)

The CFLASH\_UMISR2 provides a means to evaluate the array integrity.

The CFLASH\_UMISR2 represents the bits 95:64 of the whole 144 bits word (2 double words including ECC).

The CFLASH\_UMISR2 is not accessible whenever CFLASH\_MCR[*DONE*] or CFLASH\_UT0[*AID*] are low: reading returns indeterminate data while writing has no effect.

Offset: 0x00050

Access: Read/write

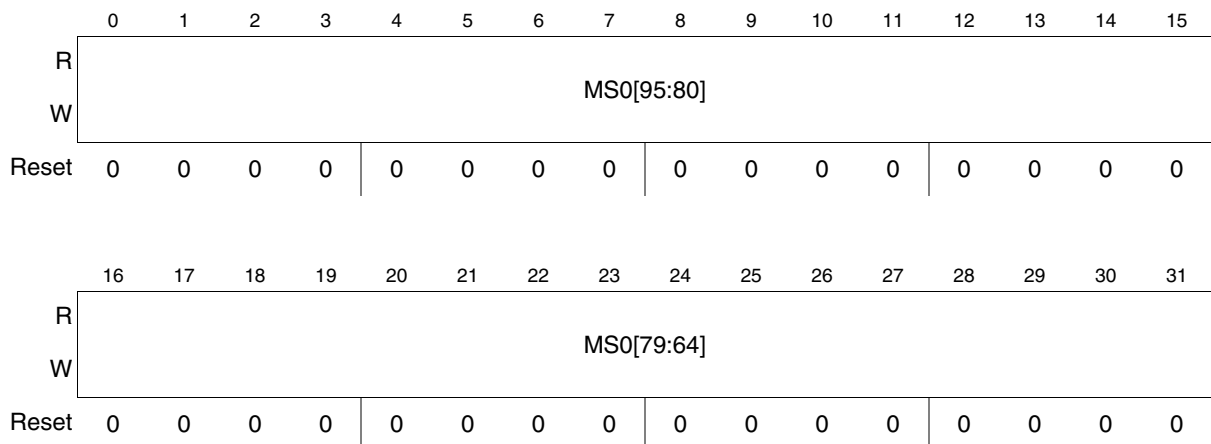


Figure 30-18. CFlash User Multiple Input Signature Register 2 (CFLASH\_UMISR2)

Table 30-29. CFLASH\_UMISR2 field descriptions

| Field      | Description   |
|------------|---|
| MS0[95:64] | <i>Multiple Input Signature, bits 95–64</i><br>These bits represent the MISR value obtained accumulating the bits 95:64 of all the pages read from the flash memory.<br>The MS can be seeded to any value by writing the CFLASH_UMISR2. |

### 30.5.1.14 CFlash User Multiple Input Signature Register 3 (CFLASH\_UMISR3)

The CFLASH\_UMISR3 provides a mean to evaluate the array integrity.

The CFLASH\_UMISR3 represents the bits 127:96 of the whole 144 bits word (2 double words including ECC).

The CFLASH\_UMISR3 is not accessible whenever CFLASH\_MCR[DONE] or CFLASH\_UT0[AID] are low: reading returns indeterminate data while writing has no effect.

Offset: 0x00054

Access: Read/write

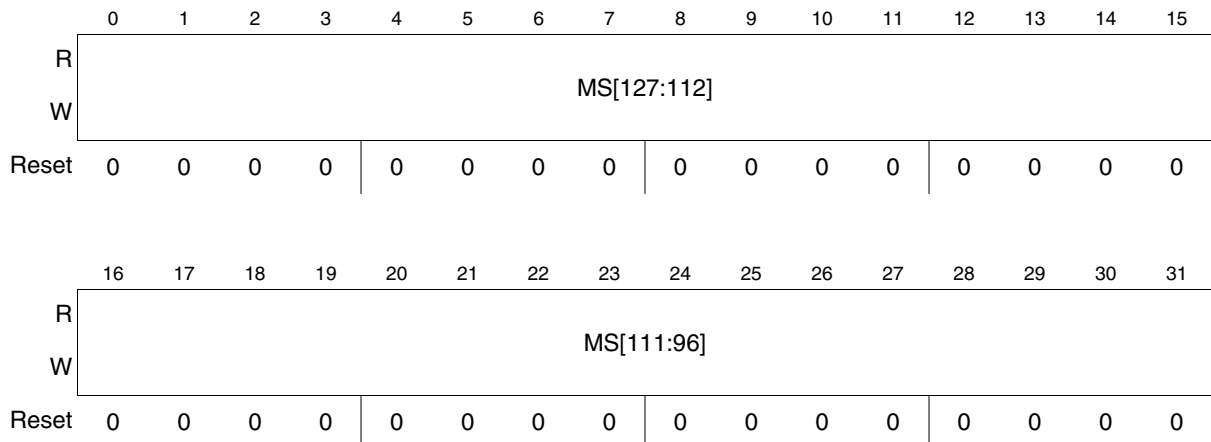


Figure 30-19. CFlash User Multiple Input Signature Register 3 (CFLASH\_UMISR3)

Table 30-30. CFLASH\_UMISR3 field descriptions

| Field      | Description  |
|------------|--|
| MS[127:96] | <p><i>Multiple Input Signature, bits 127–96</i></p> <p>These bits represent the MISR value obtained accumulating the bits 127:96 of all the pages read from the flash memory.</p> <p>The MS can be seeded to any value by writing the CFLASH_UMISR3.</p> |

### 30.5.1.15 CFlash User Multiple Input Signature Register 4 (CFLASH\_UMISR4)

The CFLASH\_UMISR4 provides a mean to evaluate the array integrity.

The CFLASH\_UMISR4 represents the ECC bits of the whole 144 bits word (2 double words including ECC): bits 8:15 are ECC bits for the odd double word, and bits 24:31 are the ECC bits for the even double word; bits 4:5 and 20:21 of MISR are respectively the double and single ECC error detection for odd and even double word.

The CFLASH\_UMISR4 is not accessible whenever CFLASH\_MCR[*DONE*] or CFLASH\_UT0[*AID*] are low: reading returns indeterminate data while writing has no effect.

Offset: 0x00058

Access: Read/write

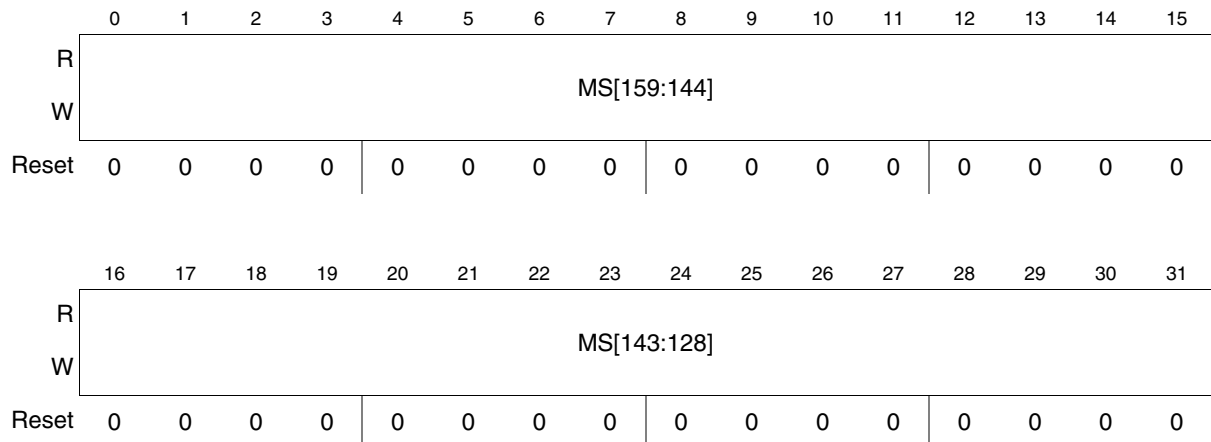


Figure 30-20. CFlash User Multiple Input Signature Register 4 (CFLASH\_UMISR4)

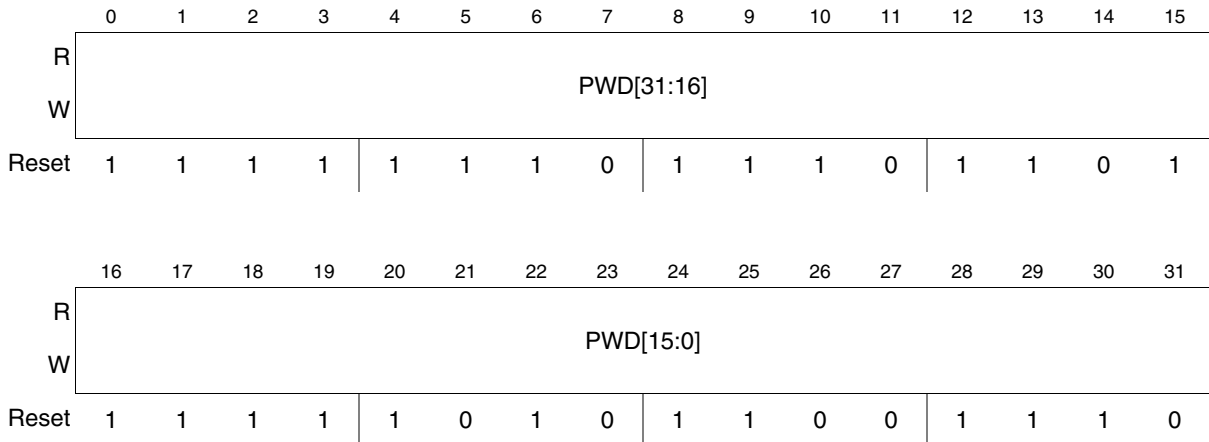
Table 30-31. CFLASH\_UMISR4 field descriptions

| Field       | Description  |
|-------------|--|
| MS[159:128] | <p><i>Multiple Input Signature, bits 159–128</i></p> <p>These bits represent the MISR value obtained accumulating:</p> <ul style="list-style-type: none"> <li>• The 8 ECC bits for the even double word (on MS[135:128]);</li> <li>• The single ECC error detection for even double word (on MS138);</li> <li>• The double ECC error detection for even double word (on MS139);</li> <li>• The 8 ECC bits for the odd double word (on MS[151:144]);</li> <li>• The single ECC error detection for odd double word (on MS154);</li> <li>• The double ECC error detection for odd double word (on MS155).</li> </ul> <p>The MS can be seeded to any value by writing the CFLASH_UMISR4 register.</p> |

### 30.5.1.16 CFlash Nonvolatile Private Censorship Password 0 Register (NVPWD0)

The Nonvolatile Private Censorship Password 0 register contains the 32 LSB of the password used to validate the censorship information contained in NVSCC0–1 registers.

Offset: 0x203DD8 Access: Read/write



**Figure 30-21. CFlash Nonvolatile Private Censorship Password 0 Register (NVPWD0)**

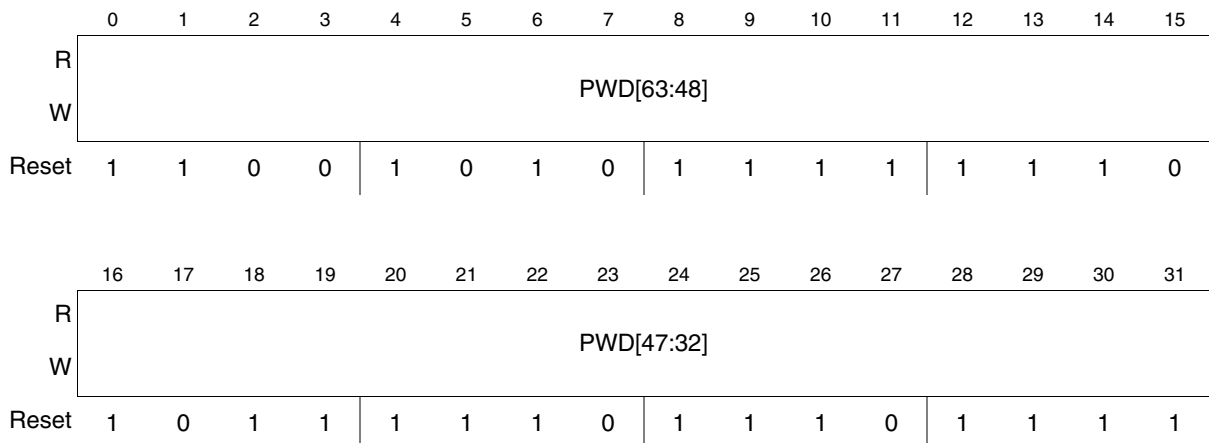
**Table 30-32. NVPWD0 field descriptions**

| Field     | Description  |
|-----------|--|
| PWD[31:0] | Password, bits 31–0<br>These bits represent the 32 LSB of the private censorship password. |

### 30.5.1.17 CFlash Nonvolatile Private Censorship Password 1 Register (NVPWD1)

The Nonvolatile Private Censorship Password 1 register contains the 32 MSB of the password used to validate the censorship information contained in NVSCC0–1 registers.

Offset: 0x203DDC Access: Read/write



**Figure 30-22. CFlash Nonvolatile Private Censorship Password 1 Register (NVPWD1)**

Table 30-33. NVPWD1 field descriptions

| Field      | Description  |
|------------|--|
| PWD[63:32] | <i>Password, bits 63–32</i><br>These bits represent the 32 MSB of the Private Censorship Password. |

**NOTE**

In a secured device, starting with a serial boot, it is possible to read the content of the four flash locations where the RCHW can be stored. For example if the RCHW is stored at address 0x00000000, the reads at address 0x00000000, 0x00000004, 0x00000008, and 0x0000000C will return a correct value. Any other flash address cannot be accessed.

**30.5.1.18 CFlash Nonvolatile System Censorship Control 0 register (NVSCC0)**

The NVSCC0 register stores the 32 LSB of the Censorship Control Word of the device.

The NVSCC0 is a nonvolatile register located in the Shadow sector: it is read during the reset phase of the flash memory module and the protection mechanisms are activated consequently.

The parts are delivered uncensored to the user.

Offset: 0x203DE0

Access: Read/write

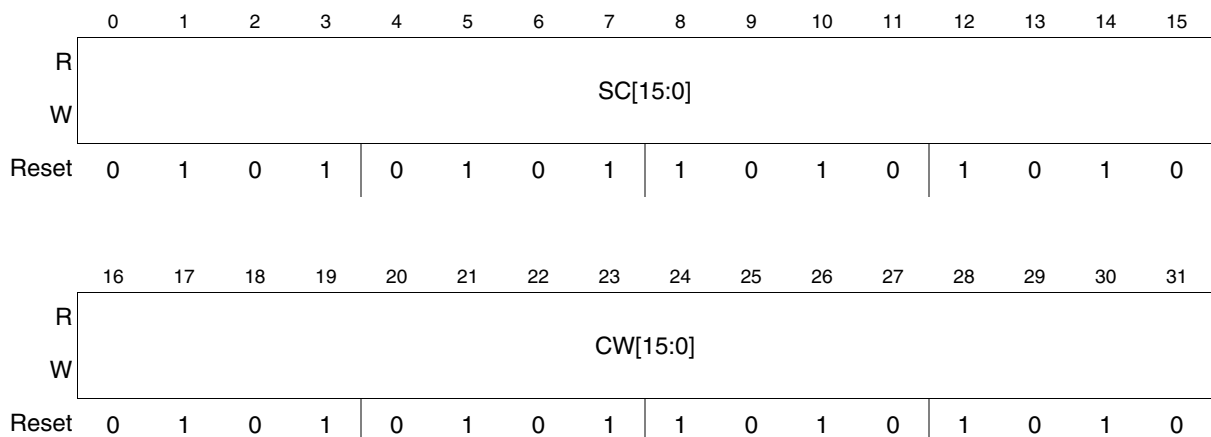


Figure 30-23. CFlash Nonvolatile System Censorship Control 0 register (NVSCC0)

**Table 30-34. NVSCC0 field descriptions**

| Field    | Description   |
|----------|---|
| SC[15:0] | <i>Serial Censorship control word, bits 15-0</i><br>These bits represent the 16 LSB of the Serial Censorship Control Word (SCCW).<br>If SC15-0 = 0x55AA and NVSCC1 = NVSCC0 the Public Access is disabled.<br>If SC15-0 ≠ 0x55AA or NVSCC1 ≠ NVSCC0 the Public Access is enabled. |
| CW[15:0] | <i>Censorship control Word, bits 15-0</i><br>These bits represent the 16 LSB of the Censorship Control Word (CCW).<br>If CW15-0 = 0x55AA and NVSCC1 = NVSCC0 the Censored Mode is disabled.<br>If CW15-0 ≠ 0x55AA or NVSCC1 ≠ NVSCC0 the Censored Mode is enabled.                |

### 30.5.1.19 CFlash Nonvolatile System Censorship Control 1 register (NVSCC1)

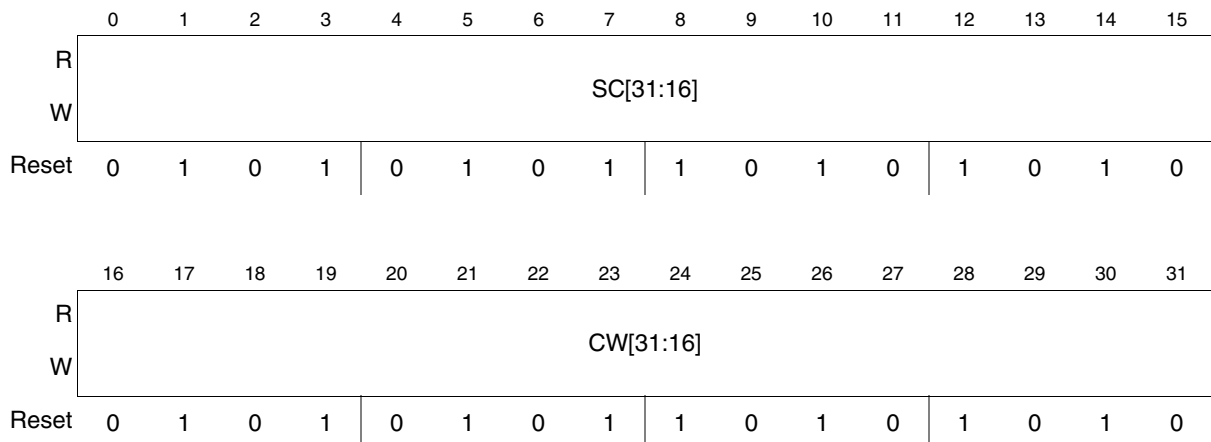
The NVSCC1 register stores the 32 MSB of the Censorship Control Word of the device.

The NVSCC1 is a nonvolatile register located in the Shadow sector: it is read during the reset phase of the flash memory module and the protection mechanisms are activated consequently.

The parts are delivered uncensored to the user.

Offset: 0x203DE4

Access: Read/write

**Figure 30-24. CFlash Nonvolatile System Censorship Control 1 register (NVSCC1)****Table 30-35. NVSCC1 field descriptions**

| Field     | Description  |
|-----------|--|
| SC[31:16] | <i>Serial Censorship control word, bits 31-16</i><br>These bits represent the 16 MSB of the Serial Censorship Control Word (SCCW).<br>If SC15-0 = 0x55AA and NVSCC1 = NVSCC0 the Public Access is disabled.<br>If SC15-0 ≠ 0x55AA or NVSCC1 ≠ NVSCC0 the Public Access is enabled. |
| CW[31:16] | <i>Censorship control Word, bits 31-16</i><br>These bits represent the 16 MSB of the Censorship Control Word (CCW).<br>If CW15-0 = 0x55AA and NVSCC1 = NVSCC0 the Censored Mode is disabled.<br>If CW15-0 ≠ 0x55AA or NVSCC1 ≠ NVSCC0 the Censored Mode is enabled.                |



### 30.5.1.20 CFlash Nonvolatile User Options register (NVUSRO)

The Nonvolatile User Options Register contains configuration information for the user application.

The NVUSRO register is a 64-bit register, of which the 32 most significant bits 63:32 are “don’t care” and are used to manage ECC codes.

Offset: 0x203E18

Access: Read/write

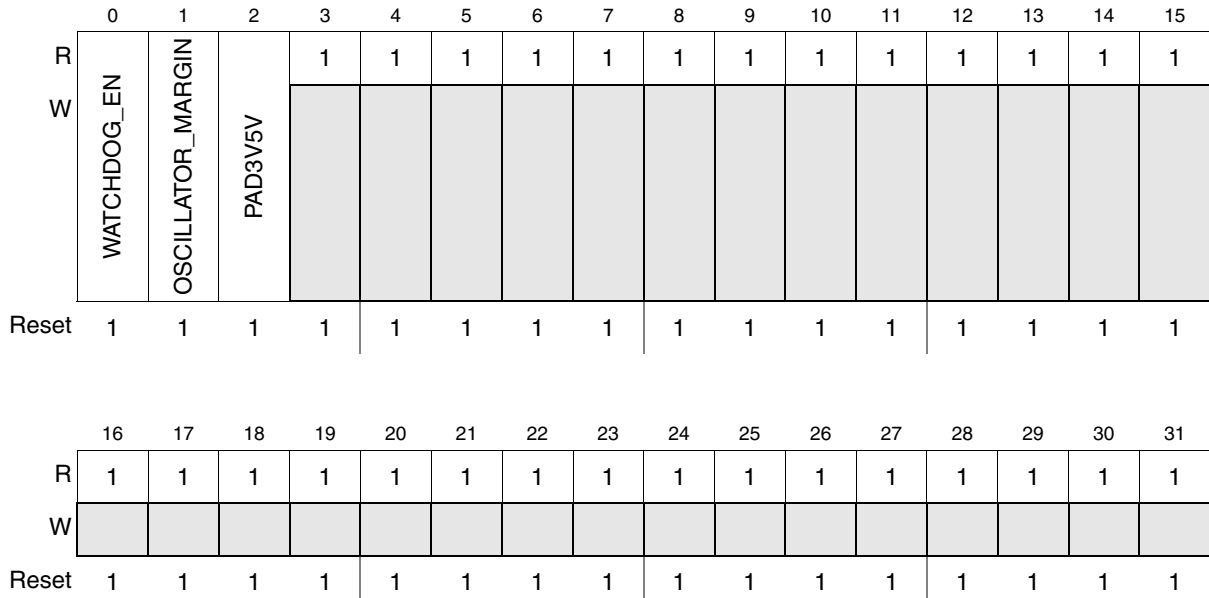


Figure 30-25. CFlash Nonvolatile User Options register (NVUSRO)

Table 30-36. NVUSRO field descriptions

| Field             | Description   |
|-------------------|---|
| WATCHDOG_EN       | <i>Watchdog Enable</i><br>0 Disable after reset<br>1 Enable after reset<br>Default manufacturing value before flash memory initialization is 1  |
| OSCILLATOR_MARGIN | <i>Oscillator Margin</i><br>0 Low consumption configuration (4 MHz/8 MHz)<br>1 High margin configuration (4 MHz/16 MHz)<br>Default manufacturing value before flash memory initialization is 1                                  |
| PAD3V5V           | <i>PAD3V5V</i><br>0 High voltage supply is 5.0 V<br>1 High voltage supply is 3.3 V<br>Default manufacturing value before flash memory initialization is 1 (3.3 V), which should ensure correct minimum slope for boundary scan. |

## 30.5.2 DFlash register description

### 30.5.2.1 DFlash Module Configuration Register (DFLASH\_MCR)

The Module Configuration Register is used to enable and monitor all modify operations of the flash memory module.

Address offset: 0x0000

Access: Read/write

|       |     |   |   |   |   |      |   |   |     |   |    |    |    |    |     |    |
|-------|-----|---|---|---|---|------|---|---|-----|---|----|----|----|----|-----|----|
|       | 0   | 1 | 2 | 3 | 4 | 5    | 6 | 7 | 8   | 9 | 10 | 11 | 12 | 13 | 14  | 15 |
| R     | EDC | 0 | 0 | 0 | 0 | SIZE |   | 0 | LAS |   |    | 0  | 0  | 0  | MAS |    |
| W     | w1c |   |   |   |   |      |   |   |     |   |    |    |    |    |     |    |
| Reset | 0   | 0 | 0 | 0 | 0 | 1    | 1 | 0 | 0   | 1 | 1  | 0  | 0  | 0  | 0   | 0  |

|       |     |     |    |    |      |      |     |    |    |    |    |     |      |     |      |     |
|-------|-----|-----|----|----|------|------|-----|----|----|----|----|-----|------|-----|------|-----|
|       | 16  | 17  | 18 | 19 | 20   | 21   | 22  | 23 | 24 | 25 | 26 | 27  | 28   | 29  | 30   | 31  |
| R     | EER | RWE | 0  | 0  | PEAS | DONE | PEG | 0  | 0  | 0  | 0  | PGM | PSUS | ERS | ESUS | EHV |
| W     | w1c | w1c |    |    |      |      |     |    |    |    |    |     |      |     |      |     |
| Reset | 0   | 0   | 0  | 0  | 0    | 1    | 1   | 0  | 0  | 0  | 0  | 0   | 0    | 0   | 0    | 0   |

Figure 30-26. DFlash Module Configuration Register (DFLASH\_MCR)

Table 30-37. DFLASH\_MCR field descriptions

| Field | Description   |
|-------|---|
| EDC   | <p><i>ECC Data Correction</i></p> <p>EDC provides information on previous reads. If an ECC single error detection and correction occurred, the EDC bit is set to 1. This bit must then be cleared, or a reset must occur before this bit will return to a 0 state. This bit may not be set to 1 by the user.</p> <p>In the event of an ECC double error detection, this bit will not be set.</p> <p>If EDC is not set, or remains 0, this indicates that all previous reads (from the last reset, or clearing of EDC) were not corrected through ECC.</p> <p>The function of this bit is device dependent and it can be configured to be disabled.</p> <p>0 Reads are occurring normally.</p> <p>1 An ECC single error occurred and was corrected during a previous read.</p> |
| SIZE  | <p><i>Array Space Size</i></p> <p>The value of SIZE field is dependent upon the size of the flash memory module. See <a href="#">Table 30-38</a>.</p>   |
| LAS   | <p><i>Low Address Space</i></p> <p>The value of the LAS field corresponds to the configuration of the low address space. See <a href="#">Table 30-39</a>.</p>   |
| MAS   | <p><i>Mid Address Space</i></p> <p>The value of the MAS field corresponds to the configuration of the mid address space. See <a href="#">Table 30-40</a>.</p>   |

Table 30-37. DFLASH\_MCR field descriptions (continued)

| Field | Description   |
|-------|---|
| EER   | <p><i>ECC Event Error</i></p> <p>EER provides information on previous reads. If an ECC double error detection occurred, the EER bit is set to 1.</p> <p>This bit must then be cleared, or a reset must occur before this bit will return to a 0 state. This bit may not be set to 1 by the user.</p> <p>In the event of an ECC single error detection and correction, this bit will not be set.</p> <p>If EER is not set, or remains 0, this indicates that all previous reads (from the last reset, or clearing of EER) were correct.</p> <p>0 Reads are occurring normally.<br/>1 An ECC double error occurred during a previous read.</p>  |
| RWE   | <p><i>Read-While-Write Event Error</i></p> <p>RWE provides information on previous reads when a Modify operation is on going. If a RWW Error occurs, the RWE bit will be set to 1. Read-While-Write Error means that a read access to the flash memory Matrix has occurred while the FPEC was performing a program or erase operation or an array integrity check.</p> <p>This bit must then be cleared, or a reset must occur before this bit will return to a 0 state. This bit may not be set to 1 by the user.</p> <p>If RWE is not set, or remains 0, this indicates that all previous RWW reads (from the last reset, or clearing of RWE) were correct.</p> <p>0 Reads are occurring normally.<br/>1 A RWW Error occurred during a previous read.</p>   |
| PEAS  | <p><i>Program/Erase Access Space</i></p> <p>PEAS indicates which space is valid for program and erase operations: main array space or shadow/test space.</p> <p>PEAS = 0 indicates that the main address space is active for all flash memory module program and erase operations.</p> <p>PEAS = 1 indicates that the test or shadow address space is active for program and erase. The value in PEAS is captured and held with the first interlock write done for Modify operations. The value of PEAS is retained between sampling events (that is, subsequent first interlock writes).</p> <p>0 Shadow/Test address space is disabled for program/erase and main address space enabled.<br/>1 Shadow/Test address space is enabled for program/erase and main address space disabled.</p>  |
| DONE  | <p><i>Modify Operation Done</i></p> <p>DONE indicates if the flash memory module is performing a high voltage operation. DONE is set to 1 on termination of the flash memory module reset.</p> <p>DONE is cleared to 0 just after a 0-to-1 transition of EHV, which initiates a high voltage operation, or after resuming a suspended operation.</p> <p>DONE is set to 1 at the end of program and erase high voltage sequences.</p> <p>DONE is set to 1 (within <math>t_{PABT}</math> or <math>t_{EABT}</math>, equal to P/E Abort Latency) after a 1-to-0 transition of EHV, which aborts a high voltage program/erase operation.</p> <p>DONE is set to 1 (within <math>t_{ESUS}</math>, time equals to Erase Suspend Latency) after a 0-to-1 transition of ESUS, which suspends an erase operation.</p> <p>0 Flash memory is executing a high voltage operation.<br/>1 Flash memory is not executing a high voltage operation.</p> |

Table 30-37. DFLASH\_MCR field descriptions (continued)

| Field | Description   |
|-------|---|
| PEG   | <p><i>Program/Erase Good</i></p> <p>The PEG bit indicates the completion status of the last flash memory program or erase sequence for which high voltage operations were initiated. The value of PEG is updated automatically during the program and erase high voltage operations.</p> <p>Aborting a program/erase high voltage operation will cause PEG to be cleared to 0, indicating the sequence failed.</p> <p>PEG is set to 1 when the flash memory module is reset, unless a flash memory initialization error has been detected.</p> <p>The value of PEG is valid only when PGM = 1 and/or ERS = 1 and after DONE transitions from 0-to-1 due to an abort or the completion of a program/erase operation. PEG is valid until PGM/ERS makes a 1-to-0 transition or EHV makes a 0-to-1 transition.</p> <p>The value in PEG is not valid after a 0-to-1 transition of DONE caused by ESUS being set to logic 1.</p> <p>If program or erase are attempted on blocks that are locked, the response will be PEG = 1, indicating that the operation was successful, and the content of the block were properly protected from the program or erase operation.</p> <p>If a program operation tries to program at 1 bits that are at 0, the program operation is correctly executed on the new bits to be programmed at 0, but PEG is cleared, indicating that the requested operation has failed.</p> <p>In array integrity check or margin read, PEG is set to 1 when the operation is completed, regardless the occurrence of any error. The presence of errors can be detected only comparing checksum value stored in UMIRS0-1.</p> <p>Aborting an array integrity check or a margin read operation will cause PEG to be cleared to 0, indicating the sequence failed.</p> <p>0 Program or erase operation failed; or program, erase, array integrity check, or margin mode aborted.</p> <p>1 Program or erase operation successful, or array integrity check or margin mode completed.</p> |
| PGM   | <p><i>Program</i></p> <p>PGM is used to set up the flash memory module for a program operation.</p> <p>A 0-to-1 transition of PGM initiates a program sequence.</p> <p>A 1-to-0 transition of PGM ends the program sequence.</p> <p>PGM can be set only under User mode read (ERS is low and DFLASH_UT0[AIE] is low).</p> <p>PGM can be cleared by the user only when EHV is low and DONE is high.</p> <p>PGM is cleared on reset.</p> <p>0 Flash memory is not executing a program sequence.</p> <p>1 Flash memory is executing a program sequence.</p>  |
| PSUS  | <p><i>Program Suspend</i></p> <p>A write to this bit has no effect, but the written data can be read back.</p>  |
| ERS   | <p><i>Erase</i></p> <p>ERS is used to set up the flash memory module for an erase operation.</p> <p>A 0-to-1 transition of ERS initiates an erase sequence.</p> <p>A 1-to-0 transition of ERS ends the erase sequence.</p> <p>ERS can be set only under User mode read (PGM is low and DFLASH_UT0[AIE] is low).</p> <p>ERS can be cleared by the user only when ESUS and EHV are low and DONE is high.</p> <p>ERS is cleared on reset.</p> <p>0 Flash memory is not executing an erase sequence.</p> <p>1 Flash memory is executing an erase sequence.</p>  |

Table 30-37. DFLASH\_MCR field descriptions (continued)

| Field | Description  |
|-------|--|
| ESUS  | <p><i>Erase Suspend</i></p> <p>ESUS is used to indicate that the flash memory module is in erase suspend or in the process of entering a Suspend state. The flash memory module is in erase suspend when ESUS = 1 and DONE = 1.</p> <p>ESUS can be set high only when ERS and EHV are high and PGM is low.</p> <p>A 0-to-1 transition of ESUS starts the sequence that sets DONE and places the flash memory in erase suspend. The flash memory module enters Suspend within <math>t_{ESUS}</math> of this transition.</p> <p>ESUS can be cleared only when DONE and EHV are high and PGM is low.</p> <p>A 1-to-0 transition of ESUS with EHV = 1 starts the sequence that clears DONE and returns the module to erase.</p> <p>The flash memory module cannot exit erase suspend and clear DONE while EHV is low.</p> <p>ESUS is cleared on reset.</p> <p>0 Erase sequence is not suspended.<br/>1 Erase sequence is suspended.</p>  |
| EHV   | <p><i>Enable High Voltage</i></p> <p>The EHV bit enables the flash memory module for a high voltage program/erase operation. EHV is cleared on reset.</p> <p>EHV must be set after an interlock write to start a program/erase sequence. EHV may be set under one of the following conditions:</p> <ul style="list-style-type: none"> <li>• Erase (ERS = 1, ESUS = 0, DFLASH_UT0[AIE] = 0)</li> <li>• Program (ERS = 0, ESUS = 0, PGM = 1, DFLASH_UT0[AIE] = 0)</li> </ul> <p>In normal operation, a 1-to-0 transition of EHV with DONE high and ESUS low terminates the current program/erase high voltage operation.</p> <p>When an operation is aborted, there is a 1-to-0 transition of EHV with DONE low and the eventual Suspend bit low. An abort causes the value of PEG to be cleared, indicating a failing program/erase; address locations being operated on by the aborted operation contain indeterminate data after an abort. A suspended operation cannot be aborted.</p> <p>Aborting a high voltage operation will leave the flash memory module addresses in an indeterminate data state. This may be recovered by executing an erase on the affected blocks.</p> <p>EHV may be written during Suspend. EHV must be high to exit Suspend. EHV may not be written after ESUS is set and before DONE transitions high. EHV may not be cleared after ESUS is cleared and before DONE transitions low.</p> <p>0 Flash memory is not enabled to perform an high voltage operation.<br/>1 Flash memory is enabled to perform an high voltage operation.</p> |

Table 30-38. Array space size

| SIZE | Array space size   |
|------|--------------------|
| 000  | 128 KB             |
| 001  | 256 KB             |
| 010  | 512 KB             |
| 011  | Reserved (1024 KB) |
| 100  | Reserved (1536 KB) |
| 101  | Reserved (2048 KB) |

**Table 30-38. Array space size (continued)**

| SIZE | Array space size |
|------|------------------|
| 110  | 64 KB            |
| 111  | Reserved         |

**Table 30-39. Low address space configuration**

| LAS | Low address space sectorization        |
|-----|--|
| 000 | Reserved                               |
| 001 | Reserved                               |
| 010 | 32 KB + 2 × 16 KB + 2 × 32 KB + 128 KB |
| 011 | Reserved                               |
| 100 | Reserved                               |
| 101 | Reserved                               |
| 110 | 4 × 16 KB                              |
| 111 | Reserved                               |

**Table 30-40. Mid address space configuration**

| MAS | Mid address space sectorization |
|-----|---------------------------------|
| 0   | 2 × 128KB                       |
| 1   | Reserved                        |

A number of DFLASH\_MCR bits are protected against write when another bit, or set of bits, is in a specific state. These write locks are covered on a bit by bit basis in the preceding description, but those locks do not consider the effects of trying to write two or more bits simultaneously.

The flash memory module does not allow the user to write bits simultaneously, which would put the device into an illegal state. This is implemented through a priority mechanism among the bits. The bit changing priorities are detailed in the [Table 30-41](#).

**Table 30-41. DFLASH\_MCR bits set/clear priority levels**

| Priority level | DFLASH_MCR bits |
|----------------|-----------------|
| 1              | ERS             |
| 2              | PGM             |
| 3              | EHV             |
| 4              | ESUS            |

If the user attempts to write two or more DFLASH\_MCR bits simultaneously then only the bit with the lowest priority level is written.

If Stall/Abort-While-Write is enabled and an erase operation is started on one sector while fetching code from another then the following sequence is executed:

- CPU is stalled when flash is unavailable
- PEG flag set (stall case) or reset (abort case)
- Interrupt triggered if enabled

If Stall/Abort-While-Write is used, then software should ignore the setting of the RWE flag. The RWE flag should be cleared after each HV operation.

If Stall/Abort-While-Write is not used the application software should handle RWE error. See [Section 30.8.10, Read-while-write functionality](#).

### 30.5.2.2 DFlash Low/Mid Address Space Block Locking Register (DFLASH\_LML)

The DFlash Low/Mid Address Space Block Locking register provides a means to protect blocks from being modified. These bits, along with bits in the DFLASH\_SLL register, determine if the block is locked from program or erase. An OR of DFLASH\_LML and DFLASH\_SLL determine the final lock status.

Offset: 0x0004

Access: Read/write

|   | 0   | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11   | 12 | 13 | 14 | 15 |
|---|-----|---|---|---|---|---|---|---|---|---|----|------|----|----|----|----|
| R | LME | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | TSLK | 0  | 0  | 0  | 0  |
| W |     |   |   |   |   |   |   |   |   |   |    |      |    |    |    |    |

Reset Defined by DFLASH\_NVLML at DFlash Test sector address 0xC03DE8. This location is user OTP (One-Time Programmable). The DFLASH\_NVLML register influences only the R/W bits of the DFLASH\_LML register.

|   | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28  | 29 | 30 | 31 |
|---|----|----|----|----|----|----|----|----|----|----|----|----|-----|----|----|----|
| R | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | LLK |    |    |    |
| W |    |    |    |    |    |    |    |    |    |    |    |    |     |    |    |    |

Reset Defined by DFLASH\_NVLML at DFlash Test sector address 0xC03DE8. This location is user OTP (One-Time Programmable). The DFLASH\_NVLML register influences only the R/W bits of the DFLASH\_LML register.

**Figure 30-27. DFlash Low/Mid Address Space Block Locking Register (DFLASH\_LML)**

Table 30-42. DFLASH\_LML field descriptions

| Field | Description   |
|-------|---|
| LME   | <p><i>Low/Mid Address Space Block Enable</i></p> <p>This bit is used to enable the Lock registers (TSLK, MLK1-0 and LLK15-0) to be set or cleared by registers writes.</p> <p>This bit is a status bit only. The method to set this bit is to write a password, and if the password matches, the LME bit will be set to reflect the status of enabled, and is enabled until a reset operation occurs. For LME the password 0xA1A11111 must be written to the DFLASH_LML register.</p> <p>0 Low Address Locks are disabled: TSLK, MLK1-0 and LLK15-0 cannot be written.<br/> 1 Low Address Locks are enabled: TSLK, MLK1-0 and LLK15-0 can be written.</p>   |
| TSLK  | <p><i>Test/Shadow Address Space Block Lock</i></p> <p>This bit is used to lock the block of Test and Shadow address space from program and erase (erase is any case forbidden for Test block).</p> <p>A value of 1 in the TSLK register signifies that the Test/shadow sector is locked for program and erase.</p> <p>A value of 0 in the TSLK register signifies that the Test/shadow sector is available to receive program and erase pulses.</p> <p>The TSLK register is not writable once an interlock write is completed until DFLASH_MCR[DONE] is set at the completion of the requested operation. Likewise, the TSLK register is not writable if a high voltage operation is suspended.</p> <p>Upon reset, information from the TestFlash block is loaded into the TSLK register. The TSLK bit may be written as a register. Reset will cause the bit to go back to its TestFlash block value. The default value of the TSLK bit (assuming erased fuses) would be locked.</p> <p>TSLK is not writable unless LME is high.</p> <p>0 Test/Shadow address space block is unlocked and can be modified (also if DFLASH_SLL[STSLK] = 0).<br/> 1 Test/Shadow address space block is locked and cannot be modified.</p>  |
| LLK   | <p><i>Low Address Space Block Lock</i></p> <p>This field is used to lock the blocks of low address space from program and erase. LLK[3:0] are related to sectors B1F3-0, respectively. LLK[15:4] are not used for this memory cut.</p> <p>A value of 1 in a bit of the LLK field signifies that the corresponding block is locked for program and erase.</p> <p>A value of 0 in a bit of the LLK field signifies that the corresponding block is available to receive program and erase pulses.</p> <p>The LLK field is not writable after an interlock write is completed until DFLASH_MCR[DONE] is set at the completion of the requested operation. Likewise, the LLK field is not writable if a high voltage operation is suspended.</p> <p>Upon reset, information from the TestFlash block is loaded into the LLK field. The LLK field may be written as a register. Reset will cause the field to go back to its TestFlash block value. The default value of the LLK field (assuming erased fuses) would be locked.</p> <p>In the event that blocks are not present (due to configuration or total memory size), the LLK field will default to locked, and will not be writable. The reset value will always be 1 (independent of the TestFlash block), and register writes will have no effect.</p> <p>In the 64 KB flash memory module bits LLK[15:4] are read-only and locked at 1.</p> <p>LLK is not writable unless LME is high.</p> <p>0 Low address space block is unlocked and can be modified (also if DFLASH_SLL[SLK] = 0).<br/> 1 Low address space block is locked and cannot be modified.</p> |



### 30.5.2.2.1 DFlash Nonvolatile Low/Mid Address Space Block Locking Register (DFLASH\_NVLML)

The DFLASH\_LML register has a related Nonvolatile Low/Mid Address Space Block Locking register located in TestFlash that contains the default reset value for DFLASH\_LML. During the reset phase of the flash memory module, the DFLASH\_NVLML register content is read and loaded into the DFLASH\_LML.

The DFLASH\_NVLML register is a 64-bit register, of which the 32 most significant bits 63:32 are “don’t care” and are used to manage ECC codes.

Offset: 0xC03DE8

Access: Read/write

|       | 0   | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11   | 12 | 13 | 14 | 15 |
|-------|-----|---|---|---|---|---|---|---|---|---|----|------|----|----|----|----|
| R     | LME | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1  | TSLK | 1  | 1  | 1  | 1  |
| W     |     |   |   |   |   |   |   |   |   |   |    |      |    |    |    |    |
| Reset | 1   | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1  | 1    | 1  | 1  | 1  | 1  |

|       | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28  | 29 | 30 | 31 |
|-------|----|----|----|----|----|----|----|----|----|----|----|----|-----|----|----|----|
| R     | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | LLK |    |    |    |
| W     |    |    |    |    |    |    |    |    |    |    |    |    |     |    |    |    |
| Reset | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1   | 1  | 1  | 1  |

Figure 30-28. DFlash Nonvolatile Low/Mid address space block Locking register (DFLASH\_NVLML)

Table 30-43. DFLASH\_NVLML field descriptions

| Field | Description   |
|-------|---|
| LME   | <p><i>Low/Mid Address Space Block Enable</i></p> <p>This bit is used to enable the lock registers (TSLK, MLK1-0 and LLK15-0) to be set or cleared by registers writes.</p> <p>This bit is a status bit only. The method to set this bit is to write a password, and if the password matches, the LME bit will be set to reflect the status of enabled, and is enabled until a reset operation occurs. For LME the password 0xA1A11111 must be written to the DFLASH_LML register.</p> <p>0 Low Address Locks are disabled: TSLK, MLK1-0, and LLK15-0 cannot be written.<br/> 1 Low Address Locks are enabled: TSLK, MLK1-0, and LLK15-0 can be written.</p>   |
| TSLK  | <p><i>Test/Shadow Address Space Block Lock</i></p> <p>This bit is used to lock the block of Test and Shadow address space from program and erase (erase is any case forbidden for Test block).</p> <p>A value of 1 in the TSLK register signifies that the Test/shadow sector is locked for program and erase.</p> <p>A value of 0 in the TSLK register signifies that the Test/shadow sector is available to receive program and erase pulses.</p> <p>The TSLK register is not writable once an interlock write is completed until DFLASH_MCR[DONE] is set at the completion of the requested operation. Likewise, the TSLK register is not writable if a high voltage operation is suspended.</p> <p>Upon reset, information from the TestFlash block is loaded into the TSLK register. The TSLK bit may be written as a register. Reset will cause the bit to go back to its TestFlash block value. The default value of the TSLK bit (assuming erased fuses) would be locked.</p> <p>TSLK is not writable unless LME is high.</p> <p>0 Test/Shadow address space block is unlocked and can be modified (also if DFLASH_SLL[STSLK]= 0).<br/> 1 Test/Shadow address space block is locked and cannot be modified.</p>   |
| LLK   | <p><i>Low Address Space Block Lock</i></p> <p>These bits are used to lock the blocks of low address space from program and erase. LLK[3:0] are related to sectors B1F3-0, respectively. LLK[15:4] are not used for this memory cut.</p> <p>A value of 1 in a bit of the LLK register signifies that the corresponding block is locked for program and erase.</p> <p>A value of 0 in a bit of the LLK register signifies that the corresponding block is available to receive program and erase pulses.</p> <p>The LLK register is not writable once an interlock write is completed until DFLASH_MCR[DONE] is set at the completion of the requested operation. Likewise, the LLK register is not writable if a high voltage operation is suspended.</p> <p>Upon reset, information from the TestFlash block is loaded into the LLK registers. The LLK bits may be written as a register. Reset will cause the bits to go back to their TestFlash block value. The default value of the LLK bits (assuming erased fuses) would be locked.</p> <p>In the event that blocks are not present (due to configuration or total memory size), the LLK bits will default to locked, and will not be writable. The reset value will always be 1 (independent of the TestFlash block), and register writes will have no effect.</p> <p>In the 64 KB flash memory module bits LLK[15:4] are read-only and locked at 1.</p> <p>LLK is not writable unless LME is high.</p> <p>0 Low address space block is unlocked and can be modified (also if DFLASH_SLL[SLK] = 0).<br/> 1 Low address space block is locked and cannot be modified.</p> |

### 30.5.2.3 DFlash Secondary Low/Mid Address Space Block Locking Register (DFLASH\_SLL)

The DFlash Secondary Low/Mid Address Space Block Locking Register provides an alternative means to protect blocks from being modified. These bits, along with bits in the DFLASH\_LML register, determine if the block is locked from program or erase. An OR of DFLASH\_LML and DFLASH\_SLL determines the final lock status.

Offset: 0x000C

Access: Read/write

|   |     |   |   |   |   |   |   |   |   |   |    |       |    |    |    |    |
|---|-----|---|---|---|---|---|---|---|---|---|----|-------|----|----|----|----|
|   | 0   | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11    | 12 | 13 | 14 | 15 |
| R | SLE | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | STSLK | 0  | 0  | 0  | 0  |
| W |     |   |   |   |   |   |   |   |   |   |    |       |    |    |    |    |

Reset Defined by DFLASH\_NVSLI at DFlash Test Sector Address 0xC03DF8. This location is user OTP (One-Time Programmable). The DFLASH\_NVSLI register influences only the R/W bits of the DFLASH\_SLL register.

|   |    |    |    |    |    |    |    |    |    |    |    |    |     |    |    |    |
|---|----|----|----|----|----|----|----|----|----|----|----|----|-----|----|----|----|
|   | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28  | 29 | 30 | 31 |
| R | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | SLK |    |    |    |
| W |    |    |    |    |    |    |    |    |    |    |    |    |     |    |    |    |

Reset Defined by DFLASH\_NVSLI at DFlash Test Sector Address 0xC03DF8. This location is user OTP (One-Time Programmable). The DFLASH\_NVSLI register influences only the R/W bits of the DFLASH\_SLL register.

**Figure 30-29. DFlash Secondary Low/Mid Address Space Block Locking register (DFLASH\_SLL)**

Table 30-44. DFLASH\_SLL field descriptions

| Field | Description  |
|-------|--|
| SLE   | <p><i>Secondary Low/Mid Address Space Block Enable</i></p> <p>This bit is used to enable the Lock registers (STSLK, SMK1-0 and SLK15-0) to be set or cleared by registers writes.</p> <p>This bit is a status bit only. The method to set this bit is to write a password, and if the password matches, the SLE bit will be set to reflect the status of enabled, and is enabled until a reset operation occurs. For SLE the password 0xC3C33333 must be written to the DFLASH_SLL register.</p> <p>0 Secondary Low/Mid Address Locks are disabled: STSLK, SMK1-0, and SLK15-0 cannot be written.</p> <p>1 Secondary Low/Mid Address Locks are enabled: STSLK, SMK1-0, and SLK15-0 can be written.</p>   |
| STSLK | <p><i>Secondary Test/Shadow Address Space Block Lock</i></p> <p>This bit is used as an alternate means to lock the block of Test and Shadow address space from program and erase (erase is any case forbidden for Test block).</p> <p>A value of 1 in the STSLK register signifies that the Test/shadow sector is locked for program and erase.</p> <p>A value of 0 in the STSLK register signifies that the Test/shadow sector is available to receive program and erase pulses.</p> <p>The STSLK register is not writable once an interlock write is completed until DFLASH_MCR[DONE] is set at the completion of the requested operation. Likewise, the STSLK register is not writable if a high voltage operation is suspended.</p> <p>Upon reset, information from the TestFlash block is loaded into the STSLK register. The STSLK bit may be written as a register. Reset will cause the bit to go back to its TestFlash block value. The default value of the STSLK bit (assuming erased fuses) would be locked. STSLK is not writable unless SLE is high.</p> <p>0 Test/Shadow address space block is unlocked and can be modified (also if DFLASH_LML[TSLK] = 0).</p> <p>1 Test/Shadow address space block is locked and cannot be modified.</p>   |
| SLK   | <p><i>Secondary Low Address Space Block Lock</i></p> <p>These bits are used as an alternate means to lock the blocks of Low address space from program and erase.</p> <p>SLK[3:0] are related to sectors B1F3-0, respectively. SLK[15:4] are not used for this memory cut.</p> <p>A value of 1 in a bit of the SLK register signifies that the corresponding block is locked for program and erase.</p> <p>A value of 0 in a bit of the SLK register signifies that the corresponding block is available to receive program and erase pulses.</p> <p>The SLK register is not writable once an interlock write is completed until DFLASH_MCR[DONE] is set at the completion of the requested operation. Likewise, the SLK register is not writable if a high voltage operation is suspended.</p> <p>Upon reset, information from the TestFlash block is loaded into the SLK registers. The SLK bits may be written as a register. Reset will cause the bits to go back to their TestFlash block value. The default value of the SLK bits (assuming erased fuses) would be locked.</p> <p>In the event that blocks are not present (due to configuration or total memory size), the SLK bits will default to locked, and will not be writable. The reset value will always be 1 (independent of the TestFlash block), and register writes will have no effect.</p> <p>In the 64 KB flash memory module bits SLK[15:4] are read-only and locked at 1.</p> <p>SLK is not writable unless SLE is high.</p> <p>0 Low address space block is unlocked and can be modified (also if DFLASH_LML[LLK] = 0).</p> <p>1 Low address space block is locked and cannot be modified.</p> |

### 30.5.2.3.1 DFlash Nonvolatile Secondary Low/Mid Address Space Block Locking Register (DFLASH\_NVSL)

The DFLASH\_SLL register has a related Nonvolatile Secondary Low/Mid Address Space Block Locking register located in TestFlash that contains the default reset value for DFLASH\_SLL. During the reset phase of the flash memory module, the DFLASH\_NVSL register content is read and loaded into the DFLASH\_SLL.

The DFLASH\_NVSL register is a 64-bit register, of which the 32 most significant bits 63:32 are “don’t care” and are used to manage ECC codes.

Offset: 0xC03DF8

Access: Read/write

|       | 0   | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11    | 12 | 13 | 14 | 15 |
|-------|-----|---|---|---|---|---|---|---|---|---|----|-------|----|----|----|----|
| R     | SLE | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1  | STSLK | 1  | 1  | 1  | 1  |
| W     |     |   |   |   |   |   |   |   |   |   |    |       |    |    |    |    |
| Reset | 1   | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1  | 1     | 1  | 1  | 1  | 1  |

|       | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28  | 29 | 30 | 31 |
|-------|----|----|----|----|----|----|----|----|----|----|----|----|-----|----|----|----|
| R     | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | SLK |    |    |    |
| W     |    |    |    |    |    |    |    |    |    |    |    |    |     |    |    |    |
| Reset | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1   | 1  | 1  | 1  |

**Figure 30-30. DFlash Nonvolatile Secondary Low/mid address space block Locking register (DFLASH\_NVSL)**

Table 30-45. DFLASH\_NVSLI field descriptions

| Field | Description  |
|-------|--|
| SLE   | <p><i>Secondary Low/Mid Address Space Block Enable</i></p> <p>This bit is used to enable the Lock registers (STSLK, SMK1-0 and SLK15-0) to be set or cleared by registers writes.</p> <p>This bit is a status bit only. The method to set this bit is to write a password, and if the password matches, the SLE bit will be set to reflect the status of enabled, and is enabled until a reset operation occurs. For SLE the password 0xC3C33333 must be written to the DFLASH_SLL register.</p> <p>0 Secondary Low/Mid Address Locks are disabled: STSLK, SMK1-0, and SLK15-0 cannot be written.</p> <p>1 Secondary Low/Mid Address Locks are enabled: STSLK, SMK1-0, and SLK15-0 can be written.</p>   |
| STSLK | <p><i>Secondary Test/Shadow Address Space Block Lock</i></p> <p>This bit is used as an alternate means to lock the block of Test and Shadow address space from program and erase (erase is any case forbidden for Test block).</p> <p>A value of 1 in the STSLK register signifies that the Test/shadow sector is locked for program and erase.</p> <p>A value of 0 in the STSLK register signifies that the Test/shadow sector is available to receive program and erase pulses.</p> <p>The STSLK register is not writable once an interlock write is completed until DFLASH_MCR[DONE] is set at the completion of the requested operation. Likewise, the STSLK register is not writable if a high voltage operation is suspended.</p> <p>Upon reset, information from the TestFlash block is loaded into the STSLK register. The STSLK bit may be written as a register. Reset will cause the bit to go back to its TestFlash block value. The default value of the STSLK bit (assuming erased fuses) would be locked. STSLK is not writable unless SLE is high.</p> <p>0 Test/Shadow address space block is unlocked and can be modified (also if DFLASH_LML[TSLK] = 0).</p> <p>1 Test/Shadow address space block is locked and cannot be modified.</p>   |
| SLK   | <p><i>Secondary Low Address Space Block Lock</i></p> <p>These bits are used as an alternate means to lock the blocks of low address space from program and erase.</p> <p>SLK[3:0] are related to sectors B1F3-0, respectively. SLK[15:4] are not used for this memory cut.</p> <p>A value of 1 in a bit of the SLK register signifies that the corresponding block is locked for program and erase.</p> <p>A value of 0 in a bit of the SLK register signifies that the corresponding block is available to receive program and erase pulses.</p> <p>The SLK register is not writable once an interlock write is completed until DFLASH_MCR[DONE] is set at the completion of the requested operation. Likewise, the SLK register is not writable if a high voltage operation is suspended.</p> <p>Upon reset, information from the TestFlash block is loaded into the SLK registers. The SLK bits may be written as a register. Reset will cause the bits to go back to their TestFlash block value. The default value of the SLK bits (assuming erased fuses) would be locked.</p> <p>In the event that blocks are not present (due to configuration or total memory size), the SLK bits will default to locked, and will not be writable. The reset value will always be 1 (independent of the TestFlash block), and register writes will have no effect.</p> <p>In the 64 KB flash memory module bits SLK[15:4] are read-only and locked at 1.</p> <p>SLK is not writable unless SLE is high.</p> <p>0 Low address space block is unlocked and can be modified (also if DFLASH_LML[LLK] = 0).</p> <p>1 Low address space block is locked and cannot be modified.</p> |

### 30.5.2.4 DFlash Low/Mid Address Space Block Select Register (DFLASH\_LMS)

The DFLASH\_LMS register provides a means to select blocks to be operated on during erase.

Offset: 0x00010

Access: Read/write

|       |    |    |    |    |    |    |    |    |    |    |    |    |     |    |    |    |
|-------|----|----|----|----|----|----|----|----|----|----|----|----|-----|----|----|----|
|       | 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12  | 13 | 14 | 15 |
| R     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 0  | 0  | 0  |
| W     |    |    |    |    |    |    |    |    |    |    |    |    |     |    |    |    |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 0  | 0  | 0  |
|       | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28  | 29 | 30 | 31 |
| R     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | LSL |    |    |    |
| W     |    |    |    |    |    |    |    |    |    |    |    |    |     |    |    |    |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 0  | 0  | 0  |

Figure 30-31. DFlash Low/Mid Address Space Block Select Register (DFLASH\_LMS)

Table 30-46. DFLASH\_LMS field descriptions

| Field | Description   |
|-------|---|
| LSL   | <p><i>Low Address Space Block Select</i></p> <p>A value of 1 in the select register signifies that the block is selected for erase.</p> <p>A value of 0 in the select register signifies that the block is not selected for erase. The reset value for the select register is 0, or not selected.</p> <p>LSL[3:0] are related to sectors B1F3-0, respectively. LSL[15:4] are not used for this memory cut.</p> <p>The blocks must be selected (or deselected) before doing an erase interlock write as part of the erase sequence. The select register is not writable once an interlock write is completed or if a high voltage operation is suspended.</p> <p>In the event that blocks are not present (due to configuration or total memory size), the corresponding LSL bits will default to not selected, and will not be writable. The reset value will always be 0, and register writes will have no effect.</p> <p>In the 80 KB flash memory module bits LSL[15:4] are read-only and locked at 0.</p> <p>0 Low address space block is not selected for erase.</p> <p>1 Low address space block is selected for erase.</p> |

### 30.5.2.5 DFlash Address Register (DFLASH\_ADR)

The DFLASH\_ADR provides the first failing address in the event module failures (ECC, RWW or FPEC) occur or the first address at which an ECC single error correction occurs.

Address offset: 0x00018

Access: Read

|       |      |      |      |      |      |      |     |     |     |      |      |      |      |      |      |      |
|-------|------|------|------|------|------|------|-----|-----|-----|------|------|------|------|------|------|------|
|       | 0    | 1    | 2    | 3    | 4    | 5    | 6   | 7   | 8   | 9    | 10   | 11   | 12   | 13   | 14   | 15   |
| R     | 0    | 0    | 0    | 0    | 0    | 0    | 0   | 0   | 0   | AD22 | AD21 | AD20 | AD19 | AD18 | AD17 | AD16 |
| W     |      |      |      |      |      |      |     |     |     |      |      |      |      |      |      |      |
| Reset | 0    | 0    | 0    | 0    | 0    | 0    | 0   | 0   | 0   | 0    | 0    | 0    | 0    | 0    | 0    | 0    |
|       | 16   | 17   | 18   | 19   | 20   | 21   | 22  | 23  | 24  | 25   | 26   | 27   | 28   | 29   | 30   | 31   |
| R     | AD15 | AD14 | AD13 | AD12 | AD11 | AD10 | AD9 | AD8 | AD7 | AD6  | AD5  | AD4  | AD3  | 0    | 0    | 0    |
| W     |      |      |      |      |      |      |     |     |     |      |      |      |      |      |      |      |
| Reset | 0    | 0    | 0    | 0    | 0    | 0    | 0   | 0   | 0   | 0    | 0    | 0    | 0    | 0    | 0    | 0    |

Figure 30-32. DFlash Address Register (DFLASH\_ADR)

Table 30-47. DFLASH\_ADR field descriptions

| Field    | Description  |
|----------|--|
| AD[22:3] | <p><i>Address 22-3</i></p> <p>The Address Register provides the first failing address in the event of ECC error (DFLASH_MCR[EER] set) or the first failing address in the event of RWW error (DFLASH_MCR[RWE] set), or the address of a failure that may have occurred in a FPEC operation (DFLASH_MCR[PEG] cleared). The Address Register also provides the first address at which an ECC single error correction occurs (DFLASH_MCR[EDC] set), if the device is configured to show this feature.</p> <p>The ECC double error detection takes the highest priority, followed by the RWW error, the FPEC error and the ECC single error correction. When accessed DFLASH_ADR will provide the address related to the first event occurred with the highest priority. The priorities between these four possible events is summarized in the <a href="#">Table 30-48</a>.</p> <p>This address is always a double word address that selects 64 bits.</p> <p>In case of a simultaneous ECC double error detection on both double words of the same page, bit AD3 will output 0. The same is valid for a simultaneous ECC single error correction on both double words of the same page.</p> <p>In User mode, the Address register is read only.</p> |

Table 30-48. DFLASH\_ADR content: priority list

| Priority level | Error flag          | DFLASH_ADR content                           |
|----------------|---------------------|--|
| 1              | DFLASH_MCR[EER] = 1 | Address of first ECC double error            |
| 2              | DFLASH_MCR[RWE] = 1 | Address of first RWW error                   |
| 3              | DFLASH_MCR[PEG] = 0 | Address of first FPEC Error                  |
| 4              | DFLASH_MCR[EDC] = 1 | Address of first ECC single error correction |

### 30.5.2.6 DFlash User Test 0 register (DFLASH\_UT0)

The User Test Registers provide the user with the ability to test features on the flash memory module.



The User Test 0 Register allows to control the way in which the flash memory content check is done. Bits MRE, MRV, AIS, EIE, and DSI[7:0] of the User Test 0 Register are not accessible whenever DFLASH\_MCR[DONE] or DFLASH\_UT0[AID] are low: reading returns indeterminate data while writing has no effect.

Offset: 0x0003C

Access: Read/write

|       |     |    |    |    |    |    |    |    |     |    |     |     |     |     |     |     |
|-------|-----|----|----|----|----|----|----|----|-----|----|-----|-----|-----|-----|-----|-----|
|       | 0   | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8   | 9  | 10  | 11  | 12  | 13  | 14  | 15  |
| R     | UTE | 0  | 0  | 0  | 0  | 0  | 0  | 0  | DSI |    |     |     |     |     |     |     |
| W     |     |    |    |    |    |    |    |    |     |    |     |     |     |     |     |     |
| Reset | 0   | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 0  | 0   | 0   | 0   | 0   | 0   | 0   |
|       | 16  | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24  | 25 | 26  | 27  | 28  | 29  | 30  | 31  |
| R     | 0   | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | X  | MRE | MRV | EIE | AIS | AIE | AID |
| W     |     |    |    |    |    |    |    |    |     |    |     |     |     |     |     |     |
| Reset | 0   | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 0  | 0   | 0   | 0   | 0   | 0   | 1   |

Figure 30-33. DFlash User Test 0 register (DFLASH\_UT0)

Table 30-49. DFLASH\_UT0 field descriptions

| Field | Description  |
|-------|--|
| UTE   | <p><i>User Test Enable</i></p> <p>This status bit gives indication when User Test is enabled. All bits in DFLASH_UT0-2 and DFLASH_UMISR0-4 are locked when this bit is 0.</p> <p>This bit is not writable to a 1, but may be cleared. The reset value is 0.</p> <p>The method to set this bit is to provide a password, and if the password matches, the UTE bit is set to reflect the status of enabled, and is enabled until it is cleared by a register write.</p> <p>For UTE the password 0xF9F99999 must be written to the DFLASH_UT0 register.</p> |
| DSI   | <p><i>Data Syndrome Input</i></p> <p>These bits represent the input of Syndrome bits of ECC logic used in the ECC logic check. Bits DSI[7:0] correspond to the 8 syndrome bits on a double word.</p> <p>These bits are not accessible whenever DFLASH_MCR[DONE] or DFLASH_UT0[AID] are low: reading returns indeterminate data while writing has no effect.</p> <p>0 The syndrome bit is forced at 0.</p> <p>1 The syndrome bit is forced at 1.</p>  |
| X     | <p><i>Reserved</i></p> <p>This bit can be written and its value can be read back, but there is no function associated.</p> <p>This bit is not accessible whenever DFLASH_MCR[DONE] or DFLASH_UT0[AID] are low: reading returns indeterminate data while writing has no effect.</p>   |

Table 30-49. DFLASH\_UT0 field descriptions (continued)

| Field | Description   |
|-------|---|
| MRE   | <p><i>Margin Read Enable</i></p> <p>MRE enables margin reads to be done. This bit, combined with MRV, enables regular User mode reads to be replaced by margin reads. Margin reads are only active during array integrity checks; normal user reads are not affected by MRE. This bit is not accessible whenever DFLASH_MCR[DONE] or DFLASH_UT0[AID] are low: reading returns indeterminate data while writing has no effect.</p> <p>0 Margin reads are not enabled. All reads are User mode reads.<br/>1 Margin reads are enabled.</p>   |
| MRV   | <p><i>Margin Read Value</i></p> <p>If MRE is high, MRV selects the margin level that is being checked. Margin can be checked to an erased level (MRV = 1) or to a programmed level (MRV = 0). This bit is not accessible whenever DFLASH_MCR[DONE] or DFLASH_UT0[AID] are low: reading returns indeterminate data while writing has no effect.</p> <p>0 Zeros (programmed) margin reads are requested (if MRE = 1).<br/>1 Ones (erased) margin reads are requested (if MRE = 1).</p>  |
| EIE   | <p><i>ECC Data Input Enable</i></p> <p>EIE enables the ECC logic check operation to be done. This bit is not accessible whenever DFLASH_MCR[DONE] or DFLASH_UT0[AID] are low: reading returns indeterminate data while writing has no effect.</p> <p>0 ECC logic check is not enabled.<br/>1 ECC logic check is enabled.</p>  |
| AIS   | <p><i>Array Integrity Sequence</i></p> <p>AIS determines the address sequence to be used during array integrity checks or margin read. The default sequence (AIS = 0) is meant to replicate sequences normal user code follows, and thoroughly checks the read propagation paths. This sequence is proprietary. The alternative sequence (AIS = 1) is just logically sequential. Proprietary sequence is forbidden in margin read.</p> <p>It should be noted that the time to run a sequential sequence is significantly shorter than the time to run the proprietary sequence.</p> <p>This bit is not accessible whenever DFLASH_MCR[DONE] or DFLASH_UT0[AID] are low: reading returns indeterminate data while writing has no effect.</p> <p>0 Array integrity sequence is proprietary sequence.<br/>1 Array integrity or margin read sequence is sequential.</p> |
| AIE   | <p><i>Array Integrity Enable</i></p> <p>AIE set to 1 starts the array integrity check done on all selected and unlocked blocks. The pattern is selected by AIS, and the MISR (DFLASH_UMISR0-4) can be checked after the operation is complete, to determine if a correct signature is obtained. AIE can be set only if DFLASH_MCR[ERS], DFLASH_MCR[PGM], and DFLASH_MCR[EHV] are all low.</p> <p>0 Array integrity checks are not enabled.<br/>1 Array integrity checks are enabled.</p>  |
| AID   | <p><i>Array Integrity Done</i></p> <p>aid will be cleared upon an array integrity check being enabled (to signify the operation is on-going). Once completed, AID will be set to indicate that the array integrity check is complete. At this time the MISR (DFLASH_UMISR0-4) can be checked.</p> <p>0 Array integrity check is on-going.<br/>1 Array integrity check is done.</p>  |

### 30.5.2.7 DFlash User Test 1 register (DFLASH\_UT1)

The DFLASH\_UT1 register allows to enable the checks on the ECC logic related to the 32 LSB of the double word.

The User Test 1 Register is not accessible whenever DFLASH\_MCR[DONE] or DFLASH\_UT0[AID] are low: reading returns indeterminate data while writing has no effect.

Address offset: 0x00040

Access: Read/write

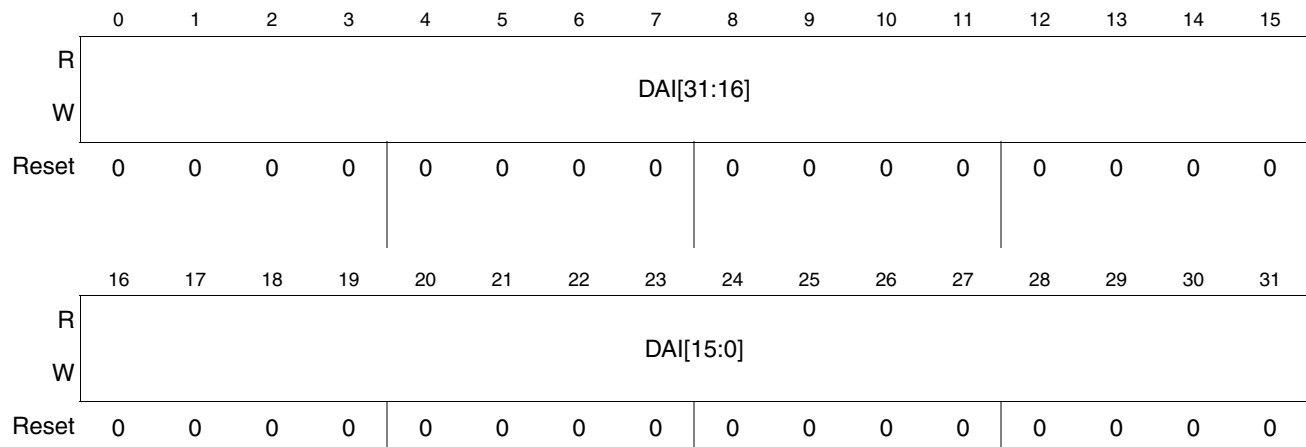


Figure 30-34. DFlash User Test 1 register (DFLASH\_UT1)

Table 30-50. DFLASH\_UT1 field descriptions

| Field      | Description  |
|------------|--|
| DAI[31:16] | <p><i>Data Array Input, bits 31-0</i></p> <p>These bits represent the input of even word of ECC logic used in the ECC logic check. Bits DAI[31:00] correspond to the 32 array bits representing Word 0 within the double word.</p> <p>0 The array bit is forced at 0.</p> <p>1 The array bit is forced at 1.</p> |

### 30.5.2.8 DFlash User Test 2 register (DFLASH\_UT2)

The DFLASH\_UT2 register allows to enable the checks on the ECC logic related to the 32 MSB of the double word.

The User Test 2 Register is not accessible whenever DFLASH\_MCR[DONE] or DFLASH\_UT0[AID] are low: reading returns indeterminate data while writing has no effect.

Offset: 0x00044

Reset value: 0x0000\_0000

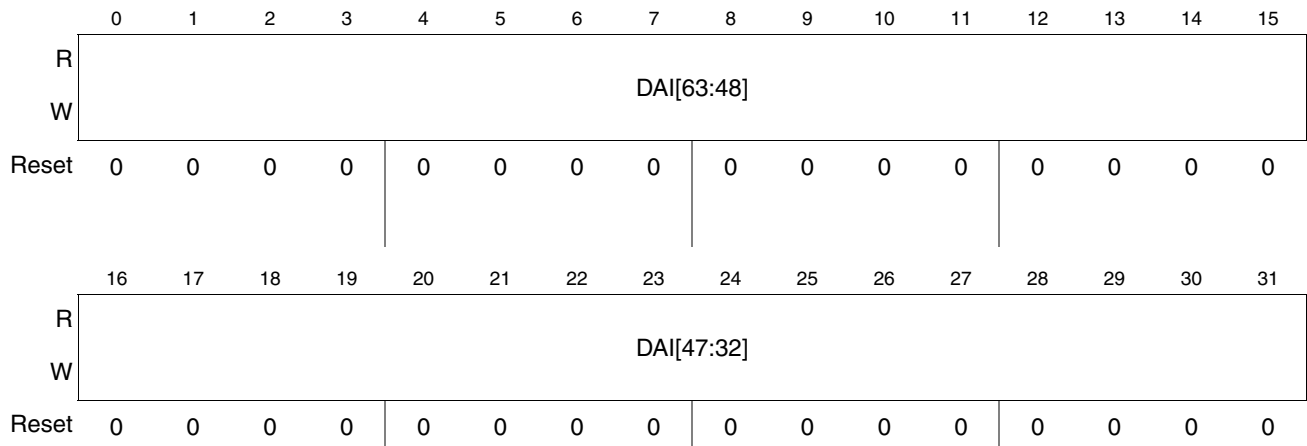


Figure 30-35. DFlash User Test 2 register (DFLASH\_UT2)

Table 30-51. DFLASH\_UT2 field descriptions

| Field      | Description   |
|------------|---|
| DAI[63:32] | <p><i>Data Array Input, bits 63-32</i></p> <p>These bits represent the input of odd word of ECC logic used in the ECC logic check. Bits DAI[63:32] correspond to the 32 array bits representing Word 1 within the double word.</p> <p>0 The array bit is forced at 0.<br/>1 The array bit is forced at 1.</p> |

### 30.5.2.9 DFlash User Multiple Input Signature Register 0 (DFLASH\_UMISR0)

The DFLASH\_UMISR0 provides a means to evaluate the array integrity.

The DFLASH\_UMISR0 represents the bits 31:0 of the whole 144 bits word (2 double words including ECC).

The DFLASH\_UMISR0 is not accessible whenever DFLASH\_MCR[DONE] or DFLASH\_UT0[AID] are low: reading returns indeterminate data while writing has no effect.

Address offset: 0x00048

Reset value: 0x0000\_0000

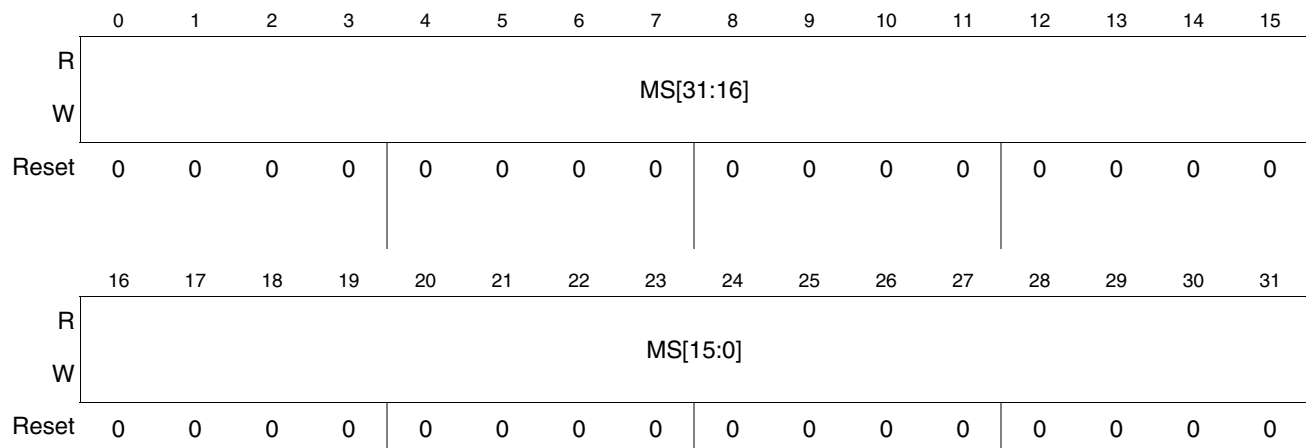


Figure 30-36. DFlash User Multiple Input Signature Register 0 (DFLASH\_UMISR0)

Table 30-52. DFLASH\_UMISR0 field descriptions

| Field    | Description   |
|----------|---|
| MS[31:0] | <p><i>Multiple Input Signature, bits 31–0</i></p> <p>These bits represent the MISR value obtained accumulating the bits 31:0 of all the pages read from the flash memory.</p> <p>The MS can be seeded to any value by writing the DFLASH_UMISR0 register.</p> |

### 30.5.2.10 DFlash User Multiple Input Signature Register 1 (DFLASH\_UMISR1)

The DFLASH\_UMISR1 provides a mean to evaluate the array integrity.

The DFLASH\_UMISR1 represents the bits 63:32 of the whole 144 bits word (2 double words including ECC).

The DFLASH\_UMISR1 is not accessible whenever DFLASH\_MCR[DONE] or DFLASH\_UT0[AID] are low: reading returns indeterminate data while writing has no effect.

Address offset: 0x0004C

Reset value: 0x0000\_0000

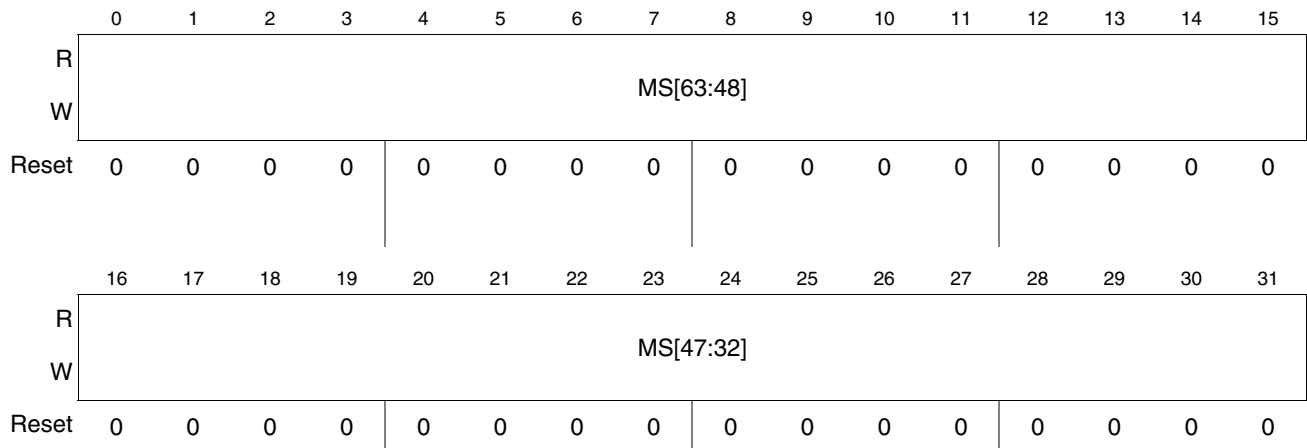


Figure 30-37. DFlash User Multiple Input Signature Register 1 (DFLASH\_UMISR1)

Table 30-53. DFLASH\_UMISR1 field descriptions

| Field     | Description   |
|-----------|---|
| MS[63:32] | <i>Multiple Input Signature, bits 63-32</i><br>These bits represent the MISR value obtained accumulating the bits 63:32 of all the pages read from the flash memory.<br>The MS can be seeded to any value by writing the DFLASH_UMISR1. |

### 30.5.2.11 DFlash User Multiple Input Signature Register 2 (DFLASH\_UMISR2)

The DFLASH\_UMISR2 provides a mean to evaluate the array integrity.

The DFLASH\_UMISR2 represents the bits 95:64 of the whole 144 bits word (2 double words including ECC).

The DFLASH\_UMISR2 is not accessible whenever DFLASH\_MCR[*DONE*] or DFLASH\_UT0[*AID*] are low: reading returns indeterminate data while writing has no effect.

Address offset: 0x00050

Reset value: 0x0000\_0000

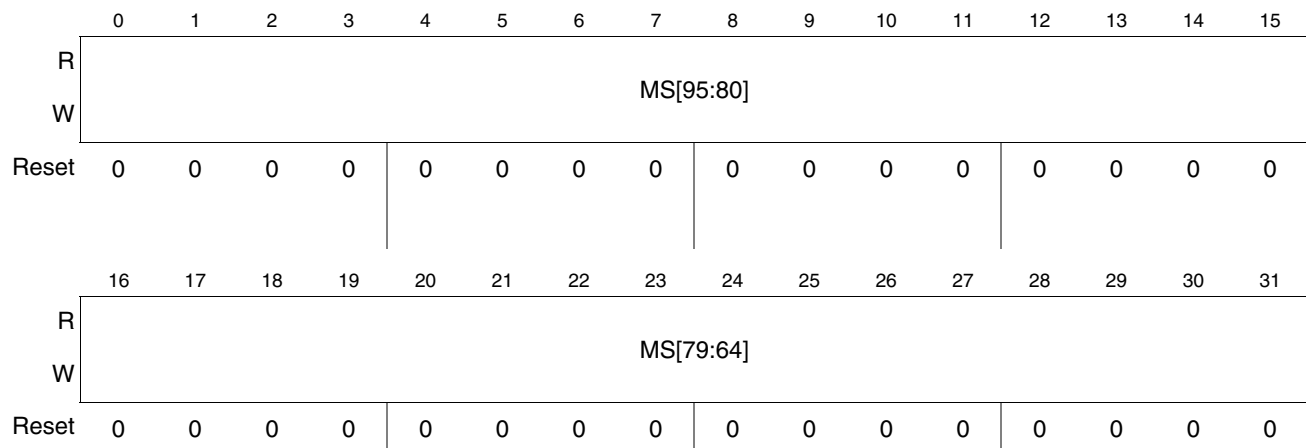


Figure 30-38. DFlash User Multiple Input Signature Register 2 (DFLASH\_UMISR2)

Table 30-54. DFLASH\_UMISR2 field descriptions

| Field     | Description  |
|-----------|--|
| MS[95:64] | <p><i>Multiple Input Signature, bits 95-64</i></p> <p>These bits represent the MISR value obtained accumulating the bits 95:64 of all the pages read from the flash memory.</p> <p>The MS can be seeded to any value by writing the DFLASH_UMISR2.</p> |

### 30.5.2.12 DFlash User Multiple Input Signature Register 3 (DFLASH\_UMISR3)

The DFLASH\_UMISR3 provides a mean to evaluate the array integrity.

The DFLASH\_UMISR3 represents the bits 127:96 of the whole 144 bits word (2 double words including ECC).

The DFLASH\_UMISR3 is not accessible whenever DFLASH\_MCR[DONE] or DFLASH\_UT0[AID] are low: reading returns indeterminate data while writing has no effect.

Address offset: 0x00054

Access: Read/write

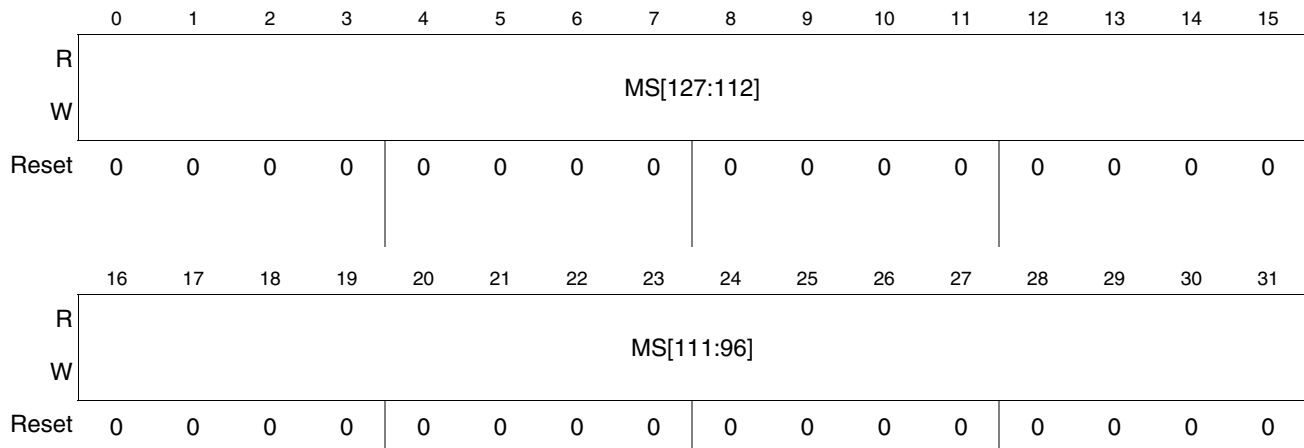


Figure 30-39. DFlash User Multiple Input Signature Register 3 (DFLASH\_UMISR3)

Table 30-55. DFLASH\_UMISR3 field descriptions

| Field      | Description  |
|------------|--|
| MS[127:96] | <p><i>Multiple Input Signature, bits 127:96</i></p> <p>These bits represent the MISR value obtained accumulating the bits 127:96 of all the pages read from the flash memory.</p> <p>The MS can be seeded to any value by writing the DFLASH_UMISR3.</p> |

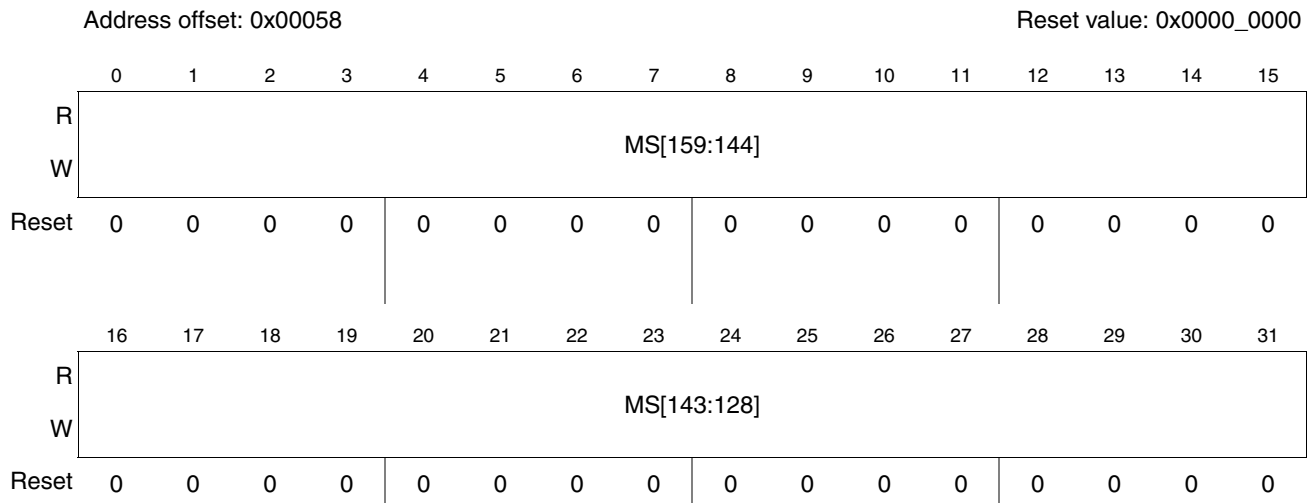
### 30.5.2.13 DFlash User Multiple Input Signature Register 4 (DFLASH\_UMISR4)

The Multiple Input Signature Register provides a mean to evaluate the array integrity.

The User Multiple Input Signature Register 4 represents the ECC bits of the whole 144 bits word (2 double words including ECC): bits 23-168:15 are ECC bits for the odd double word, and bits 7-024:31 are the ECC bits for the even double word; bits 27-264:5 and 11-1020:21 of MISR are respectively the double and single ECC error detection for odd and even double word.

The DFLASH\_UMISR4 Register is not accessible whenever DFLASH\_MCR[*DONE*] or DFLASH\_UT0[*AID*] are low: reading returns indeterminate data while writing has no effect.





**Figure 30-40. DFlash User Multiple Input Signature Register 4 (DFLASH\_UMISR4)**

**Table 30-56. DFLASH\_UMISR4 field descriptions**

| Field       | Description  |
|-------------|--|
| MS[159:128] | <p><i>Multiple Input Signature, bits 159-128</i></p> <p>These bits represent the MISR value obtained accumulating:</p> <ul style="list-style-type: none"> <li>• The 8 ECC bits for the even double word (on MS[135:128]);</li> <li>• The single ECC error detection for even double word (on MS138);</li> <li>• The double ECC error detection for even double word (on MS139);</li> <li>• The 8 ECC bits for the odd double word (on MS[151:144]);</li> <li>• The single ECC error detection for odd double word (on MS154);</li> <li>• The double ECC error detection for odd double word (on MS155).</li> </ul> <p>The MS can be seeded to any value by writing the DFLASH_UMISR4 register.</p> |

## 30.6 Programming considerations

In the following sections, register names can refer to the CFlash or DFlash versions of those registers. Thus, for example, the term MCR can refer to the CFLASH\_MCR or DFLASH\_MCR, based on context.

### 30.6.1 Modify operation

All modify operations of the flash memory module are managed through the flash memory user registers interface.

All the sectors of the flash memory module belong to the same partition (Bank), therefore when a Modify operation is active on some sectors no read access is possible on any other sector (Read-While-Write is not supported).

During a flash memory modify operation any attempt to read any flash memory location will output invalid data and bit MCR[RWE] will be automatically set. This means that the flash memory module is not fetchable when a modify operation is active and these commands must be executed from another memory (internal SRAM or another flash memory module).

If during a Modify Operation a reset occurs, the operation is suddenly terminated and the macrocell is reset to Read Mode. The data integrity of the flash memory section where the Modify Operation has been terminated is not guaranteed: the interrupted flash memory Modify Operation must be repeated.

In general each modify operation is started through a sequence of three steps:

1. The first instruction is used to select the desired operation by setting its corresponding selection bit in MCR (PGM or ERS) or UT0 (MRE or EIE).
2. The second step is the definition of the operands: the Address and the Data for programming or the Sectors for erase or margin read.
3. The third instruction is used to start the modify operation, by setting MCR[EHV] or UT0[AIE].

Once selected, but not yet started, one operation can be canceled by resetting the operation selection bit.

A summary of the available flash memory modify operations is shown in [Table 30-57](#).

**Table 30-57. Flash memory modify operations**

| Operation             | Select bit | Operands                             | Start bit |
|-----------------------|------------|--------------------------------------|-----------|
| Double word program   | MCR[PGM]   | Address and data by interlock writes | MCR[EHV]  |
| Sector erase          | MCR[ERS]   | LMS                                  | MCR[EHV]  |
| Array integrity check | None       | LMS                                  | UT0[AIE]  |
| Margin read           | UT0[MRE]   | UT0[MRV] + LMS                       | UT0[AIE]  |
| ECC logic check       | UT0[EIE]   | UT0[DSI], UT1, UT2                   | UT0[AIE]  |

Once the MCR[EHV] bit (or UT0[AIE]) is set, all the operands can no more be modified until the MCR[DONE] bit (or UT0[AID]) is high.

In general each modify operation is completed through a sequence of four steps:

1. Wait for operation completion: wait for the MCR[DONE] bit (or UT0[AID]) to go high.
2. Check operation result: check the MCR[PEG] bit (or compare UMISR0-4 with expected value).
3. Switch off FPEC by resetting the MCR[EHV] bit (or UT0[AIE]).
4. Deselect current operation by clearing the MCR[PGM] / MCR[ERS] fields (or UT0[MRE] /UT0[EIE]).

If the device embeds more than one flash memory module and a modify operation is on-going on one of them, then it is forbidden to start any other modify operation on the other flash memory modules.

In the following all the possible modify operations are described and some examples of the sequences needed to activate them are presented.

### 30.6.2 Double word program

A flash memory program sequence operates on any double word within the flash memory core.

As many as two words within the double word may be altered in a single program operation.

ECC is handled on a 64-bit boundary. Thus, if only one word in any given 64-bit ECC segment is programmed, the adjoining word (in that segment) should not be programmed since ECC calculation has already completed for that 64-bit segment. Attempts to program the adjoining word will probably result in an operation failure. It is recommended that all programming operations be of 64 bits. The programming operation should completely fill selected ECC segments within the double word.

Programming changes the value stored in an array bit from logic 1 to logic 0 only. Programming cannot change a stored logic 0 to a logic 1.

Addresses in locked/disabled blocks cannot be programmed.

The user may program the values in any or all of two words of a double word with a single program sequence.

Double word-bound words have addresses that differ only in address bit 2.

The program operation consists of the following sequence of events:

1. Change the value in the MCR[PGM] bit from 0 to 1.
2. Ensure the block that contains the address to be programmed is unlocked.
  - a) Write the first address to be programmed with the program data.
  - b) The flash memory module latches address bits (22:3) at this time.
  - c) The flash memory module latches data written as well.
  - d) This write is referred to as a program data interlock write. An interlock write may be as large as 64 bits, and as small as 32 bits (depending on the CPU bus).
3. If more than 1 word is to be programmed, write the additional address in the double word with data to be programmed. This is referred to as a program data write.  
The flash memory module ignores address bits (22:3) for program data writes.  
The eventual unwritten data word default to 0xFFFFFFFF.
4. Write a logic 1 to the MCR[EHV] bit to start the internal program sequence or skip to step 9 to terminate.
5. Wait until the MCR[DONE] bit goes high.
6. Confirm that the MCR[PEG] bit is 1.
7. Write a logic 0 to the MCR[EHV] bit.
8. If more addresses are to be programmed, return to step 2.
9. Write a logic 0 to the MCR[PGM] bit to terminate the program operation.

Program may be initiated with the 0-to-1 transition of the MCR[PGM] bit or by clearing the MCR[EHV] bit at the end of a previous program.

The first write after a program is initiated determines the page address to be programmed. This first write is referred to as an interlock write. The interlock write determines if the shadow, test or normal array space will be programmed by causing the MCR[PEAS] field to be set/cleared.

An interlock write must be performed before setting MCR[EHV]. The user may terminate a program sequence by clearing MCR[PGM] prior to setting MCR[EHV].

After the interlock write, additional writes only affect the data to be programmed at the word location determined by address bit 2. Unwritten locations default to a data value of 0xFFFFFFFF. If multiple writes are done to the same location the data for the last write is used in programming.

While MCR[DONE] is low and MCR[EHV] is high, the user may clear EHV, resulting in a program abort. A program abort forces the module to step 8 of the program sequence.

An aborted program will result in MCR[PEG] being set low, indicating a failed operation. MCR[DONE] must be checked to know when the aborting command has completed.

The data space being operated on before the abort will contain indeterminate data. This may be recovered by repeating the same program instruction or executing an erase of the affected blocks.

**Example 30-1. Double word program of data 0x55AA55AA at address 0x00AAA8  
and data 0xAA55AA55 at address 0x00AAAC**

---

```

MCR          = 0x00000010;          /* Set PGM in MCR: Select Operation */
(0x00AAA8)   = 0x55AA55AA;          /* Latch Address and 32 LSB data */
(0x00AAAC)   = 0xAA55AA55;          /* Latch 32 MSB data */
MCR          = 0x00000011;          /* Set EHV in MCR: Operation Start */
do
{ tmp        = MCR;                /* Loop to wait for DONE=1 */
} while ( !(tmp & 0x00000400) );    /* Read MCR */
status       = MCR & 0x00000200;    /* Check PEG flag */
MCR          = 0x00000010;          /* Reset EHV in MCR: Operation End */
MCR          = 0x00000000;          /* Reset PGM in MCR: Deselect Operation */

```

---

### 30.6.3 Sector erase

Erase changes the value stored in all bits of the selected block(s) to logic 1.

An erase sequence operates on any combination of blocks (sectors) in the low, mid or high address space, or the shadow sector (if available). The test block cannot be erased.

The erase sequence is fully automated within the flash memory. The user only needs to select the blocks to be erased and initiate the erase sequence.

Locked/disabled blocks cannot be erased.

If multiple blocks are selected for erase during an erase sequence, no specific operation order must be assumed.

The erase operation consists of the following sequence of events:

1. Change the value in the MCR[ERS] bit from 0 to 1.
2. Select the block(s) to be erased by writing 1s to the appropriate bit(s) in the LMS register. If the shadow sector is to be erased, this step may be skipped, and LMS is ignored. Note that Lock and Select are independent. If a block is selected and locked, no erase will occur.
3. Write to any address in flash memory. This is referred to as an erase interlock write.
4. Write a logic 1 to the MCR[EHV] bit to start the internal erase sequence or skip to step 9 to terminate.
5. Wait until the MCR[DONE] bit goes high.

6. Confirm MCR[PEG] = 1.
7. Write a logic 0 to the MCR[EHV] bit.
8. If more blocks are to be erased, return to step 2.
9. Write a logic 0 to the MCR[ERS] bit to terminate the erase operation.

After setting MCR[ERS], one write, referred to as an interlock write, must be performed before MCR[EHV] can be set to 1. Data words written during erase sequence interlock writes are ignored.

The user may terminate the erase sequence by clearing ERS before setting EHV.

An erase operation may be aborted by clearing MCR[EHV] assuming MCR[DONE] is low, MCR[EHV] is high, and MCR[ESUS] is low.

An erase abort forces the module to step 8 of the erase sequence.

An aborted erase will result in MCR[PEG] being set low, indicating a failed operation. MCR[DONE] must be checked to know when the aborting command has completed.

The block(s) being operated on before the abort contain indeterminate data. This may be recovered by executing an erase on the affected blocks.

The user may not abort an erase sequence while in erase suspend.

#### Example 30-2. Erase of sectors B0F1 and B0F2

---

```

MCR          = 0x00000004;          /* Set ERS in MCR: Select Operation */
LMS          = 0x00000006;          /* Set LSL2-1 in LMS: Select Sectors to erase */
(0x000000)   = 0xFFFFFFFF;         /* Latch a flash memory Address with any data */
MCR          = 0x00000005;          /* Set EHV in MCR: Operation Start */
do
{ tmp        = MCR;                /* Loop to wait for DONE=1 */
} while ( !(tmp & 0x00000400) );    /* Read MCR */
status      = MCR & 0x00000200;    /* Check PEG flag */
MCR        = 0x00000004;          /* Reset EHV in MCR: Operation End */
MCR        = 0x00000000;          /* Reset ERS in MCR: Deselect Operation */

```

### 30.6.3.1 Erase suspend/resume

The erase sequence may be suspended to allow read access to the flash memory core.

It is not possible to program or to erase during an erase suspend.

During erase suspend, all reads to blocks targeted for erase return indeterminate data.

An erase suspend can be initiated by changing the value of the MCR[ESUS] bit from 0 to 1. MCR[ESUS] can be set to 1 at any time when MCR[ERS] and MCR[EHV] are high and MCR[PGM] is low. A 0-to-1 transition of MCR[ESUS] causes the module to start the sequence that places it in erase suspend.

The user must wait until MCR[DONE] = 1 before the module is suspended and further actions are attempted. MCR[DONE] will go high no more than  $t_{ESUS}$  after MCR[ESUS] is set to 1.

Once suspended, the array may be read. flash memory core reads while MCR[ESUS] = 1 from the block(s) being erased return indeterminate data.

### Example 30-3. Sector erase suspend

---

```
MCR          = 0x00000007;          /* Set ESUS in MCR: Erase Suspend */
do
/* Loop to wait for DONE=1 */
{ tmp      = MCR;          /* Read MCR */
} while ( !(tmp & 0x00000400) );
```

Notice that there is no need to clear MCR[EHV] and MCR[ERS] in order to perform reads during erase suspend.

The erase sequence is resumed by writing a logic 0 to MCR[ESUS].

MCR[EHV] must be set to 1 before MCR[ESUS] can be cleared to resume the operation.

The module continues the erase sequence from one of a set of predefined points. This may extend the time required for the erase operation.

### Example 30-4. Sector erase resume

---

```
MCR          = 0x00000005;          /* Reset ESUS in MCR: Erase Resume */
```

## 30.6.3.2 User Test mode

The user can perform specific tests to check flash memory module integrity by putting the flash memory module in User Test mode.

Three kinds of test can be performed:

- Array integrity self check
- Margin read
- ECC logic check

The User Test Mode is equivalent to a Modify operation: read accesses attempted by the user during User Test Mode generates a Read-While-Write Error (MCR[RWE] set).

It is not allowed to perform User Test operations on the Test and shadow sectors.

### 30.6.3.2.1 Array integrity self check

Array integrity is checked using a predefined address sequence (proprietary), and this operation is executed on selected and unlocked blocks. Once the operation is completed, the results of the reads can be checked by reading the MISR value (stored in UMISR0–4), to determine if an incorrect read, or ECC detection was noted.

The internal MISR calculator is a 32-bit register.

The 128-bit data, the 16 ECC data, and the single and double ECC errors of the two double words are therefore captured by the MISR through five different read accesses at the same location.

The whole check is done through five complete scans of the memory address space:

1. The first pass will scan only bits 31:0 of each page.

2. The second pass will scan only bits 63:32 of each page.
3. The third pass will scan only bits 95:64 of each page.
4. The fourth pass will scan only bits 127:96 of each page.
5. The fifth pass will scan only the ECC bits (8 + 8) and the single and double ECC errors (2 + 2) of both double words of each page.

The 128-bit data and the 16 ECC data are sampled before the eventual ECC correction, while the single and double error flags are sampled after the ECC evaluation.

Only data from existing and unlocked locations are captured by the MISR.

The MISR can be seeded to any value by writing the UMISR0–4 registers.

The Array Integrity Self Check consists of the following sequence of events:

1. Set UT0[UTE] by writing the related password in UT0.
2. Select the block(s) to be checked by writing 1s to the appropriate bit(s) in the LMS register. Note that Lock and Select are independent. If a block is selected and locked, no Array Integrity Check will occur.
3. Set eventually UT0[AIS] bit for a sequential addressing only.
4. Write a logic 1 to the UT0[AIE] bit to start the Array Integrity Check.
5. Wait until the UT0[AID] bit goes high.
6. Compare UMISR0-4 content with the expected result.
7. Write a logic 0 to the UT0[AIE] bit.
8. If more blocks are to be checked, return to step 2.

It is recommended to leave UT0[AIS] at 0 and use the proprietary address sequence that checks the read path more fully, although this sequence takes more time. During the execution of the Array Integrity Check operation it is forbidden to modify the content of Block Select (LMS) and Lock (LML, SLL) registers, otherwise the MISR value can vary in an unpredictable way. While UT0[AID] is low and UT0[AIE] is high, the User may clear AIE, resulting in a Array Integrity Check abort.

UT0[AID] must be checked to know when the aborting command has completed.

#### Example 30-5. Array integrity check of sectors B0F1 and B0F2

```

UT0          = 0xF9F99999;          /* Set UTE in UT0: Enable User Test */
LMS          = 0x00000006;          /* Set LSL2-1 in LMS: Select Sectors */
UT0          = 0x80000002;          /* Set AIE in UT0: Operation Start */
do
/* Loop to wait for AID=1 */
{ tmp        = UT0;                /* Read UT0 */
} while ( !(tmp & 0x00000001) );
data0        = UMISR0;              /* Read UMISR0 content*/
data1        = UMISR1;              /* Read UMISR1 content*/
data2        = UMISR2;              /* Read UMISR2 content*/
data3        = UMISR3;              /* Read UMISR3 content*/
data4        = UMISR4;              /* Read UMISR4 content*/
UT0          = 0x00000000;          /* Reset UTE and AIE in UT0: Operation End */

```

### 30.6.3.2.2 Margin read

Margin read procedure (either Margin 0 or Margin 1), can be run on unlocked blocks in order to unbalance the Sense Amplifiers, respect to standard read conditions, so that all the read accesses reduce the margin vs. 0 ( $UT0[MRV] = 0$ ) or vs. 1 ( $UT0[MRV] = 1$ ). Locked sectors are ignored by MISR calculation and ECC flagging. The results of the margin reads can be checked comparing checksum value in UMISR0–4. Since Margin reads are done at voltages that differ than the normal read voltage, lifetime expectancy of the flash memory macrocell is impacted by the execution of margin reads. Doing margin reads repetitively results in degradation of the flash memory array, and shorten expected lifetime experienced at normal read levels. For these reasons the margin read usage is allowed only in Factory, while it is forbidden to use it inside the user application.

In any case the charge losses detected through the margin read cannot be considered failures of the device and no failure analysis will be opened on them. The margin read setup operation consists of the following sequence of events:

1. Set  $UT0[UTE]$  by writing the related password in  $UT0$ .
2. Select the block(s) to be checked by writing 1s to the appropriate bit(s) in the LMS register.

Note that Lock and Select are independent. If a block is selected and locked, no Array Integrity Check will occur.

3. Set  $T0.AIS$  bit for a sequential addressing only.
4. Change the value in the  $UT0[MRE]$  bit from 0 to 1.
5. Select the margin level:  $UT0[MRV]=0$  for 0s margin,  $UT0[MRV]=1$  for 1s margin.
6. Write a logic 1 to the  $UT0[AIE]$  bit to start the margin read setup or skip to step 6 to terminate.
7. Wait until the  $UT0[AID]$  bit goes high.
8. Compare UMISR0-4 content with the expected result.
9. Write a logic 0 to the  $UT0[AIE]$ ,  $UT0[MRE]$ , and  $UT0[MRV]$  bits.
10. If more blocks are to be checked, return to step 2.

It is mandatory to leave  $UT0[AIS]$  at 1 and use the linear address sequence, the usage of the proprietary sequence in margin read is forbidden.

During the execution of the margin read operation it is forbidden to modify the content of Block Select (LMS) and Lock (LML, SLL) registers, otherwise the MISR value can vary in an unpredictable way.

The read accesses will be done with the addition of a proper number of Wait States to guarantee the correctness of the result.

While  $UT0[AID]$  is low and  $UT0[AIE]$  is high, the User may clear AIE, resulting in a Array Integrity Check abort.

$UT0[AID]$  must be checked to know when the aborting command has completed.

#### Example 30-6. Margin read setup versus 1s

---

```

UMISR0      = 0x00000000;          /* Reset UMISR0 content */
UMISR1      = 0x00000000;          /* Reset UMISR1 content */
UMISR2      = 0x00000000;          /* Reset UMISR2 content */
UMISR3      = 0x00000000;          /* Reset UMISR3 content */

```



```

UMISR4      = 0x00000000;          /* Reset UMISR4 content */
UT0         = 0xF9F99999;          /* Set UTE in UT0: Enable User Test */
LMS         = 0x00000006;          /* Set LSL2-1 in LMS: Select Sectors */
UT0         = 0x80000004;          /* Set AIS in UT0: Select Operation */
UT0         = 0x80000024;          /* Set MRE in UT0: Select Operation */
UT0         = 0x80000034;          /* Set MRV in UT0: Select Margin versus 1's */
UT0         = 0x80000036;          /* Set AIE in UT0: Operation Start */
do
/* Loop to wait for AID=1 */
{ tmp = UT0;                       /* Read UT0 */
} while ( !(tmp & 0x00000001) );
data0       = UMISR0;              /* Read UMISR0 content*/
data1       = UMISR1;              /* Read UMISR1 content*/
data2       = UMISR2;              /* Read UMISR2 content*/
data3       = UMISR3;              /* Read UMISR3 content*/
data4       = UMISR4;              /* Read UMISR4 content*/
UT0         = 0x80000034;          /* Reset AIE in UT0: Operation End */
UT0         = 0x00000000;          /* Reset UTE, MRE, MRV, AIS in UT0: Deselect Op. */

```

To exit from the margin read mode, a read reset operation must be executed.

### 30.6.3.2.3 ECC logic check

ECC logic can be checked by forcing the input of ECC logic: The 64 bits of data and the 8 bits of ECC syndrome can be individually forced and they will drive simultaneously at the same value the ECC logic of the whole page (2 double words).

The results of the ECC logic check can be verified by reading the MISR value.

The ECC logic check operation consists of the following sequence of events:

1. Set UT0[UTE] by writing the related password in UT0.
2. Write in UT1[DAI31–0] and UT2[DAI63–32] the double word input value.
3. Write in UT0[DSI7–0] the Syndrome Input value.
4. Select the ECC logic check: write a logic 1 to the UT0[EIE] bit.
5. Write a logic 1 to the UT0[AIE] bit to start the ECC logic check.
6. Wait until the UT0[AID] bit goes high.
7. Compare UMISR0–4 content with the expected result.
8. Write a logic 0 to the UT0[AIE] bit.

Notice that when UT0[AID] is low, UMISR0–4, UT1–2, and bits MRE, MRV, EIE, AIS, and DSI7–0 of UT0 are not accessible: reading returns indeterminate data and write has no effect.

#### Example 30-7. ECC logic check

```

UT0         = 0xF9F99999;          /* Set UTE in UT0: Enable User Test */
UT1         = 0x55555555;          /* Set DAI31-0 in UT1: Even Word Input Data */
UT2         = 0xAAAAAAAA;          /* Set DAI63-32 in UT2: Odd Word Input Data */
UT0         = 0x80FF0000;          /* Set DSI7-0 in UT0: Syndrome Input Data */
UT0         = 0x80FF0008;          /* Set EIE in UT0: Select ECC Logic Check */
UT0         = 0x80FF000A;          /* Set AIE in UT0: Operation Start */
do
/* Loop to wait for AID=1 */
{ tmp
/* Read UT0 */
= UT0;
} while ( !(tmp & 0x00000001) );
data0       = UMISR0;              /* Read UMISR0 content (expected 0x55555555) */

```

```

data1          = UMISR1;          /* Read UMISR1 content (expected 0xAAAAAAAA) */
data2          = UMISR2;          /* Read UMISR2 content (expected 0x55555555) */
data3          = UMISR3;          /* Read UMISR3 content (expected 0xAAAAAAAA) */
data4          = UMISR4;          /* Read UMISR4 content (expected 0x00FF00FF) */
UT0            = 0x00000000;      /* Reset UTE, AIE and EIE in UT0: Operation End */

```

### 30.6.3.3 Error Correction Code (ECC)

The flash memory module provides a method to improve the reliability of the data stored in flash memory: the usage of an error correction code. The word size is fixed at 64 bits.

Eight ECC bits, programmed to guarantee a single error correction and a double error detection (SEC-DED), are associated to each 64-bit double word.

ECC circuitry provides correction of single bit faults and is used to achieve automotive reliability targets. Some units will experience single bit corrections throughout the life of the product with no impact to product reliability.

#### 30.6.3.3.1 ECC algorithms

The flash memory module supports one ECC algorithm: “All 1s No Error”. A modified Hamming code is used that ensures the all erased state (that is, 0xFFFF....FFFF) data is a valid state, and will not cause an ECC error. This allows the user to perform a blank check after a sector erase operation.

#### 30.6.3.4 EEPROM emulation

The chosen ECC algorithm allows some bit manipulations so that a double word can be rewritten several times without needing an erase of the sector. This allows to use a double word to store flags useful for the EEPROM emulation. As an example, the chosen ECC algorithm allows to start from an All 1s double word value and rewrite whichever of its four 16-bit half-words to an All 0s content by keeping the same ECC value.

Table 30-58 shows a set of double words sharing the same ECC value.

**Table 30-58. Bit manipulation: double words with the same ECC value**

| Double word           | ECC all 1s no error |
|-----------------------|---------------------|
| 0xFFFF_FFFF_FFFF_FFFF | 0xFF                |
| 0xFFFF_FFFF_FFFF_0000 | 0xFF                |
| 0xFFFF_FFFF_0000_FFFF | 0xFF                |
| 0xFFFF_0000_FFFF_FFFF | 0xFF                |
| 0x0000_FFFF_FFFF_FFFF | 0xFF                |
| 0xFFFF_FFFF_0000_0000 | 0xFF                |
| 0xFFFF_0000_FFFF_0000 | 0xFF                |
| 0x0000_FFFF_FFFF_0000 | 0xFF                |
| 0xFFFF_0000_0000_FFFF | 0xFF                |
| 0x0000_FFFF_0000_FFFF | 0xFF                |

**Table 30-58. Bit manipulation: double words with the same ECC value (continued)**

| Double word           | ECC all 1s no error |
|-----------------------|---------------------|
| 0x0000_0000_FFFF_FFFF | 0xFF                |
| 0xFFFF_0000_0000_0000 | 0xFF                |
| 0x0000_FFFF_0000_0000 | 0xFF                |
| 0x0000_0000_0000_0000 | 0xFF                |

When some flash memory sectors are used to perform an Eeprom Emulation, it is recommended for safety reasons to reserve at least 3 sectors to this purpose.

#### 30.6.3.4.1 All 1s No Error

The All 1s No Error algorithm detects as valid any double word read on a just erased sector (all the 72 bits are 1s).

This option allows to perform a Blank Check after a Sector erase operation.

#### 30.6.3.5 Protection strategy

Two kinds of protection are available: Modify Protection to avoid unwanted program/erase in flash memory sectors and Censored Mode to avoid piracy.

##### 30.6.3.5.1 Modify protection

The flash memory modify protection information is stored in nonvolatile flash memory cells located in the TestFlash. This information is read once during the flash memory initialization phase following the exiting from Reset and is stored in volatile registers that act as actuators.

The reset state of all the volatile modify protection registers is the protected state.

All the nonvolatile modify protection registers can be programmed through a normal double word program operation at the related locations in TestFlash.

The nonvolatile modify protection registers cannot be erased.

- The nonvolatile modify protection registers are physically located in TestFlash their bits can be programmed to 0 only once and they can no more be restored to 1.
- The Volatile Modify Protection Registers are Read/Write registers whose bits can be written at 0 or 1 by the user application.

A software mechanism is provided to independently lock/unlock each low, mid, and high address space block against program and erase.

Software locking is done through the LML register.

An alternate means to enable software locking for blocks of low address space only is through the SLL.

All these registers have a nonvolatile image stored in TestFlash (NVLML, NVSLL), so that the locking information is kept on reset.

On delivery the TestFlash nonvolatile image is at all 1s, meaning all sectors are locked.

By programming the nonvolatile locations in TestFlash the selected sectors can be unlocked.

Being the TestFlash One-Time Programmable (that is, not erasable), once unlocked the sectors cannot be locked again.

Of course, on the contrary, all the volatile registers can be written at 0 or 1 at any time, therefore the user application can lock and unlock sectors when desired.

### 30.6.3.5.2 Censored mode

The Censored Mode information is stored in nonvolatile flash memory cells located in the Shadow Sector. This information is read once during the flash memory initialization phase following the exiting from Reset and is stored in volatile registers that act as actuators.

The reset state of all the Volatile Censored Mode registers is the protected state.

All the nonvolatile Censored Mode registers can be programmed through a normal double word program operation at the related locations in the Shadow Sector.

The nonvolatile Censored Mode registers can be erased by erasing the Shadow Sector.

- The nonvolatile Censored Mode Registers are physically located in the Shadow Sector their bits can be programmed to 0 and restored to 1 by erasing the Shadow Sector.
- The Volatile Censored Mode Registers are registers not accessible by the user application.

The flash memory module provides two levels of protection against piracy:

- If bits CW15:0 of NVSCC0 are programmed at 0x55AA and NVSC1 = NVSCC0 the Censored Mode is disabled, while all the other possible values enable the Censored Mode.
- If bits SC15:0 of NVSCC0 are programmed at 0x55AA and NVSC1 = NVSCC0 the Public Access is disabled, while all the other possible values enable the Public Access.

The parts are delivered to the user with Censored Mode and Public Access disabled.

## 30.7 Platform flash memory controller

### 30.7.1 Introduction

The platform flash memory controller acts as the interface between the system bus (AHB-Lite 2.v6) and as many as two banks of integrated flash memory arrays (Program and Data). It intelligently converts the protocols between the system bus and the dedicated flash memory array interfaces.

A block diagram of the e200z0h Power Architecture reduced product platform (RPP) reference design is shown below in [Figure 30-41](#) with the platform flash memory controller module and its attached off-platform flash memory arrays highlighted.

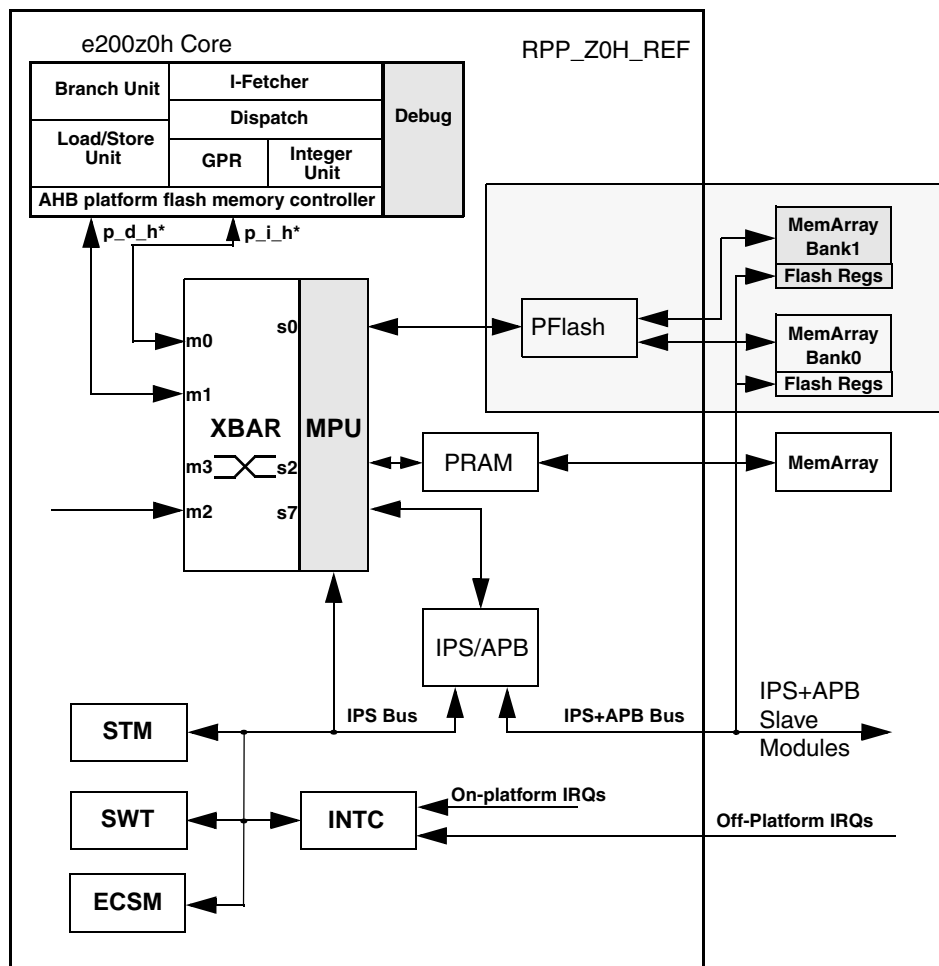


Figure 30-41. Power Architecture e200z0h RPP reference platform block diagram

The module list includes:

- Power Architecture e200z0h (Harvard) core
- AHB crossbar switch “lite” (XBAR)
- Memory Protection Unit (MPU)
- Platform flash memory controller with connections to 2 memory banks
- Platform SRAM memory controller (PRAM)
- AHB-to-IPS/APB bus controller (PBRIDGE) for access to on- and off-platform slave modules
- Interrupt Controller (INTC)
- 4-channel System Timers (STM)
- Software Watchdog Timer (SWT)
- Error Correction Status Module (ECSM)

Throughout this document, several important terms are used to describe the platform flash memory controller module and its connections. These terms are defined here:

- **Port** — This is used to describe the AMBA-AHB connection(s) into the platform flash memory controller. From an architectural and programming model viewpoint, the definition supports as many as two AHB ports, even though this specific controller only supports a single AHB connection.
- **Bank** — This term is used to describe the attached flash memories. From the platform flash memory controller’s perspective, there may be one or two attached banks of flash memory. The code flash memory is required and always attached to bank0. Additionally, there is a data flash memory attached to bank1. The platform flash memory controller interface supports two separate connections, one to each memory bank.
- **Array** — Within each memory bank, there is one flash memory array instantiations.
- **Page** — This value defines the number of bits read from the flash memory array in a single access. For this controller and memory, the page size is 128 bits (16 bytes).

The nomenclature “page buffers and “line buffers” are used interchangeably.

### 30.7.1.1 Overview

The platform flash memory controller supports a 32-bit data bus width at the AHB port and connections to 128-bit read data interfaces from two memory banks, where each bank contains one instantiations of the flash memory array. One flash memory bank is connected to the code flash memory and the other bank is connected to the optional data flash memory. The memory controller capabilities vary between the two banks with each bank’s functionality optimized with the typical use cases associated with the attached flash memory. As an example, the platform flash memory controller logic associated with the code flash memory bank contains a four-entry page buffer, each entry containing 128 bits of data (1 flash memory page) plus an associated controller that prefetches sequential lines of data from the flash memory array into the buffer, while the controller logic associated with the data flash memory bank only supports a 128-bit register that serves as a temporary page holding register and does not support any prefetching. Prefetch buffer hits from the code flash memory bank support zero-wait AHB data phase responses. AHB read requests that miss the buffers generate the needed flash memory array access and are forwarded to the AHB upon completion, typically incurring two wait-states at an operating frequency of 60–64 MHz.

This memory controller is optimized for applications where a cacheless processor core, e.g., the Power e200z0h, is connected through the platform to on-chip memories, e.g., flash memory and SRAM, where the processor and platform operate at the same frequency. For these applications, the 2-stage pipeline AMBA-AHB system bus is effectively mapped directly into stages of the processor’s pipeline and zero wait-state responses for most memory accesses are critical for providing the required level of system performance.

### 30.7.1.2 Features

The following list summarizes the key features of the platform flash memory controller:

- Dual array interfaces support up to a total of 16 MB of flash memory, partitioned as two separate 8 MB banks

- Single AHB port interface supports a 32-bit data bus. All AHB aligned and unaligned reads within the 32-bit container are supported. Only aligned word writes are supported.
- Array interfaces support a 128-bit read data bus and a 64-bit write data bus for each bank
- Interface with code flash memory (bank0) provides configurable read buffering and page prefetch support. Four page read buffers (each 128 bits wide) and a prefetch controller are used to support single-cycle read responses (zero AHB data phase wait-states) for hits in the buffers. The buffers implement a least recently used replacement algorithm to maximize performance.
- Interface with optional data flash memory (bank1) includes a 128-bit register to temporarily hold a single flash memory page. This logic supports single-cycle read responses (zero AHB data phase wait-states) for accesses that hit in the holding register. There is no support for prefetching associated with this bank.
- Programmable response for read-while-write sequences including support for stall-while-write, optional stall notification interrupt, optional flash memory operation abort, and optional abort notification interrupt
- Separate and independent configurable access timing (on a per bank basis) to support use across a wide range of platforms and frequencies
- Support of address-based read access timing for emulation of other memory types
- Support for reporting of single- and multi-bit flash memory ECC events
- Typical operating configuration loaded into programming model by system reset

## 30.7.2 Memory map and register description

Two memory maps are associated with the platform flash memory controller: one for the flash memory space and another for the program-visible control and configuration registers. The flash memory space is accessed via the AMBA-AHB port and the program-visible registers are accessed via the slave peripheral bus. Details on both memory spaces are provided in [Section 30.7.2.1, Memory map](#).

There are no program-visible registers that physically reside inside the platform flash memory controller. Rather, the platform flash memory controller receives control and configuration information from the flash memory array controller(s) to determine the operating configuration. These are part of the flash memory array's configuration registers mapped into its slave peripheral (IPS) address space but are described here.

### 30.7.2.1 Memory map

First, consider the flash memory space accessed via transactions from the platform flash memory controller's AHB port.

To support the two separate flash memory banks, each as large as 8 MB in size, the platform flash memory controller uses address bit 23 (haddr[23]) to steer the access to the appropriate memory bank. In addition to the actual flash memory regions, the system memory map includes shadow and test sectors. The program-visible control and configuration registers associated with each memory array are included in the slave peripheral address region. The system memory map defines one code flash memory array and one data flash memory array. See [Table 30-59](#).

**Table 30-59. Flash memory-related regions in the system memory map**

| Start address | End address | Size [KB] | Region                                   |
|---------------|-------------|-----------|--|
| 0x0008_0000   | 0x000F_FFFF | 512       | Code flash memory array 1                |
| 0x0010_0000   | 0x0017_FFFF | 512       | Code flash memory array 2                |
| 0x0018_0000   | 0x001F_FFFF | 512       | Reserved                                 |
| 0x0020_0000   | 0x0027_FFFF | 16        | Code flash memory array 0: Shadow sector |
| 0x0028_0000   | 0x002F_FFFF | 1536      | Reserved                                 |
| 0x0040_0000   | 0x0040_3FFF | 16        | Code flash memory array 0: Test sector   |
| 0x0040_4000   | 0x007F_FFFF | 4078      | Reserved                                 |
| 0x0080_0000   | 0x0080_FFFF | 64        | Data flash memory array 0                |
| 0x0081_0000   | 0x00BF_FFFF | 4032      | Reserved                                 |
| 0x00C0_0000   | 0x00C7_FFFF | 16        | Data flash memory array 0: Test sector   |
| 0x00C8_0000   | 0x00FF_FFFF | 3584      | Reserved                                 |
| 0x0100_0000   | 0x1FFF_FFFF | 507904    | Emulation mapping                        |
| 0xC3F8_8000   | 0xC3F8_BFFF | 16        | Code flash memory array 0 configuration  |
| 0xC3F8_C000   | 0xC3F8_FFFF | 16        | Data flash memory array 0 configuration  |

For additional information on the address-based read access timing for emulation of other memory types, see [Section 30.8.11, Wait-state emulation](#).

Next, consider the memory map associated with the control and configuration registers.

Regardless of the number of populated banks or the number of flash memory arrays included in a given bank, the configuration of the platform flash memory controller is wholly specified by the platform flash memory controller registers associated with code flash memory array 0. The code array0 register settings define the operating behavior of **both** flash memory banks; it is recommended that the platform flash memory controller registers for all physically present arrays be set to the code flash memory array0 values.

#### NOTE

To perform program and erase operations, the control registers in the actual referenced flash memory array must be programmed, but the configuration of the platform flash memory controller module is defined by the platform flash controller registers of code array0.

The 32-bit memory map for the platform flash memory controller control registers is shown in [Table 30-60](#). The base address of the controller is 0xC3F8\_8000.

**Table 30-60. Platform flash memory controller 32-bit memory map**

| Address offset | Register  | Location                    |
|----------------|---|-----------------------------|
| 0x1C           | Platform Flash Configuration Register 0 (PFCR0) | <a href="#">on page 873</a> |



Table 30-60. Platform flash memory controller 32-bit memory map (continued)

| Address offset | Register  | Location                    |
|----------------|---|-----------------------------|
| 0x20           | Platform Flash Configuration Register 1 (PFCR1)   | <a href="#">on page 876</a> |
| 0x24           | Platform Flash Access Protection Register (PFAPR) | <a href="#">on page 877</a> |

See the MPC5606BK data sheet for detailed settings for different values of frequency.

### 30.7.2.2 Register description

This section details the individual registers of the platform flash memory controller.

Flash memory configuration registers must be written only with 32-bit write operations to avoid any issues associated with register incoherency caused by bits spanning smaller-size (8- or 16-bit) boundaries.

#### 30.7.2.2.1 Platform Flash Configuration Register 0 (PFCR0)

This register defines the configuration associated with the code flash memory bank0. It includes fields that provide specific information for as many as two separate AHB ports (p0 and the optional p1). For the platform flash memory controller module, the fields associated with AHB port p1 are ignored. The register is described in [Figure 30-42](#) and [Table 30-61](#).

#### NOTE

Do not execute code from flash memory when you are programming PFCR0. If you wish to program PFCR0, execute your application code from RAM.

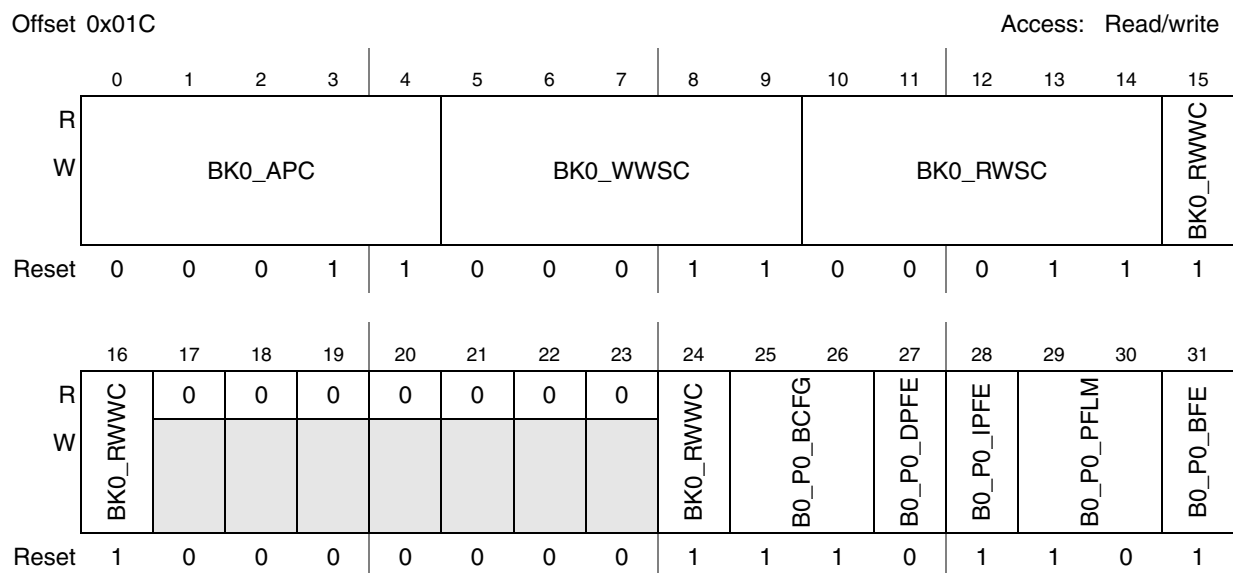


Figure 30-42. PFlash Configuration Register 0 (PFCR0)

Table 30-61. PFCR0 field descriptions

| Field    | Description   |
|----------|---|
| BK0_APC  | The setting for APC must be the same as RWSC.   |
| BK0_WWSC | <p>Bank0 Write Wait-State Control</p> <p>This field is used to control the number of wait-states to be added to the flash memory array access time for writes. This field must be set to a value appropriate to the operating frequency of the PFlash. The required settings are documented in the device data sheet. Higher operating frequencies require non-zero settings for this field for proper flash memory operation. This field is set to an appropriate value by hardware reset.</p> <p>00000 No additional wait-states are added<br/> 00001 One additional wait-state is added<br/> 00010 Two additional wait-states are added<br/> ...<br/> 11111 31 additional wait-states are added</p> <p><b>Note:</b> The Platform Flash Memory Controller does not support Write Wait-State Control since this capability is not supported by the flash memory array.</p>   |
| BK0_RWSC | <p>Bank0 Read Wait-State Control</p> <p>This field is used to control the number of wait-states to be added to the flash memory array access time for reads. This field must be set to a value corresponding to the operating frequency of the PFlash and the actual read access time of the PFlash. The required settings are documented in the device datasheet.</p> <p>00000 No additional wait-states are added<br/> 00001 One additional wait-state is added<br/> 00010 Two additional wait-states are added<br/> ...<br/> 11111 31 additional wait-states are added</p> <p><b>Note:</b> The setting for RWSC must be the same as APC.</p>   |
| BK0_RWWC | <p>Bank0 Read-While-Write Control</p> <p>This 3-bit field defines the controller response to flash memory reads while the array is busy with a program (write) or erase operation.</p> <p>0— This state should be avoided. Setting to this state can cause unpredictable operation.<br/> 111 Generate a bus stall for a read while write/erase, disable the stall notification interrupt, disable the abort + abort notification interrupt<br/> 110 Generate a bus stall for a read while write/erase, enable the stall notification interrupt, disable the abort + abort notification interrupt<br/> 101 Generate a bus stall for a read while write/erase, enable the operation abort, disable the abort notification interrupt<br/> 100 Generate a bus stall for a read while write/erase, enable the operation abort and the abort notification interrupt</p> <p>This field is set to 0b111 by hardware reset enabling the stall-while-write/erase and disabling the abort and notification interrupts.</p> |

Table 30-61. PFCR0 field descriptions (continued)

| Field      | Description   |
|------------|---|
| B0_P0_BCFG | <p>Bank0, Port 0 Page Buffer Configuration</p> <p>This field controls the configuration of the four page buffers in the PFlash controller. The buffers can be organized as a pool of available resources, or with a fixed partition between instruction and data buffers.</p> <p>If enabled, when a buffer miss occurs, it is allocated to the least recently used buffer within the group and the just-fetched entry then marked as most recently used. If the flash memory access is for the next-sequential line, the buffer is not marked as most recently used until the given address produces a buffer hit.</p> <p>00 All four buffers are available for any flash memory access, that is, there is no partitioning of the buffers based on the access type.<br/> 01 Reserved<br/> 10 The buffers are partitioned into two groups with buffers 0 and 1 allocated for instruction fetches and buffers 2 and 3 for data accesses.<br/> 11 The buffers are partitioned into two groups with buffers 0,1,2 allocated for instruction fetches and buffer 3 for data accesses.</p> <p>This field is set to 2b11 by hardware reset.</p> |
| B0_P0_DPFE | <p>Bank0, Port 0 Data Prefetch Enable</p> <p>This field enables or disables prefetching initiated by a data read access. This field is cleared by hardware reset. Prefetching can be enabled/disabled on a per Master basis at PFAPR[MxPFD].</p> <p>0 No prefetching is triggered by a data read access<br/> 1 If page buffers are enabled (B0_P0_BFE = 1), prefetching is triggered by any data read access</p>  |
| B0_P0_IPFE | <p>Bank0, Port 0 Instruction Prefetch Enable</p> <p>This field enables or disables prefetching initiated by an instruction fetch read access. This field is set by hardware reset. Prefetching can be enabled/disabled on a per Master basis at PFAPR[MxPFD].</p> <p>0 No prefetching is triggered by an instruction fetch read access<br/> 1 If page buffers are enabled (B0_P0_BFE = 1), prefetching is triggered by any instruction fetch read access</p>  |
| B0_P0_PFLM | <p>Bank0, Port 0 Prefetch Limit</p> <p>This field controls the prefetch algorithm used by the PFlash controller. This field defines the prefetch behavior. In all situations when enabled, only a single prefetch is initiated on each buffer miss or hit. This field is set to 2b10 by hardware reset.</p> <p>00 No prefetching is performed.<br/> 01 The referenced line is prefetched on a buffer miss, that is, <i>prefetch on miss</i>.<br/> 1– The referenced line is prefetched on a buffer miss, or the next sequential page is prefetched on a buffer hit (if not already present), that is, <i>prefetch on miss or hit</i>.</p>   |
| B0_P0_BFE  | <p>Bank0, Port 0 Buffer Enable</p> <p>This bit enables or disables page buffer read hits. It is also used to invalidate the buffers. This bit is set by hardware reset.</p> <p>0 The page buffers are disabled from satisfying read requests, and all buffer valid bits are cleared.<br/> 1 The page buffers are enabled to satisfy read requests on hits. Buffer valid bits may be set when the buffers are successfully filled.</p>   |

### 30.7.2.2.2 Platform Flash Configuration Register 1 (PFCR1)

This register defines the configuration associated with flash memory bank1. This corresponds to the data flash memory. It includes fields that provide specific information for as many as two separate AHB ports (p0 and the optional p1). For the platform flash memory controller module, the fields associated with AHB port p1 are ignored. The register is described below in [Figure 30-43](#) and [Table 30-62](#).

**NOTE**

Do not execute code from flash memory when you are programming PFCR1. If you wish to program PFCR1, execute your application code from RAM.

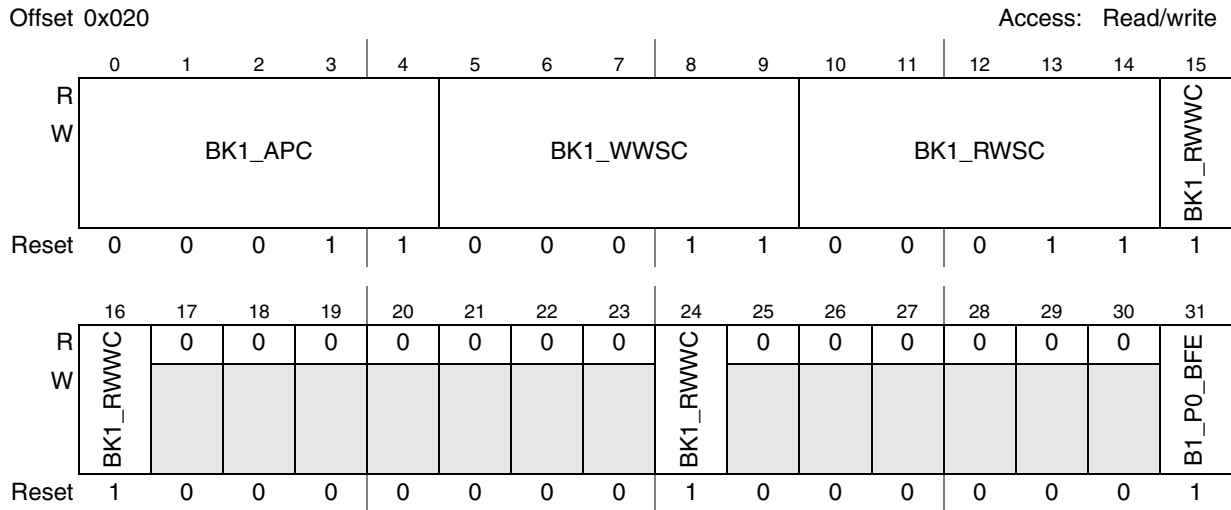


Figure 30-43. PFlash Configuration Register 1 (PFCR1)

Table 30-62. PFCR1 field descriptions

| Field    | Description  |
|----------|--|
| BK1_APC  | The setting for APC must be the same as RWSC.  |
| BK1_WWSC | <p>Bank1 Write Wait-State Control</p> <p>This field is used to control the number of wait-states to be added to the flash memory array access time for writes. This field must be set to a value appropriate to the operating frequency of the PFlash. The required settings are documented in the device data sheet. Higher operating frequencies require non-zero settings for this field for proper flash memory operation. This field is set to an appropriate value by hardware reset.</p> <p>00000 No additional wait-states are added<br/>                     00001 One additional wait-state is added<br/>                     00010 Two additional wait-states are added<br/>                     ...<br/>                     11111 31 additional wait-states are added</p> <p>This field is ignored in single bank flash memory configurations.</p> <p><b>Note:</b> The Platform Flash Memory Controller does not support Write Wait-State Control since this capability is not supported by the flash memory array.</p> |

Table 30-62. PFCR1 field descriptions (continued)

| Field     | Description   |
|-----------|---|
| BK1_RWSC  | <p>Bank1 Read Wait-State Control</p> <p>This field is used to control the number of wait-states to be added to the flash memory array access time for reads. This field must be set to a value corresponding to the operating frequency of the PFlash and the actual read access time of the PFlash. The required settings are documented in the device data sheet.</p> <p>00000 No additional wait-states are added<br/> 00001 One additional wait-state is added<br/> 00010 Two additional wait-states are added<br/> ...<br/> 11111 31 additional wait-states are added</p> <p>This field is ignored in single bank flash memory configurations.<br/> <b>Note:</b> The setting for RWSC must be the same as APC.</p>   |
| BK1_RWWC  | <p>Bank1 Read-While-Write Control</p> <p>This 3-bit field defines the controller response to flash memory reads while the array is busy with a program (write) or erase operation.</p> <p>0— This state is not supported<br/> 111 Generate a bus stall for a read while write/erase, disable the stall notification interrupt, disable the abort + abort notification interrupt<br/> 110 Generate a bus stall for a read while write/erase, enable the stall notification interrupt, disable the abort + abort notification interrupt<br/> 101 Generate a bus stall for a read while write/erase, enable the operation abort, disable the abort notification interrupt<br/> 100 Generate a bus stall for a read while write/erase, enable the operation abort and the abort notification interrupt</p> <p>This field is set to 0b111 by hardware reset enabling the stall-while-write/erase and disabling the abort and notification interrupts.</p> <p>This field is ignored in single bank flash memory configurations.</p> |
| B1_PO_PFE | <p>Bank1, Port 0 Buffer Enable</p> <p>This bit enables or disables read hits from the 128-bit holding register. It is also used to invalidate the contents of the holding register. This bit is set by hardware reset, enabling the use of the holding register.</p> <p>0 The holding register is disabled from satisfying read requests.<br/> 1 The holding register is enabled to satisfy read requests on hits.</p>  |

### 30.7.2.2.3 Platform Flash Access Protection Register (PFAPR)

The PFlash Access Protection Register (PFAPR) is used to control read and write accesses to the flash memory based on system master number. Prefetching capabilities are defined on a per master basis. This register also defines the arbitration mode for controllers supporting two AHB ports. The register is described below in [Figure 30-44](#) and [Table 30-63](#).

The contents of the register are loaded from location 0x203E00 of the shadow region in the code flash memory (bank0) array at reset. To temporarily change the values of any of the fields in the PFAPR, a write to the IPS-mapped register is performed. To change the values loaded into the PFAPR *at reset*, the word

location at address 0x203E00 of the shadow region in the flash memory array must be programmed using the normal sequence of operations. The reset value shown in Table 30-44 reflects an erased or unprogrammed value from the shadow region.

Offset 0x024 Access: Read/write

|   |   |   |   |   |   |   |   |   |   |   |    |    |    |       |    |       |
|---|---|---|---|---|---|---|---|---|---|---|----|----|----|-------|----|-------|
|   | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13    | 14 | 15    |
| R | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | M2PFD | 0  | M0PFD |
| W |   |   |   |   |   |   |   |   |   |   |    |    |    |       |    |       |

Reset Defined by NVPFAPR at CFlash Test Sector Address 0x203E00

|   |    |    |    |    |    |    |    |    |    |    |      |    |    |    |      |    |
|---|----|----|----|----|----|----|----|----|----|----|------|----|----|----|------|----|
|   | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26   | 27 | 28 | 29 | 30   | 31 |
| R | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | M2AP |    | 0  | 0  | M0AP |    |
| W |    |    |    |    |    |    |    |    |    |    |      |    |    |    |      |    |

Reset Defined by NVPFAPR at CFlash Test Sector Address 0x203E00

**Figure 30-44. PFlash Access Protection Register (PFAPR)**

**Table 30-63. PFAPR field descriptions**

| Field | Description  |
|-------|--|
| M2PFD | <p>eDMA Master 2 Prefetch Disable</p> <p>This field controls whether prefetching may be triggered based on the master number of the requesting AHB master. This field is further qualified by the PFCR0[B0_Px_DPFE, B0_Px_IPFE, Bx_Py_BFE] bits. For master numbering, see Table 19-1.</p> <p>0 Prefetching may be triggered by this master<br/>                     1 No prefetching may be triggered by this master</p>        |
| M0PFD | <p>e200z0 core Master 0 Prefetch Disable</p> <p>This field controls whether prefetching may be triggered based on the master number of the requesting AHB master. This field is further qualified by the PFCR0[B0_Px_DPFE, B0_Px_IPFE, Bx_Py_BFE] bits. For master numbering, see Table 19-1.</p> <p>0 Prefetching may be triggered by this master<br/>                     1 No prefetching may be triggered by this master</p> |

Table 30-63. PFAPR field descriptions (continued)

| Field | Description   |
|-------|---|
| M2AP  | eDMA Master 2 Access Protection<br>These fields control whether read and write accesses to the flash memory are allowed based on the master number of the initiating module. For master numbering, see Table 19-1.<br><br>00 No accesses may be performed by this master<br>01 Only read accesses may be performed by this master<br>10 Only write accesses may be performed by this master<br>11 Both read and write accesses may be performed by this master        |
| M0AP  | e200z0 core Master 0 Access Protection<br>These fields control whether read and write accesses to the flash memory are allowed based on the master number of the initiating module. For master numbering, see Table 19-1.<br><br>00 No accesses may be performed by this master<br>01 Only read accesses may be performed by this master<br>10 Only write accesses may be performed by this master<br>11 Both read and write accesses may be performed by this master |

### 30.7.2.2.3.1 Nonvolatile Platform Flash Access Protection Register (NVPFAPR)

The NVPFAPR register has a related Nonvolatile PFAPR located in the Shadow Sector that contains the default reset value for PFAPR. During the reset phase of the flash memory module, the NVPFAPR register content is read and loaded into the PFAPR.

The NVPFAPR register is a 64-bit register, of which the 32 most significant bits 63:32 are “don’t care” and are used to manage ECC codes.

Offset: 0x203E00

Access: Read/write

|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13    | 14 | 15    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|-------|----|-------|
| R     | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1  | 1  | 1  | M2PFD | 1  | M0PFD |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |       |    |       |
| Reset | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1  | 1  | 1  | 1     | 1  | 1     |

|       | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26   | 27 | 28 | 29 | 30   | 31 |
|-------|----|----|----|----|----|----|----|----|----|----|------|----|----|----|------|----|
| R     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | M2AP |    | 0  | 0  | M0AP |    |
| W     |    |    |    |    |    |    |    |    |    |    |      |    |    |    |      |    |
| Reset | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1    | 1  | 1  | 1  | 1    | 1  |

Figure 30-45. Nonvolatile Platform Flash Access Protection Register (NVPFAPR)

**Table 30-64. NVPFAPR field descriptions**

| Field | Description                       |
|-------|-----------------------------------|
| M2PFD | See <a href="#">Table 30-63</a> . |
| M0PFD | See <a href="#">Table 30-63</a> . |
| M2AP  | See <a href="#">Table 30-63</a> . |
| M0AP  | See <a href="#">Table 30-63</a> . |

## 30.8 Functional description

The platform flash memory controller interfaces between the AHB system bus and the flash memory arrays.

The platform flash memory controller generates read and write enables, the flash memory array address, write size, and write data as inputs to the flash memory array. The platform flash memory controller captures read data from the flash memory array interface and drives it onto the AHB. As many as four pages of data (128-bit width) from bank0 are buffered by the platform flash memory controller. Lines may be prefetched in advance of being requested by the AHB interface, allowing single-cycle (zero AHB wait-states) read data responses on buffer hits.

Several prefetch control algorithms are available for controlling page read buffer fills. Prefetch triggering may be restricted to instruction accesses only, data accesses only, or may be unrestricted. Prefetch triggering may also be controlled on a per-master basis.

Buffers may also be selectively enabled or disabled for allocation by instruction and data prefetch. See [Section 30.7.2.2.1, Platform Flash Configuration Register 0 \(PFCR0\)](#), and [Section 30.7.2.2.2, Platform Flash Configuration Register 1 \(PFCR1\)](#).

Access protections may be applied on a per-master basis for both reads and writes to support security and privilege mechanisms. See [Section 30.7.2.2.3, Platform Flash Access Protection Register \(PFAPR\)](#).

Throughout this discussion, `bkn_` is used as a prefix to refer to two signals, each for each bank: `bk0_` and `bk1_`. Also, the nomenclature `Bx_Py_RegName` is used to reference a program-visible register field associated with bank “x” and port “y”.

### 30.8.1 Access protections

The platform flash memory controller provides programmable configurable access protections for both read and write cycles from masters via the PFlash Access Protection Register (PFAPR). It allows restriction of read and write requests on a per-master basis. This functionality is described in [Section 30.7.2.2.3, Platform Flash Access Protection Register \(PFAPR\)](#). Detection of a protection violation results in an error response from the platform flash memory controller on the AHB transfer.

### 30.8.2 Read cycles – Buffer miss

Read cycles from the flash memory array are initiated by the platform flash memory controller. The platform flash memory controller then waits for the programmed number of read wait-states before



sampling the read data from the flash memory array. This data is normally stored in the least-recently updated page read buffer for bank0 in parallel with the requested data being forwarded to the AHB. For bank1, the data is captured in the page-wide temporary holding register as the requested data is forwarded to the AHB bus.

If the flash memory access was the direct result of an AHB transaction, the page buffer is marked as most recently used as it is being loaded. If the flash memory access was the result of a speculative prefetch to the next sequential line, it is first loaded into the least recently used buffer. The status of this buffer is not changed to most recently used until a subsequent buffer hit occurs.

### 30.8.3 Read cycles – Buffer hit

Single cycle read responses to the AHB are possible with the platform flash memory controller when the requested read access was previously loaded into one of the bank0 page buffers. In these buffer hit cases, read data is returned to the AHB data phase with a zero wait-state response.

Likewise, the bank1 logic includes a single 128-bit temporary holding register and sequential accesses that “hit” in this register are also serviced with a zero wait-state response.

### 30.8.4 Write cycles

Write cycles are initiated by the platform flash memory controller. The platform flash memory controller then waits for the appropriate number of write wait-states before terminating the write operation.

### 30.8.5 Error termination

The first case that can cause an error response to the AHB is when an access is attempted by an AHB master whose corresponding Read Access Control or Write Access Control settings do not allow the access, thus causing a protection violation. In this case, the platform flash memory controller does not initiate a flash memory array access.

The second case that can cause an error response to the AHB is when an access is performed to the flash memory array and is terminated with a flash memory error response. See [Section 30.8.7, Flash error response operation](#). This may occur for either a read or a write operation.

A third case involves an attempted read access while the flash memory array is busy doing a write (program) or erase operation if the appropriate read-while-write control field is programmed for this response. The 3-bit read-while-write control allows for immediate termination of an attempted read, or various stall-while-write/erase operations are occurring.

### 30.8.6 Access pipelining

The platform flash memory controller does not support access pipelining since this capability is not supported by the flash memory array. As a result, the APC (Address Pipelining Control) field must be the same value as the RWSC (Read Wait-State Control).

### 30.8.7 Flash error response operation

The flash memory array may signal an error response to terminate a requested access with an error. This may occur due to an uncorrectable ECC error, or because of improper sequencing during program/erase operations. When an error response is received, the platform flash memory controller does not update or validate a bank0 page read buffer nor the bank1 temporary holding register. An error response may be signaled on read or write operations. For additional information on the system registers that capture the faulting address, attributes, data, and ECC information, see [Chapter 34, Error Correction Status Module \(ECSM\)](#).

### 30.8.8 Bank0 page read buffers and prefetch operation

The logic associated with bank0 of the platform flash memory controller contains four 128-bit page read buffers that are used to hold instructions and data read from the flash memory array. Each buffer operates independently, and is filled using a single array access. The buffers are used for both prefetch and normal demand fetches.

For the general case, a page buffer is written at the completion of an error-free flash memory access and the valid bit asserted. Subsequent flash memory accesses that hit the buffer, that is, the current access address matches the address stored in the buffer, can be serviced in 0 AHB wait-states as the stored read data is routed from the given page buffer back to the requesting bus master.

As noted in [Section 30.8.7, Flash error response operation](#), a page buffer is *not* marked as valid if the flash memory array access terminated with any type of transfer error. However, the result is that flash memory array accesses that are tagged with a single-bit correctable ECC event are loaded into the page buffer and validated. For additional comments on this topic, see [Section 30.8.8.4, Buffer invalidation](#).

Prefetch triggering is controllable on a per-master and access-type basis. Bus masters may be enabled or disabled from triggering prefetches, and triggering may be further restricted based on whether a read access is for instruction or data. A read access to the platform flash memory controller may trigger a prefetch to the next sequential page of array data on the first idle cycle following the request. The access address is incremented to the next-higher 16-byte boundary, and a flash memory array prefetch is initiated if the data is not already resident in a page buffer. Prefetched data is always loaded into the least recently used buffer.

Buffers may be in one of six states, listed here in order of priority:

1. Invalid — The buffer contains no valid data.
2. Used — The buffer contains valid data that has been provided to satisfy an AHB burst type read.
3. Valid — The buffer contains valid data that has been provided to satisfy an AHB single type read.
4. Prefetched — The buffer contains valid data that has been prefetched to satisfy a potential future AHB access.
5. Busy AHB — The buffer is currently being used to satisfy an AHB burst read.
6. Busy Fill — The buffer has been allocated to receive data from the flash memory array, and the array access is still in progress.

Selection of a buffer to be loaded on a miss is based on the following replacement algorithm:

1. First, the buffers are examined to determine if there are any invalid buffers. If there are multiple invalid buffers, the one to be used is selected using a simple numeric priority, where buffer 0 is selected first, then buffer 1, etc.
2. If there are no invalid buffers, the least recently used buffer is selected for replacement.

Once the candidate page buffer has been selected, the flash memory array is accessed and read data loaded into the buffer. If the buffer load was in response to a miss, the just-loaded buffer is immediately marked as most recently used. If the buffer load was in response to a speculative fetch to the next-sequential line address after a buffer hit, the recently used status is not changed. Rather, it is marked as most recently used only after a subsequent buffer hit.

This policy maximizes performance based on reference patterns of flash memory accesses and allows for prefetched data to remain valid when non-prefetch enabled bus masters are granted flash memory access.

Several algorithms are available for prefetch control that trade off performance versus power. They are defined by the Bx\_Py\_PFLM (prefetch limit) register field. More aggressive prefetching increases power slightly due to the number of wasted (discarded) prefetches, but may increase performance by lowering average read latency.

In order for prefetching to occur, a number of control bits must be enabled. Specifically, the global buffer enable (PFCRn[Bx\_Py\_BFE]) must be set, the prefetch limit (PFCRn[Bx\_Py\_PFLM]) must be non-zero, either instruction prefetching (PFCRn[Bx\_Py\_IPFE]) or data prefetching (PFCRn[Bx\_Py\_DPFE]) enabled, and Master Access must be enabled (PFAPR[MxPFD]). See [Section 30.7.2.2, Register description](#), for a description of these control fields.

### 30.8.8.1 Instruction/Data prefetch triggering

Prefetch triggering may be enabled for instruction reads via the Bx\_Py\_IPFE control field, while prefetching for data reads is enabled via the Bx\_Py\_DPFE control field. Additionally, the Bx\_Py\_PFLIM field must be set to enable prefetching. Prefetches are never triggered by write cycles.

### 30.8.8.2 Per-master prefetch triggering

Prefetch triggering may be also controlled for individual bus masters. See [Section 30.7.2.2.3, Platform Flash Access Protection Register \(PFAPR\)](#), for details on these controls.

### 30.8.8.3 Buffer allocation

Allocation of the line read buffers is controlled via page buffer configuration (Bx\_Py\_BCFG) field. This field defines the operating organization of the four page buffers. The buffers can be organized as a pool of available resources (with all four buffers in the pool) or with a fixed partition between buffers allocated to instruction or data accesses. For the fixed partition, two configurations are supported. In one configuration, buffers 0 and 1 are allocated for instruction fetches and buffers 2 and 3 for data accesses. In the second configuration, buffers 0, 1, and 2 are allocated for instruction fetches and buffer 3 reserved for data accesses.

### 30.8.8.4 Buffer invalidation

The page read buffers may be invalidated under hardware or software control.

At the beginning of all program/erase operations, the flash memory array will invalidate the page read buffers. Buffer invalidation occurs at the next AHB non-sequential access boundary, but does not affect a burst from a page read buffer that is in progress.

Software may invalidate the buffers by clearing the `Bx_Py_BFE` bit, which also disables the buffers. Software may then re-assert the `Bx_Py_BFE` bit to its previous state, and the buffers will have been invalidated.

One special case needing software invalidation relates to page buffer hits on flash memory data that was tagged with a single-bit ECC event on the original array access. Recall that the page buffer structure includes an status bit signaling the array access detected and corrected a single-bit ECC error. On all subsequent buffer hits to this type of page data, a single-bit ECC event is signaled by the platform flash memory controller. Depending on the specific hardware configuration, this reporting of a single-bit ECC event may generate an ECC alert interrupt. In order to prevent repeated ECC alert interrupts, the page buffers need to be invalidated by software after the first notification of the single-bit ECC event.

Finally, the buffers are invalidated by hardware on any non-sequential access with a non-zero value on `haddr[28:24]` to support wait-state emulation.

### 30.8.9 Bank1 Temporary Holding Register

Recall the bank1 logic within the platform flash memory controller includes a single 128-bit data register, used for capturing read data. Since this bank does not support prefetching, the read data for the referenced address is bypassed directly back to the AHB data bus. The page is also loaded into the temporary data register and subsequent accesses to this page can hit from this register, if it is enabled (`B1_P0_BFE`).

For the general case, a temporary holding register is written at the completion of an error-free flash memory access and the valid bit asserted. Subsequent flash memory accesses that hit the buffer, that is, the current access address matches the address stored in the temporary holding register, can be serviced in 0 AHB wait-states as the stored read data is routed from the temporary register back to the requesting bus master.

The contents of the holding register are invalidated by the flash memory array at the beginning of all program/erase operations and on any non-sequential access with a non-zero value on `haddr[28:24]` (to support wait-state emulation) in the same manner as the bank0 page buffers. Additionally, the `B1_P0_BFE` register bit can be cleared by software to invalidate the contents of the holding register.

As noted in [Section 30.8.7, Flash error response operation](#), the temporary holding register is *not* marked as valid if the flash memory array access terminated with any type of transfer error. However, the result is that flash memory array accesses that are tagged with a single-bit correctable ECC event are loaded into the temporary holding register and validated. Accordingly, one special case needing software invalidation relates to holding register hits on flash memory data that was tagged with a single-bit ECC event. Depending on the specific hardware configuration, the reporting of a single-bit ECC event may generate an ECC alert interrupt. In order to prevent repeated ECC alert interrupts, the page buffers need to be invalidated by software after the first notification of the single-bit ECC event.

The bank1 temporary holding register effectively operates like a single page buffer.

### 30.8.10 Read-while-write functionality

The platform flash memory controller supports various programmable responses for read accesses while the flash memory is busy performing a write (program) or erase operation. For all situations, the platform flash memory controller uses the state of the flash memory array's MCR[*DONE*] output to determine if it is busy performing some type of high voltage operation, namely, if MCR[*DONE*] = 0, the array is busy.

Specifically, two 3-bit read-while-write (BK<sub>*n*</sub>\_RWWC) control register fields define the platform flash memory controller's response to these types of access sequences. The following unique responses are defined by the BK<sub>*n*</sub>\_RWWC setting: one that immediately reports an error on an attempted read and four settings that support various stall-while-write capabilities. Consider the details of these settings.

- BK<sub>*n*</sub>\_RWWC = 0b111  
This defines the basic stall-while-write capability and represents the default reset setting. For this mode, the platform flash memory controller module simply stalls any read reference until the flash memory has completed its program/erase operation. If a read access arrives while the array is busy or if MCR[*DONE*] goes low while a read is still in progress, the AHB data phase is stalled and the read access address is saved. Once the array has completed its program/erase operation, the platform flash memory controller uses the saved address and attribute information to create a pseudo address phase cycle to retry the read reference and sends the registered information to the array. Once the retried address phase is complete, the read is processed normally and once the data is valid, it is forwarded to the AHB bus to terminate the system bus transfer.
- BK<sub>*n*</sub>\_RWWC = 0b110  
This setting is similar to the basic stall-while-write capability provided when BK<sub>*n*</sub>\_RWWC = 0b111 with the added ability to generate a notification interrupt if a read arrives while the array is busy with a program/erase operation. There are two notification interrupts, one for each bank (see [Chapter 18, Interrupt Controller \(INTC\)](#)).
- BK<sub>*n*</sub>\_RWWC = 0b101  
Again, this setting provides the basic stall-while-write capability with the added ability to abort any program/erase operation if a read access is initiated. For this setting, the read request is captured and retried as described for the basic stall-while-write, plus the program/erase operation is aborted by the platform flash memory controller. For this setting, no notification interrupts are generated.
- BK<sub>*n*</sub>\_RWWC = 0b100  
This setting provides the basic stall-while-write capability with the ability to abort any program/erase operation if a read access is initiated plus the generation of an abort notification interrupt. For this setting, the read request is captured and retried as described for the basic stall-while-write, the program/erase operation is aborted by the platform flash memory controller and an abort notification interrupt generated. There are two abort notification interrupts, one for each bank.

As detailed above, a total of four interrupt requests are associated with the stall-while-write functionality. These interrupt requests are captured as part of ECSM's interrupt register and logically summed together to form a single request to the interrupt controller.

**Table 30-65. Platform flash memory controller stall-while-write interrupts**

| MIR[n]      | Interrupt description                                      |
|-------------|--|
| ECSM.MIR[0] | Platform flash memory bank0 abort notification, MIR[FB0AI] |
| ECSM.MIR[1] | Platform flash memory bank0 stall notification, MIR[FB0SI] |
| ECSM.MIR[2] | Platform flash memory bank1 abort notification, MIR[FB1AI] |
| ECSM.MIR[3] | Platform flash memory bank1 stall notification, MIR[FB1SI] |

### 30.8.11 Wait-state emulation

Emulation of other memory array timings are supported by the platform flash memory controller on read cycles to the flash memory. This functionality may be useful to maintain the access timing for blocks of memory that were used to overlay flash memory blocks for the purpose of system calibration or tuning during code development.

The platform flash memory controller inserts additional wait-states according to the values of `haddr[28:24]`. When these inputs are non-zero, additional cycles are added to AHB read cycles. Write cycles are not affected. In addition, no page read buffer prefetches are initiated, and buffer hits are ignored.

[Table 30-66](#) and [Table 30-67](#) show the relationship of `haddr[28:24]` to the number of additional primary wait-states. These wait-states are applied to the initial access of a burst fetch or to single-beat read accesses on the AHB system bus.

Note that the wait-state specification consists of two components: `haddr[28:26]` and `haddr[25:24]`, and effectively extends the flash memory read by  $(8 \times \text{haddr}[25:24] + \text{haddr}[28:26])$  cycles.

**Table 30-66. Additional wait-state encoding**

| Memory address<br><code>haddr[28:26]</code> | Additional wait-states |
|---|------------------------|
| 000   | 0                      |
| 001   | 1                      |
| 010   | 2                      |
| 011   | 3                      |
| 100   | 4                      |
| 101   | 5                      |
| 110   | 6                      |
| 111   | 7                      |

[Table 30-67](#) shows the relationship of `haddr[25:24]` to the number of additional wait-states. These are applied in addition to those specified by `haddr[28:26]`, and thus extend the total wait-state specification capability.

**Table 30-67. Extended additional wait-state encoding**

| <b>Memory address<br/>haddr[25:24]</b> | <b>Additional wait-states<br/>(added to those specified by haddr[28:26])</b> |
|--|--|
| 00                                     | 0  |
| 01                                     | 8  |
| 10                                     | 16   |
| 11                                     | 24   |

This page is intentionally left blank.



# Chapter 31

## Static RAM (SRAM)

### 31.1 Introduction

This device has as much as 80 KB of general-purpose static RAM (SRAM).

The SRAM provides the following features:

- SRAM can be read/written from any bus master
- Byte, halfword, and word addressable
- ECC (error correction code) protected with single-bit correction and double-bit detection

**Table 31-1. SRAM behavior in chip modes**

| SRAM —<br>96 KB | Address                      | MC_PCU control<br>register | Run modes<br>(DRUN, RUN0..3) | Low-power modes |           |            |
|-----------------|------------------------------|----------------------------|------------------------------|-----------------|-----------|------------|
|                 |                              |                            |                              | HALT            | STOP      | STANDBY    |
| 8 KB            | 0x4000_0000 –<br>0x4000_1FFF | PCONF0[<mode>]             | Always on                    | Always on       | Always on | Always on  |
| 24 KB           | 0x4000_2000 –<br>0x4000_BFFF | PCONF2[<mode>]             | On/off                       | On/off          | On/off    | On/off     |
| 64 KB           | 0x4000_C000 –<br>0x4001_7FFF | PCONF3[<mode>]             | On/off                       | On/off          | On/off    | Always off |

### 31.2 Low power configuration

In order to reduce leakage a portion of the SRAM can be switched off/unpowered during standby mode.

**Table 31-2. Low power configuration**

| Mode                      | Configuration   |
|---------------------------|---|
| RUN, TEST, SAFE, and STOP | The entire SRAM is powered and operational.   |
| STANDBY                   | Either 8 KB or 32 KB of the SRAM remains powered. This option is software-selectable. |

### 31.3 Register memory map

The L2SRAM occupies 80 KB of memory starting at the base address as shown in [Table 31-3](#).

**Table 31-3. SRAM memory map**

| Address            | Register name | Register description | Size  |
|--------------------|---------------|----------------------|-------|
| 0x4000_0000 (Base) | —             | SRA                  | 80 KB |

The internal SRAM has no registers. Registers for the SRAM ECC are located in the ECSM [see [Chapter 34, Error Correction Status Module \(ECSM\)](#), for more information].

## 31.4 SRAM ECC mechanism

The SRAM ECC detects the following conditions and produces the following results:

- Detects and corrects all 1-bit errors
- Detects and flags all 2-bit errors as non-correctable errors
- Detects 39-bit reads (32-bit data bus plus the 7-bit ECC) that return all zeros or all ones, asserts an error indicator on the bus cycle, and sets the error flag

SRAM does not detect all errors greater than 2 bits.

Internal SRAM write operations are performed on the following byte boundaries:

- 1 byte (0:7 bits)
- 2 bytes (0:15 bits)
- 4 bytes or 1 word (0:31 bits)

If the entire 32 data bits are written to SRAM, no read operation is performed and the ECC is calculated across the 32-bit data bus. The 8-bit ECC is appended to the data segment and written to SRAM.

If the write operation is less than the entire 32-bit data width (1 or 2-byte segment), the following occurs:

1. The ECC mechanism checks the entire 32-bit data bus for errors, detecting and either correcting or flagging errors.
2. The write data bytes (1 or 2-byte segment) are merged with the corrected 32 bits on the data bus.
3. The ECC is then calculated on the resulting 32 bits formed in the previous step.
4. The 7-bit ECC result is appended to the 32 bits from the data bus, and the 39-bit value is then written to SRAM.

### 31.4.1 Access timing

The system bus is a two-stage pipelined bus, which makes the timing of any access dependent on the access during the previous clock cycle. [Table 31-4](#) lists the various combinations of read and write operations to SRAM and the number of wait states used for the each operation. The table columns contain the following information:

- Current operation — Lists the type of SRAM operation currently executing
- Previous operation — Lists the valid types of SRAM operations that can precede the current SRAM operation (valid operation during the preceding clock)
- Wait states — Lists the number of wait states (bus clocks) the operation requires, which depends on the combination of the current and previous operation

Table 31-4. Number of wait states required for SRAM operations

| Operation type | Current operation               | Previous operation          | Number of wait states required       |
|----------------|---------------------------------|-----------------------------|--------------------------------------|
| Read           | Read                            | Idle                        | 1                                    |
|                |                                 | Pipelined read              |                                      |
|                |                                 | 8, 16 or 32-bit write       | 0<br>(read from the same address)    |
|                |                                 |                             | 1<br>(read from a different address) |
|                | Pipelined read                  | Read                        | 0                                    |
| Write          | 8 or 16-bit write               | Idle                        | 1                                    |
|                |                                 | Read                        |                                      |
|                |                                 | Pipelined 8 or 16-bit write | 2                                    |
|                |                                 | 32-bit write                |                                      |
|                |                                 | 8 or 16-bit write           | 0<br>(write to the same address)     |
|                | Pipelined 8, 16 or 32-bit write | 8, 16 or 32-bit write       | 0                                    |
|                | 32-bit write                    | Idle                        | 0                                    |
|                |                                 | 32-bit write                |                                      |
| Read           |                                 |                             |                                      |

### 31.4.2 Reset effects on SRAM accesses

Asynchronous reset will possibly corrupt SRAM if it asserts during a read or write operation to SRAM. The completion of that access depends on the cycle at which the reset occurs. Data read from or written to SRAM before the reset event occurred is retained, and no other address locations are accessed or changed. In case of no access ongoing when reset occurs, the SRAM corruption does not happen.

Instead, synchronous reset (SW reset) should be used in controlled function (without SRAM accesses) in case an initialization procedure without SRAM initialization is needed.

## 31.5 Functional description

ECC checks are performed during the read portion of an SRAM ECC read/write (R/W) operation, and ECC calculations are performed during the write portion of a R/W operation. Because the ECC bits can contain random data after the device is powered on, the SRAM must be initialized by executing 32-bit write operations prior to any read accesses. This is also true for implicit read accesses caused by any write accesses of less than 32 bits as discussed in [Section 31.4, SRAM ECC mechanism](#).


## 31.6 Initialization and application information

To use the SRAM, the ECC must check all bits that require initialization after power on. All writes must specify an even number of registers performed on 32-bit word-aligned boundaries. If the write is not the

entire 32 bits (8 or 16 bits), a read / modify / write operation is generated that checks the ECC value upon the read. See [Section 31.4, SRAM ECC mechanism](#).

---

## —— Integrity ——



This page is intentionally left blank.

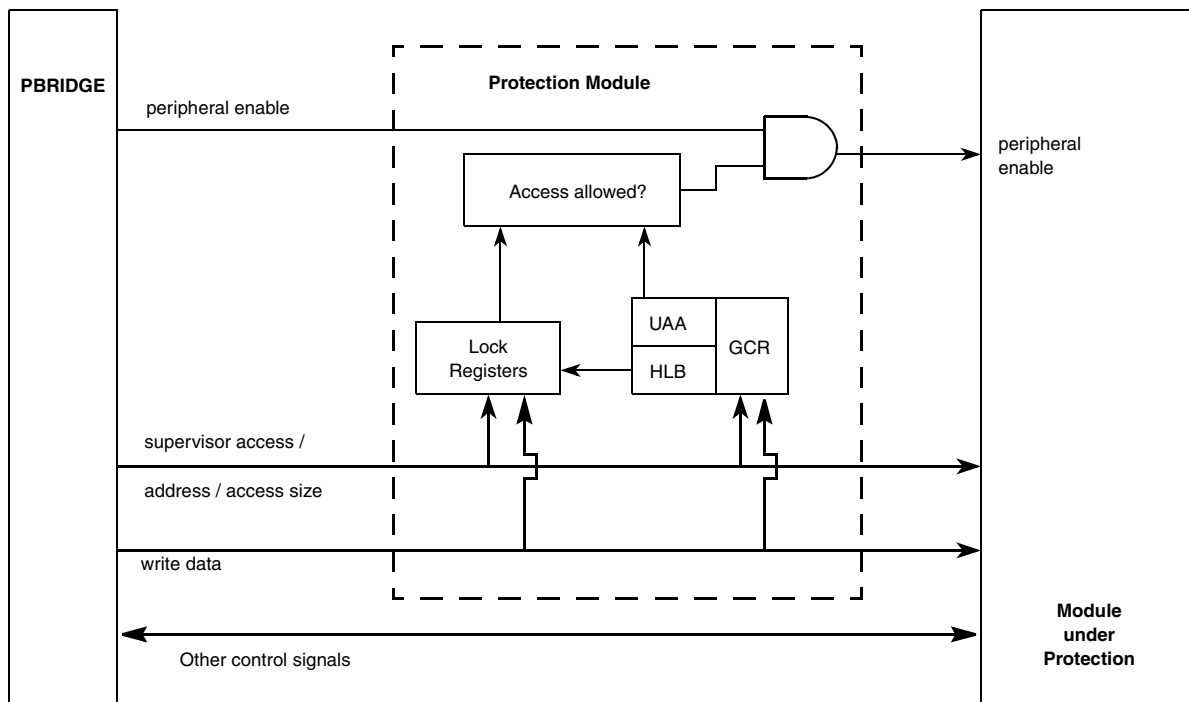
# Chapter 32

## Register Protection

### 32.1 Introduction

The Register Protection module offers a mechanism to protect defined memory-mapped address locations in a module under protection from being written. The address locations that can be protected are module-specific.

The protection module is located between the module under protection and the peripheral bridge. This is shown in [Figure 32-1](#).



**Figure 32-1. Register Protection block diagram**

Please see the “Registers Under Protection” appendix for the list of protected registers.

### 32.2 Features

The Register Protection includes these distinctive features:

- Restrict write accesses for the module under protection to supervisor mode only
- Lock registers for first 6 KB of memory-mapped address space
- Address mirror automatically sets corresponding lock bit
- Once configured lock bits can be protected from changes

### 32.3 Modes of operation

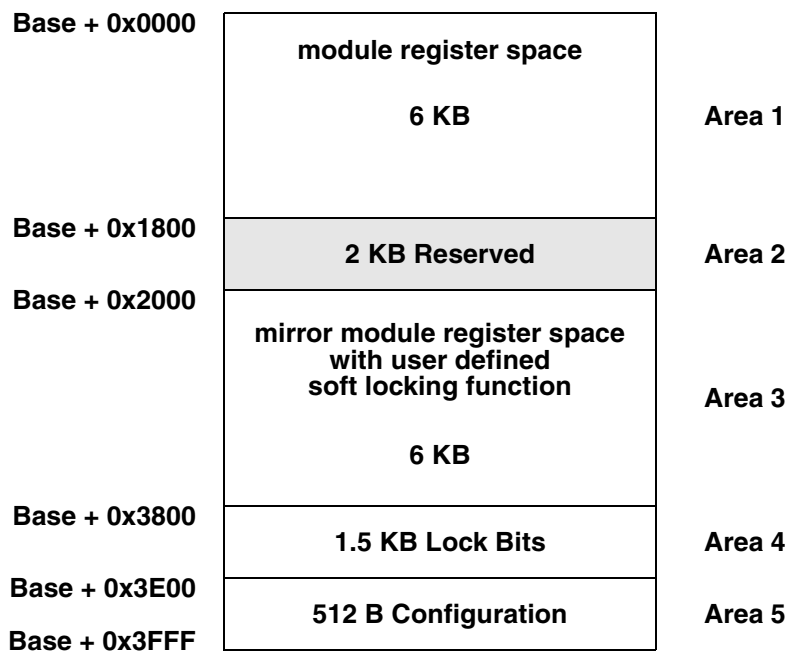
The Register Protection module is operable when the module under protection is operable.

### 32.4 External signal description

There are no external signals.

### 32.5 Memory map and register description

This section provides a detailed description of the memory map of a module using the Register Protection. The original 16 KB module memory space is divided into five areas as shown in [Figure 32-2](#).



**Figure 32-2. Register protection memory diagram**

Area 1 spans 6 KB and holds the normal functional module registers and is transparent for all read/write operations.

Area 2 spans 2 KB starting at address 0x1800. It is a reserved area that cannot be accessed.

Area 3 spans 6 KB, starting at address 0x2000 and is a mirror of area 1. A read/write access to a 0x2000+X address will reads/writes the register at address X. As a side effect, a write access to address 0x2000+X sets the optional soft lock bits for address X in the same cycle as the register at address X is written. Not all registers in area 1 need to have protection defined by associated soft lock bits. For unprotected registers



at address Y, accesses to address 0x2000+Y will be identical to accesses at address Y. Only for registers implemented in area 1 and defined as protectable soft lock bits are available in area 4.

Area 4 is 1.5 KB and holds the soft lock bits, one bit per byte in area 1. The four soft lock bits associated with a module register word are arranged at byte boundaries in the memory map. The soft lock bit registers can be directly written using a bit mask.

Area 5 is 512 byte and holds the configuration bits of the protection mode. There is one configuration hard lock bit per module that prevents all further modifications to the soft lock bits and can only be cleared by a system reset once set. The other bits, if set, will allow user access to the protected module.

If any locked byte is accessed with a write transaction, a transfer error will be issued to the system and the write transaction will not be executed. This is true even if not all accessed bytes are locked.

Accessing unimplemented 32-bit registers in Areas 4 and 5 results in a transfer error.

### 32.5.1 Memory map

Table 32-1 gives an overview on the Register Protection registers implemented.

**Table 32-1. Register protection memory map**

| Address offset | Register  | Location                    |
|----------------|---|-----------------------------|
| 0x0000         | Module Register 0 (MR0)   | <a href="#">on page 898</a> |
| 0x0001         | Module Register 1 (MR1)   | <a href="#">on page 898</a> |
| 0x0002         | Module Register 2 (MR2)   | <a href="#">on page 898</a> |
| 0x0003–0x17FF  | Module Register 3 (MR3) - Module Register 6143 (MR6143)   | <a href="#">on page 898</a> |
| 0x1800–0x1FFF  | Reserved  | —                           |
| 0x2000         | Module Register 0 (MR0) + Set soft lock bit 0 (LMR0)  | <a href="#">on page 898</a> |
| 0x2001         | Module Register 1 (MR1) + Set soft lock bit 1 (LMR1)  | <a href="#">on page 898</a> |
| 0x2002–0x37FF  | Module Register 2 (MR2) + Set soft lock bit 2 (LMR2) –<br>Module Register 6143 (MR6143) + Set soft lock bit 6143 (LMR6143)  | <a href="#">on page 898</a> |
| 0x3800         | Soft Lock Bit Register 0 (SLBR0): soft lock bits 0-3  | <a href="#">on page 898</a> |
| 0x3801         | Soft Lock Bit Register 1 (SLBR1): soft lock bits 4-7  | <a href="#">on page 898</a> |
| 0x3802–0x3DFF  | Soft Lock Bit Register 2 (SLBR2): soft lock bits 8-11 –<br>Soft Lock Bit Register 1535 (SLBR1535): soft lock bits 6140-6143 | <a href="#">on page 898</a> |
| 0x3E00–0x3FFB  | Reserved  | —                           |
| 0x3FFC         | Global Configuration Register (GCR)   | <a href="#">on page 899</a> |

#### NOTE

Reserved registers in area #2 will be handled according to the protected IP (module under protection).

## 32.5.2 Register description

### 32.5.2.1 Module Registers (MR0-6143)

This is the lower 6 KB module memory space that holds all the functional registers of the module that is protected by the Register Protection module.

### 32.5.2.2 Module Register and Set Soft Lock Bit (LMR0-6143)

This is memory area #3 that provides mirrored access to the MR0-6143 registers with the side effect of setting soft lock bits in case of a write access to a MR that is defined as protectable by the locking mechanism. Each MR is protectable by one associated bit in a SLBR<sub>n</sub>.SLB<sub>m</sub>, according to the mapping described in Table 32-2.

### 32.5.2.3 Soft Lock Bit Register (SLBR0-1535)

These registers hold the soft lock bits for the protected registers in memory area #1.

Address 0x3800-0x3DFF

Access: Read always  
Supervisor write

|       |     |     |     |     |      |      |      |      |
|-------|-----|-----|-----|-----|------|------|------|------|
|       | 0   | 1   | 2   | 3   | 4    | 5    | 6    | 7    |
| R     | 0   | 0   | 0   | 0   | SLB0 | SLB1 | SLB2 | SLB3 |
| W     | WE0 | WE1 | WE2 | WE3 |      |      |      |      |
| Reset | 0   | 0   | 0   | 0   | 0    | 0    | 0    | 0    |

Figure 32-3. Soft Lock Bit Register (SLBR<sub>n</sub>)

Table 32-2. SLBR<sub>n</sub> field descriptions

| Field                        | Description   |
|------------------------------|---|
| WE0<br>WE1<br>WE2<br>WE3     | Write Enable Bits for soft lock bits (SLB):<br>WE0 enables writing to SLB0<br>WE1 enables writing to SLB1<br>WE2 enables writing to SLB2<br>WE3 enables writing to SLB3<br><br>1 Value is written to SLB<br>0 SLB is not modified   |
| SLB0<br>SLB1<br>SLB2<br>SLB3 | Soft lock bits for one MR <sub>n</sub> register:<br>SLB0 can block accesses to MR[n × 4 + 0]<br>SLB1 can block accesses to MR[n × 4 + 1]<br>SLB2 can block accesses to MR[n × 4 + 2]<br>SLB3 can block accesses to MR[n × 4 + 3]<br><br>1 Associated MR <sub>n</sub> byte is locked against write accesses<br>0 Associated MR <sub>n</sub> byte is unprotected and writable |

Figure 32-3 gives some examples how SLBR<sub>n</sub>.SLB and MR<sub>n</sub> go together.

Table 32-3. Soft lock bits vs. protected address

| Soft lock bit | Protected address |
|---------------|-------------------|
| SLBR0.SLB0    | MR0               |
| SLBR0.SLB1    | MR1               |
| SLBR0.SLB2    | MR2               |
| SLBR0.SLB3    | MR3               |
| SLBR1.SLB0    | MR4               |
| SLBR1.SLB1    | MR5               |
| SLBR1.SLB2    | MR6               |
| SLBR1.SLB3    | MR7               |
| SLBR2.SLB0    | MR8               |
| ...           | ...               |

### 32.5.2.4 Global Configuration Register (GCR)

This register is used to make global configurations related to register protection.

Address 0x3FFC Access: Read Always Supervisor write

|       |     |    |    |    |    |    |    |    |     |    |    |    |    |    |    |    |
|-------|-----|----|----|----|----|----|----|----|-----|----|----|----|----|----|----|----|
|       | 0   | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8   | 9  | 10 | 11 | 12 | 13 | 14 | 15 |
| R     | HLB | 0  | 0  | 0  | 0  | 0  | 0  | 0  | UAA | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |     |    |    |    |    |    |    |    |     |    |    |    |    |    |    |    |
| Reset | 0   | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
|       | 16  | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24  | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| R     | 0   | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |     |    |    |    |    |    |    |    |     |    |    |    |    |    |    |    |
| Reset | 0   | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 0  | 0  | 0  | 0  | 0  | 0  | 0  |

Figure 32-4. Global Configuration Register (GCR)

Table 32-4. GCR field descriptions

| Field | Description   |
|-------|---|
| HLB   | <p>Hard Lock Bit.<br/>This register cannot be cleared once it is set by software. It can only be cleared by a system reset.</p> <p>1 All SLB bits are write protected and cannot be modified<br/>0 All SLB bits are accessible and can be modified.</p>   |
| UAA   | <p>User Access Allowed.</p> <p>1 The registers in the module under protection can be accessed in the mode defined for the module registers without any additional restrictions.<br/>0 The registers in the module under protection can only be written in supervisor mode. All write accesses in non-supervisor mode are not executed and a transfer error is issued. This access restriction is in addition to any access restrictions imposed by the protected IP module.</p> |

**NOTE**

The GCR.UAA bit has no effect on the allowed access modes for the registers in the Register Protection module.

**32.6 Functional description****32.6.1 General**

This module provides a generic register (address) write-protection mechanism. The protection size can be:

- 32-bit (address == multiples of 4)
- 16-bit (address == multiples of 2)
- 8-bit (address == multiples of 1)
- unprotected (address == multiples of 1)

Which addresses are protected and the protection size depend on the SoC and/or module. Therefore this section can just give examples for various protection configurations.

For all addresses that are protected there are SLBR $n$ .SLB $m$  bits that specify whether the address is locked. When an address is locked it can be read but not written in any mode (supervisor/normal). If an address is unprotected the corresponding SLBR $n$ .SLB $m$  bit is always 0b0 no matter what software is writing to.

**32.6.2 Change lock settings**

To change the setting whether an address is locked or unlocked the corresponding SLBR $n$ .SLB $m$  bit needs to be changed. This can be done using the following methods:

- Modify the SLBR $n$ .SLB $m$  directly by writing to area #4
- Set the SLBR $n$ .SLB $m$  bit(s) by writing to the mirror module space (area #3)

Both methods are explained in the following sections.

### 32.6.2.1 Change lock settings directly via area #4

Memory area #4 contains the lock bits. They can be modified by writing to them. Each  $SLBRn.SLBm$  bit has a mask bit  $SLBRn.WEm$ , which protects it from being modified. This masking makes clear-modify-write operations unnecessary.

Figure 32-5 shows two modification examples. In the left example there is a write access to the  $SLBRn$  register specifying a mask value that allows modification of all  $SLBRn.SLBm$  bits. The example on the right specifies a mask that only allows modification of the bits  $SLBRn.SLB[3:1]$ .

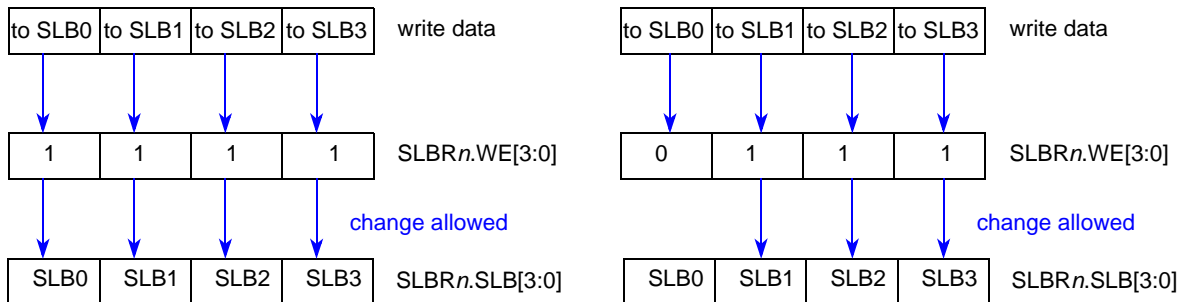


Figure 32-5. Change lock settings directly via Area #4

Figure 32-5 shows four registers that can be protected 8-bit wise. In Figure 32-6 registers with 16-bit protection and in Figure 32-7 registers with 32-bit protection are shown:

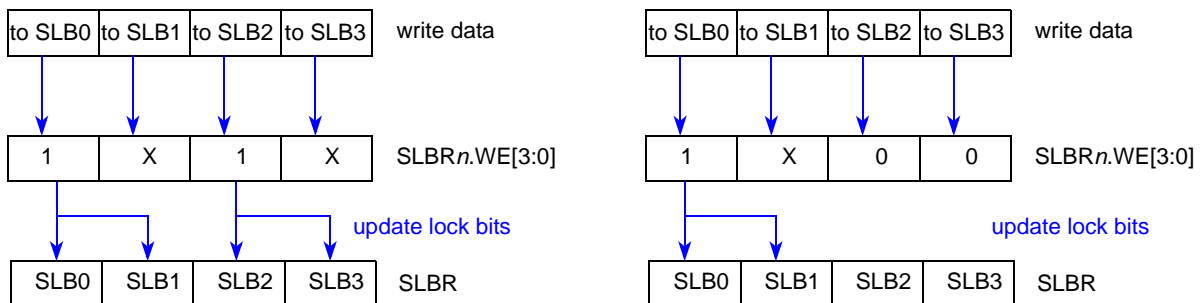


Figure 32-6. Change lock settings for 16-bit protected addresses

On the right side of Figure 32-6 it is shown that the data written to  $SLBRn.SLB[0]$  is automatically written to  $SLBRn.SLB[1]$  also. This is done as the address reflected by  $SLBRn.SLB[0]$  is protected 16-bit wise. Note that in this case the write enable  $SLBRn.WE[0]$  must be set while  $SLBRn.WE[1]$  does not matter. As the enable bits  $SLBRn.WE[3:2]$  are cleared the lock bits  $SLBRn.SLB[3:2]$  remain unchanged.

In the example on the left side of Figure 32-6 the data written to  $SLBRn.SLB[0]$  is mirrored to  $SLBRn.SLB[1]$  and the data written to  $SLBRn.SLB[2]$  is mirrored to  $SLBRn.SLB[3]$  as for both registers the write enables are set.

In Figure 32-7 a 32-bit wise protected register is shown. When  $SLBRn.WE[0]$  is set the data written to  $SLBRn.SLB[0]$  is automatically written to  $SLBRn.SLB[3:1]$  also. Otherwise  $SLBRn.SLB[3:0]$  remains unchanged.

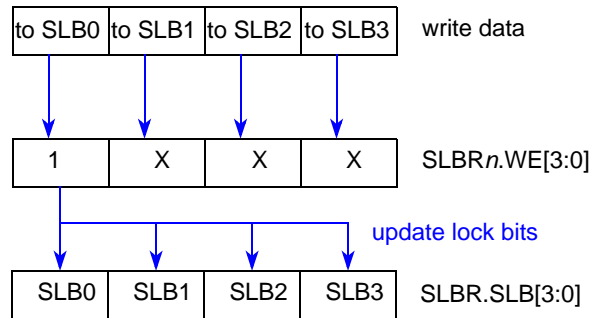


Figure 32-7. Change lock settings for 32-bit protected addresses

In Figure 32-8 an example is shown that has a mixed protection size configuration:

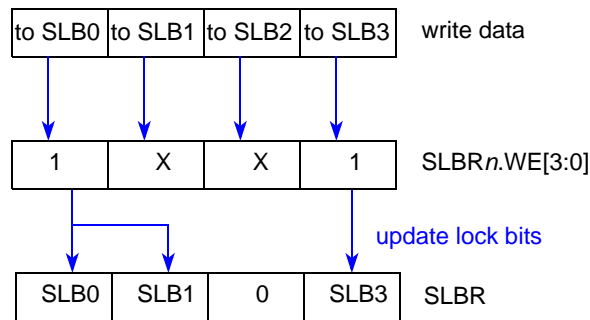
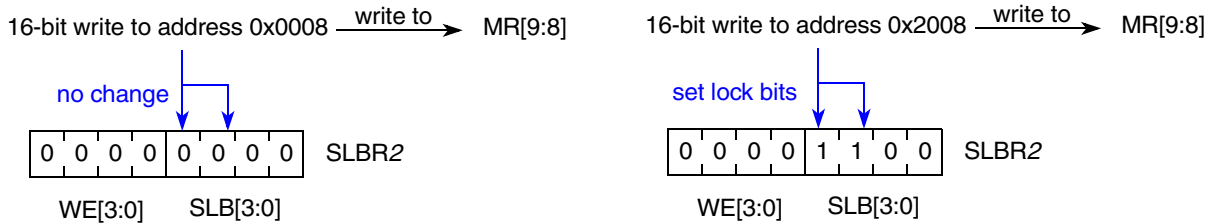


Figure 32-8. Change lock settings for mixed protection

The data written to  $SLBRn.SLB[0]$  is mirrored to  $SLBRn.SLB[1]$  as the corresponding register is 16-bit protected. The data written to  $SLBRn.SLB[2]$  is blocked as the corresponding register is unprotected. The data written to  $SLBRn.SLB[3]$  is written to  $SLBRn.SLB[3]$ .

### 32.6.2.2 Enable locking via mirror module space (area #3)

It is possible to enable locking for a register after writing to it. To do so the mirrored module address space must be used. Figure 32-9 shows one example:

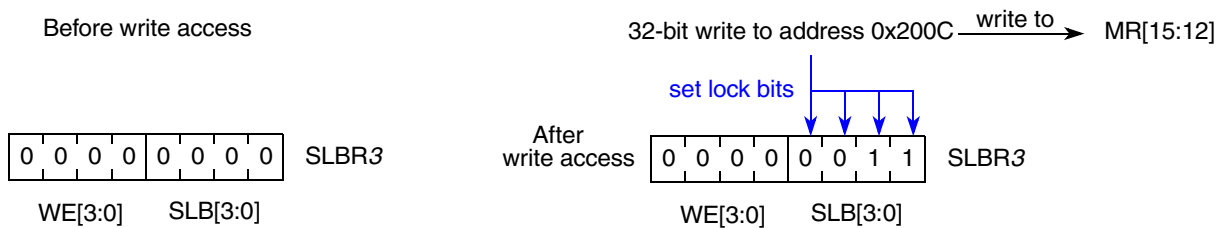


**Figure 32-9. Enable locking via mirror module space (Area #3)**

When writing to address 0x0008 the registers MR9 and MR8 in the protected module are updated. The corresponding lock bits remain unchanged (left part of [Figure 32-6](#)).

When writing to address 0x2008 the registers MR9 and MR8 in the protected module are updated. The corresponding lock bits SLBR2.SLB[1:0] are set while the lock bits SLBR2.SLB[3:2] remain unchanged (right part of [Figure 32-6](#)).

[Figure 32-10](#) shows an example where some addresses are protected and some are not:



**Figure 32-10. Enable locking for protected and unprotected addresses**

In the example in [Figure 32-10](#) addresses 0x0C and 0x0D are unprotected. Therefore their corresponding lock bits SLBR3.SLB[1:0] are always 0b0 (shown in bold). When doing a 32-bit write access to address 0x200C only lock bits SLBR3.SLB[3:2] are set while bits SLBR3.SLB[1:0] stay 0b0.

#### NOTE

Lock bits can only be set via writes to the mirror module space. Reads from the mirror module space will not change the lock bits.

### 32.6.2.3 Write protection for locking bits

Changing the locking bits through any of the procedures mentioned in [Section 32.6.2.1, Change lock settings directly via area #4](#) and [Section 32.6.2.2, Enable locking via mirror module space \(area #3\)](#) is only possible as long as the bit GCR.HLB is cleared. Once this bit is set, the locking bits can no longer be modified until there is a system reset.

### 32.6.3 Access errors

The protection module generates transfer errors under several circumstances. For the area definition refer to [Figure 32-2](#).

1. If accessing area #1 or area #3, the protection module transfers any access error from the underlying Module under Protection.
2. If user mode is not allowed, user write attempts to all areas will assert a transfer error and the writes will be blocked.
3. Access attempts to the reserved area #2 cause a transfer error to be asserted.
4. Access attempts to unimplemented 32-bit registers in area #4 or area #5 cause a transfer error to be asserted.
5. Attempted writes to a register in area #1 or area #3 with soft lock bit set for any of the affected bytes causes a transfer error to be asserted and the write is blocked. The complete write operation to non-protected bytes in this word is ignored.
6. If writing to a soft lock register in area #4 with the hard lock bit being set a transfer error is asserted.
7. Any write operation in any access mode to area #3 while GCR.HLB is set result in a error.

## 32.7 Reset

The reset state of each individual bit is shown within the Register Description section (See [Section 32.5.2, Register description](#)). In summary, after reset, locking for all MR<sub>n</sub> registers is disabled. The registers can be accessed in Supervisor Mode only.

## 32.8 Protected registers

For MPC5606BK the Register Protection module protects the registers shown in [Table 32-5](#).

**Table 32-5. Protected registers**

| Module   | Register | Protected size (bits) | Module base address | Register offset | Protected bits |
|--|----------|-----------------------|---------------------|-----------------|----------------|
| <b>Code flash memory 0, 4 registers to protect</b> |          |                       |                     |                 |                |
| Code Flash 0                                       | MCR      | 32                    | C3F88000            | 000             | bits[0:31]     |
| Code Flash 0                                       | PFCR0    | 32                    | C3F88000            | 01C             | bits[0:31]     |
| Code Flash 0                                       | PFCR1    | 32                    | C3F88000            | 020             | bits[0:31]     |
| Code Flash 0                                       | PFAPR    | 32                    | C3F88000            | 024             | bits[0:31]     |
| <b>Data flash memory, 1 register to protect</b>    |          |                       |                     |                 |                |
| Data Flash   | MCR      | 32                    | C3F8C000            | 000             | bits[0:31]     |
| <b>SIU lite, 64 registers to protect</b>           |          |                       |                     |                 |                |
| SIUL   | IRER     | 32                    | C3F90000            | 018             | bits[0:31]     |
| SIUL   | IREER    | 32                    | C3F90000            | 028             | bits[0:31]     |



Table 32-5. Protected registers (continued)

| Module | Register | Protected size (bits) | Module base address | Register offset | Protected bits |
|--------|----------|-----------------------|---------------------|-----------------|----------------|
| SIUL   | IFEER    | 32                    | C3F90000            | 02C             | bits[0:31]     |
| SIUL   | IFER     | 32                    | C3F90000            | 030             | bits[0:31]     |
| SIUL   | PCR0     | 16                    | C3F90000            | 040             | bits[0:15]     |
| SIUL   | PCR1     | 16                    | C3F90000            | 042             | bits[0:15]     |
| SIUL   | PCR2     | 16                    | C3F90000            | 044             | bits[0:15]     |
| SIUL   | PCR3     | 16                    | C3F90000            | 046             | bits[0:15]     |
| SIUL   | PCR4     | 16                    | C3F90000            | 048             | bits[0:15]     |
| SIUL   | PCR5     | 16                    | C3F90000            | 04A             | bits[0:15]     |
| SIUL   | PCR6     | 16                    | C3F90000            | 04C             | bits[0:15]     |
| SIUL   | PCR7     | 16                    | C3F90000            | 04E             | bits[0:15]     |
| SIUL   | PCR8     | 16                    | C3F90000            | 050             | bits[0:15]     |
| SIUL   | PCR9     | 16                    | C3F90000            | 052             | bits[0:15]     |
| SIUL   | PCR10    | 16                    | C3F90000            | 054             | bits[0:15]     |
| SIUL   | PCR11    | 16                    | C3F90000            | 056             | bits[0:15]     |
| SIUL   | PCR12    | 16                    | C3F90000            | 058             | bits[0:15]     |
| SIUL   | PCR13    | 16                    | C3F90000            | 05A             | bits[0:15]     |
| SIUL   | PCR14    | 16                    | C3F90000            | 05C             | bits[0:15]     |
| SIUL   | PCR15    | 16                    | C3F90000            | 05E             | bits[0:15]     |
| SIUL   | PCR16    | 16                    | C3F90000            | 060             | bits[0:15]     |
| SIUL   | PCR17    | 16                    | C3F90000            | 062             | bits[0:15]     |
| SIUL   | PCR18    | 16                    | C3F90000            | 064             | bits[0:15]     |
| SIUL   | PCR19    | 16                    | C3F90000            | 066             | bits[0:15]     |
| SIUL   | PCR34    | 16                    | C3F90000            | 084             | bits[0:15]     |
| SIUL   | PCR35    | 16                    | C3F90000            | 086             | bits[0:15]     |
| SIUL   | PCR36    | 16                    | C3F90000            | 088             | bits[0:15]     |
| SIUL   | PCR37    | 16                    | C3F90000            | 08A             | bits[0:15]     |
| SIUL   | PCR38    | 16                    | C3F90000            | 08C             | bits[0:15]     |
| SIUL   | PCR39    | 16                    | C3F90000            | 08E             | bits[0:15]     |
| SIUL   | PCR40    | 16                    | C3F90000            | 090             | bits[0:15]     |
| SIUL   | PCR41    | 16                    | C3F90000            | 092             | bits[0:15]     |
| SIUL   | PCR42    | 16                    | C3F90000            | 094             | bits[0:15]     |
| SIUL   | PCR43    | 16                    | C3F90000            | 096             | bits[0:15]     |

Table 32-5. Protected registers (continued)

| Module | Register | Protected size (bits) | Module base address | Register offset | Protected bits |
|--------|----------|-----------------------|---------------------|-----------------|----------------|
| SIUL   | PCR44    | 16                    | C3F90000            | 098             | bits[0:15]     |
| SIUL   | PCR45    | 16                    | C3F90000            | 09A             | bits[0:15]     |
| SIUL   | PCR46    | 16                    | C3F90000            | 09C             | bits[0:15]     |
| SIUL   | PCR47    | 16                    | C3F90000            | 09E             | bits[0:15]     |
| SIUL   | PCR64    | 16                    | C3F90000            | 0C0             | bits[0:15]     |
| SIUL   | PCR65    | 16                    | C3F90000            | 0C2             | bits[0:15]     |
| SIUL   | PCR66    | 16                    | C3F90000            | 0C4             | bits[0:15]     |
| SIUL   | PCR67    | 16                    | C3F90000            | 0C6             | bits[0:15]     |
| SIUL   | PCR68    | 16                    | C3F90000            | 0C8             | bits[0:15]     |
| SIUL   | PCR69    | 16                    | C3F90000            | 0CA             | bits[0:15]     |
| SIUL   | PCR70    | 16                    | C3F90000            | 0CC             | bits[0:15]     |
| SIUL   | PCR71    | 16                    | C3F90000            | 0CE             | bits[0:15]     |
| SIUL   | PCR72    | 16                    | C3F90000            | 0D0             | bits[0:15]     |
| SIUL   | PCR73    | 16                    | C3F90000            | 0D2             | bits[0:15]     |
| SIUL   | PCR74    | 16                    | C3F90000            | 0D4             | bits[0:15]     |
| SIUL   | PCR75    | 16                    | C3F90000            | 0D6             | bits[0:15]     |
| SIUL   | PCR76    | 16                    | C3F90000            | 0D8             | bits[0:15]     |
| SIUL   | PCR77    | 16                    | C3F90000            | 0DA             | bits[0:15]     |
| SIUL   | PCR78    | 16                    | C3F90000            | 0DC             | bits[0:15]     |
| SIUL   | PCR79    | 16                    | C3F90000            | 0DE             | bits[0:15]     |
| SIUL   | PCR88    | 16                    | C3F90000            | 0F0             | bits[0:15]     |
| SIUL   | PCR89    | 16                    | C3F90000            | 0F2             | bits[0:15]     |
| SIUL   | PCR90    | 16                    | C3F90000            | 0F4             | bits[0:15]     |
| SIUL   | PCR91    | 16                    | C3F90000            | 0F6             | bits[0:15]     |
| SIUL   | PCR92    | 16                    | C3F90000            | 0F8             | bits[0:15]     |
| SIUL   | PCR93    | 16                    | C3F90000            | 0FA             | bits[0:15]     |
| SIUL   | PCR94    | 16                    | C3F90000            | 0FC             | bits[0:15]     |
| SIUL   | PCR95    | 16                    | C3F90000            | 0FE             | bits[0:15]     |
| SIUL   | PCR96    | 16                    | C3F90000            | 100             | bits[0:15]     |
| SIUL   | PCR97    | 16                    | C3F90000            | 102             | bits[0:15]     |
| SIUL   | PCR98    | 16                    | C3F90000            | 104             | bits[0:15]     |
| SIUL   | PCR99    | 16                    | C3F90000            | 106             | bits[0:15]     |

Table 32-5. Protected registers (continued)

| Module | Register | Protected size (bits) | Module base address | Register offset | Protected bits |
|--------|----------|-----------------------|---------------------|-----------------|----------------|
| SIUL   | PCR100   | 16                    | C3F90000            | 108             | bits[0:15]     |
| SIUL   | PCR101   | 16                    | C3F90000            | 10A             | bits[0:15]     |
| SIUL   | PCR102   | 16                    | C3F90000            | 10C             | bits[0:15]     |
| SIUL   | PCR103   | 16                    | C3F90000            | 10E             | bits[0:15]     |
| SIUL   | PCR104   | 16                    | C3F90000            | 110             | bits[0:15]     |
| SIUL   | PCR105   | 16                    | C3F90000            | 112             | bits[0:15]     |
| SIUL   | PCR106   | 16                    | C3F90000            | 114             | bits[0:15]     |
| SIUL   | PCR107   | 16                    | C3F90000            | 116             | bits[0:15]     |
| SIUL   | PCR108   | 16                    | C3F90000            | 118             | bits[0:15]     |
| SIUL   | PCR109   | 16                    | C3F90000            | 11A             | bits[0:15]     |
| SIUL   | PCR110   | 16                    | C3F90000            | 11C             | bits[0:15]     |
| SIUL   | PCR111   | 16                    | C3F90000            | 11E             | bits[0:15]     |
| SIUL   | PCR112   | 16                    | C3F90000            | 120             | bits[0:15]     |
| SIUL   | PCR113   | 16                    | C3F90000            | 122             | bits[0:15]     |
| SIUL   | PCR114   | 16                    | C3F90000            | 124             | bits[0:15]     |
| SIUL   | PCR115   | 16                    | C3F90000            | 126             | bits[0:15]     |
| SIUL   | PCR123   | 16                    | C3F90000            | 136             | bits[0:15]     |
| SIUL   | PCR124   | 16                    | C3F90000            | 138             | bits[0:15]     |
| SIUL   | PCR125   | 16                    | C3F90000            | 13A             | bits[0:15]     |
| SIUL   | PCR126   | 16                    | C3F90000            | 13C             | bits[0:15]     |
| SIUL   | PCR127   | 16                    | C3F90000            | 13E             | bits[0:15]     |
| SIUL   | PCR128   | 16                    | C3F90000            | 140             | bits[0:15]     |
| SIUL   | PCR129   | 16                    | C3F90000            | 142             | bits[0:15]     |
| SIUL   | PCR130   | 16                    | C3F90000            | 144             | bits[0:15]     |
| SIUL   | PCR131   | 16                    | C3F90000            | 146             | bits[0:15]     |
| SIUL   | PCR132   | 16                    | C3F90000            | 148             | bits[0:15]     |
| SIUL   | PCR133   | 16                    | C3F90000            | 14A             | bits[0:15]     |
| SIUL   | PCR134   | 16                    | C3F90000            | 14C             | bits[0:15]     |
| SIUL   | PCR135   | 16                    | C3F90000            | 14E             | bits[0:15]     |
| SIUL   | PSMI0_3  | 8                     | C3F90000            | 500             | bits[0:7]      |
| SIUL   | PSMI4_7  | 8                     | C3F90000            | 504             | bits[0:7]      |
| SIUL   | PSMI8_11 | 8                     | C3F90000            | 508             | bits[0:7]      |

Table 32-5. Protected registers (continued)

| Module | Register  | Protected size (bits) | Module base address | Register offset | Protected bits |
|--------|-----------|-----------------------|---------------------|-----------------|----------------|
| SIUL   | PSMI12_15 | 8                     | C3F90000            | 50C             | bits[0:7]      |
| SIUL   | PSMI16_19 | 8                     | C3F90000            | 510             | bits[0:7]      |
| SIUL   | PSMI20_23 | 32                    | C3F90000            | 514             | bits[0:7]      |
| SIUL   | PSMI24_27 | 32                    | C3F90000            | 518             | bits[0:7]      |
| SIUL   | PSMI28_31 | 32                    | C3F90000            | 51C             | bits[0:7]      |
| SIUL   | PSMI32_35 | 32                    | C3F90000            | 520             | bits[0:7]      |
| SIUL   | PSMI36_39 | 32                    | C3F90000            | 524             | bits[0:7]      |
| SIUL   | PSMI40_43 | 32                    | C3F90000            | 528             | bits[0:7]      |
| SIUL   | PSMI44_47 | 32                    | C3F90000            | 52C             | bits[0:7]      |
| SIUL   | PSMI48_51 | 32                    | C3F90000            | 530             | bits[0:7]      |
| SIUL   | PSMI52_55 | 32                    | C3F90000            | 534             | bits[0:7]      |
| SIUL   | PSMI56_59 | 32                    | C3F90000            | 538             | bits[0:7]      |
| SIUL   | PSMI61_63 | 32                    | C3F90000            | 53C             | bits[0:7]      |
| SIUL   | IFMC0     | 32                    | C3F90000            | 1000            | bits[0:31]     |
| SIUL   | IFMC1     | 32                    | C3F90000            | 1004            | bits[0:31]     |
| SIUL   | IFMC2     | 32                    | C3F90000            | 1008            | bits[0:31]     |
| SIUL   | IFMC3     | 32                    | C3F90000            | 100C            | bits[0:31]     |
| SIUL   | IFMC4     | 32                    | C3F90000            | 1010            | bits[0:31]     |
| SIUL   | IFMC5     | 32                    | C3F90000            | 1014            | bits[0:31]     |
| SIUL   | IFMC6     | 32                    | C3F90000            | 1018            | bits[0:31]     |
| SIUL   | IFMC7     | 32                    | C3F90000            | 101C            | bits[0:31]     |
| SIUL   | IFMC8     | 32                    | C3F90000            | 1020            | bits[0:31]     |
| SIUL   | IFMC9     | 32                    | C3F90000            | 1024            | bits[0:31]     |
| SIUL   | IFMC10    | 32                    | C3F90000            | 1028            | bits[0:31]     |
| SIUL   | IFMC11    | 32                    | C3F90000            | 102C            | bits[0:31]     |
| SIUL   | IFMC12    | 32                    | C3F90000            | 1030            | bits[0:31]     |
| SIUL   | IFMC13    | 32                    | C3F90000            | 1034            | bits[0:31]     |
| SIUL   | IFMC14    | 32                    | C3F90000            | 1038            | bits[0:31]     |
| SIUL   | IFMC15    | 32                    | C3F90000            | 103C            | bits[0:31]     |
| SIUL   | IFMC16    | 32                    | C3F90000            | 1040            | bits[0:31]     |
| SIUL   | IFMC17    | 32                    | C3F90000            | 1044            | bits[0:31]     |
| SIUL   | IFMC18    | 32                    | C3F90000            | 1048            | bits[0:31]     |

Table 32-5. Protected registers (continued)

| Module  | Register      | Protected size (bits) | Module base address | Register offset | Protected bits |
|---|---------------|-----------------------|---------------------|-----------------|----------------|
| SIUL  | IFMC19        | 32                    | C3F90000            | 104C            | bits[0:31]     |
| SIUL  | IFMC20        | 32                    | C3F90000            | 1050            | bits[0:31]     |
| SIUL  | IFMC21        | 32                    | C3F90000            | 1054            | bits[0:31]     |
| SIUL  | IFMC22        | 32                    | C3F90000            | 1058            | bits[0:31]     |
| SIUL  | IFMC23        | 32                    | C3F90000            | 105C            | bits[0:31]     |
| SIUL  | IFCPR         | 32                    | C3F90000            | 1080            | bits[0:31]     |
| <b>Mode Entry Module, 41 registers to protect</b> |               |                       |                     |                 |                |
| MC ME   | ME_ME         | 32                    | C3FDC000            | 008             | bits[0:31]     |
| MC ME   | ME_IM         | 32                    | C3FDC000            | 010             | bits[0:31]     |
| MC ME   | ME_TEST_MC    | 32                    | C3FDC000            | 024             | bits[0:31]     |
| MC ME   | ME_SAFE_MC    | 32                    | C3FDC000            | 028             | bits[0:31]     |
| MC ME   | ME_DRUN_MC    | 32                    | C3FDC000            | 02C             | bits[0:31]     |
| MC ME   | ME_RUN0_MC    | 32                    | C3FDC000            | 030             | bits[0:31]     |
| MC ME   | ME_RUN1_MC    | 32                    | C3FDC000            | 034             | bits[0:31]     |
| MC ME   | ME_RUN2_MC    | 32                    | C3FDC000            | 038             | bits[0:31]     |
| MC ME   | ME_RUN3_MC    | 32                    | C3FDC000            | 03C             | bits[0:31]     |
| MC ME   | ME_HALT_MC    | 32                    | C3FDC000            | 040             | bits[0:31]     |
| MC ME   | ME_STOP_MC    | 32                    | C3FDC000            | 048             | bits[0:31]     |
| MC ME   | ME_STANDBY_MC | 32                    | C3FDC000            | 054             | bits[0:31]     |
| MC ME   | ME_RUN_PC0    | 32                    | C3FDC000            | 080             | bits[0:31]     |
| MC ME   | ME_RUN_PC1    | 32                    | C3FDC000            | 084             | bits[0:31]     |
| MC ME   | ME_RUN_PC2    | 32                    | C3FDC000            | 088             | bits[0:31]     |
| MC ME   | ME_RUN_PC3    | 32                    | C3FDC000            | 08C             | bits[0:31]     |
| MC ME   | ME_RUN_PC4    | 32                    | C3FDC000            | 090             | bits[0:31]     |
| MC ME   | ME_RUN_PC5    | 32                    | C3FDC000            | 094             | bits[0:31]     |
| MC ME   | ME_RUN_PC6    | 32                    | C3FDC000            | 098             | bits[0:31]     |
| MC ME   | ME_RUN_PC7    | 32                    | C3FDC000            | 09C             | bits[0:31]     |
| MC ME   | ME_LP_PC0     | 32                    | C3FDC000            | 0A0             | bits[0:31]     |
| MC ME   | ME_LP_PC1     | 32                    | C3FDC000            | 0A4             | bits[0:31]     |
| MC ME   | ME_LP_PC2     | 32                    | C3FDC000            | 0A8             | bits[0:31]     |
| MC ME   | ME_LP_PC3     | 32                    | C3FDC000            | 0AC             | bits[0:31]     |
| MC ME   | ME_LP_PC4     | 32                    | C3FDC000            | 0B0             | bits[0:31]     |

Table 32-5. Protected registers (continued)

| Module   | Register          | Protected size (bits) | Module base address | Register offset | Protected bits |
|--|-------------------|-----------------------|---------------------|-----------------|----------------|
| MC ME  | ME_LP_PC5         | 32                    | C3FDC000            | 0B4             | bits[0:31]     |
| MC ME  | ME_LP_PC6         | 32                    | C3FDC000            | 0B8             | bits[0:31]     |
| MC ME  | ME_LP_PC7         | 32                    | C3FDC000            | 0BC             | bits[0:31]     |
| MC ME  | ME_PCTL[4..7]     | 32                    | C3FDC000            | 0C4             | bits[0:31]     |
| MC ME  | ME_PCTL[16..19]   | 32                    | C3FDC000            | 0D0             | bits[0:31]     |
| MC ME  | ME_PCTL[20..23]   | 32                    | C3FDC000            | 0D4             | bits[0:31]     |
| MC ME  | ME_PCTL[32..35]   | 32                    | C3FDC000            | 0E0             | bits[0:31]     |
| MC ME  | ME_PCTL[44..47]   | 32                    | C3FDC000            | 0EC             | bits[0:31]     |
| MC ME  | ME_PCTL[48..51]   | 32                    | C3FDC000            | 0F0             | bits[0:31]     |
| MC ME  | ME_PCTL[56..59]   | 32                    | C3FDC000            | 0F8             | bits[0:31]     |
| MC ME  | ME_PCTL[60..63]   | 32                    | C3FDC000            | 0FC             | bits[0:31]     |
| MC ME  | ME_PCTL[68..71]   | 32                    | C3FDC000            | 104             | bits[0:31]     |
| MC ME  | ME_PCTL[72..75]   | 32                    | C3FDC000            | 108             | bits[0:31]     |
| MC ME  | ME_PCTL[88..91]   | 32                    | C3FDC000            | 118             | bits[0:31]     |
| MC ME  | ME_PCTL[92..95]   | 32                    | C3FDC000            | 11C             | bits[0:31]     |
| MC ME  | ME_PCTL[104..107] | 32                    | C3FDC000            | 128             | bits[0:31]     |
| <b>Clock Generation Module, 3 registers to protect</b> |                   |                       |                     |                 |                |
| MC CGM   | CGM_OC_EN         | 8                     | C3FE0000            | 373             | bits[0:7]      |
| MC CGM   | CGM_OCDS_SC       | 8                     | C3FE0000            | 374             | bits[0:7]      |
| MC CGM   | CGM_SC_DC[0..3]   | 32                    | C3FE0000            | 37C             | bits[0:31]     |
| <b>CMU, 1 register to protect</b>                      |                   |                       |                     |                 |                |
| CMU  | CMU_CSR           | 8                     | C3FE00E0            | 103             | bits[24:31]    |
| <b>Reset Generation Module, 7 registers to protect</b> |                   |                       |                     |                 |                |
| MC RGM   | RGM_FERD          | 16                    | C3FE4000            | 004             | bits[0:15]     |
| MC RGM   | RGM_DERD          | 16                    | C3FE4000            | 006             | bits[0:15]     |
| MC RGM   | RGM_FEAR          | 16                    | C3FE4000            | 010             | bits[0:15]     |
| MC RGM   | RGM_DEAR          | 16                    | C3FE4000            | 012             | bits[0:15]     |
| MC RGM   | RGM_FESS          | 16                    | C3FE4000            | 018             | bits[0:15]     |
| MC RGM   | RGM_STDBY         | 16                    | C3FE4000            | 01A             | bits[0:15]     |
| MC RGM   | RGM_FBRE          | 16                    | C3FE4000            | 01C             | bits[0:15]     |
| <b>Power Control Unit, 2 registers to protect</b>      |                   |                       |                     |                 |                |

Table 32-5. Protected registers (continued)

| Module | Register | Protected size (bits) | Module base address | Register offset | Protected bits |
|--------|----------|-----------------------|---------------------|-----------------|----------------|
| MC PCU | PCONF2   | 32                    | C3FE8000            | 008             | bits[0:31]     |
| MC PCU | PCONF3   | 32                    | C3FE8000            | 00C             | bits[0:31]     |

This page is intentionally left blank.



# Chapter 33

## Software Watchdog Timer (SWT)

### 33.1 Overview

The SWT is a peripheral module that can prevent system lockup in situations such as software getting trapped in a loop or if a bus transaction fails to terminate. When enabled, the SWT requires periodic execution of a watchdog servicing sequence. Writing the sequence resets the timer to a specified time-out period. If this servicing action does not occur before the timer expires the SWT generates an interrupt or hardware reset. The SWT can be configured to generate a reset or interrupt on an initial time-out, a reset is always generated on a second consecutive time-out.

The SWT provides a window functionality. When this functionality is programmed, the servicing action should take place within the defined window. When occurring outside the defined period, the SWT generates a reset.

### 33.2 Features

The SWT has the following features:

- 32-bit time-out register to set the time-out period
- The unique SWT counter clock is the undivided slow internal RC oscillator 128 kHz (SIRC), no other clock source can be selected
- Programmable selection of window mode or regular servicing
- Programmable selection of reset or interrupt on an initial time-out
- Master access protection
- Hard and soft configuration lock bits
- The SWT is started on exit of power-on phase (RGM phase 2) to monitor flash boot sequence phase. It is then reset during RGM phase3 and optionally enabled when platform reset is released depending on value of flash user option bit 31 (WATCHDOG\_EN).

### 33.3 Modes of operation

The SWT supports three device modes of operation: normal, debug, and stop. When the SWT is enabled in normal mode, its counter runs continuously. In debug mode, operation of the counter is controlled by the FRZ bit in the SWT\_CR. If the FRZ bit is set, the counter is stopped in debug mode, otherwise it continues to run. In STOP mode, operation of the counter is controlled by the STP bit in the SWT\_CR. If the STP bit is set, the counter is stopped in STOP mode, otherwise it continues to run. On exit from STOP mode, the SWT will continue from the state it was before entering this mode.

The software watchdog is not available during standby. On exit from standby, the SWT behaves in a usual “out of reset” situation.

## 33.4 External signal description

The SWT module does not have any external interface signals.

## 33.5 Memory map and register description

The SWT programming model has six 32-bit registers. The programming model can only be accessed using 32-bit (word) accesses. References using a different size are invalid. Other types of invalid accesses include: writes to read only registers, incorrect values written to the service register when enabled, accesses to reserved addresses, and accesses by masters without permission. If the SWT\_CR[RIA] bit is set, then the SWT generates a system reset on an invalid access. Otherwise, a bus error is generated. If either the HLK or SLK bits in the SWT\_CR are set, then the SWT\_CR, SWT\_TO, SWT\_SK, and SWT\_WN registers are read-only. point save

### 33.5.1 Memory map

The SWT memory map is shown in [Table 33-1](#). The reset values of SWT\_CR, SWT\_TO, and SWT\_WN are device-specific. These values are determined by SWT inputs.

**Table 33-1. SWT memory map**

| Base address: 0xFFF3_8000 |                                      |                     |                           |                             |
|---------------------------|--------------------------------------|---------------------|---------------------------|-----------------------------|
| Address offset            | Register                             | Access <sup>1</sup> | Reset Value <sup>23</sup> | Location                    |
| 0x0000                    | SWT Control Register (SWT_CR)        | R/W                 | 0x4000_011U               | <a href="#">on page 915</a> |
| 0x0004                    | SWT Interrupt Register (SWT_IR)      | R/W                 | 0x0000_0000               | <a href="#">on page 916</a> |
| 0x0008                    | SWT Time-Out Register (SWT_TO)       | R/W                 | 0x0000_0500               | <a href="#">on page 917</a> |
| 0x000C                    | SWT Window Register (SWT_WN)         | R/W                 | 0x0000_0000               | <a href="#">on page 917</a> |
| 0x0010                    | SWT Service Register (SWT_SR)        | W                   | 0x0000_0000               | <a href="#">on page 918</a> |
| 0x0014                    | SWT Counter Output Register (SWT_CO) | R                   | 0x0000_0000               | <a href="#">on page 918</a> |
| 0x0018                    | SWT Service Key Register (SWT_SK)    | W                   | 0x0000_0000               | <a href="#">on page 919</a> |
| 0x001C–0xFFFF             | Reserved                             |                     |                           |                             |

<sup>1</sup> In this column, R/W = Read/Write, R = Read-only, and W = Write-only.

<sup>2</sup> In this column, the format of the reset value indicates the register width. Thus, reset values of the formats 0xHH, 0xHHHH, and 0xHHHH\_HHHH, where H is a hexadecimal digit, indicate 8-, 16-, and 32-bit registers, respectively.

<sup>3</sup> In this column, the symbol “U” indicates one or more bits in a byte are undefined at reset. See the associated description for more information.

## 33.5.2 Register description

### 33.5.2.1 SWT Control Register (SWT\_CR)

The SWT\_CR contains fields for configuring and controlling the SWT. The reset value of this register is device specific. Some devices can be configured to automatically clear the SWT\_CR[WEN] bit during the boot process. If either the SWT\_CR[HLK] or the SWT\_CR[SLK] bit is set, this register is read-only.

| Address: Base + 0x0000 |     |     |     |     |     |     |     | Access: User read/write |   |   |    |    |    |    |    |    |
|------------------------|-----|-----|-----|-----|-----|-----|-----|-------------------------|---|---|----|----|----|----|----|----|
|                        | 0   | 1   | 2   | 3   | 4   | 5   | 6   | 7                       | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| R                      | MAP | MAP | MAP | MAP | MAP | MAP | MAP | MAP                     | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W                      | 0   | 1   | 2   | 3   | 4   | 5   | 6   | 7                       |   |   |    |    |    |    |    |    |
| Reset <sup>1</sup>     | 1   | 0   | 0   | 0   | 0   | 0   | 0   | 0                       | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       | 16 | 17 | 18 | 19 | 20 | 21 | 22  | 23  | 24  | 25  | 26  | 27  | 28  | 29  | 30  | 31  |
|-------|----|----|----|----|----|----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| R     | 0  | 0  | 0  | 0  | 0  | 0  | KEY | RIA | WND | ITR | HLK | SLK | CSL | STP | FRZ | WEN |
| W     |    |    |    |    |    |    |     |     |     |     |     |     |     |     |     |     |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 1   | 0   | 0   | 0   | 1   | 1   | 0   | 1   | –   |

<sup>1</sup> The reset value for the SWT\_CR is device-specific.

**Figure 33-1. SWT Control Register (SWT\_CR)**

This last bit is cleared when exiting ME RESET mode in case flash user option bit 31 (WATCHDOG\_EN) is 0.

**Table 33-2. SWT\_CR field descriptions**

| Field   | Description  |
|---------|--|
| MAP $n$ | Master Access Protection for Master $n$ .<br>Allows specific master to update watchdog. MAP0 = CPU, MAP2 = eDMA.<br>The platform bus master assignments are device-specific.<br>0 Access for the master is not enabled<br>1 Access for the master is enabled |
| KEY     | Keyed Service Mode.<br>0 Fixed Service Sequence, the fixed sequence 0xA602, 0xB480 is used to service the watchdog.<br>1 Keyed Service Mode, two pseudorandom key value are used to service the watchdog.  |
| RIA     | Reset on Invalid Access.<br>0 Invalid access to the SWT generates a bus error.<br>1 Invalid access to the SWT causes a system reset if WEN = 1.  |
| WND     | Window Mode.<br>0 Regular mode, service sequence can be done at any time.<br>1 Windowed mode, the service sequence is only valid when the down counter is less than the value in the SWT_WN register.  |
| ITR     | Interrupt Then Reset.<br>0 Generate a reset on a time-out.<br>1 Generate an interrupt on an initial time-out, reset on a second consecutive time-out.  |

**Table 33-2. SWT\_CR field descriptions**

| Field | Description   |
|-------|---|
| HLK   | Hard Lock. This bit is only cleared at reset.<br>0 SWT_CR, SWT_TO, SWT_WN, and SWT_SK are read/write registers if SLK = 0.<br>1 SWT_CR, SWT_TO, SWT_WN, and SWT_SK are read-only registers.   |
| SLK   | Soft Lock. This bit is cleared by writing the unlock sequence to the SWT Service Register (SWT_SR).<br>0 SWT_CR, SWT_TO, SWT_WN, and SWT_SK are read/write registers if HLK = 0.<br>1 SWT_CR, SWT_TO, SWT_WN, and SWT_SK are read-only registers.                         |
| CSL   | Clock Selection. Selects the SIRC oscillator clock that drives the internal timer.<br>CSL bit can be written. The status of the bit has no effect on counter clock selection on MPC5606BK device.<br>0 System clock (Not applicable in MPC5606BK).<br>1 Oscillator clock. |
| STP   | Stop Mode Control. Allows the watchdog timer to be stopped when the device enters STOP mode.<br>0 SWT counter continues to run in STOP mode.<br>1 SWT counter is stopped in STOP mode.  |
| FRZ   | Debug Mode Control. Allows the watchdog timer to be stopped when the device enters debug mode.<br>0 SWT counter continues to run in debug mode.<br>1 SWT counter is stopped in debug mode.  |
| WEN   | Watchdog Enabled.<br>0 SWT is disabled.<br>1 SWT is enabled.  |

### 33.5.2.2 SWT Interrupt Register (SWT\_IR)

The SWT Interrupt Register (SWT\_IR) contains the time-out interrupt flag.

Address: Base + 0x0004 Access: User read/write

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| R     | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W     |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|       |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |     |
|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|
|       | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31  |
| R     | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | TIF |
| W     |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | w1c |
| Reset | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   |

**Figure 33-2. SWT Interrupt Register (SWT\_IR)**

Table 33-3. SWT\_IR field descriptions

| Field | Description  |
|-------|--|
| TIF   | Time-out Interrupt Flag. The flag and interrupt are cleared by writing a 1 to this bit. Writing a 0 has no effect.<br>0 No interrupt request.<br>1 Interrupt request due to an initial time-out. |

### 33.5.2.3 SWT Time-Out Register (SWT\_TO)

The SWT Time-Out (SWT\_TO) register contains the 32-bit time-out period. The reset value for this register is device specific. If either the SWT\_CR[HLK] or the SWT\_CR[SLK] bit is set, this register is read-only.

Address: Base + 0x0008

Access: User read/write

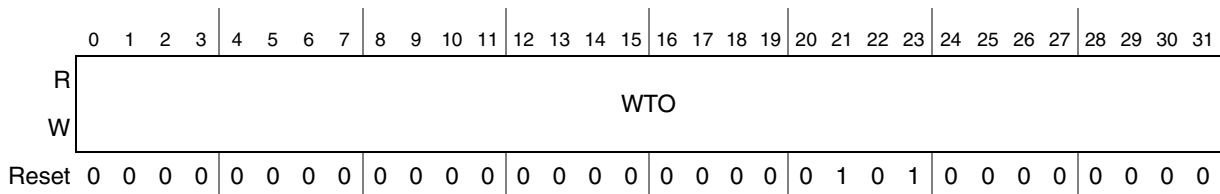


Figure 33-3. SWT Time-Out Register (SWT\_TO)

The default counter value (SWT\_TO\_RST) is 1280 (0x0000\_0500 hexadecimal), which corresponds to around 10 ms with a 128 kHz clock.

Table 33-4. SWT\_TO Register field descriptions

| Field | Description   |
|-------|---|
| WTO   | Watchdog time-out period in clock cycles. When the service sequence is written or when the SWT is enabled, an internal 32-bit down counter is loaded with the greater of this value or 0x100. |

### 33.5.2.4 SWT Window Register (SWT\_WN)

The SWT Window (SWT\_WN) register contains the 32-bit window start value. This register is cleared on reset. If either the SWT\_CR[HLK] or the SWT\_CR[SLK] bit is set, this register is read-only.

Address: Base + 0x000C

Access: User read/write

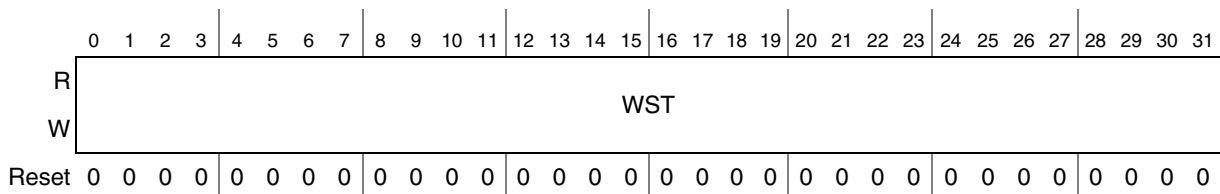


Figure 33-4. SWT Window Register (SWT\_WN)

**Table 33-5. SWT\_WN Register field descriptions**

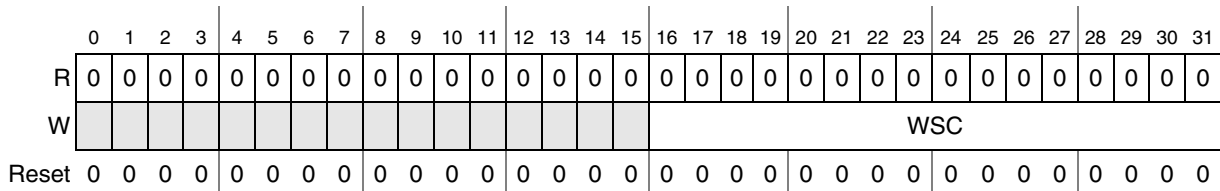
| Field | Description   |
|-------|---|
| WST   | Window start value. When window mode is enabled, the service sequence can only be written when the internal down counter is less than this value. |

### 33.5.2.5 SWT Service Register (SWT\_SR)

The SWT Time-Out (SWT\_SR) service register is the target for service sequence writes used to reset the watchdog timer.

Address: Base + 0x0010

Access: User read/write



**Figure 33-5. SWT Service Register (SWT\_SR)**

**Table 33-6. SWT\_SR field descriptions**

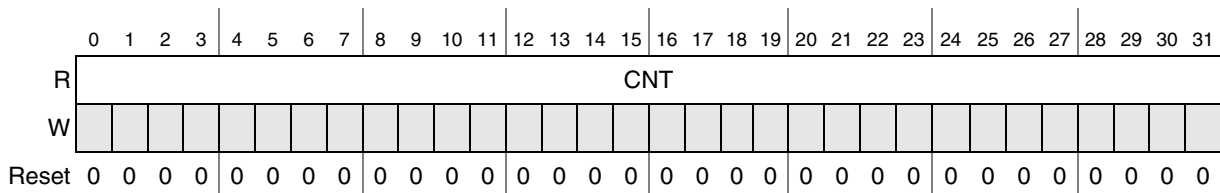
| Field | Description   |
|-------|---|
| WSC   | Watchdog Service Code. This field is used to service the watchdog and to clear the soft lock bit (SWT_CR[SLK]). To service the watchdog, the value 0xA602 followed by 0xB480 is written to the WSC field. To clear the soft lock bit (SWT_CR.SLK), the value 0xC520 followed by 0xD928 is written to the WSC field. |

### 33.5.2.6 SWT Counter Output Register (SWT\_CO)

The SWT Counter Output (SWT\_CO) register is a read only register that shows the value of the internal down counter when the SWT is disabled.

Address: Base + 0x0014

Access: User read-only



**Figure 33-6. SWT Counter Output Register (SWT\_CO)**

Table 33-7. SWT\_CO field descriptions

| Field | Description  |
|-------|--|
| CNT   | Watchdog Count. When the watchdog is disabled (SWT_CR[WEN] = 0), this field shows the value of the internal down counter. When the watchdog is enabled, the value of this field is 0x0000_0000. Values in this field can lag behind the internal counter value for as many as 6 system clock cycles plus 8 counter clock cycles. Therefore, the value read from this field immediately after disabling the watchdog may be higher than the actual value of the internal counter. |

### 33.5.2.7 SWT Service Key Register (SWT\_SK)

The SWT Service Key (SWT\_SK) register holds the previous (or initial) service key value. If either the SWT\_CR[HCLK] or the SWT\_CR[SLK] bit is set, this register is read-only.

Address: Base + 0x0018

Access: User read/write

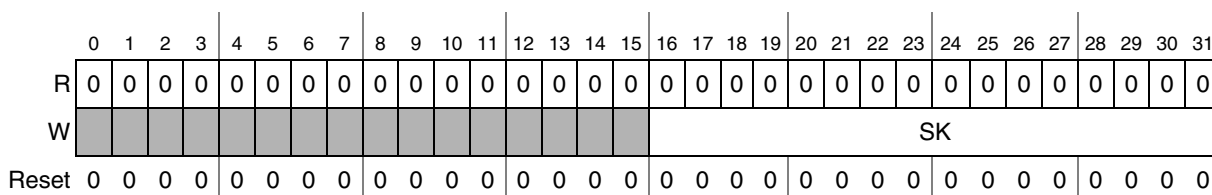


Figure 7. SWT Service Register (SWT\_SK)

Table 8. SWT\_SK field descriptions

| Field | Description   |
|-------|---|
| SK    | Service Key. This field is the previous (or initial) service key value used in keyed service mode. If SWT_CR[KEY] is set, the next key value to be written to the SWT_SK is $(17 \times SK + 3) \bmod 2^{16}$ . |

## 33.6 Functional description

The SWT is a 32-bit timer designed to enable the system to recover in situations such as software getting trapped in a loop or if a bus transaction fails to terminate. It includes a control register (SWT\_CR), an interrupt register (SWT\_IR), time-out register (SWT\_TO), a window register (SWT\_WN), a service register (SWT\_SR), a counter output register (SWT\_CO), and a service key register (SWT\_SK).

The SWT\_CR includes bits to enable the timer, set configuration options and lock configuration of the module. The watchdog is enabled by setting the SWT\_CR[WEN] bit. The reset value of the SWT\_CR[WEN] bit is device specific 1 (enabled). This last bit is cleared when exiting ME RESET mode in case flash user option bit 31 (WATCHDOG\_EN) is 0. If the reset value of this bit is 1, the watchdog starts operation automatically after reset is released. Some devices can be configured to clear this bit automatically during the boot process.

The SWT\_TO register holds the watchdog time-out period in clock cycles unless the value is less than 0x100, in which case the time-out period is set to 0x100. This time-out period is loaded into an internal 32-bit down counter when the SWT is enabled and each time a valid service sequence is written. The

SWT\_CR.CSL bit selects which clock (system or oscillator) is used to drive the down counter. The reset value of the SWT\_TO register is device-specific as described previously.

The configuration of the SWT can be locked through use of either a soft lock or a hard lock. In either case, when locked the SWT\_CR, SWT\_TO, SWT\_WN, and SWT\_SK registers are read only. The hard lock is enabled by setting the SWT\_CR[HCLK] bit, which can only be cleared by a reset. The soft lock is enabled by setting the SWT\_CR[SLK] bit and is cleared by writing the unlock sequence to the service register. The unlock sequence is a write of 0xC520 followed by a write of 0xD928 to the SWT\_SR[WSC] field. There is no timing requirement between the two writes. The unlock sequence logic ignores service sequence writes and recognizes the 0xC520, 0xD928 sequence regardless of previous writes. The unlock sequence can be written at any time and does not require the SWT\_CR[WEN] bit to be set.

When enabled, the SWT requires periodic execution of the watchdog servicing sequence. Writing the proper sequence of values loads the internal down counter with the time-out period. There is no timing requirement between the two writes and the service sequence logic ignores unlock sequence writes. If the SWT\_CR[KEY] bit = 0, the fixed sequence 0xA602, 0xB480 is written to the SWT\_SR[WSC] field to service the watchdog. If the SWT\_CR[KEY] bit = 1, then two pseudorandom keys are written to the SWT\_SR[WSC] field to service the watchdog. The key values are determined by the pseudorandom key generator defined in Equation 33-1.

**Eqn. 33-1**

$$SK_{n+1} = (17 * SK_n + 3) \bmod 2^{16}$$

This algorithm generates a sequence of  $2^{16}$  different key values before repeating. The state of the key generator is held in the SWT\_SK register.

For example, if SWT\_SK[SK] is 0x0100 then the service sequence keys are 0x1103, 0x2136. In this mode, each time a valid key is written to the SWT\_SR register, the SWT\_SK register is updated. So, after servicing the watchdog by writing 0x1103 and then 0x2136 to the SWT\_SR[WSC] field, SWT\_SK[SK] is 0x2136 and the next key sequence is 0x3499, 0x7E2C.

Accesses to SWT registers occur with no peripheral bus wait states. (The peripheral bus bridge may add one or more system wait states.) However, due to synchronization logic in the SWT design, recognition of the service sequence or configuration changes may require as long as 3 system plus 7 counter clock cycles.

If window mode is enabled (SWT\_CR[WND] bit is set), the service sequence must be performed in the last part of the time-out period defined by the window register. The window is open when the down counter is less than the value in the SWT\_WN register. Outside of this window, service sequence writes are invalid accesses and generate a bus error or reset depending on the value of the SWT\_CR[RIA] bit. For example, if the SWT\_TO register is set to 5000 and the SWT\_WN register is set to 1000, then the service sequence must be performed in the last 20% of the time-out period. There is a short lag in the time it takes for the window to open due to synchronization logic in the watchdog design. This delay could be as long as 3 system plus 4 counter clock cycles.

The interrupt then reset bit (SWT\_CR[ITR]) controls the action taken when a time-out occurs. If the SWT\_CR[ITR] bit is not set, a reset is generated immediately on a time-out. If the SWT\_CR[ITR] bit is set, an initial time-out causes the SWT to generate an interrupt and load the down counter with the time-out



period. If the service sequence is not written before the second consecutive time-out, the SWT generates a system reset. The interrupt is indicated by the time-out interrupt flag (SWT\_IR[TIF]). The interrupt request is cleared by writing a one to the SWT\_IR[TIF] bit.

The SWT\_CO register shows the value of the down counter when the watchdog is disabled. When the watchdog is enabled this register is cleared. The value shown in this register can lag behind the value in the internal counter for as long as 6 system plus 8 counter clock cycles.

The SWT\_CO can be used during a software self test of the SWT. For example, the SWT can be enabled and not serviced for a fixed period of time less than the time-out value. Then the SWT can be disabled (SWT\_CR[WEN] cleared) and the value of the SWT\_CO read to determine if the internal down counter is working properly.

#### **NOTE**

Watchdog is disabled at the start of BAM execution. In the case of an unexpected issue during BAM execution, the CPU may be stalled and an external reset needs to be generated to recover.

This page is intentionally left blank.

# Chapter 34

## Error Correction Status Module (ECSM)

### 34.1 Introduction

The Error Correction Status Module (ECSM) provides a myriad of miscellaneous control functions for the device including program-visible information about configuration and revision levels, a reset status register, and information on memory errors reported by error-correcting codes.

### 34.2 Overview

The Error Correction Status Module is mapped into the IPS space and supports a number of miscellaneous control functions for the device.

The AIPS is the interface between the Advanced High performance Bus (AHB) interface and on-chip IPS peripherals. IPS peripherals are modules that contain readable/writable control and status registers. The AHB master reads and writes these registers through the AIPS. The AIPS generates module enables, the module address, transfer attributes, byte enables, and write data. These elements then function as inputs to the IPS peripherals.

- IPS — Inter Peripheral Subsystem
- AIPS — interface between the Advanced High performance Bus (AHB) interface and on-chip IPS peripherals
- AHB — Advance High-performance Bus

### 34.3 Features

The ECSM includes these features:

- Program-visible information on the device configuration and revision
- Registers for capturing information on memory errors due to error-correction codes
- Registers to specify the generation of single- and double-bit memory data inversions for test purposes to check ECC protection

### 34.4 Memory map and register description

This section details the programming model for the Error Correction Status Module. This is a 128-byte space mapped to the region serviced by an IPS bus controller.

#### 34.4.1 Memory map

The Error Correction Status Module does not include any logic that provides access control. Rather, this function is supported using the standard access control logic provided by the IPS controller.

[Table 34-1](#) shows the ECSM's memory map.

Table 34-1. ECSM memory map

| Base address: 0xFFF4_0000 |   |                             |
|---------------------------|---|-----------------------------|
| Address offset            | Register  | Location                    |
| 0x00                      | Processor Core Type Register (PCT)                    | <a href="#">on page 925</a> |
| 0x02                      | SoC-Defined Platform Revision Register (REV)          | <a href="#">on page 925</a> |
| 0x04                      | Reserved  |                             |
| 0x08                      | IPS On-Platform Module Configuration Register (IOPMC) | <a href="#">on page 925</a> |
| 0x0C–0x1E                 | Reserved  |                             |
| 0x1F                      | Miscellaneous Interrupt Register (MIR)                | <a href="#">on page 927</a> |
| 0x20–0x23                 | Reserved  |                             |
| 0x24                      | Miscellaneous User-Defined Control Register (MUDCR)   | <a href="#">on page 928</a> |
| 0x28–0x42                 | Reserved  |                             |
| 0x43                      | ECC Configuration Register (ECR)                      | <a href="#">on page 929</a> |
| 0x44–0x46                 | Reserved  |                             |
| 0x47                      | ECC Status Register (ESR)                             | <a href="#">on page 931</a> |
| 0x48–0x49                 | Reserved  |                             |
| 0x4A                      | ECC Error Generation Register (EEGR)                  | <a href="#">on page 932</a> |
| 0x4C–0x4F                 | Reserved  |                             |
| 0x50                      | Platform Flash ECC Address Register (PFEAR)           | <a href="#">on page 935</a> |
| 0x54–0x55                 | Reserved  |                             |
| 0x56                      | Platform Flash ECC Master Number Register (PFEMR)     | <a href="#">on page 936</a> |
| 0x57                      | Platform Flash ECC Attributes Register (PFEAT)        | <a href="#">on page 936</a> |
| 0x58–0x5B                 | Reserved  |                             |
| 0x5C                      | Platform Flash ECC Data Register (PFEDR)              | <a href="#">on page 937</a> |
| 0x60                      | Platform RAM ECC Address Register (PREAR)             | <a href="#">on page 938</a> |
| 0x64                      | Reserved  |                             |
| 0x65                      | Platform RAM ECC Syndrome Register (PRESR)            | <a href="#">on page 938</a> |
| 0x66                      | Platform RAM ECC Master Number Register (PREMR)       | <a href="#">on page 940</a> |
| 0x67                      | Platform RAM ECC Attributes Register (PREAT)          | <a href="#">on page 941</a> |
| 0x68–0x6B                 | Reserved  |                             |
| 0x6C                      | Platform RAM ECC Data Register (PREDR)                | <a href="#">on page 942</a> |

### 34.4.2 Register description

Attempted accesses to reserved addresses result in an error termination, while attempted writes to read-only registers are ignored and do not terminate with an error. Unless noted otherwise, writes to the

programming model must match the size of the register, e.g., an n-bit register only supports n-bit writes, etc. Attempted writes of a different size than the register width produce an error termination of the bus cycle and no change to the targeted register.

### 34.4.2.1 Processor Core Type Register (PCT)

The PCT is a 16-bit read-only register specifying the architecture of the processor core in the device. The state of this register is defined by a module input signal; it can only be read from the IPS programming model. Any attempted write is ignored.

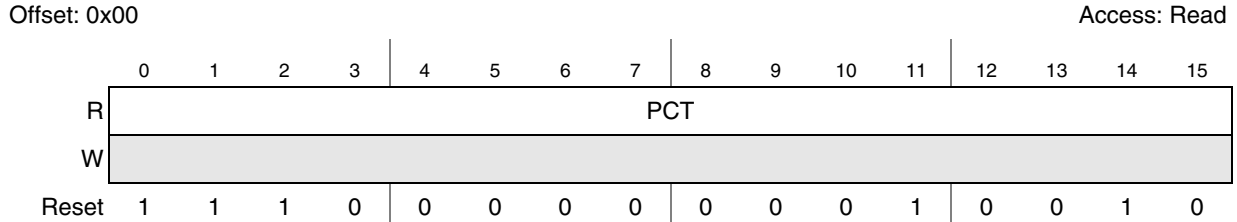


Figure 34-1. Processor Core Type Register (PCT)

Table 34-2. PCT field descriptions

| Field | Description         |
|-------|---------------------|
| PCT   | Processor Core Type |

### 34.4.2.2 SoC-Defined Platform Revision Register (REV)

The REV is a 16-bit read-only register specifying a revision number. The state of this register is defined by an input signal; it can only be read from the IPS programming model. Any attempted write is ignored.

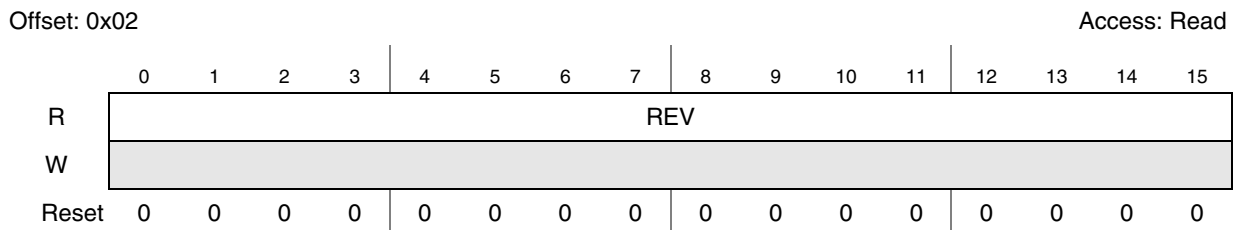


Figure 34-2. SoC-Defined Platform Revision Register (REV)

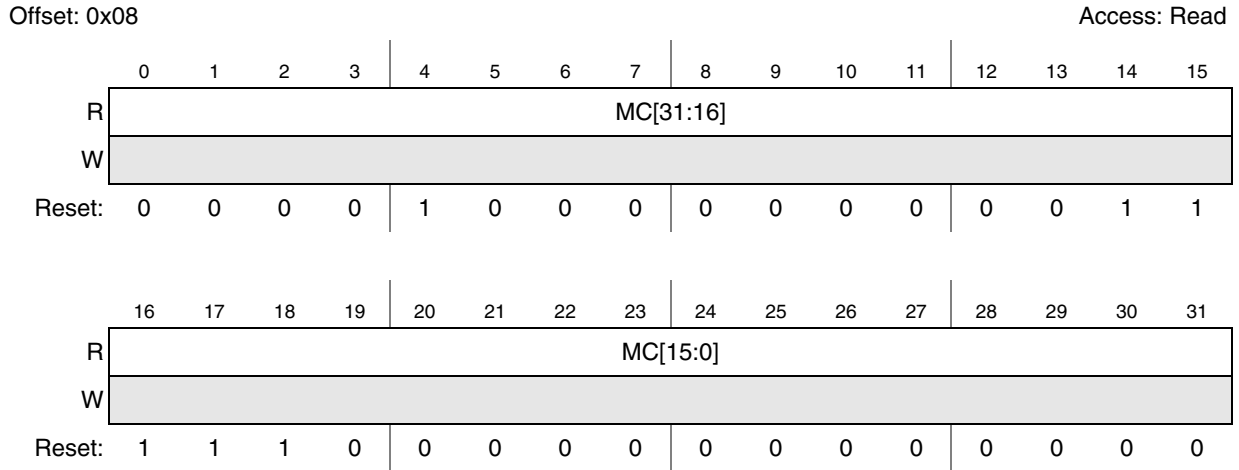
Table 34-3. REV field descriptions

| Field | Description  |
|-------|--|
| REV   | <b>Revision</b><br>The REV field is specified by an input signal to define a software-visible revision number. |

### 34.4.2.3 IPS On-Platform Module Configuration Register (IOPMC)

The IOPMC is a 32-bit read-only register identifying the presence/absence of the 32 low-order IPS peripheral modules connected to the primary IPI slave bus controller. The state of this register is defined

by a module input signal; it can only be read from the IPS programming model. Any attempted write is ignored.



**Figure 34-3. IPS On-Platform Module Configuration Register (IOPMC)**

**Table 34-4. IOPMC field descriptions**

| Field | Description  |
|-------|--|
| MC    | <b>IPS Module Configuration</b><br>MC[n] = 0 if an IPS module connection to decoded slot n is absent<br>MC[n] = 1 if an IPS module connection to decoded slot n is present |

### 34.4.2.4 Miscellaneous Interrupt Register (MIR)

All interrupt requests associated with ECSM are collected in the MIR. This includes the processor core system bus fault interrupt.

During the appropriate interrupt service routine handling these requests, the interrupt source contained in the MIR must be explicitly cleared. See [Figure 34-4](#) and [Table 34-5](#).

|              |       |       |       |                 |   |   |   |   |
|--------------|-------|-------|-------|-----------------|---|---|---|---|
| Offset: 0x1F |       |       |       | Access: Special |   |   |   |   |
|              | 0     | 1     | 2     | 3               | 4 | 5 | 6 | 7 |
| R            | FB0AI | FB0SI | FB1AI | FB1SI           | 0 | 0 | 0 | 0 |
| W            | 1     | 1     | 1     | 1               |   |   |   |   |
| Reset:       | 0     | 0     | 0     | 0               | 0 | 0 | 0 | 0 |

**Figure 34-4. Miscellaneous Interrupt (MIR) Register**

**Table 34-5. MIR field descriptions**

| Field | Description   |
|-------|---|
| FB0AI | <b>Flash Bank 0 Abort Interrupt</b><br>0 A flash bank 0 abort has not occurred.<br>1 A flash bank 0 abort has occurred. The interrupt request is negated by writing a 1 to this bit. Writing a 0 has no effect. |
| FB0SI | <b>Flash Bank 0 Stall Interrupt</b><br>0 A flash bank 0 stall has not occurred.<br>1 A flash bank 0 stall has occurred. The interrupt request is negated by writing a 1 to this bit. Writing a 0 has no effect. |
| FB1AI | <b>Flash Bank 1 Abort Interrupt</b><br>0 A flash bank 1 abort has not occurred.<br>1 A flash bank 1 abort has occurred. The interrupt request is negated by writing a 1 to this bit. Writing a 0 has no effect. |
| FB1SI | <b>Flash Bank 1 Stall Interrupt</b><br>0 A flash bank 1 stall has not occurred.<br>1 A flash bank 1 stall has occurred. The interrupt request is negated by writing a 1 to this bit. Writing a 0 has no effect. |

### 34.4.2.5 Miscellaneous User-Defined Control Register (MUDCR)

The MUDCR provides a program-visible register for user-defined control functions. It typically is used as configuration control for miscellaneous SoC-level modules. The contents of this register is simply output from the ECSM to other modules where the user-defined control functions are implemented.

Offset: 0x24

Access: Read/write

|        |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
|--------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|
|        | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| R      | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |
| W      |   |   |   |   |   |   |   |   |   |   |    |    |    |    |    |    |
| Reset: | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0  | 0  | 0  | 0  | 0  | 0  |

|        |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|--------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|        | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| R      | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |
| W      |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| Reset: | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |

Figure 34-5. Miscellaneous User-Defined Control (MUDCR) Register

Table 34-6. MUDCR field descriptions

| Field     | Description   |
|-----------|---|
| MUDCR[31] | <p><b>XBAR force_round_robin bit</b></p> <p>This bit is used to drive the force_round_robin bit of the XBAR. This will force the slaves into round robin mode of arbitration rather than fixed mode (unless a master is using priority elevation, which forces the design back into fixed mode regardless of this bit). By setting the hardware definition to ENABLE_ROUND_ROBIN_RESET, this bit will reset to 1.</p> <p>1 XBAR is in round robin mode<br/>0 XBAR is in fixed priority mode</p> |

### 34.4.2.6 ECC registers

For designs including error-correcting code (ECC) implementations to improve the quality and reliability of memories, there are a number of program-visible registers for the sole purpose of reporting and logging of memory failures. These registers include:

- ECC Configuration Register (ECR)
- ECC Status Register (ESR)
- ECC Error Generation Register (EEGR)
- Platform Flash ECC Address Register (PFEAR)
- Platform Flash ECC Master Number Register (PFEMR)
- Platform Flash ECC Attributes Register (PFEAT)
- Platform Flash ECC Data Register (PFEDR)



- Platform RAM ECC Address Register (PREAR)
- Platform RAM ECC Syndrome Register (PRESR)
- Platform RAM ECC Master Number Register (PREMR)
- Platform RAM ECC Attributes Register (PREAT)
- Platform RAM ECC Data Register (PREDR)

The details on the ECC registers are provided in the subsequent sections.

#### 34.4.2.6.1 ECC Configuration Register (ECR)

The ECC Configuration Register is an 8-bit control register for specifying which types of memory errors are reported. In all systems with ECC, the occurrence of a non-correctable error causes the current access to be terminated with an error condition. In many cases, this error termination is reported directly by the initiating bus master. However, there are certain situations where the occurrence of this type of non-correctable error is not reported by the master. Examples include speculative instruction fetches that are discarded due to a change-of-flow operation, and buffered operand writes. The ECC reporting logic in the ECSM provides an optional error interrupt mechanism to signal all non-correctable memory errors. In addition to the interrupt generation, the ECSM captures specific information (memory address, attributes and data, bus master number, etc.), which may be useful for subsequent failure analysis.

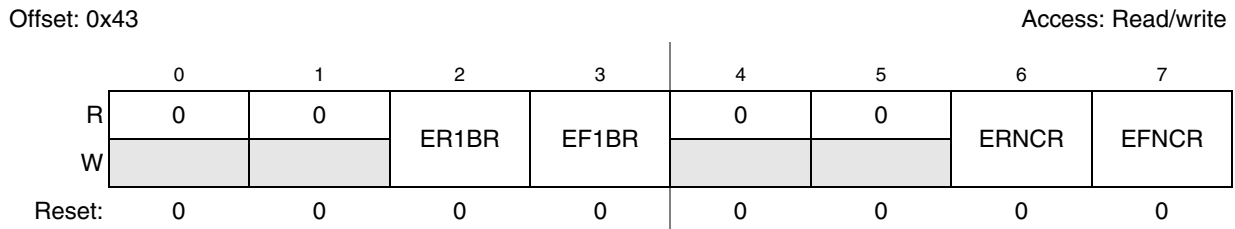


Figure 34-6. ECC Configuration (ECR) Register

Table 34-7. ECR field descriptions

| Field | Description   |
|-------|---|
| ER1BR | <p><b>Enable SRAM 1-bit Reporting</b></p> <p>This bit can only be set if the SoC-configurable input enable signal is asserted. The occurrence of a single-bit SRAM correction generates a ECSM ECC interrupt request as signaled by the assertion of ESR[R1BC]. The address, attributes, and data are also captured in the PREAR, PRESR, PREMR, PREAT, and PREDR registers.</p> <p>0 Reporting of single-bit SRAM corrections is disabled.<br/>1 Reporting of single-bit SRAM corrections is enabled.</p> |
| EF1BR | <p><b>Enable Flash 1-bit Reporting</b></p> <p>This bit can only be set if the SoC-configurable input enable signal is asserted. The occurrence of a single-bit flash correction generates a ECSM ECC interrupt request as signaled by the assertion of ESR[F1BC]. The address, attributes, and data are also captured in the PFEAR, PFEMR, PFEAT, and PFEDR registers.</p> <p>0 Reporting of single-bit flash corrections is disabled.<br/>1 Reporting of single-bit flash corrections is enabled.</p>    |

Table 34-7. ECR field descriptions (continued)

| Field | Description  |
|-------|--|
| ERNCR | <p><b>Enable SRAM Non-Correctable Reporting</b></p> <p>The occurrence of a non-correctable multi-bit SRAM error generates a ECSM ECC interrupt request as signaled by the assertion of ESR[RNCE]. The faulting address, attributes, and data are also captured in the PREAR, PRESR, PREMR, PREAT, and PREDR registers.</p> <p>0 Reporting of non-correctable SRAM errors is disabled.<br/>1 Reporting of non-correctable SRAM errors is enabled.</p> |
| EFNCR | <p><b>Enable Flash Non-Correctable Reporting</b></p> <p>The occurrence of a non-correctable multi-bit flash error generates a ECSM ECC interrupt request as signaled by the assertion of ESR[FNCE]. The faulting address, attributes, and data are also captured in the PFEAR, PFEMR, PFEAT, and PFEDR registers.</p> <p>0 Reporting of non-correctable flash errors is disabled.<br/>1 Reporting of non-correctable flash errors is enabled.</p>    |

### 34.4.2.6.2 ECC Status Register (ESR)

The ECC Status Register is an 8-bit control register for signaling which types of properly enabled ECC events have been detected. The ESR signals the last, properly enabled memory event to be detected. ECC interrupt generation is separated into single-bit error detection/correction, uncorrectable error detection and the combination of the two as defined by the following boolean equations:

ECSM\_ECC1BIT\_IRQ

= ECR[ER1BR] & ESR[R1BC]// ram, 1-bit correction  
 | ECR[EF1BR] & ESR[F1BC]// flash, 1-bit correction

ECSM\_ECCRNCR\_IRQ

= ECR[ERNCR] & ESR[RNCE]// ram, noncorrectable error

ECSM\_ECCFNCR\_IRQ

= ECR[EFNCR] & ESR[FNCE]// flash, noncorrectable error

ECSM\_ECC2BIT\_IRQ

= ECSM\_ECCRNCR\_IRQ// ram, noncorrectable error  
 | ECSM\_ECCFNCR\_IRQ// flash, noncorrectable error

ECSM\_ECC\_IRQ

= ECSM\_ECC1BIT\_IRQ // 1-bit correction  
 | ECSM\_ECC2BIT\_IRQ// noncorrectable error

where the combination of a properly enabled category in the ECR and the detection of the corresponding condition in the ESR produces the interrupt request.

The ECSM allows a maximum of one bit of the ESR to be asserted at any given time. This preserves the association between the ESR and the corresponding address and attribute registers, which are loaded on each occurrence of an properly enabled ECC event. If there is a pending ECC interrupt and another properly enabled ECC event occurs, the ECSM hardware automatically handles the ESR reporting, clearing the previous data and loading the new state, thus guaranteeing that only a single flag is asserted.

To maintain the coherent software view of the reported event, the following sequence in the ECSM error interrupt service routine is suggested:

1. Read the ESR and save it.
2. Read and save all the address and attribute reporting registers.
3. Re-read the ESR and verify the current contents matches the original contents. If the two values are different, go back to step 1 and repeat.
4. When the values are identical, write a 1 to the asserted ESR flag to negate the interrupt request.

Offset: 0x47

Access: Read/write

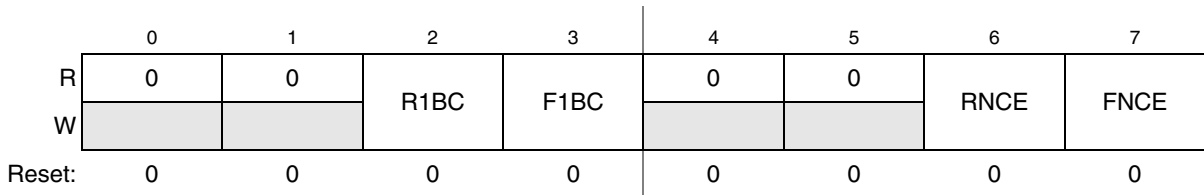


Figure 34-7. ECC Status Register (ESR)

Table 34-8. ESR field descriptions

| Field | Description  |
|-------|--|
| R1BC  | <p><b>SRAM 1-bit Correction</b></p> <p>This bit can only be set if ECR[EPR1BR] is asserted. The occurrence of a properly enabled single-bit SRAM correction generates a ECSM ECC interrupt request. The address, attributes, and data are also captured in the PREAR, PRESR, PREMR, PREAT, and PREDR registers. To clear this interrupt flag, write a 1 to this bit. Writing a 0 has no effect.</p> <p>0 No reportable single-bit SRAM correction has been detected.<br/>1 A reportable single-bit SRAM correction has been detected.</p>                          |
| F1BC  | <p><b>Flash Memory 1-bit Correction</b></p> <p>This bit can only be set if ECR[EPF1BR] is asserted. The occurrence of a properly enabled single-bit flash memory correction generates a ECSM ECC interrupt request. The address, attributes, and data are also captured in the PFEAR, PFEMR, PFEAT, and PFEDR registers. To clear this interrupt flag, write a 1 to this bit. Writing a 0 has no effect.</p> <p>0 No reportable single-bit flash memory correction has been detected.<br/>1 A reportable single-bit flash memory correction has been detected.</p> |
| RNCE  | <p><b>SRAM Non-Correctable Error</b></p> <p>The occurrence of a properly enabled non-correctable SRAM error generates a ECSM ECC interrupt request. The faulting address, attributes, and data are also captured in the PREAR, PRESR, PREMR, PREAT, and PREDR registers. To clear this interrupt flag, write a 1 to this bit. Writing a 0 has no effect.</p> <p>0 No reportable non-correctable SRAM error has been detected.<br/>1 A reportable non-correctable SRAM error has been detected.</p>   |
| FNCE  | <p><b>Flash Memory Non-Correctable Error</b></p> <p>The occurrence of a properly enabled non-correctable flash memory error generates a ECSM ECC interrupt request. The faulting address, attributes, and data are also captured in the PFEAR, PFEMR, PFEAT, and PFEDR registers. To clear this interrupt flag, write a 1 to this bit. Writing a 0 has no effect.</p> <p>0 No reportable non-correctable flash memory error has been detected.<br/>1 A reportable non-correctable flash memory error has been detected.</p>  |

In the event that multiple status flags are signaled simultaneously, ECSM records the event with the R1BC as highest priority, then F1BC, then RNCE, and finally FNCE.

### 34.4.2.6.3 ECC Error Generation Register (EEGR)

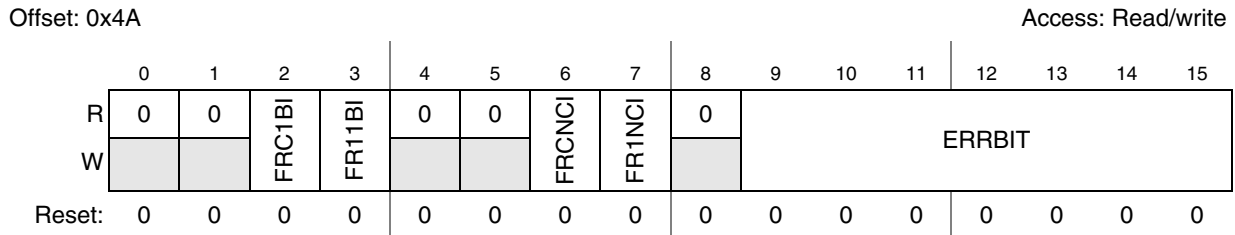
The ECC Error Generation Register is a 16-bit control register used to force the generation of single- and double-bit data inversions in the memories with ECC, most notably the SRAM. This capability is provided for two purposes:

- It provides a software-controlled mechanism for injecting errors into the memories during data writes to verify the integrity of the ECC logic.

- It provides a mechanism to allow testing of the software service routines associated with memory error logging.

It should be noted that while the EEGR is associated with the SRAM, similar capabilities exist for the flash, that is, the ability to program the non-volatile memory with single- or double-bit errors is supported for the same two reasons previously identified.

For both types of memories (SRAM and flash), the intent is to generate errors during data write cycles, such that subsequent reads of the corrupted address locations generate ECC events, either single-bit corrections or double-bit non-correctable errors that are terminated with an error response.



**Figure 34-8. ECC Error Generation Register (EEGR)**

**Table 34-9. EEGR field descriptions**

| Field  | Description   |
|--------|---|
| FRC1BI | <p><b>Force SRAM Continuous 1-bit Data Inversions</b></p> <p>The assertion of this bit forces the SRAM controller to create 1-bit data inversions, as defined by the bit position specified in ERRBIT[6:0], continuously on every write operation.</p> <p>The normal ECC generation takes place in the SRAM controller, but then the polarity of the bit position defined by ERRBIT is inverted to introduce a 1-bit ECC event in the SRAM.</p> <p>After this bit has been enabled to generate another continuous 1-bit data inversion, it must be cleared before being set again to properly re-enable the error generation logic.</p> <p>This bit can only be set if the same SoC configurable input enable signal (as that used to enable single-bit correction reporting) is asserted.</p> <p>0 No SRAM continuous 1-bit data inversions are generated.<br/>1 1-bit data inversions in the SRAM are continuously generated.</p> |
| FR11BI | <p><b>Force SRAM One 1-bit Data Inversion</b></p> <p>The assertion of this bit forces the SRAM controller to create one 1-bit data inversion, as defined by the bit position specified in ERRBIT[6:0], on the first write operation after this bit is set.</p> <p>The normal ECC generation takes place in the SRAM controller, but then the polarity of the bit position defined by ERRBIT is inverted to introduce a 1-bit ECC event in the SRAM.</p> <p>After this bit has been enabled to generate a single 1-bit data inversion, it must be cleared before being set again to properly re-enable the error generation logic.</p> <p>This bit can only be set if the same SoC configurable input enable signal (as that used to enable single-bit correction reporting) is asserted.</p> <p>0 No SRAM single 1-bit data inversion is generated.<br/>1 One 1-bit data inversion in the SRAM is generated.</p>                    |

Table 34-9. EEGR field descriptions (continued)

| Field  | Description   |
|--------|---|
| FRCNCI | <p><b>Force SRAM Continuous Non-correctable Data Inversions</b></p> <p>The assertion of this bit forces the SRAM controller to create 2-bit data inversions, as defined by the bit position specified in ERRBIT[6:0] and the overall odd parity bit, continuously on every write operation.</p> <p>After this bit has been enabled to generate another continuous non-correctable data inversion, it must be cleared before being set again to properly re-enable the error generation logic.</p> <p>The normal ECC generation takes place in the SRAM controller, but then the polarity of the bit position defined by ERRBIT and the overall odd parity bit are inverted to introduce a 2-bit ECC error in the SRAM.</p> <p>0 No SRAM continuous 2-bit data inversions are generated.<br/>1 2-bit data inversions in the SRAM are continuously generated.</p>   |
| FR1NCI | <p><b>Force SRAM One Non-correctable Data Inversions</b></p> <p>The assertion of this bit forces the SRAM controller to create one 2-bit data inversion, as defined by the bit position specified in ERRBIT[6:0] and the overall odd parity bit, on the first write operation after this bit is set.</p> <p>The normal ECC generation takes place in the SRAM controller, but then the polarity of the bit position defined by ERRBIT and the overall odd parity bit are inverted to introduce a 2-bit ECC error in the SRAM.</p> <p>After this bit has been enabled to generate a single 2-bit error, it must be cleared before being set again to properly re-enable the error generation logic.</p> <p>0 No SRAM single 2-bit data inversions are generated.<br/>1 One 2-bit data inversion in the SRAM is generated.</p>  |
| ERRBIT | <p><b>Error Bit Position</b></p> <p>The vector defines the bit position that is complemented to create the data inversion on the write operation. For the creation of 2-bit data inversions, the bit specified by this field plus the odd parity bit of the ECC code are inverted.</p> <p>The SRAM controller follows a vector bit ordering scheme where LSB = 0. Errors in the ECC syndrome bits can be generated by setting this field to a value greater than the SRAM width. For example, consider a 32-bit SRAM implementation.</p> <p>The 32-bit ECC approach requires 7 code bits for a 32-bit word. For PRAM data width of 32 bits, the actual SRAM (32b data + 7b for ECC) = 39 bits. The following association between the ERRBIT field and the corrupted memory bit is defined:</p> <p>if ERRBIT = 0, then SRAM[0] of the odd bank is inverted<br/>if ERRBIT = 1, then SRAM[1] of the odd bank is inverted<br/>...<br/>if ERRBIT = 31, then SRAM[31] of the odd bank is inverted<br/>if ERRBIT = 64, then ECC Parity[0] of the odd bank is inverted<br/>if ERRBIT = 65, then ECC Parity[1] of the odd bank is inverted<br/>...<br/>if ERRBIT = 70, then ECC Parity[6] of the odd bank is inverted</p> <p>For ERRBIT values of 32 to 63 and greater than 70, no bit position is inverted.</p> |

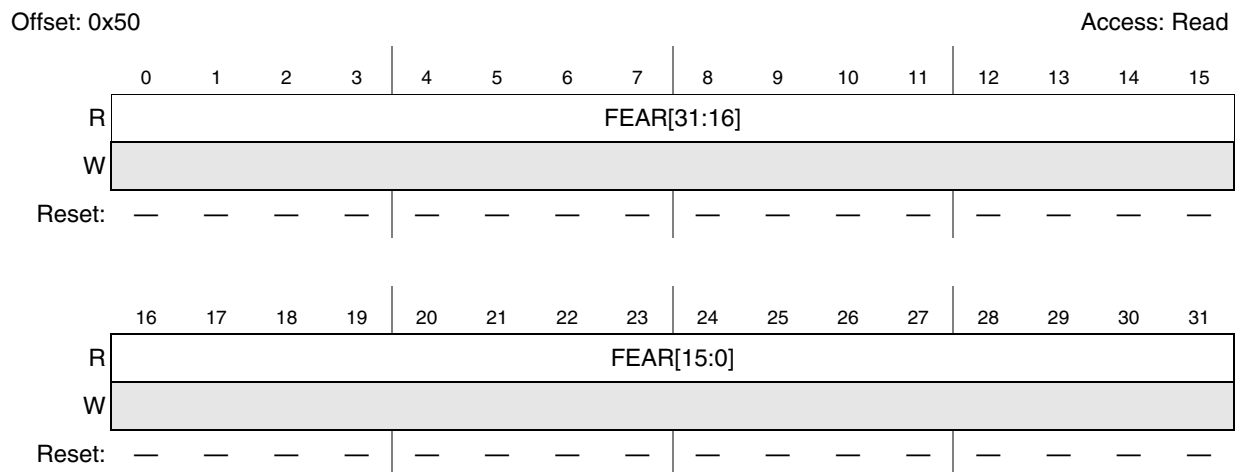
If an attempt to force a non-correctable inversion (by asserting EEGR[FRCNCI] or EEGR[FRC1NCI]) and EEGR[ERRBIT] equals 64, then no data inversion will be generated.

The only allowable values for the 4 control bit enables {FR11BI, FRC1BI, FRCNCI, FR1NCI} are {0,0,0,0}, {1,0,0,0}, {0,1,0,0}, {0,0,1,0}, and {0,0,0,1}. All other values result in undefined behavior.

#### 34.4.2.6.4 Platform Flash ECC Address Register (PFEAR)

The PFEAR is a 32-bit register for capturing the address of the last, properly enabled ECC event in the flash memory. Depending on the state of the ECC Configuration Register, an ECC event in the flash causes the address, attributes, and data associated with the access to be loaded into the PFEAR, PFEMR, PFEAT, and PFEDR registers, and the appropriate flag (F1BC or FNCE) in the ECC Status Register to be asserted.

This register can only be read from the IPS programming model; any attempted write is ignored.



**Figure 34-9. Platform Flash ECC Address Register (PFEAR)**

**Table 34-10. PFEAR field descriptions**

| Field | Description   |
|-------|---|
| FEAR  | <b>Flash ECC Address Register</b><br>This 32-bit register contains the faulting access address of the last, properly enabled flash ECC event. |

### 34.4.2.6.5 Platform Flash ECC Master Number Register (PFEMR)

The PFEMR is a 4-bit register for capturing the XBAR bus master number of the last, properly enabled ECC event in the flash memory. Depending on the state of the ECC Configuration Register, an ECC event in the flash causes the address, attributes, and data associated with the access to be loaded into the PFEAR, PFEMR, PFEAT, and PFEDR registers, and the appropriate flag (F1BC or FNCE) in the ECC Status Register to be asserted.

This register can only be read from the IPS programming model; any attempted write is ignored.

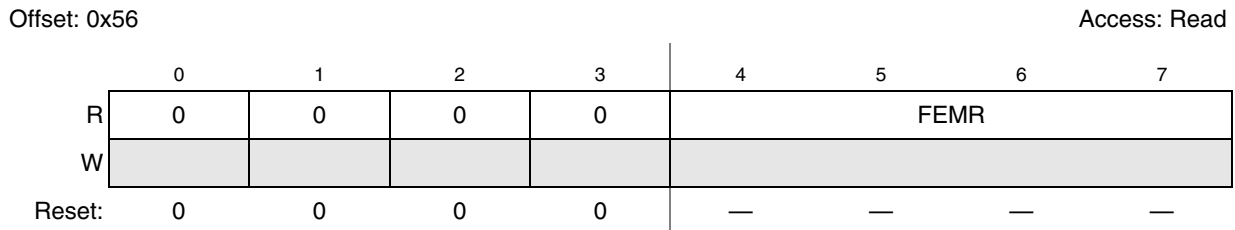


Figure 34-10. Platform Flash ECC Master Number Register (PFEMR)

Table 34-11. PFEMR field descriptions

| Field | Description  |
|-------|--|
| FEMR  | <b>Flash ECC Master Number Register</b><br>This 4-bit register contains the XBAR bus master number of the faulting access of the last, properly enabled flash ECC event. |

### 34.4.2.6.6 Platform Flash ECC Attributes Register (PFEAT)

The PFEAT is an 8-bit register for capturing the XBAR bus master attributes of the last, properly enabled ECC event in the flash memory. Depending on the state of the ECC Configuration Register, an ECC event in the flash causes the address, attributes, and data associated with the access to be loaded into the PFEAR, PFEMR, PFEAT, and PFEDR registers, and the appropriate flag (F1BC or FNCE) in the ECC Status Register to be asserted.

This register can only be read from the IPS programming model; any attempted write is ignored.

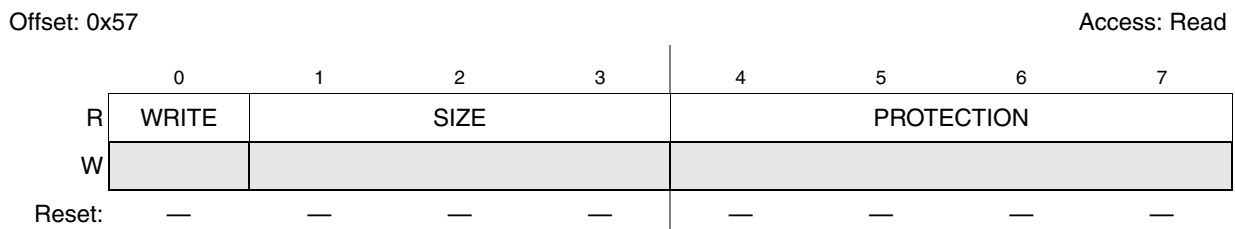


Figure 34-11. Platform Flash ECC Attributes Register (PFEAT)

Table 34-12. PFEAT field descriptions

| Field | Description   |
|-------|---|
| WRITE | <b>AMBA-AHB HWRITE</b><br>0 AMBA-AHB read access<br>1 AMBA-AHB write access |



Table 34-12. PFEAT field descriptions (continued)

| Field      | Description  |
|------------|--|
| SIZE       | <b>AMBA-AHB HSIZE[2:0]</b><br>000 8-bit AMBA-AHB access<br>001 16-bit AMBA-AHB access<br>010 32-bit AMBA-AHB access<br>1xx Reserved  |
| PROTECTION | <b>AMBA-AHB HPROT[3:0]</b><br>Protection[3]: Cacheable 0 = Non-cacheable, 1 = Cacheable<br>Protection[2]: Bufferable 0 = Non-bufferable, 1 = Bufferable<br>Protection[1]: Mode 0 = User mode, 1 = Supervisor mode<br>Protection[0]: Type 0 = I-Fetch, 1 = Data |

#### 34.4.2.6.7 Platform Flash ECC Data Register (PFEDR)

The PFEDR is a 32-bit register for capturing the data associated with the last, properly enabled ECC event in the flash memory. Depending on the state of the ECC Configuration Register, an ECC event in the flash causes the address, attributes, and data associated with the access to be loaded into the PFEAR, PFEMR, PFEAT, and PFEDR registers, and the appropriate flag (F1BC or FNCE) in the ECC Status Register to be asserted.

The data captured on a multi-bit non-correctable ECC error is undefined.

This register can only be read from the IPS programming model; any attempted write is ignored.

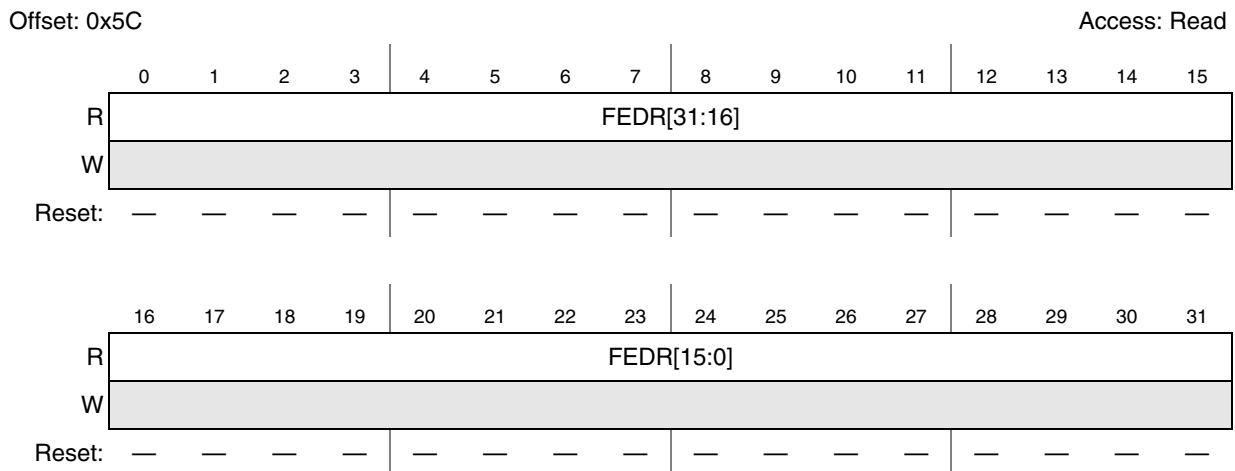


Figure 34-12. Platform Flash ECC Data Register (PFEDR)

Table 34-13. PFEDR field descriptions

| Field | Description  |
|-------|--|
| FEDR  | <b>Flash ECC Data Register</b><br>This 32-bit register contains the data associated with the faulting access of the last, properly enabled flash ECC event. The register contains the data value taken directly from the data bus. |

### 34.4.2.6.8 Platform RAM ECC Address Register (PREAR)

The PREAR is a 32-bit register for capturing the address of the last, properly enabled ECC event in the SRAM memory. Depending on the state of the ECC Configuration Register, an ECC event in the SRAM causes the address, attributes, and data associated with the access to be loaded into the PREAR, PRESR, PREMR, PREAT, and PREDR registers, and the appropriate flag (R1BC or RNCE) in the ECC Status Register to be asserted.

This register can only be read from the IPS programming model; any attempted write is ignored.

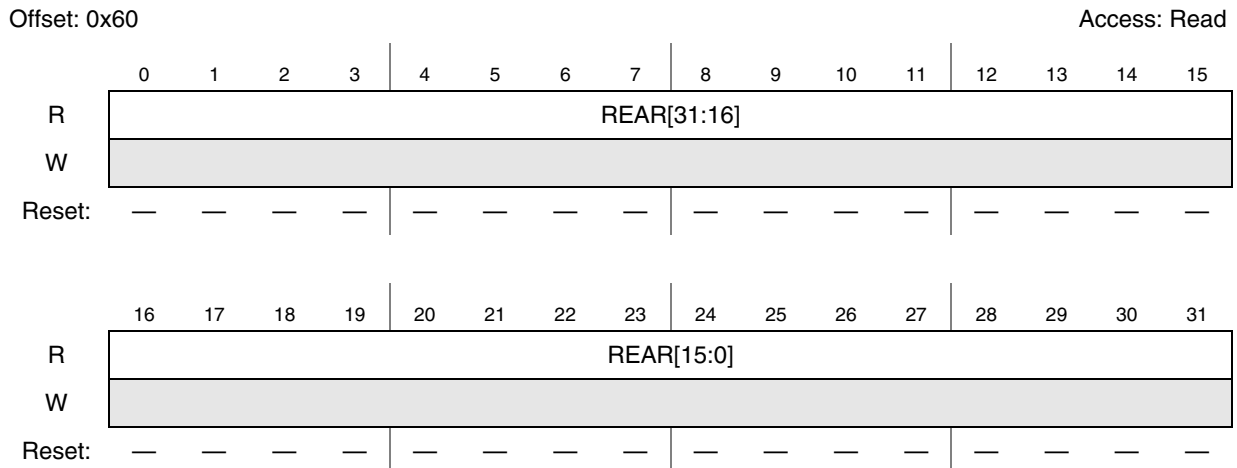


Figure 34-13. Platform RAM ECC Address Register (PREAR)

Table 34-14. PREAR field descriptions

| Field | Description   |
|-------|---|
| REAR  | <b>SRAM ECC Address Register</b><br>This 32-bit register contains the faulting access address of the last, properly enabled SRAM ECC event. |

### 34.4.2.6.9 Platform RAM ECC Syndrome Register (PRESR)

The PRESR is an 8-bit register for capturing the error syndrome of the last, properly enabled ECC event in the SRAM memory. Depending on the state of the ECC Configuration Register, an ECC event in the SRAM causes the address, attributes, and data associated with the access to be loaded into the PREAR, PRESR, PREMR, PREAT, and PREDR registers, and the appropriate flag (R1BC or RNCE) in the ECC Status Register to be asserted.

This register can only be read from the IPS programming model; any attempted write is ignored.

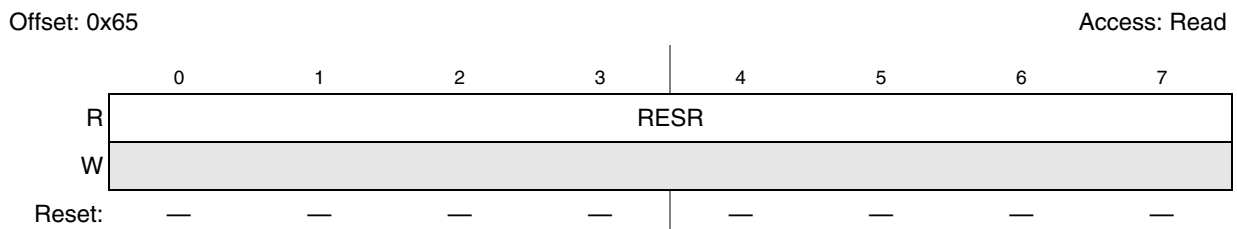


Figure 34-14. Platform RAM ECC Syndrome Register (PRESR)

Table 34-15. PRESR field descriptions

| Field | Description   |
|-------|---|
| RESR  | <p><b>SRAM ECC Syndrome Register</b></p> <p>This 8-bit syndrome field includes 6 bits of Hamming decoded parity plus an odd-parity bit for the entire 39-bit (32-bit data + 7 ECC) code word. The upper 7 bits of the syndrome specify the exact bit position in error for single-bit correctable codewords, and the combination of a non-zero 7-bit syndrome plus overall incorrect parity bit signal a multi-bit, non-correctable error.</p> <p>For correctable single-bit errors, the mapping shown in <a href="#">Table 34-16</a> associates the upper 7 bits of the syndrome with the data bit in error.</p> |

[Table 34-16](#) associates the upper 7 bits of the ECC syndrome with the exact data bit in error for single-bit correctable codewords. This table follows the bit vectoring notation where the LSB = 0. Note that the syndrome value of 0x01 implies no error condition but this value is not readable when the PRESR is read for the no error case.

Table 34-16. RAM syndrome mapping for single-bit correctable errors

| PRESR[RESR] | Data bit in error |
|-------------|-------------------|
| 0x00        | ECC ODD[0]        |
| 0x01        | No error          |
| 0x02        | ECC ODD[1]        |
| 0x04        | ECC ODD[2]        |
| 0x06        | DATA ODD BANK[31] |
| 0x08        | ECC ODD[3]        |
| 0x0a        | DATA ODD BANK[30] |
| 0x0c        | DATA ODD BANK[29] |
| 0x0e        | DATA ODD BANK[28] |
| 0x10        | ECC ODD[4]        |
| 0x12        | DATA ODD BANK[27] |
| 0x14        | DATA ODD BANK[26] |
| 0x16        | DATA ODD BANK[25] |
| 0x18        | DATA ODD BANK[24] |
| 0x1a        | DATA ODD BANK[23] |
| 0x1c        | DATA ODD BANK[22] |
| 0x50        | DATA ODD BANK[21] |
| 0x20        | ECC ODD[5]        |
| 0x22        | DATA ODD BANK[20] |
| 0x24        | DATA ODD BANK[19] |
| 0x26        | DATA ODD BANK[18] |

**Table 34-16. RAM syndrome mapping for single-bit correctable errors (continued)**

| PRESR[RESR]        | Data bit in error  |
|--------------------|--------------------|
| 0x28               | DATA ODD BANK[17]  |
| 0x2a               | DATA ODD BANK[16]  |
| 0x2c               | DATA ODD BANK[15]  |
| 0x58               | DATA ODD BANK[14]  |
| 0x30               | DATA ODD BANK[13]  |
| 0x32               | DATA ODD BANK[12]  |
| 0x34               | DATA ODD BANK[11]  |
| 0x64               | DATA ODD BANK[10]  |
| 0x38               | DATA ODD BANK[9]   |
| 0x62               | DATA ODD BANK[8]   |
| 0x70               | DATA ODD BANK[7]   |
| 0x60               | DATA ODD BANK[6]   |
| 0x40               | ECC ODD[6]         |
| 0x42               | DATA ODD BANK[5]   |
| 0x44               | DATA ODD BANK[4]   |
| 0x46               | DATA ODD BANK[3]   |
| 0x48               | DATA ODD BANK[2]   |
| 0x4a               | DATA ODD BANK[1]   |
| 0x4c               | DATA ODD BANK[0]   |
| 0x03,0x05.....0x4d | Multiple bit error |
| > 0x4d             | Multiple bit error |

#### 34.4.2.6.10 Platform RAM ECC Master Number Register (PREMR)

The PREMR is a 4-bit register for capturing the XBAR bus master number of the last, properly enabled ECC event in the SRAM memory. Depending on the state of the ECC Configuration Register, an ECC event in the SRAM causes the address, attributes, and data associated with the access to be loaded into the PREAR, PRESR, PREMR, PREAT, and PREDR registers, and the appropriate flag (R1BC or RNCE) in the ECC Status Register to be asserted.

See [Chapter 19, Crossbar Switch \(XBAR\)](#), for a listing of XBAR bus master numbers.

This register can only be read from the IPS programming model; any attempted write is ignored.

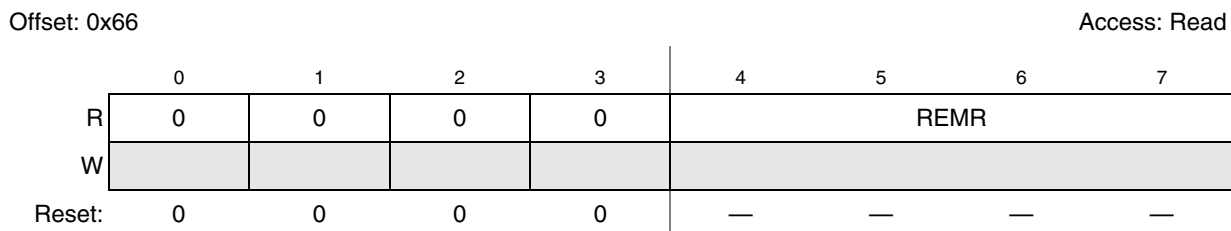


Figure 34-15. Platform RAM ECC Master Number Register (PREMR)

Table 34-17. PREMR field descriptions

| Field | Description  |
|-------|--|
| REMR  | <b>SRAM ECC Master Number Register</b><br>This 4-bit register contains the XBAR bus master number of the faulting access of the last, properly enabled SRAM ECC event.<br>See <a href="#">Chapter 19, Crossbar Switch (XBAR)</a> , for a listing of XBAR bus master numbers. |

#### 34.4.2.6.11 Platform RAM ECC Attributes Register (PREAT)

The PREAT is an 8-bit register for capturing the XBAR bus master attributes of the last, properly enabled ECC event in the SRAM memory. Depending on the state of the ECC Configuration Register, an ECC event in the SRAM causes the address, attributes, and data associated with the access to be loaded into the PREAR, PRESR, PREMR, PREAT, and PREDR registers, and the appropriate flag (RIBC or RNCE) in the ECC Status Register to be asserted.



Figure 34-16. Platform RAM ECC Attributes Register (PREAT)

Table 34-18. PREAT field descriptions

| Field      | Description  |
|------------|--|
| WRITE      | <b>XBAR HWRITE</b><br>0 XBAR read access<br>1 XBAR write access  |
| SIZE       | <b>XBAR HSIZE[2:0]</b><br>000 8-bit XBAR access<br>001 16-bit XBAR access<br>010 32-bit XBAR access<br>1xx Reserved  |
| PROTECTION | <b>XBAR HPROT[3:0]</b><br>Protection[3]: Cacheable 0 = Non-cacheable, 1 = Cacheable<br>Protection[2]: Bufferable 0 = Non-bufferable, 1 = Bufferable<br>Protection[1]: Mode 0 = User mode, 1 = Supervisor mode<br>Protection[0]: Type 0 = I-Fetch, 1 = Data |

### 34.4.2.6.12 Platform RAM ECC Data Register (PREDR)

The PREDR is a 32-bit register for capturing the data associated with the last, properly enabled ECC event in the SRAM memory. Depending on the state of the ECC Configuration Register, an ECC event in the SRAM causes the address, attributes, and data associated with the access to be loaded into the PREAR, PRESR, PREMR, PREAT, and PREDR registers, and the appropriate flag (R1BC or RNCE) in the ECC Status Register to be asserted.

The data captured on a multi-bit non-correctable ECC error is undefined.

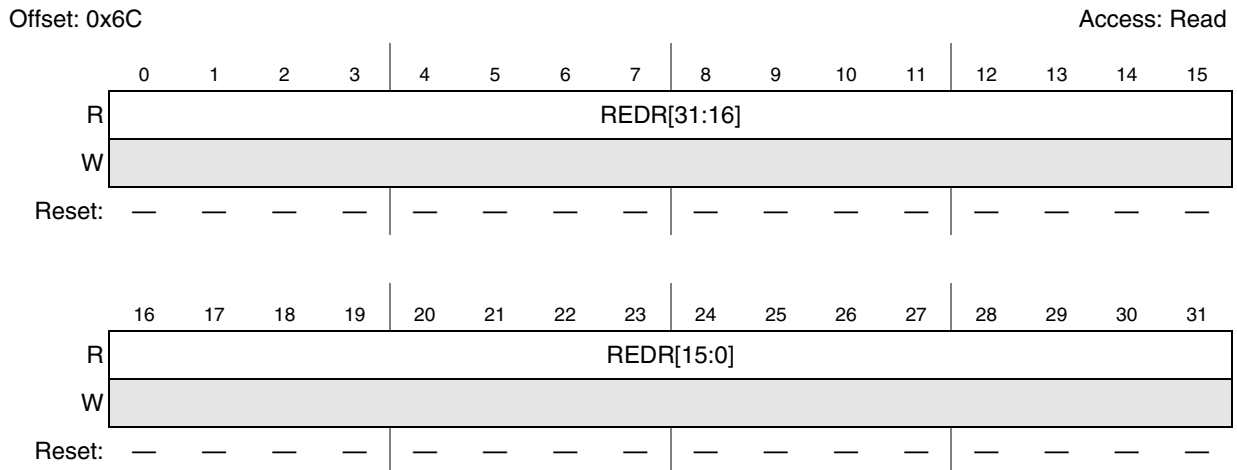


Figure 34-17. Platform RAM ECC Data Register (PREDR)

Table 34-19. PREDR field descriptions


| Field | Description  |
|-------|--|
| REDR  | <b>SRAM ECC Data Register</b><br>This 32-bit register contains the data associated with the faulting access of the last, properly enabled SRAM ECC event. The register contains the data value taken directly from the data bus. |

### 34.4.3 Register protection

Logic exists that restricts accesses to INTC, ECSM, MPU, STM, and SWT to supervisor mode only. Accesses in User mode are not possible.



# ———— Debug ————



This page is intentionally left blank.



# Chapter 35

## IEEE 1149.1 Test Access Port Controller (JTAGC)

### 35.1 Introduction

The JTAG port of the device consists of three inputs and one output. These pins include test data input (TDI), test data output (TDO), test mode select (TMS), and test clock input (TCK). TDI, TDO, TMS, and TCK are compliant with the IEEE 1149.1-2001 standard and are shared with the NDI through the test access port (TAP) interface.

### 35.2 Block diagram

Figure 35-1 is a block diagram of the JTAG Controller (JTAGC) block.

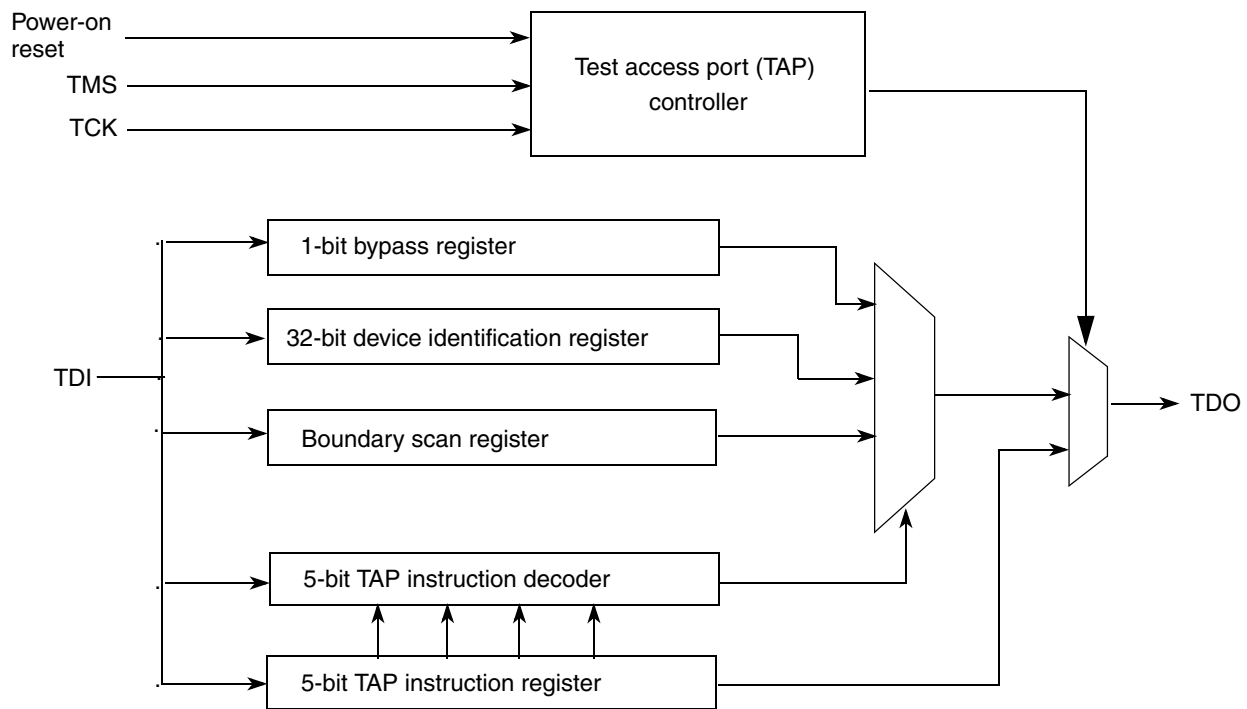


Figure 35-1. JTAG controller block diagram

### 35.3 Overview

The JTAGC provides the means to test chip functionality and connectivity while remaining transparent to system logic when not in TEST mode. Testing is performed via a boundary scan technique, as defined in the IEEE 1149.1-2001 standard. In addition, instructions can be executed that allow the Test Access Port (TAP) to be shared with other modules on the MCU. All data input to and output from the JTAGC is communicated in serial format.

## 35.4 Features

The JTAGC is compliant with the IEEE 1149.1-2001 standard, and supports the following features:

- IEEE 1149.1-2001 Test Access Port (TAP) interface
- Four pins (TDI, TMS, TCK, and TDO)—Refer to [Section 35.6, External signal description](#)
- A 5-bit instruction register that supports several IEEE 1149.1-2001 defined instructions, as well as several public and private MCU specific instructions
- Two test data registers:
  - Bypass register
  - Device identification register
- A TAP controller state machine that controls the operation of the data registers, instruction register, and associated circuitry

## 35.5 Modes of operation

The JTAGC uses a power-on reset indication as its primary reset signals. Several IEEE 1149.1-2001 defined TEST modes are supported, as well as a bypass mode.

### 35.5.1 Reset

The JTAGC is placed in reset when the TAP controller state machine is in the TEST-LOGIC-RESET state. The TEST-LOGIC-RESET state is entered upon the assertion of the power-on reset signal, or through TAP controller state machine transitions controlled by TMS. Asserting power-on reset results in asynchronous entry into the reset state. While in reset, the following actions occur:

- The TAP controller is forced into the test-logic-reset state, thereby disabling the test logic and allowing normal operation of the on-chip system logic to continue unhindered.
- The instruction register is loaded with the IDCODE instruction.

In addition, execution of certain instructions can result in assertion of the internal system reset. These instructions include EXTEST.

### 35.5.2 IEEE 1149.1-2001 defined test modes

The JTAGC supports several IEEE 1149.1-2001 defined TEST modes. The TEST mode is selected by loading the appropriate instruction into the instruction register while the JTAGC is enabled. Supported test instructions include EXTEST, SAMPLE, and SAMPLE/PRELOAD. Each instruction defines the set of data registers that can operate and interact with the on-chip system logic while the instruction is current. Only one test data register path is enabled to shift data between TDI and TDO for each instruction.

The boundary scan register is external to JTAGC but can be accessed by JTAGC TAP through EXTEST, SAMPLE, SAMPLE/PRELOAD instructions. The functionality of each TEST mode is explained in more detail in [Section 35.8.4, JTAGC instructions](#).

### 35.5.2.1 Bypass mode

When no test operation is required, the BYPASS instruction can be loaded to place the JTAGC into bypass mode. While in bypass mode, the single-bit bypass shift register is used to provide a minimum-length serial path to shift data between TDI and TDO.

### 35.5.2.2 TAP sharing mode

There are three selectable auxiliary TAP controllers that share the TAP with the JTAGC. Selectable TAP controllers include the PLATFORM. The instructions required to grant ownership of the TAP to the auxiliary TAP controllers are ACCESS\_AUX\_TAP\_ONCE and ACCESS\_AUX\_TAP\_TCU. Instruction opcodes for each instruction are shown in [Table 35-3](#).

When the access instruction for an auxiliary TAP is loaded, control of the JTAG pins is transferred to the selected TAP controller. Any data input via TDI and TMS is passed to the selected TAP controller, and any TDO output from the selected TAP controller is sent back to the JTAGC to be output on the pins. The JTAGC regains control of the JTAG port during the UPDATE-DR state if the PAUSE-DR state was entered. Auxiliary TAP controllers are held in RUN-TEST/IDLE while they are inactive.

## 35.6 External signal description

The JTAGC consists of four signals that connect to off-chip development tools and allow access to test support functions. The JTAGC signals are outlined in [Table 35-1](#):

**Table 35-1. JTAG signal properties**

| Name | I/O | Function         | Reset State |
|------|-----|------------------|-------------|
| TCK  | I   | Test clock       | Pull Up     |
| TDI  | I   | Test data in     | Pull Up     |
| TDO  | O   | Test data out    | High Z      |
| TMS  | I   | Test mode select | Pull Up     |

The JTAGC pins are shared with GPIO. TDO at reset is a input pad and output direction control from JTAGC. Once TAP enters shift-ir or shift-dr then output direction control from JTAGC, which allows the value to see on pad. It is up to the user to configure them as GPIOs accordingly.

## 35.7 Memory map and register description

This section provides a detailed description of the JTAGC registers accessible through the TAP interface, including data registers and the instruction register. Individual bit-level descriptions and reset states of each register are included. These registers are not memory-mapped and can only be accessed through the TAP.

### 35.7.1 Instruction register

The JTAGC uses a 5-bit instruction register as shown in [Table 35-2](#). The instruction register allows instructions to be loaded into the module to select the test to be performed or the test data register to be

accessed or both. Instructions are shifted in through TDI while the TAP controller is in the Shift-IR state, and latched on the falling edge of TCK in the Update-IR state. The latched instruction value can only be changed in the update-IR and test-logic-reset TAP controller states. Synchronous entry into the test-logic-reset state results in the IDCODE instruction being loaded on the falling edge of TCK. Asynchronous entry into the test-logic-reset state results in asynchronous loading of the IDCODE instruction. During the capture-IR TAP controller state, the instruction shift register is loaded with the value 0b10101, making this value the register’s read value when the TAP controller is sequenced into the Shift-IR state.

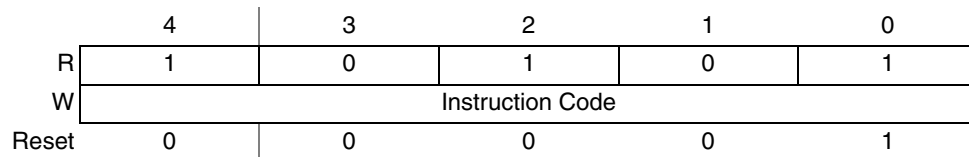


Figure 35-2. 5-bit instruction register

### 35.7.2 Bypass register

The bypass register is a single-bit shift register path selected for serial data transfer between TDI and TDO when the BYPASS, or reserve instructions are active. After entry into the capture-DR state, the single-bit shift register is set to a logic 0. Therefore, the first bit shifted out after selecting the bypass register is always a logic 0.

### 35.7.3 Device identification register

The device identification register, shown in Table 35-3, allows the part revision number, design center, part identification number, and manufacturer identity code to be determined through the TAP. The device identification register is selected for serial data transfer between TDI and TDO when the IDCODE instruction is active. Entry into the capture-DR state while the device identification register is selected loads the IDCODE into the shift register to be shifted out on TDO in the Shift-DR state. No action occurs in the update-DR state.

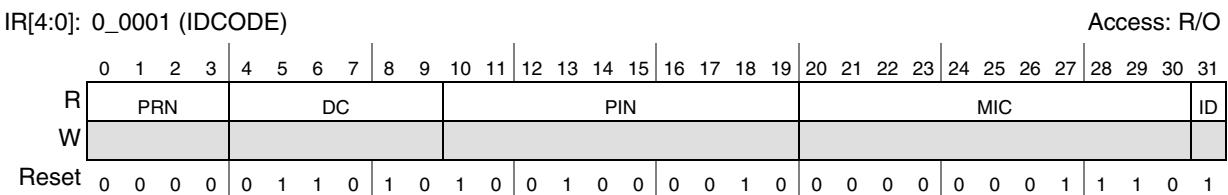


Figure 35-3. Device identification register

Table 35-2. Device identification register field descriptions

| Field      | Description  |
|------------|--|
| 0–3<br>PRN | Part revision number. Contains the revision number of the device. This field changes with each revision of the device or module. |
| 4–9<br>DC  | Design center. For the MPC5606BK this value is 0x1A.   |

**Table 35-2. Device identification register field descriptions**

| Field        | Description  |
|--------------|--|
| 10–19<br>PIN | Part identification number. Contains the part number of the device. For the MPC5606BK, this value is 0x244.                      |
| 20–30<br>MIC | Manufacturer identity code. Contains the reduced Joint Electron Device Engineering Council (JEDEC) ID for Freescale, 0xE         |
| 31<br>ID     | IDCODE register ID. Identifies this register as the device identification register and not the bypass register. Always set to 1. |

### 35.7.4 Boundary scan register

The boundary scan register is connected between TDI and TDO when the EXTEST, SAMPLE or SAMPLE/PRELOAD instructions are active. It is used to capture input pin data, force fixed values on output pins, and select a logic value and direction for bidirectional pins. Each bit of the boundary scan register represents a separate boundary scan register cell, as described in the IEEE 1149.1-2001 standard and discussed in [Section 35.8.5, Boundary scan](#). The size of the boundary scan register is 464 bits.

## 35.8 Functional description

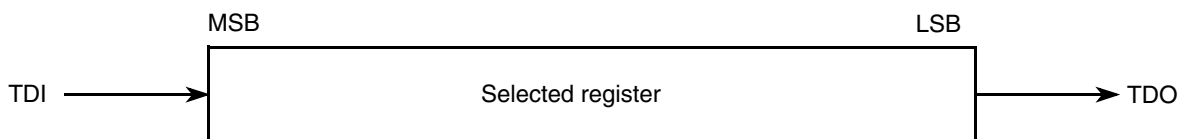
### 35.8.1 JTAGC reset configuration

While in reset, the TAP controller is forced into the test-logic-reset state, thus disabling the test logic and allowing normal operation of the on-chip system logic. In addition, the instruction register is loaded with the IDCODE instruction.

### 35.8.2 IEEE 1149.1-2001 (JTAG) Test Access Port

The JTAGC uses the IEEE 1149.1-2001 TAP for accessing registers. This port can be shared with other TAP controllers on the MCU. For more detail on TAP sharing via JTAGC instructions refer to [Section 35.8.4.2, ACCESS\\_AUX\\_TAP\\_x instructions](#).

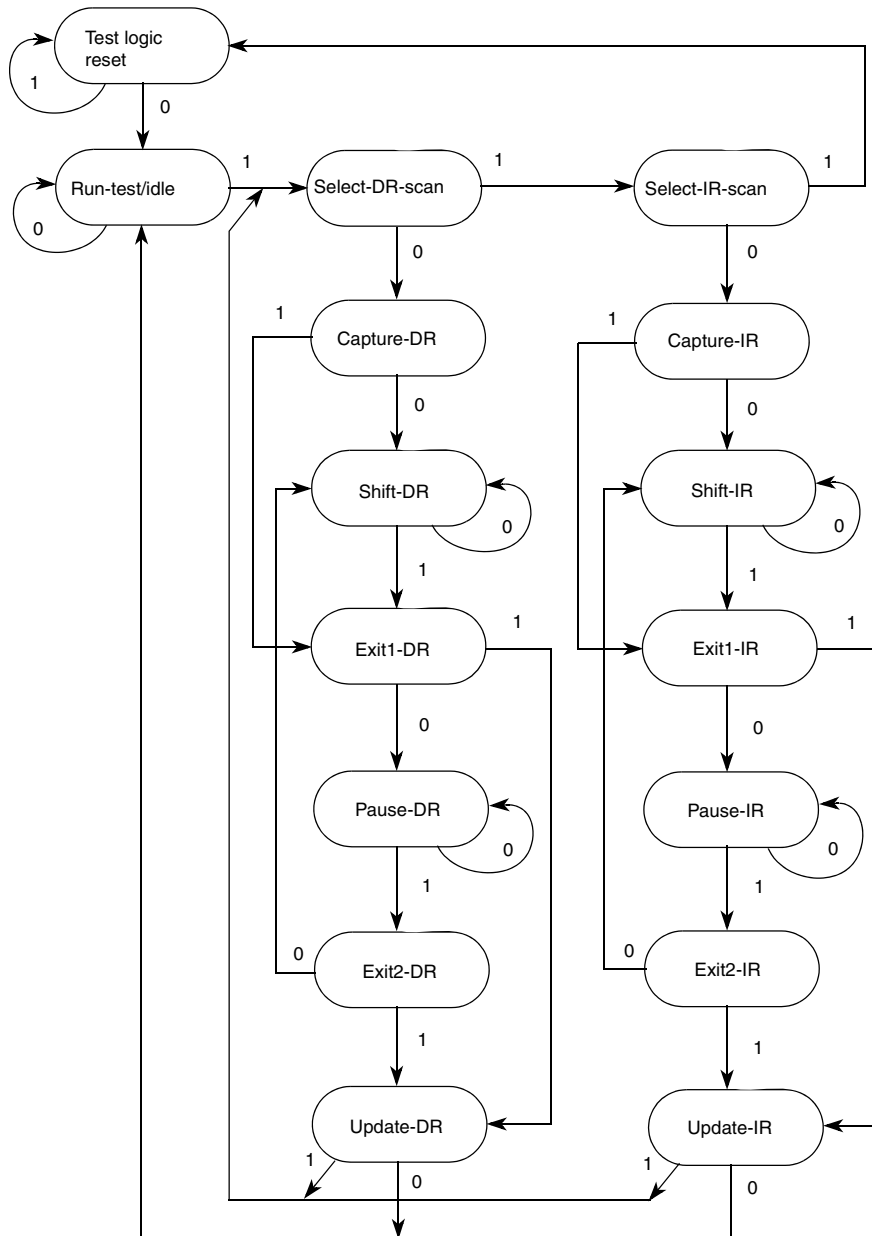
Data is shifted between TDI and TDO through the selected register starting with the least significant bit, as illustrated in [Figure 35-4](#). This applies for the instruction register, test data registers, and the bypass register.

**Figure 35-4. Shifting data through a register**

### 35.8.3 TAP controller state machine

The TAP controller is a synchronous state machine that interprets the sequence of logical values on the TMS pin. Figure 35-5 shows the machine's states. The value shown next to each state is the value of the TMS signal sampled on the rising edge of the TCK signal.

As Figure 35-5 shows, holding TMS at logic 1 while clocking TCK through a sufficient number of rising edges also causes the state machine to enter the test-logic-reset state.



NOTE: The value shown adjacent to each state transition in this figure represents the value of TMS at the time of a rising edge of TCK.

Figure 35-5. IEEE 1149.1-2001 TAP controller finite state machine

### 35.8.3.1 Selecting an IEEE 1149.1-2001 register

Access to the JTAGC data registers is done by loading the instruction register with any of the JTAGC instructions while the JTAGC is enabled. Instructions are shifted in via the select-IR-scan path and loaded in the update-IR state. At this point, all data register access is performed via the select-DR-scan path.

The select-DR-scan path is used to read or write the register data by shifting in the data (LSB first) during the shift-DR state. When reading a register, the register value is loaded into the IEEE 1149.1-2001 shifter during the capture-DR state. When writing a register, the value is loaded from the IEEE 1149.1-2001 shifter to the register during the update-DR state. When reading a register, there is no requirement to shift out the entire register contents. Shifting can be terminated after fetching the required number of bits.

## 35.8.4 JTAGC instructions

This section gives an overview of each instruction, refer to the IEEE 1149.1-2001 standard for more details.

The JTAGC implements the IEEE 1149.1-2001 defined instructions listed in [Table 35-3](#).

**Table 35-3. JTAG instructions**

| Instruction                         | Code[4:0]               | Instruction Summary   |
|-------------------------------------|-------------------------|---|
| IDCODE                              | 00001                   | Selects device identification register for shift  |
| SAMPLE/PRELOAD                      | 00010                   | Selects boundary scan register for shifting, sampling, and preloading without disturbing functional operation |
| SAMPLE                              | 00011                   | Selects boundary scan register for shifting and sampling without disturbing functional operation              |
| EXTEST                              | 00100                   | Selects boundary scan register while applying preloaded values to output pins and asserting functional reset  |
| ACCESS_AUX_TAP_ONCE                 | 10001                   | Grants the PLATFORM ownership of the TAP  |
| Reserved                            | 10010                   | —   |
| BYPASS                              | 11111                   | Selects bypass register for data operations   |
| Factory Debug Reserved <sup>1</sup> | 00101<br>00110<br>01010 | Intended for factory debug only   |
| Reserved <sup>2</sup>               | All other codes         | Decoded to select bypass register   |

<sup>1</sup> Intended for factory debug, and not customer use

<sup>2</sup> Freescale reserves the right to change the decoding of reserved instruction codes

### 35.8.4.1 BYPASS instruction

BYPASS selects the bypass register, creating a single-bit shift register path between TDI and TDO. BYPASS enhances test efficiency by reducing the overall shift path when no test operation of the MCU is required. This allows more rapid movement of test data to and from other components on a board that are required to perform test functions. While the BYPASS instruction is active the system logic operates normally.

### 35.8.4.2 ACCESS\_AUX\_TAP\_x instructions

The ACCESS\_AUX\_TAP\_x instructions allow the Nexus modules on the MCU to take control of the TAP. When this instruction is loaded, control of the TAP pins is transferred to the selected auxiliary TAP controller. Any data input via TDI and TMS is passed to the selected TAP controller, and any TDO output from the selected TAP controller is sent back to the JTAGC to be output on the pins. The JTAGC regains control of the JTAG port during the UPDATE-DR state if the PAUSE-DR state was entered. Auxiliary TAP controllers are held in RUN-TEST/IDLE while they are inactive.

### 35.8.4.3 EXTEST — External Test Instruction

EXTEST selects the boundary scan register as the shift path between TDI and TDO. It allows testing of off-chip circuitry and board-level interconnections by driving preloaded data contained in the boundary scan register onto the system output pins. Typically, the preloaded data is loaded into the boundary scan register using the SAMPLE/PRELOAD instruction before the selection of EXTEST. EXTEST asserts the internal system reset for the MCU to force a predictable internal state while performing external boundary scan operations.

### 35.8.4.4 IDCODE instruction

IDCODE selects the 32-bit device identification register as the shift path between TDI and TDO. This instruction allows interrogation of the MCU to determine its version number and other part identification data. IDCODE is the instruction placed into the instruction register when the JTAGC is reset.

### 35.8.4.5 SAMPLE instruction

The SAMPLE instruction obtains a sample of the system data and control signals present at the MCU input pins and just before the boundary scan register cells at the output pins. This sampling occurs on the rising edge of TCK in the capture-DR state when the SAMPLE instruction is active. The sampled data is viewed by shifting it through the boundary scan register to the TDO output during the Shift-DR state. There is no defined action in the update-DR state. Both the data capture and the shift operation are transparent to system operation.

During the SAMPLE instruction, the following pad status is enforced:

- Weak pull is disabled (independent from PCRx[WPE])
- Analog switch is disabled (independent of PCRx[APC])
- Slew rate control is forced to the slowest configuration (independent from PCRx[SRC[1]])

### 35.8.4.6 SAMPLE/PRELOAD instruction

The SAMPLE/PRELOAD instruction has two functions:

- The SAMPLE part of the instruction samples the system data and control signals on the MCU input pins and just before the boundary scan register cells at the output pins. This sampling occurs on the rising-edge of TCK in the capture-DR state when the SAMPLE/PRELOAD instruction is active. The sampled data is viewed by shifting it through the boundary scan register to the TDO output



during the shift-DR state. Both the data capture and the shift operation are transparent to system operation.

- The PRELOAD part of the instruction initializes the boundary scan register cells before selecting the EXTEST instructions to perform boundary scan tests. This is achieved by shifting in initialization data to the boundary scan register during the shift-DR state. The initialization data is transferred to the parallel outputs of the boundary scan register cells on the falling edge of TCK in the update-DR state. The data is applied to the external output pins by the EXTEST instruction. System operation is not affected.

During the SAMPLE/PRELOAD instruction, the following pad status is enforced:

- Weak pull is disabled (independent from PCR<sub>x</sub>[WPE])
- Analog switch is disabled (independent of PCR<sub>x</sub>[APC])
- Slew rate control is forced to the slowest configuration (independent from PCR<sub>x</sub>[SRC[1]])

### 35.8.5 Boundary scan

The boundary scan technique allows signals at component boundaries to be controlled and observed through the shift-register stage associated with each pad. Each stage is part of a larger boundary scan register cell, and cells for each pad are interconnected serially to form a shift-register chain around the border of the design. The boundary scan register consists of this shift-register chain, and is connected between TDI and TDO when the EXTEST, SAMPLE, or SAMPLE/PRELOAD instructions are loaded. The shift-register chain contains a serial input and serial output, as well as clock and control signals.

## 35.9 e200z0 OnCE controller

The e200z0 core OnCE controller supports a complete set of Nexus 1 debug features. A complete discussion of the e200z0 OnCE debug features is available in the *e200z0 Reference Manual*.

### 35.9.1 e200z0 OnCE controller block diagram

[Figure 35-6](#) is a block diagram of the e200z0 OnCE block.

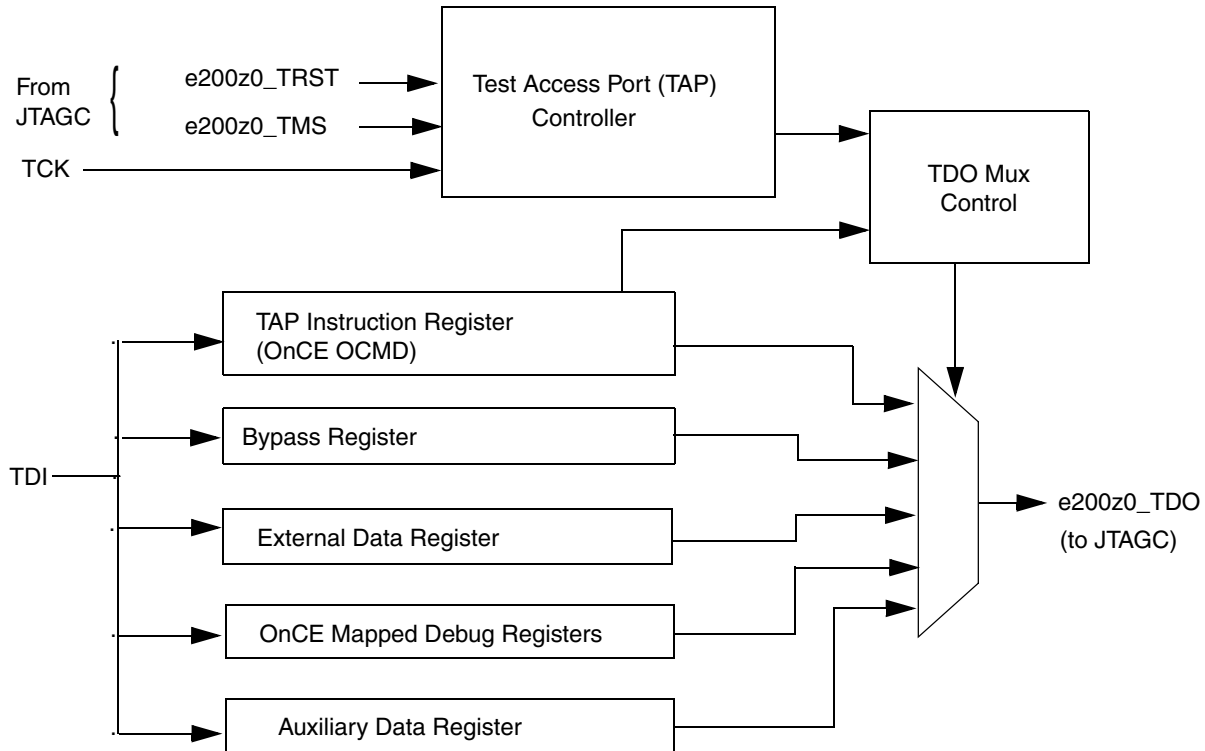


Figure 35-6. e200z0 OnCE block diagram

## 35.9.2 e200z0 OnCE controller functional description

The functional description for the e200z0 OnCE controller is the same as for the JTAGC, with the differences described below.

### 35.9.2.1 Enabling the TAP controller

To access the e200z0 OnCE controller, the proper JTAGC instruction needs to be loaded in the JTAGC instruction register, as discussed in [Section 35.5.2.2, TAP sharing mode](#).

## 35.9.3 e200z0 OnCE controller register description

Most e200z0 OnCE debug registers are fully documented in the *e200z0 Reference Manual*.

### 35.9.3.1 OnCE Command register (OCMD)

The OnCE command register (OCMD) is a 10-bit shift register that receives its serial data from the TDI pin and serves as the instruction register (IR). It holds the 10-bit commands to be used as input for the e200z0 OnCE Decoder. The OCMD is shown in [Table 35-7](#). The OCMD is updated when the TAP controller enters the update-IR state. It contains fields for controlling access to a resource, as well as controlling single-step operation and exit from OnCE mode.

Although the OCMD is updated during the update-IR TAP controller state, the corresponding resource is accessed in the DR scan sequence of the TAP controller, and as such, the update-DR state must be transitioned through in order for an access to occur. In addition, the update-DR state must also be transitioned through in order for the single-step and/or exit functionality to be performed, even though the command appears to have no data resource requirement associated with it.

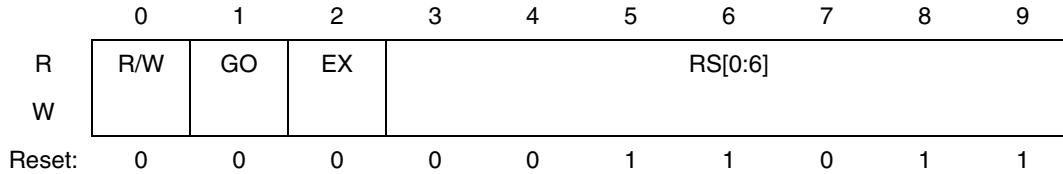


Figure 35-7. OnCE Command register (OCMD)

Table 35-4. e200z0 OnCE register addressing

| RS[0:6]             | Register selected                      |
|---------------------|--|
| 000 0000 000 0001   | Reserved                               |
| 000 0010            | JTAG ID (read-only)                    |
| 000 0011 – 000 1111 | Reserved                               |
| 001 0000            | CPU Scan Register (CPUSCR)             |
| 001 0001            | No Register Selected (Bypass)          |
| 001 0010            | OnCE Control Register (OCR)            |
| 001 0011 – 001 1111 | Reserved                               |
| 010 0000            | Instruction Address Compare 1 (IAC1)   |
| 010 0001            | Instruction Address Compare 2 (IAC2)   |
| 010 0010            | Instruction Address Compare 3 (IAC3)   |
| 010 0011            | Instruction Address Compare 4 (IAC4)   |
| 010 0100            | Data Address Compare 1 (DAC1)          |
| 010 0101            | Data Address Compare 2 (DAC2)          |
| 010 0110            | Data Value Compare 1 (DVC1)            |
| 010 0111            | Data Value Compare 2 (DVC2)            |
| 010 1000 – 010 1111 | Reserved                               |
| 011 0000            | Debug Status Register (DBSR)           |
| 011 0001            | Debug Control Register 0 (DBCR0)       |
| 011 0010            | Debug Control Register 1 (DBCR1)       |
| 011 0011            | Debug Control Register 2 (DBCR2)       |
| 011 0100 – 110 1111 | Reserved (do not access)               |
| 111 0000 – 111 1001 | General Purpose Register Selects [0:9] |
| 111 1010 – 111 1100 | Reserved                               |

Table 35-4. e200z0 OnCE register addressing (continued)

| RS[0:6]  | Register selected                      |
|----------|--|
| 111 1101 | LSRL Select<br>(factory test use only) |
| 111 1110 | Enable_OnCE                            |
| 111 1111 | Bypass                                 |

### 35.10 Initialization/application information

The test logic is a static logic design, and TCK can be stopped in either a high or low state without loss of data. However, the system clock is not synchronized to TCK internally. Any mixed operation using both the test logic and the system functional logic requires external synchronization.

To initialize the JTAGC module and enable access to registers, the following sequence is required:

1. Place the JTAGC in reset through TAP controller state machine transitions controlled by TMS
2. Load the appropriate instruction for the test or action to be performed.

# Appendix A

## Revision History

This appendix describes corrections to the *MPC5606BK Microcontroller Reference Manual*. For convenience, the corrections are grouped by revision.

### A.1 Changes between revisions 1 and 2

Table A-1. Changes between revisions 1 and 2

| Chapter                              | Description  |
|--------------------------------------|--|
| Throughout                           | Updated device name to MPC5606BK.  |
| Chapter 2, Introduction              | <p>In <a href="#">Figure 2-1 (MPC5606BK block diagram)</a>:</p> <ul style="list-style-type: none"> <li>changed “64-bit 2 × 3 Crossbar Switch” to “32-bit 3 × 3 Crossbar Switch.”</li> </ul> <p>In <a href="#">Table 2-1 (MPC5606BK family comparison)</a>, added 100LQFP option for MPC5606BK and updated the Code flash memory for 100 LQFP to 1024 KB.</p> <p>In <a href="#">Table 2-2 (MPC5606BK series block summary)</a>:</p> <ul style="list-style-type: none"> <li>changed “Supports simultaneous connections between two master ports” to “Supports simultaneous connections between three master ports and three slave ports.”</li> <li>Corrected “32 kHz oscillator” to “32 KHz oscillator.”</li> </ul> <p>In <a href="#">Section 2.4.1, e200z0h core processor</a> removed bullet item “Reservation instructions for implementing read-modify-write constructs and removed “low cost” from the 1st bullet.</p> <p>In <a href="#">Section 2.4.2, Crossbar switch (XBAR)</a>, changed “two master ports” to “three master ports.” Added bullet item for eDMA. Removed bullet item for reservation instructions.</p>   |
| Chapter 6, Clock Description         | <p>Added block for CGM_AC0_SC with inputs FIRC and FXOSC to <a href="#">Figure 6-6 (FMPLL block diagram)</a></p> <p>Added <b>Note</b>: to <a href="#">Section 6.8.4.1, Crystal clock monitor</a>:<br/> <b>Note</b>: Functional FXOSC monitoring can only be guaranteed when the FXOSC frequency is greater than <math>(FIRC / 2^{RCDIV}) + 0.5</math> MHz.</p> <p>Added <b>Note</b>: to <a href="#">Section 6.8.4.2, FMPLL clock monitor</a>:<br/> <b>Note</b>: Functional FMPLL monitoring can only be guaranteed when the FMPLL frequency is greater than <math>(FIRC / 4) + 0.5</math> MHz.</p> <p>In <a href="#">Section 6.7.7, Recommendations</a>, changed this bullet item:</p> <ul style="list-style-type: none"> <li>Use progressive clock switching if system clock changes are required while the PLL is being used as the system clock source. MOD_PERIOD, INC_STEP, SPREAD_SEL bits should be modified before activating the FM mode. Then strobe has to be generated to enable the new settings. If STRB_BYP is set to 1 then MOD_PERIOD, INC_STEP, and SPREAD_SEL can be modified only when FMPLL is in powerdown mode.</li> <li>Use progressive clock switching (FMPLL output clock can be changed when it is the system clock, but only when using progressive clock switching).</li> </ul> <p>with</p> <ul style="list-style-type: none"> <li>Use PLL progressive clock switching to ramp system clock (/8, /4, /2, /1) automatically for the case when PLL is enabled and selected as system clock.</li> <li>MOD_PERIOD, INC_STEP, SPREAD_SEL bits should be modified before activating the FM mode. Then strobe has to be generated to enable the new settings. If STRB_BYP is set to 1 then MOD_PERIOD, INC_STEP, and SPREAD_SEL can be modified only when FMPLL is in powerdown mode.</li> </ul> |
| Chapter 8, Mode Entry Module (MC_ME) | Reformatted <a href="#">Figure 8-1 (MC_ME block diagram)</a> .   |

Table A-1. Changes between revisions 1 and 2 (continued)

| Chapter  | Description   |
|--|---|
| Chapter 9,<br>Reset Generation<br>Module (MC_RGM)                              | <p>Replaced <a href="#">Section 9.4.6, Boot mode capturing</a> with the following:</p> <p>The MC_RGM samples PA[9:8] whenever RESET is asserted until five FIRC (16 MHz internal RC oscillator) clock cycles before its deassertion edge. The result of the sampling is used at the beginning of reset PHASE3 for boot mode selection and is retained after RESET has been deasserted for subsequent boots after reset sequences during which RESET is not asserted.</p> <p><b>Note:</b> In order to ensure that the boot mode is correctly captured, the application needs to apply the valid boot mode value the entire time that RESET is asserted.</p> <p>RESET can be asserted as a consequence of the internal reset generation. This will force re-sampling of the boot mode pins. (See <a href="#">Table 9-11</a> for details.)</p> |
| Chapter 11,<br>Voltage Regulators and<br>Power Supplies                        | Changed block in <a href="#">Figure 11-2 (Power domain organization)</a> from 64K RAM (PD2/PD3 to 48 KB RAM (PD2).  |
| Chapter 12,<br>Wakeup Unit (WKPU)  | Removed column for 208 pkg from <a href="#">Table 12-1</a> ; removed bullet point for 208 pkg from <a href="#">Section 12.5.2, Non-maskable interrupts</a> .  |
| Chapter 13,<br>Real Time Clock /<br>Autonomous Periodic<br>Interrupt (RTC/API) | In <a href="#">Table 13-3 (RTCC field descriptions)</a> , added Note to RTCC[APIVAL] field description:<br><b>Note:</b> API functionality starts only when APIVAL is nonzero. The first API interrupt takes two more cycles because of synchronization of APIVAL to the RTC clock, and APIVAL + 1 cycles for subsequent occurrences. After that, interrupts are periodic in nature. The minimum supported value of APIVAL is 4.   |
| Chapter 16,<br>Enhanced Direct<br>Memory Access (eDMA)                         | Replaced entire <a href="#">Section 16.5.8, Dynamic programming</a> .   |
| Chapter 19,<br>Crossbar Switch (XBAR)  | <p>Updated <a href="#">Figure 19-1 (XBAR block diagram)</a> to show eDMA as master.</p> <p>In <a href="#">Table 19-1 (XBAR switch ports for MPC5606BK)</a>:</p> <ul style="list-style-type: none"> <li>Added row for eDMA</li> <li>Swapped Logical number and physical master ID</li> </ul> <p>In <a href="#">Section 19.4, Features</a>, added bullet item for eDMA.</p> <p>Added row for eDMA to <a href="#">Table 19-2 (Hardwired bus master priorities)</a>, changed title of column for master ID to “Master #,” corrected master ID for “e200z0 core–CPU data” from 0 to 1.</p> <p>Throughout the chapter, corrected “two master ports” to “three master ports”.</p>  |
| Chapter 21,<br>Memory Protection Unit<br>(MPU)                                 | <p>In <a href="#">Section 21.5.2.1, MPU Control/Error Status Register (MPU_CESR)</a>, moved SPERR bitfield to bits 0:2, changed field tag to SPERR[0:2].</p> <p>Moved MPU_EDR<math>n</math>[EACD] bits from 8:15 to 0:7 (<a href="#">Figure 21-4</a>).</p>  |
| Chapter 23,<br>LIN Controller (LINFlex)  | <p>Added <a href="#">Section 23.8.2.1.7, Overrun</a></p> <p>In <a href="#">Section 23.7.1.8, LIN output compare register (LINOOCR)</a>, in <a href="#">Figure 23-13., LIN output compare register (LINOOCR)</a>, changed the footnote to “If LINTCSR[LTOM] = 0, these fields are read-only.”</p> <p>Changed the first sentence of <a href="#">Section 23.8.3.1, LIN timeout mode</a> to “Clearing the LTOM bit (setting its value to 0) in the LINTCSR enables the LIN timeout mode.”</p> <p>Changed the first sentence of <a href="#">Section 23.8.3.2, Output compare mode</a> to “Setting LINTCSR[LTOM] = 1 enables the output compare mode.”</p>  |

Table A-1. Changes between revisions 1 and 2 (continued)

| Chapter                                  | Description  |
|--|--|
| Chapter 24,<br>LIN Controller (LINFlexD) | <p>Added <a href="#">Section 24.7.1.5, Overrun</a>.</p> <p>In <a href="#">Section 24.7.3.2, Identifier filter submode configuration</a>, changed the second sentence to “To configure an identifier filter, the filter must first be activated by setting the corresponding bit in the IFER[FACT] field.”</p> <p>In <a href="#">Section 24.10.8, LIN output compare register (LINOOCR)</a>, in <a href="#">Figure 24-25 (LIN output compare register (LINOOCR))</a>, changed the footnote to “If LINTCSR[LTOM] = 0, these fields are read-only.”</p> <p>Changed the first sentence of <a href="#">Section 24.12.1.1, LIN timeout mode</a> to “Clearing the LTOM bit (setting its value to 0) in the LINTCSR enables the LIN timeout mode.”</p> <p>Changed the first sentence of <a href="#">Section 24.12.1.2, Output compare mode</a> to “Setting LINTCSR[LTOM] = 1 enables the output compare mode.”</p> |

Table A-1. Changes between revisions 1 and 2 (continued)

| Chapter             | Description  |
|---------------------|--|
| Chapter 25, FlexCAN | <p>Removed references throughout the chapter to “low-cost MCUs.”</p> <p>Removed <b>Note:</b> at end of <a href="#">Section 25.2.2, FlexCAN module features:</a></p> <p><b>Note:</b><br/>The individual Rx Mask per Message Buffer feature may not be available in low cost MCUs. Please consult the specific MCU documentation to find out if this feature is supported.</p> <p>Removed <b>Note:</b> above <a href="#">Table 25-2:</a></p> <p>The individual Rx Mask per Message Buffer feature may not be available in low cost MCUs. Please consult the specific MCU documentation to find out if this feature is supported. If not supported, the address range 0x0880-0x097F is considered reserved space, independent of the value of the BCC bit.</p> <p>Added this Note in the RTR field description of <a href="#">Table 25-4 (Message Buffer Structure field description):</a></p> <p><b>Note:</b> Do not configure the last Message Buffer to be the RTR frame.</p> <p>In <a href="#">Figure 25-5 (Module Configuration Register (MCR))</a>, removed the WAK_MSK and WAK_SRC fields.</p> <p>In <a href="#">Table 25-8 (MCR field descriptions)</a>, removed the WAK_MSK and WAK_SRC fields.</p> <p>Replaced the BCC bit description in <a href="#">Table 25-8 (MCR field descriptions)</a>.</p> <p>In <a href="#">Table 25-10 (CTRL field descriptions)</a>, removed the note from the CLK_SRC field.</p> <p>In <a href="#">Section 25.4.4.4, Rx Global Mask (RXGMASK) register</a>, changed “For MCUs supporting individual masks per MB, setting the BCC bit in MCR causes the RXGMASK register to have no effect on the module operation. For MCUs not supporting individual masks per MB, this register is always effective.” to “Setting the BCC bit in MCR causes the RXGMASK register to have no effect on the module operation.”</p> <p>In <a href="#">Section 25.4.4.5, Rx 14 Mask (RX14MASK) register</a>, removed the phrase “For MCUs supporting individual masks per MB” from the first paragraph.</p> <p>In <a href="#">Section 25.4.4.6, Rx 15 Mask (RX15MASK) register</a>, removed the phrase “For MCUs supporting individual masks per MB” from the first paragraph.</p> <p>Removed <b>Note:</b> in <a href="#">Section 25.4.4.13, Rx Individual Mask Registers (RXIMR0–RXIMR63):</a></p> <p><b>Note:</b><br/>The individual Rx Mask per Message Buffer feature may not be available in low cost MCUs. Please consult the specific MCU documentation to find out if this feature is supported. If not supported, the RXGMASK, RX14MASK, and RX15MASK registers are available, regardless of the value of the BCC bit.</p> <p>Updated the table title of <a href="#">Table 25-22 (Bosch CAN 2.0B standard compliant bit time segment settings)</a> from “CAN Standard Compliant Bit Time Segment Settings” to “Bosch CAN 2.0B standard compliant bit time segment settings.”</p> <p>Removed <b>Note:</b> at end of <a href="#">Section 25.5.6, Matching process:</a></p> <p><b>Note:</b><br/>The individual Rx Mask per Message Buffer feature may not be available in low cost MCUs. Please consult the specific MCU documentation to find out if this feature is supported. If not supported, the RXGMASK, RX14MASK, and RX15MASK registers are available, regardless of the value of the BCC bit.</p> <p>Changed the last sentence of <a href="#">Section 25.5.1, Overview</a> to “An MB programmed with 0000, 1000, (inactive), or 1001 (abort) will be temporarily deactivated (will not participate in the current arbitration or matching run) when the CPU writes to the C/S field of that MB (see <a href="#">Section 25.5.7.2, Message buffer deactivation</a>).</p> <p>In <a href="#">Section 25.5.9.4, Protocol timing</a>, removed the Note following <a href="#">Figure 25-16 (CAN Engine Clocking Scheme)</a>: “This clock selection feature may not be available in all MCUs. A particular MCU may not have a PLL, in which case it would have only the oscillator clock, or it may use only the PLL clock feeding the FlexCAN module. In these cases, the CLK_SRC bit in the CTRL Register has no effect on the module operation.”</p> |



Table A-1. Changes between revisions 1 and 2 (continued)

| Chapter  | Description   |
|--|---|
| Chapter 25,<br>FlexCAN (cont.)                   | In <a href="#">Section 25.5.9.4, Protocol timing</a> , updated the Note following <a href="#">Table 25-22 (Bosch CAN 2.0B standard compliant bit time segment settings)</a> to read: “Other combinations of Time Segment 1 and Time Segment 2 can be valid. It is the user’s responsibility to ensure the bit time settings are in compliance with the CAN standard. For bit time calculations, use an IPT (Information Processing Time) of 2, which is the value implemented in the FlexCAN module.”   |
| Chapter 28,<br>Analog-to-Digital Converter (ADC) | <p>Throughout this chapter, corrected references to register CWSEL to CWSELR.</p> <p>Corrected offset for CEOFCR1 in <a href="#">Table 28-10 (12-bit ADC_1 digital registers)</a> to 0x0018; changed row for 0x0018 to “reserved.” Added Note: under <a href="#">Figure 28-15 (Channel Pending Register 2 (CEOFCR2))</a>, explaining that CEOFCR1 was not implemented on ADC_1.</p> <p>Removed this paragraph in <a href="#">Section 28.3.2, Analog clock generator and conversion timings</a>:<br/>The direct clock should basically be used only in low power mode when the device is using only the 16 MHz fast internal RC oscillator, but the conversion still requires a 16 MHz clock (an 8 MHz clock is not fast enough).</p> <p>In <a href="#">Section 28.3.4.2, CTU in trigger mode</a>, replaced sentence:<br/>“If another CTU conversion is triggered before the end of the conversion, that request is discarded.”<br/>with:<br/>“If another CTU conversion is triggered before the end of the current CTU triggered conversion, the new request is discarded.”</p> <p>In <a href="#">Section 28.3.5.2, Presampling channel enable signals</a>, in <a href="#">Table 28-7 (Presampling voltage selection based on PREVALx fields)</a>, in the 01 row, changed the “Presampling voltage” field to:<br/><math>V1 = V_{DD\_HV\_ADC0} \text{ or } V_{DD\_HV\_ADC1}</math>.</p> <p>In <a href="#">Table 28-10 (12-bit ADC_1 digital registers)</a>, added the following rows for these registers:<br/>— 0x0028 CIMR1<br/>— 0x0048 DMAR1<br/>— 0x0088 PSR1<br/>— 0x0098 CTR1<br/>— 0x00A8 NCMR1<br/>— 0x00B8 JCMR1<br/>Corrected surrounding rows to reflect these changes.</p> <p>Made the following corrections to the ADCSTATUS description field in <a href="#">Table 28-12 (MSR field descriptions)</a>:<br/>The value of this parameter depends on ADC status:<br/>000 IDLE — The ADC is powered up but idle.<br/>001 Power-down — The ADC is powered down.<br/>010 Wait state — The ADC is waiting for an external multiplexer, This only occurs when the DSDR register is non-zero.<br/>011 Reserved<br/>100 Sample — The ADC is sampling the analog signal.<br/>101 Reserved<br/>110 Conversion — The ADC is converting the sampled signal.<br/>111 Reserved</p> <p>In <a href="#">Section 28.4.2.2, Main Status Register (MSR)</a>, added detailed explanations to the ADCSTATUS field.</p> <p>Corrected reset value of CTR<sub>n</sub> registers to 0x0203 <a href="#">Section 28.4.7, Conversion timing registers CTR[0..2]</a></p> <p>Corrected CDR<sub>n</sub> registers from read/write to read-only (<a href="#">Figure 28-48</a> and <a href="#">Figure 28-49</a>).<br/>Changed fields CWSELR7[WSEL_CH63] through CWSELR7[WSEL_CH60] to reserved (<a href="#">Figure 28-50</a>).</p> |

Table A-1. Changes between revisions 1 and 2 (continued)

| Chapter  | Description   |
|--|---|
| Chapter 28,<br>Analog-to-Digital<br>Converter (ADC)<br>(Contd..) | <p>Reformatted <a href="#">Table 28-14</a>.<br/> Reformatted <a href="#">Table 28-17</a>.<br/> Reformatted <a href="#">Table 28-24</a>.<br/> Reformatted <a href="#">Table 28-29</a>.<br/> Reformatted <a href="#">Table 28-31</a>.<br/> Reformatted <a href="#">Table 28-33</a>.<br/> Reformatted <a href="#">Table 28-35</a>.<br/> Reformatted <a href="#">Table 28-39</a>.<br/> Reformatted <a href="#">Table 28-41</a>.<br/> Reformatted <a href="#">Table 28-56</a>.</p> <ul style="list-style-type: none"> <li>• In <a href="#">Table 28-17 (CIMR[0..2] register description)</a>, added a row for ADC_1 CIMR1 bits “Enable bit for channel 32 to 39 (standard channels)”</li> <li>• Edited title for <a href="#">Figure 28-18 (Channel Interrupt Mask Register 1 (CIMR1) for ADC_0)</a> to specify usage with ADC_0.</li> <li>• Added <a href="#">Figure 28-19 (Channel Interrupt Mask Register 1 (CIMR1) for ADC_1)</a>.</li> <li>• In <a href="#">Table 28-24 (DMAR[0..2] register description)</a>, added a row for ADC_1 DMAR1 bits “Enable bit for channel 32 to 39 (standard channels)”</li> <li>• Edited title for <a href="#">Figure 28-27 (DMA Channel Select Register 1 (DMAR1) for ADC_0)</a> to specify usage with ADC_0.</li> <li>• Added <a href="#">Figure 28-28 (DMA Channel Select Register 1 (DMAR1) for ADC_1)</a>.</li> <li>• In <a href="#">Table 28-29 (PSR[0..2] register description)</a>, added a row for ADC_1 PSR1 bits “Enable bit for channel 32 to 39 (standard channels)”</li> <li>• Added <a href="#">Figure 28-35 (Presampling Register 1 (PSR1) for ADC_1)</a>.</li> <li>• Edited title for <a href="#">Figure 28-27 (DMA Channel Select Register 1 (DMAR1) for ADC_0)</a> to specify usage with ADC_0.</li> <li>• In <a href="#">Table 28-31 (CTR[0..2] register description)</a>, added a row for ADC_1 CTR1 “Associated to internal standard channel 32 to 39”</li> <li>• In <a href="#">Table 28-33 (NCOMR[0..2] register description)</a>, added a row for ADC_1 NCOMR1 “Enable bit of normal sampling channel 32 to 39 (standard channels)”</li> <li>• Added <a href="#">Figure 28-40 (Normal Conversion Mask Register 1 (NCOMR1) for ADC_1)</a>.</li> <li>• In <a href="#">Table 28-35 (JCMR[0..2] register description)</a>, added a row for ADC_1 JCMR1 “Enable bit of normal sampling channel 32 to 39 (standard channels)”</li> <li>• Added <a href="#">Figure 28-44 (Injected Conversion Mask Register 1 (JCMR1) for ADC_1)</a>.</li> <li>• Edited title for <a href="#">Figure 28-43 (Injected Conversion Mask Register 1 (JCMR1) for ADC_0)</a> to specify usage with ADC_0.</li> </ul> <p>Added Note: in <a href="#">Section 28.3.11, Auto-clock-off mode</a><br/> <b>Note:</b> The auto-clock-off feature cannot operate when the digital interface runs at the same rate as the analog interface. This means that when MCR.ADCCLKSEL = 1, the analog clock will not shut down in IDLE mode.</p> |
| Chapter 29,<br>Cross Triggering Unit<br>(CTU)                    | <p>At the end of <a href="#">Section 29.4.1, Event Configuration Registers (CTU_EVTCFGRx) (x = 0..63)</a>, added the following Note:</p> <p>The CTU tracks issued conversion requests to the ADC. When the ADC is being triggered by the CTU and there is a need to shut down the ADC, the ADC must be allowed to complete conversions before being shut down. This ensures that the CTU is notified of completion; if the ADC is shut down while performing a CTU-triggered conversion, the CTU is not notified and will not be able to trigger further conversions until the device is reset.</p>   |

Table A-1. Changes between revisions 1 and 2 (continued)

| Chapter   | Description  |
|---|--|
| Chapter 30,<br>Flash Memory                             | <p>Changed reserved area in <a href="#">Table 30-8</a> from 0x001C – 0x0038 to 0x001C – 0x003B.</p> <p>In the third paragraph of <a href="#">Section 30.4.2.1, CFlash module sectorization</a>, corrected “Bank 0 of the module is divided in 18 sectors “ to 14 sectors.</p> <p>Updated field description of BKx_APC and BKx_RWSC fields of PFCRx registers (in <a href="#">Section 30.7.2.2.1, Platform Flash Configuration Register 0 (PFCR0)</a> and <a href="#">Section 30.7.2.2.2, Platform Flash Configuration Register 1 (PFCR1)</a>).</p> <p>Updated <a href="#">Section 30.8.6, Access pipelining</a></p> <p>Updated <a href="#">Section 30.1, Introduction</a>.</p> <p>Updated <a href="#">Figure 30-1 (Flash memory architecture)</a>.</p> <p>Updated <a href="#">Figure 30-2 (CFlash and DFlash module structures)</a>.</p>   |
| Chapter 33,<br>Software Watchdog<br>Timer (SWT)         | <p>In <a href="#">Figure 33-1 (SWT Control Register (SWT_CR))</a> corrected reset value from 0x4000_011B to 0x800_0011B.</p> <p>Removed the following sentence from <a href="#">Section 33.5.2.1, SWT Control Register (SWT_CR)</a><br/>Default value for SWT_CR_RST is 0x4000_011B, corresponding to MAP1 = 1 (only data bus access allowed), RIA = 1 (reset on invalid SWT access), SLK = 1 (soft lock), CSL = 1 (IRC clock source for counter), FRZ = 1 (freeze on debug), WEN = 1 (watchdog enable).</p> <p>In <a href="#">Figure 33-2 (SWT Interrupt Register (SWT_IR))</a> updated the MAPn field description to:<br/>Master Access Protection for Master n.<br/>Allows specific master to update watchdog. MAP0 = CPU, MAP2=eDMA.<br/>The platform bus master assignments are device-specific.<br/>0 Access for the master is not enabled<br/>1 Access for the master is enabled</p> <p>In <a href="#">Figure 33-2 (SWT Interrupt Register (SWT_IR))</a> “SWT Interrupt Register (SWT_IR),” changed TIF bit to “w1c.”</p> <p>Added SWT Service Key Register (SWT_SK) information.</p> <p>Updated <a href="#">Section 33.6, Functional description</a> to describe additional service key functionality.</p> |
| Chapter 34,<br>Error Correction Status<br>Module (ECSM) | <p>Inserted the following in <a href="#">Section 34.2, Overview</a></p> <p>The AIPS is the interface between the Advanced High performance Bus (AHB) interface and on-chip IPS peripherals. IPS peripherals are modules that contain readable/writable control and status registers. The AHB master reads and writes these registers through the AIPS. The AIPS generates module enables, the module address, transfer attributes, byte enables, and write data. These elements then function as inputs to the IPS peripherals.</p> <ul style="list-style-type: none"> <li>• IPS — Inter Peripheral Subsystem</li> <li>• AIPS — interface between the Advanced High performance Bus (AHB) interface and on-chip IPS peripherals</li> <li>• AHB — Advance High performance Bus</li> </ul> <p>Updated reset value of PCT register to 0xE012 (<a href="#">Figure 34-1 (Processor Core Type Register (PCT))</a>) and reset value of IOPMC register to 0x0803_E000 (<a href="#">Figure 34-3 (IPS On-Platform Module Configuration Register (IOPMC))</a>).</p> <p>Removed bullet “Configuration for additional SRAM WS for system frequency above 64 + 4% MHz” from Features list.</p> <p>Removed MWCR register.</p>     |
| Appendix A,<br>Revision History                         | Added Revision History appendix  |

## **How to Reach Us:**

### **Home Page:**

[www.freescale.com](http://www.freescale.com)

### **Web Support:**

<http://www.freescale.com/support>

### **USA/Europe or Locations Not Listed:**

Freescale Semiconductor, Inc.  
Technical Information Center, EL516  
2100 East Elliot Road  
Tempe, Arizona 85284  
+1-800-521-6274 or +1-480-768-2130  
[www.freescale.com/support](http://www.freescale.com/support)

Europe, Middle East, and Africa:  
Freescale Halbleiter Deutschland GmbH  
Technical Information Center  
Schatzbogen 7  
81829 Muenchen, Germany  
+44 1296 380 456 (English)  
+46 8 52200080 (English)  
+49 89 92103 559 (German)  
+33 1 69 35 48 48 (French)  
[www.freescale.com/support](http://www.freescale.com/support)

### **Japan:**

Freescale Semiconductor Japan Ltd. Headquarters  
ARCO Tower 15F  
1-8-1, Shimo-Meguro, Meguro-ku,  
Tokyo 153-0064  
Japan  
0120 191014 or +81 3 5437 9125  
[support.japan@freescale.com](mailto:support.japan@freescale.com)

### **Asia/Pacific:**

Freescale Semiconductor China Ltd.  
Exchange Building 23F  
No. 118 Jianguo Road  
Chaoyang District  
Beijing 100022  
China  
+86 10 5879 8000  
[support.asia@freescale.com](mailto:support.asia@freescale.com)

Freescale Semiconductor Literature Distribution Center  
1-800-441-2447 or +1-303-675-2140  
Fax: +1-303-675-2150  
[LDCForFreescaleSemiconductor@hibbertgroup.com](mailto:LDCForFreescaleSemiconductor@hibbertgroup.com)

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals", must be validated for each customer application by customer's technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.

Freescale™ and the Freescale logo are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners.

The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.

All other product or service names are the property of their respective owners.  
© Freescale Semiconductor, Inc. 2010–2014. All rights reserved.

MPC5606BKRM  
Rev. 2  
05/2014

