S3C84BB/F84BB

8-BIT CMOS
MICROCONTROLLERS
USER'S MANUAL

Revision 1

SAMSUNG ELECTRONICS

Important Notice

information in this publication has been carefully checked and is believed to be entirely accurate at the time of publication. Samsung assumes no responsibility, however, for possible errors or omissions, or for any consequences resulting from the use of the information contained herein.

Samsung reserves the right to make changes in its products or product specifications with the intent to improve function or design at any time and without notice and is not required to update this documentation to reflect such changes.

This publication does not convey to a purchaser of semiconductor devices described herein any license under the patent rights of Samsung or others.

Samsung makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does Samsung 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 any consequential or incidental damages.

"Typical" parameters can and do vary in different applications. All operating parameters, including "Typicals" must be validated for each customer application by the customer's technical experts.

Samsung products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, for other applications intended to support or sustain life, or for any other application in which the failure of the Samsung product could create a situation where personal injury or death may occur.

Should the Buyer purchase or use a Samsung product for any such unintended or unauthorized application, the Buyer shall indemnify and hold Samsung and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, expenses, and reasonable attorney fees arising out of, either directly or indirectly, any claim of personal injury or death that may be associated with such unintended or unauthorized use, even if such claim alleges that Samsung was negligent regarding the design or manufacture of said product.

S3C84BB/F84BB 8-Bit CMOS Microcontrollers
User's Manual, Revision 1
Publication Number: 20-S3-C84BB/F84BB-0800

Samsung Electronics' microcontroller business has been awarded full ISO-14001 certification (BSI Certificate No. FM24653). All semiconductor products are designed and manufactured in accordance with the highest quality standards and objectives.

Samsung Electronics Co., Ltd.
San #24 Nongseo-Ri, Kiheung- Eup
Yongin-City, Kyunggi-Do, Korea
C.P.O. Box #37, Suwon 449-900

TEL: (82)-(31)-209-1907
FAX: (82)-(31)-209-1899

Home Page: http://www.intl.samsungsemi.com
Printed in the Republic of Korea

Preface

The *S3C84BB/F84BB Microcontroller User's Manual* is designed for application designers and programmers who are using the S3C84BB/84BB microcontroller for application development.
It is organized in two main parts:

Part I Programming Model Part II Hardware Descriptions

Part I contains software-related information to familiarize you with the microcontroller's architecture, programming model, instruction set, and interrupt structure. It has six chapters:

Chapter 1	Product Overview	Chapter 4	Control Registers
Chapter 2	Address Spaces	Chapter 5	Interrupt Structure
Chapter 3	Addressing Modes	Chapter 6	Instruction Set

Chapter 1, "Product Overview," is a high-level introduction to S3C84BB/F84BB with general product descriptions, as well as detailed information about individual pin characteristics and pin circuit types.

Chapter 2, "Address Spaces," describes program and data memory spaces, the internal register file, and register addressing. Chapter 2 also describes working register addressing, as well as system stack and user-defined stack operations.

Chapter 3, "Addressing Modes," contains detailed descriptions of the addressing modes that are supported by the S3C8-series CPU.

Chapter 4, "Control Registers," contains overview tables for all mapped system and peripheral control register values, as well as detailed one-page descriptions in a standardized format. You can use these easy-to-read, alphabetically organized, register descriptions as a quick-reference source when writing programs.

Chapter 5, "Interrupt Structure," describes the S3C84BB/F84BB interrupt structure in detail and further prepares you for additional information presented in the individual hardware module descriptions in Part II.

Chapter 6, "Instruction Set," describes the features and conventions of the instruction set used for all S3C8-series microcontrollers. Several summary tables are presented for orientation and reference. Detailed descriptions of each instruction are presented in a standard format. Each instruction description includes one or more practical examples of how to use the instruction when writing an application program.

A basic familiarity with the information in Part I will help you to understand the hardware module descriptions in Part II. If you are not yet familiar with the S3C-series microcontroller family and are reading this manual for the first time, we recommend that you first read Chapters 1–3 carefully. Then, briefly look over the detailed information in Chapters 4, 5, and 6. Later, you can reference the information in Part I as necessary.

Part II "hardware Descriptions," has detailed information about specific hardware components of the S3C84BB/F84BB microcontroller. Also included in Part II are electrical, mechanical, Flash MCU, and development tools data. It has 15 chapters:

Chapter 7	Clock Circuit	Chapter 15	10-bit A/D Converter
Chapter 8	RESET and Power-Down	Chapter 16	8-bit D/A Converter
Chapter 9	I/O Ports	Chapter 17	Pattern Generation Module
Chapter 10	Basic Timer	Chapter 18	Embedded Flash Memory Interface
Chapter 11	8-bit Timer A/B/C(0/1)	Chapter 19	Electrical Data
Chapter 12	16-bit Timer 1(0/1)	Chapter 20	Mechanical Data
Chapter 13	Serial I/O Port	Chapter 21	Development Tools
Chapter 14	UART(0/1)		

Two order forms are included at the back of this manual to facilitate customer order for S3C84BB/F84BB microcontrollers: the Mask ROM Order Form, and the Mask Option Selection Form. You can photocopy these forms, fill them out, and then forward them to your local Samsung Sales Representative.

Table of Contents

Part I — Programming Model

Chapter 1 Product Overview

Chapter 2 Address Spaces

Chapter 3 Addressing Modes

Table of Contents (Continued)

Chapter 4 Control Registers

Chapter 5 Interrupt Structure

Chapter 6 Instruction Set

Table of Contents (Continued)

Part II Hardware Descriptions

Chapter 7 Clock Circuit

Chapter 8 RESET and Power-Down

Chapter 9 I/O Ports

Chapter 10 Basic Timer

Table of Contents (Continued)

Chapter 11 8-bit Timer A/B/C(0/1)

Chapter 12 16-bit Timer 1(0/1)

Chapter 13 Serial I/O Port

Table of Contents (Continued)

Chapter 14 UART(0/1)

Chapter 15 10-bit A/D Converter

Chapter 16 10-bit D/A Converter

Chapter 17 Pattern Generation Module

Table of Contents (Continued)

Chapter 18 Embedded FLASH Memory Interface

Chapter 19 Electrical Data

Chapter 20 Mechanical Data

Chapter 21 Development Tools

List of Figures

List of Figures (Continued)

List of Figures (Continued)

List of Figures (Concluded)

List of Tables

List of Tables (Continued)

List of Programming Tips

List of Register Descriptions

List of Register Descriptions (Continued)

List of Instruction Descriptions

List of Instruction Descriptions (Continued)

1 PRODUCT OVERVIEW

S3C8-SERIES MICROCONTROLLERS

Samsung's S3C8-series of 8-bit single-chip CMOS microcontrollers offers a fast and efficient CPU, a wide range of integrated peripherals, and various mask-programmable ROM sizes. The major CPU features are:

— Efficient register-oriented architecture
— Selectable CPU clock sources
— Idle and Stop power-down mode released by interrupt or reset
— Built-in basic timer with watchdog function

A sophisticated interrupt structure recognizes up to eight interrupt levels. Each level can have one or more interrupt sources and vectors. Fast interrupt processing (within a minimum of four CPU clocks) can be assigned to specific interrupt levels.

S3C84BB/F84BB MICROCONTROLLER

The S3C84BB/F84BB single-chip CMOS microcontrollers are fabricated using the highly advanced CMOS process, based on Samsung’s latest CPU architecture.

The S3C84BB is a microcontroller with a 64K-byte mask-programmable ROM embedded.

The S3F84BB is a microcontroller with a 64K-byte Full-Flash ROM embedded.

Using a proven modular design approach, Samsung engineers have successfully developed the S3C84BB/F84BB by integrating the following peripheral modules with the powerful SAM8 core:

— Nine programmable I/O ports, including eight 8-bit ports and one 6-bit ports, for a total of 70 pins.
— Ten bit-programmable pins for external interrupts.
— One 8-bit basic timer for oscillation stabilization and watchdog function (system reset).
— Four 8-bit timer/counter and two 16-bit timer/counter with selectable operating modes.
— Tow asynchronous UART
— One synchronous SIO
— One 8-bit D/A converter
— 8-channel A/D converter

The S3C84BB/F84BB is versatile microcontroller for CD-ROM and ADC application, etc. They are currently available in 80-pin QFP and 80-pin TQFP package.

FEATURES

CPU

- SAM88RC CPU core

Memory

- 2064-bytes internal register file
- 64K-bytes internal program memory
  - S3C84BB: Mask ROM
  - S3F84BB: Flash type memory

Oscillation Sources

- Crystal, Ceramic
- CPU clock divider (1/1, 1/2, 1/8, 1/16)

Instruction Set

- 78 instructions
- IDLE and STOP instructions added for power-down modes

Instruction Execution Time

- 400 ns at 10-MHz f_{OSC} (minimum)

Interrupts

- 24 interrupt sources with 24 vector.
- 8 level, 24 vector interrupt structure

I/O Ports

- Total 70 bit-programmable pins

Timers and Timer/Counters

- One programmable 8-bit basic timer (**BT**) for oscillation stabilization control or watchdog-timer function.
- One 8-bit timer/counter (**Timer A**) with three operating modes; Interval mode, capture mode and PWM mode.
- One 8-bit timer/counter (**Timer B**) Carrier frequency (or PWM) generator.
- Two 8-bit timer with PWM mode (**Timer C0,C1**)
- Two 16-bit capture timer/counter (**Timer 10,11**) with two operating modes; Interval mode, Capture mode for pulse period or duty.

A/D Converter

- 10-bit resolution
- Eight analog input channels
- 20us conversion speed at 10MHz f_{ADC} clock.

D/A Converter

- 8-bit D/A Converter
- R/2R Resistor method
- One D/A output (DAOUT)

Asynchronous UART

- Full duplex 2 channels UARTs
- Programmable baud rate
- Supports serial data transmit/receive operations with 8-bit, 9-bit in UART

Synchronous SIO

- Programmable baud rate
- One synchronous serial I/O module

Pattern Generation Module

- Pattern generation module triggered by timer match signal and S/W.

Operating Temperature Range

- -25°C to + 85°C

Operating Voltage Range

- 2.7 V to 5.5 V at 10MHz f_{OSC}

Package Type

- 80 pin QFP, 80 pin TQFP

BLOCK DIAGRAM

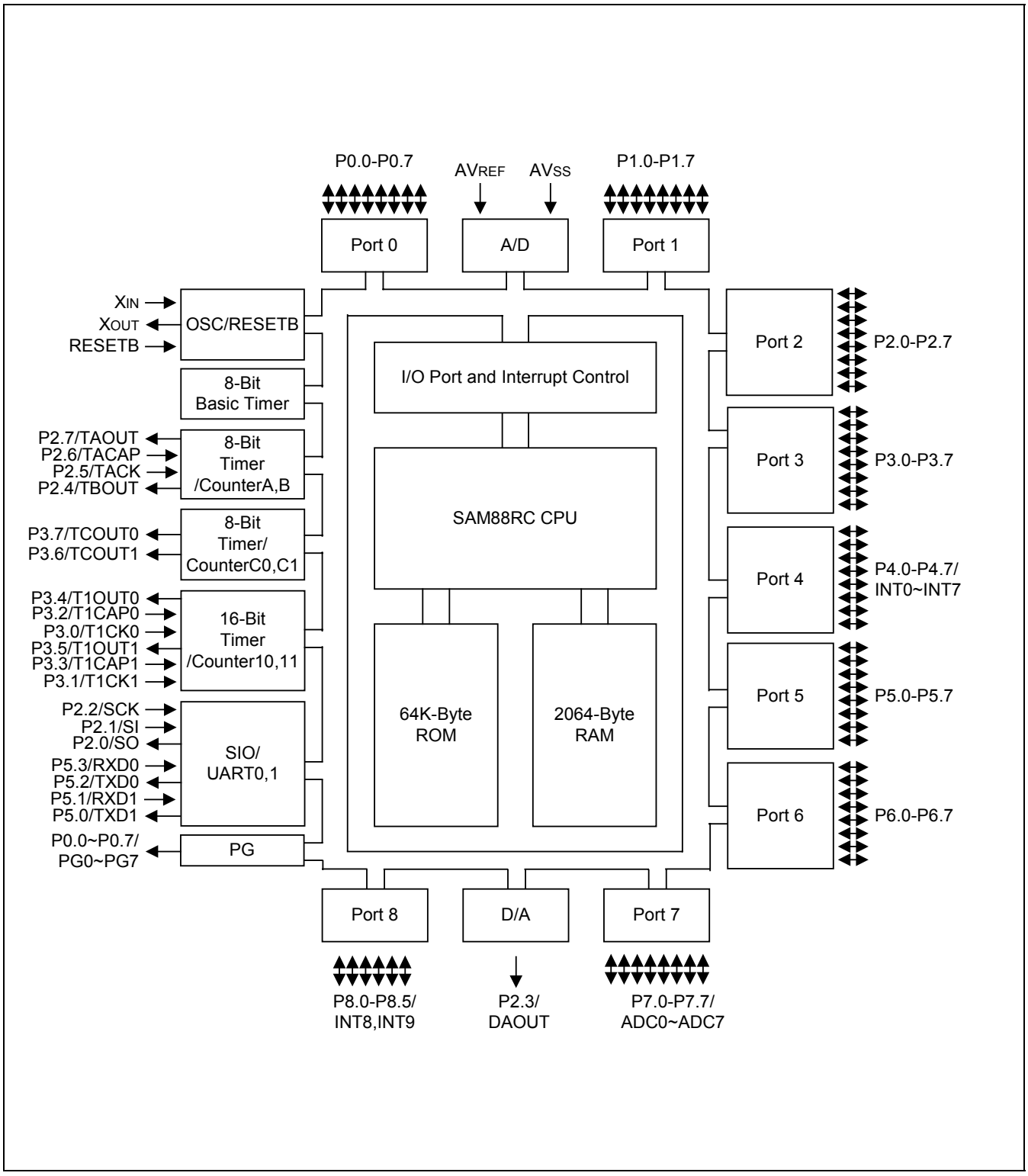


Figure 1-1. S3C84BB/F84BB Block Diagram

PIN ASSIGNMENT

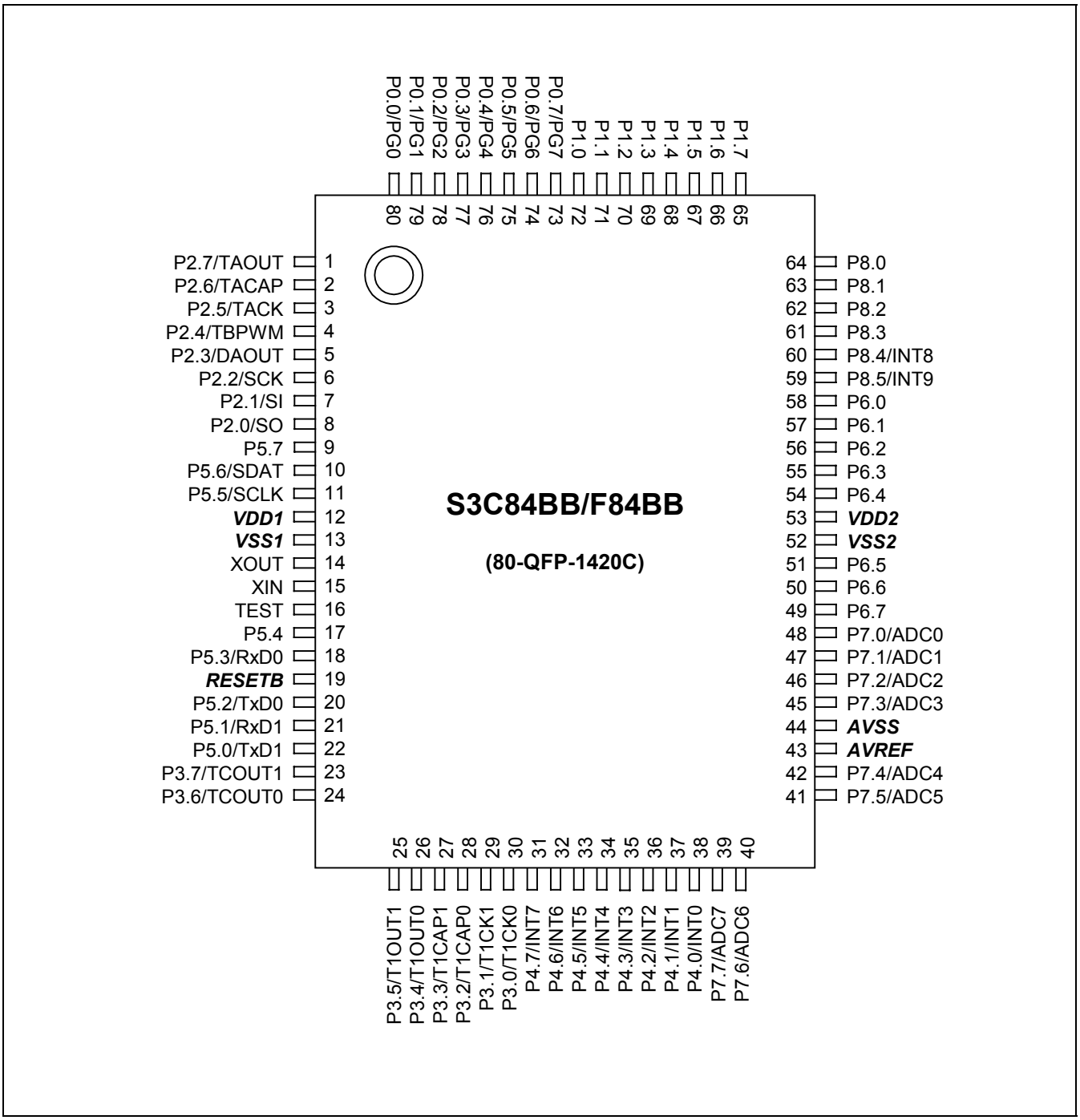


Figure 1-2. S3C84BB/F84BB Pin Assignment (80-QFP)

PIN ASSIGNMENT

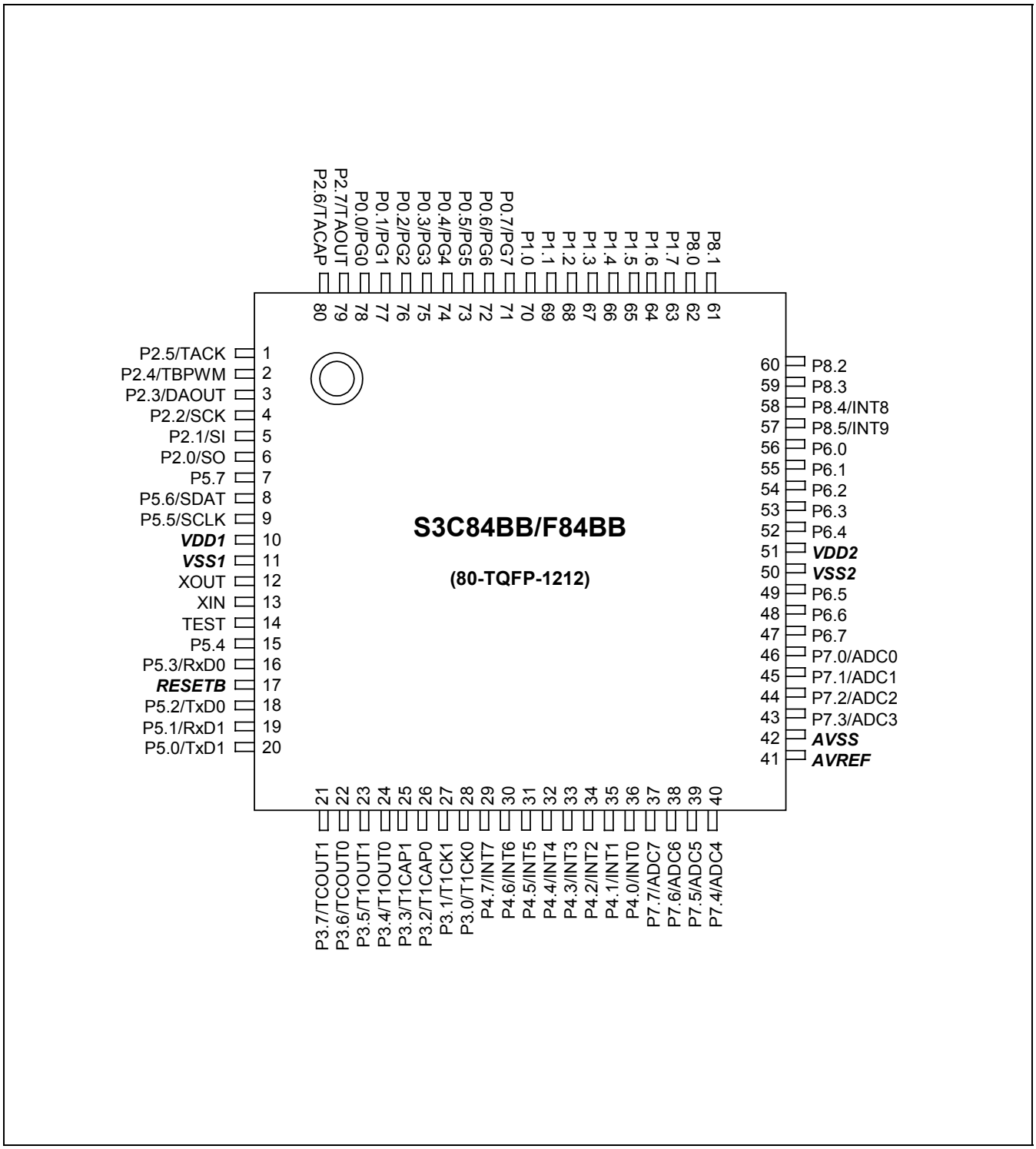


Figure 1-3. S3C84BB/F84BB Pin Assignment (80-TQFP)

PIN DESCRIPTIONS

Table 1-1. S3C84BB/F84BB Pin Descriptions (80-QFP)

Pin Name	Pin Type	Pin Description	Circuit Type	Pin Number	Share Pins
P0.0 - P0.7	I/O	Bit programmable port; input or output mode selected by software; input or push-pull output. Software assignable pull-up. Alternately, P0.0-P0.7 can be used as the PG output port (PG0-PG7).	D	80-73	PG0-PG7
P1.0 - P1.7	I/O	Bit programmable port; input or output mode selected by software; input or push-pull output. Software assignable pull-up.	D	72-65	
P2.0 - P2.7	I/O	Bit programmable port; input or output mode selected by software; input or push-pull output. Software assignable pull-up. Alternately, P2.0~P2.7 can be used as I/O for TIMERA, TIMERB, D/A, SIO	D,D-2	8-1	SO SI SCK DAOUT TBPWM TACK TACAP TAOUT
P3.0 - P3.7	I/O	Bit programmable port; input or output mode selected by software; input or push-pull output. Software assignable pull-up. Alternately, P3.0~P3.7 can be used as I/O for TIMERC0/C1, TIMER10/11	D	30–23	T1CK0 T1CK1 T1CAP0 T1CAP1 T1OUT0 T1OUT1 TCOUT0 TCOUT1

Table 1-1. S3C84BB/F84BB Pin Descriptions (80-QFP) (Continued)

Pin Name	Pin Type	Pin Description	Circuit Type	Pin Number	Share Pins
P4.0 - P4.7	I/O	Bit programmable port; input or output mode selected by software; input or push-pull output. Software assignable pull-up. P4.0-P4.7 can alternately be used as inputs for external interrupts INT0-INT7, respectively (with noise filters and interrupt controller)	D-1	38-31	INT0–INT7
P5.0 - P5.7	I/O	Bit programmable port; input or output mode selected by software; input or push-pull output. Software assignable pull-up. Alternately, P5.0~P5.3 can be used as I/O for serial por, UART0, UART1, respectively.	G	22-17,11-9	TxD1 RxD1 TxD0 RxD0
P6.0 - P6.7	O	N-channel, open-drain output only port.	F	58–54,51-49	
P7.0 - P7.7	I	General-purpose digital input ports. Alternatively used as analog input pins for A/D converter modules.	E	48-45,42-39	ADC0-ADC7
P8.0 - P8.5	I/O	Bit programmable port; input or output mode selected by software; input or push-pull output. Software assignable pull-up. P8.4, P8.5 can alternately be used as inputs for external interrupts INT8, INT9, respectively (with noise filters and interrupt controller)	D,D-1	64-59	INT8,INT9

Table 1-1. S3C84BB/F84BB Pin Descriptions (80-QFP) (Continued)

Pin Name	Pin Type	Pin Description	Circuit Type	Pin Number	Share Pins
AD0 - AD7	I	Analog input pins for A/D converter module. Alternatively used as general-purpose digital input port 7.	E	48–45 42–39	P7.0–P7.7
AVREF, AVSS	-	A/D converter reference voltage and ground	-	43, 44	-
RxD0, RxD1	I/O	Serial data RxD pin for receive input and transmit output (mode 0)	D	18, 21	P5.3, P5.1
TxD0, TxD1	O	Serial data TxD pin for transmit output and shift clock input (mode 0)	D	20, 22	P5.2, P5.0
TACK	I	External clock input pins for timer A	D	3	P2.5
TACAP	I	Capture input pins for timer A	D	2	P2.6
TAOUT	O	Pulse width modulation output pins for timer A	D	1	P2.7
TBPWM	O	Carrier frequency output pins for timer B	D	4	P2.4
TCOUT0 TCOUT1	O	Timer C 8-bit PWM mode output or counter match toggle output pins	D	24,23	P3.6,P3.7
T1CK0 T1CK1	I	External clock input pins for timer 1	D	39,30	P3.0,P3.1
T1CAP0 T1CAP1	I	Capture input pins for timer 1	D	28,27	P3.2,P3.3
T1OUT0 T1OUT1	O	Timer 1 16-bit PWM mode output or counter match toggle output pins	D	26,25	P3.4,P3.5
SI,SO,SCK	I/O	Synchronous SIO pins	D	7,8,9	P2.1,P2.0, P2.2
RESETB	I	System reset pin (pull-up resistor: 240 kΩ)	B	19	-
TEST	I	Pull – down register connected internally	-	16	-
VDD1, VDD2, VSS1, VSS2	-	Power input pins	-	12,53, 13,52	-
XIN, XOUT	-	Main oscillator pins	-	15,14	-

PIN CIRCUITS

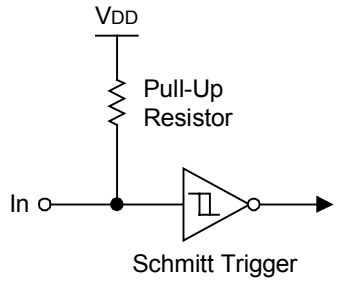


Figure 1-4. Pin Circuit Type B (RESETB)

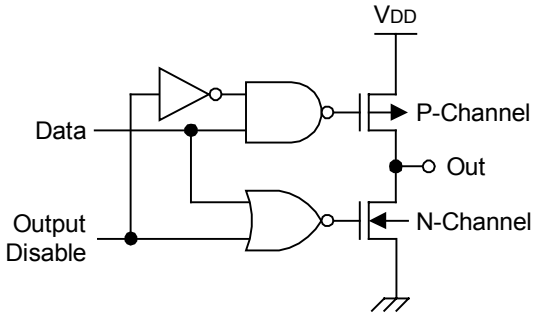


Figure 1-5. Pin Circuit Type C

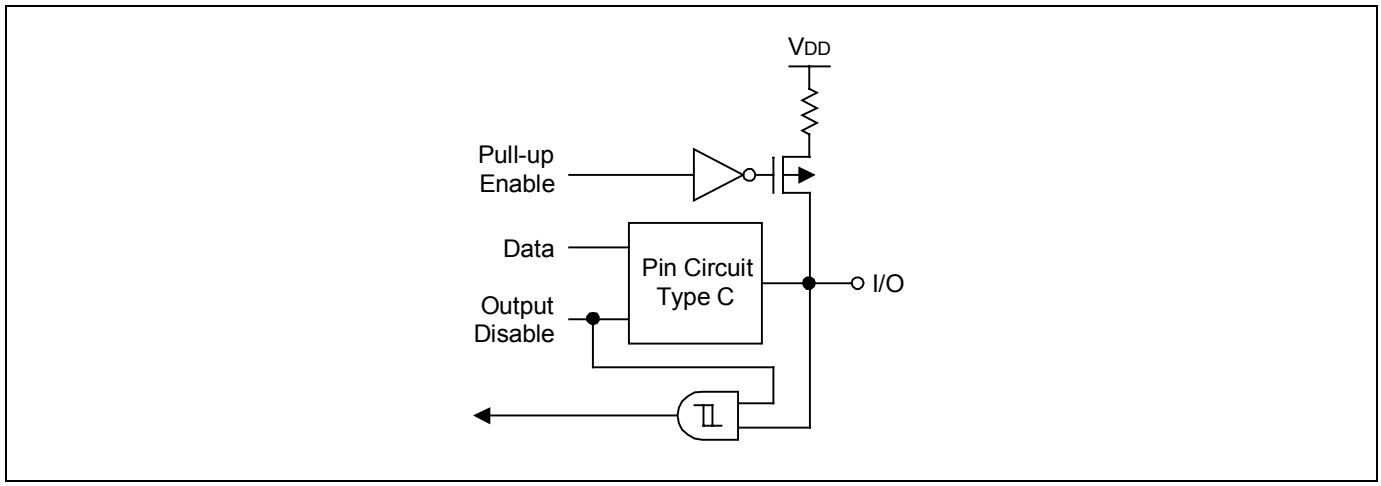


Figure 1-6. Pin Circuit Type D (P0, P1, P2 except P2.3, P3, P8 except P8.4, P8.5)

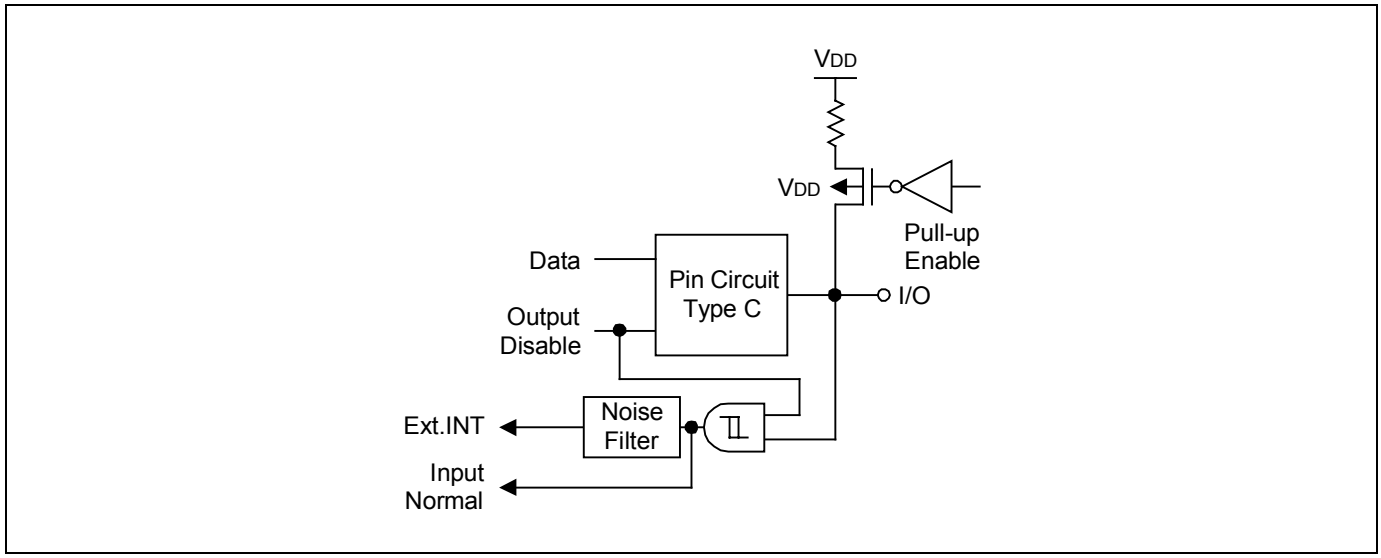


Figure 1-7. Pin Circuit Type D-1 (P4, P8.4, P8.5)

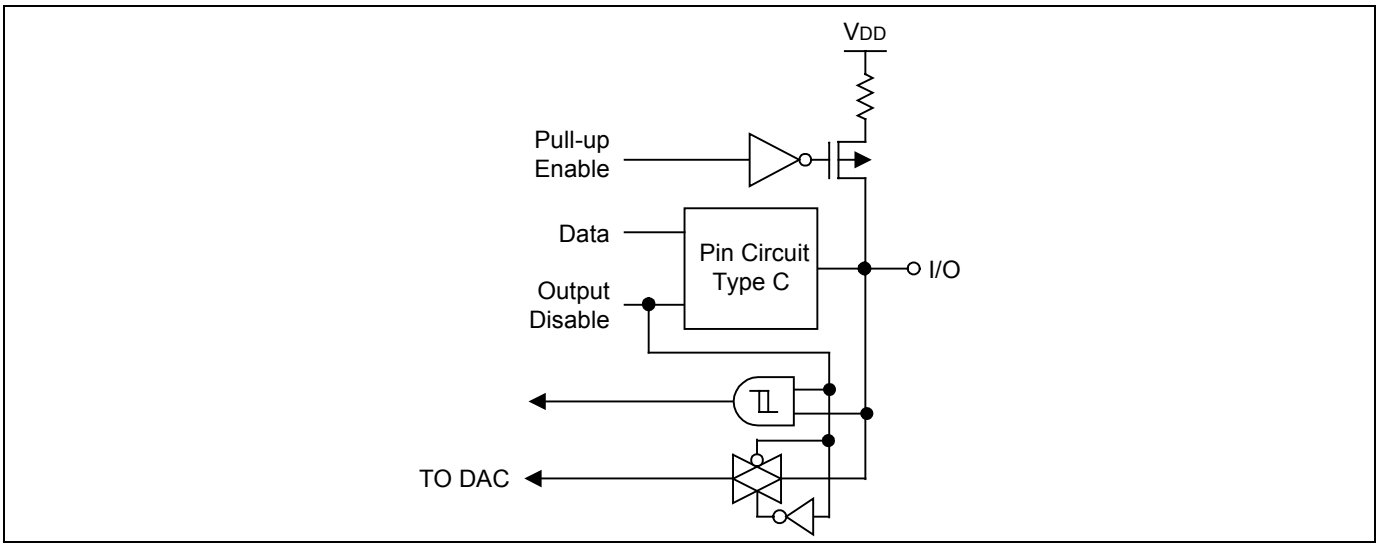


Figure 1-8. Pin Circuit Type D-2 (P2.3)

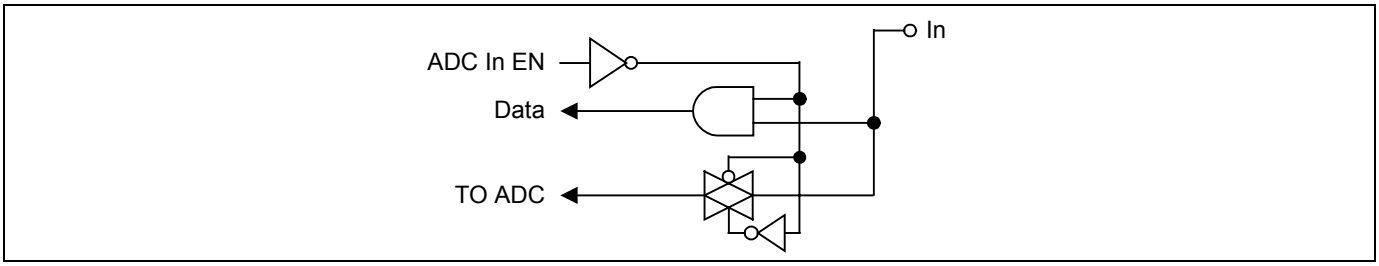


Figure 1-9. Pin Circuit Type E (ADC0-ADC7)

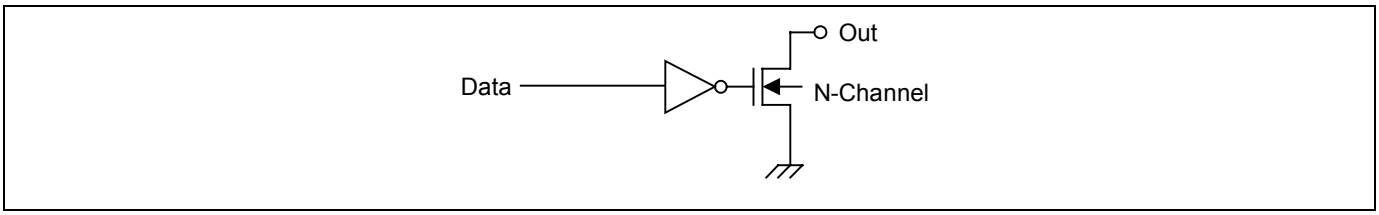


Figure 1-10. Pin Circuit Type F (P6)

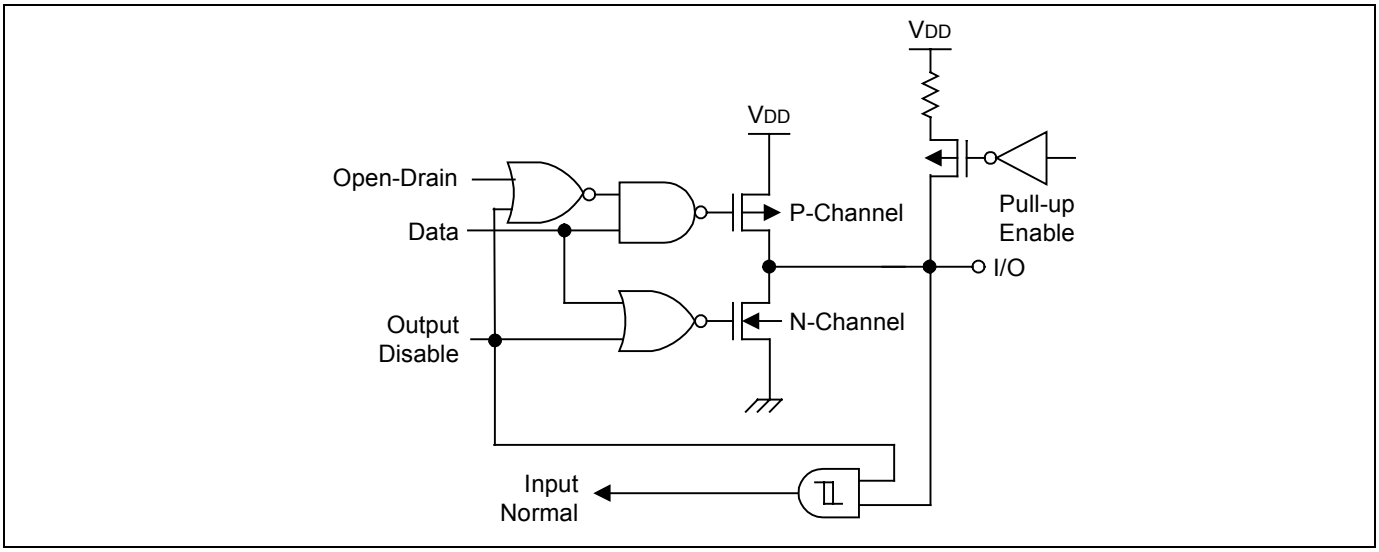


Figure 1-11. Pin Circuit Type G (P5.7-P5.4)

2 ADDRESS SPACES

OVERVIEW

The S3C84BB/F84BB microcontroller has two types of address space:

— Internal program memory (ROM)

— Internal register file (RAM)

A 16-bit address bus supports program memory operations. A separate 8-bit register bus carries addresses and data between the CPU and the register file.

The S3C84BB/F84BB has an internal 64-Kbyte mask-programmable ROM/FLASH ROM and 2064-byte RAM.

PROGRAM MEMORY (ROM)

Program memory (ROM) stores program codes or table data. The S3C84BB has 64-Kbytes of internal mask programmable program memory. The program memory address range is therefore 0H–FFFFH (see Figure 2-1).

The first 256 bytes of the ROM (0H-0FFH) are reserved for interrupt vector addresses. Unused locations in this address range can be used as normal program memory. If you use the vector address area to store a program code, be careful not to overwrite the vector addresses stored in these locations.

The ROM address at which a program execution starts after a reset is 0100H.

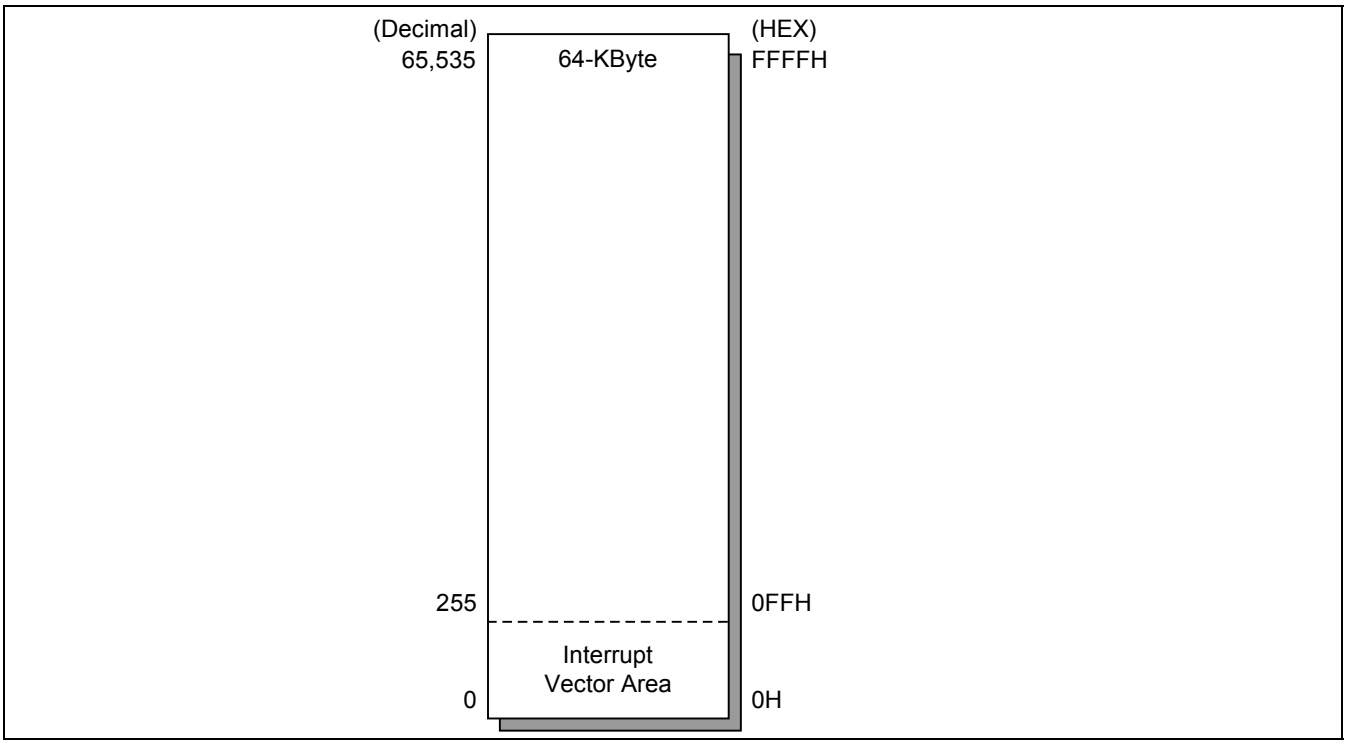


Figure 2-1. Program Memory Address Space

REGISTER ARCHITECTURE

In the S3C84BB/F84BB implementation, the upper 64-byte area of register files is expanded two 64-byte areas, called *set 1* and *set 2*. The upper 32-byte area of set 1 is further expanded two 32-byte register banks (bank 0 and bank 1), and the lower 32-byte area is a single 32-byte common area. In addition, set 2 is logically expanded 8 separately addressable register pages, page 0–page 7.

In case of S3C84BB/F84BB the total number of addressable 8-bit registers is 2,144. Of these 2,144 registers, 16 bytes are for CPU and system control registers, 64 bytes are for peripheral control and data registers, 16 bytes are used as a shared working registers, and 2,048 registers are for general-purpose use.

You can always address set 1 register locations, regardless of which of the 8 register pages is currently selected. Set 1 locations, however, can only be addressed using direct addressing modes.

The extension of register space into separately addressable areas (sets, banks, and pages) is supported by various addressing mode restrictions, the select bank instructions, SB0 and SB1, and the register page pointer (PP).

Specific register types and the area (in bytes) that they occupy in the register file are summarized in Table 2–1.

Table 2-1. S3C84BB/F84BB Register Type Summary

Register Type	Number of Bytes
General-purpose registers (including 16-byte common working register area, the 192-byte prime register area, and the 64-byte set 2 area)	2,064
CPU and system control registers	16
Mapped clock, peripheral, I/O control, and data registers	64
Total Addressable Bytes	2,144

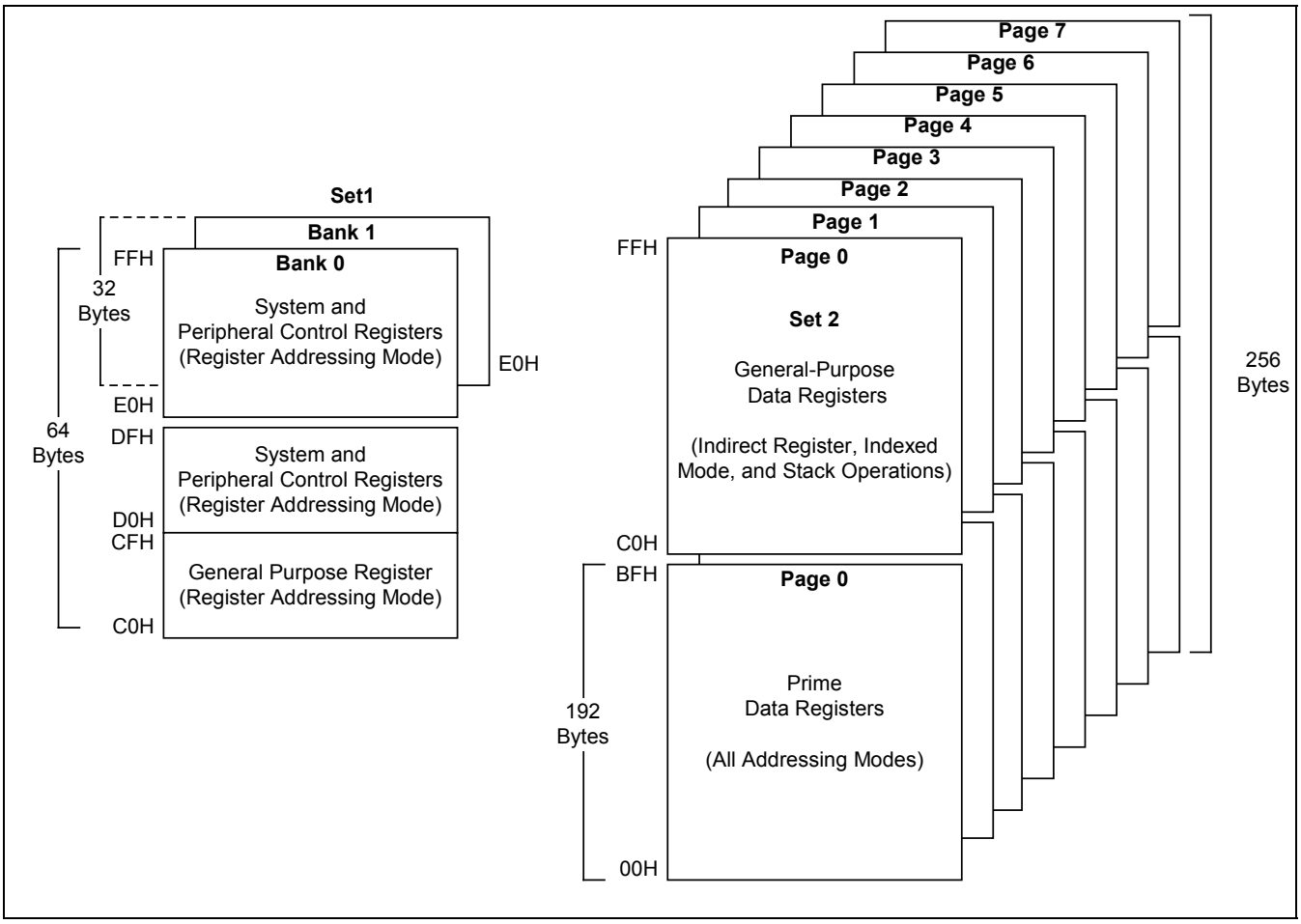


Figure 2-2. Internal Register File Organization

REGISTER PAGE POINTER (PP)

The S3C8-series architecture supports the logical expansion of the physical 2,064-byte internal register file (using an 8-bit data bus) into as many as 16 separately addressable register pages. Page addressing is controlled by the register page pointer (PP, DFH). In the S3C84BB/F84BB microcontroller, a paged register file expansion is implemented for data registers, and the register page pointer must be changed to address other pages.

After a reset, the page pointer's source value (lower nibble) and the destination value (upper nibble) are always "0000", automatically selecting page 0 as the source and destination page for register addressing.

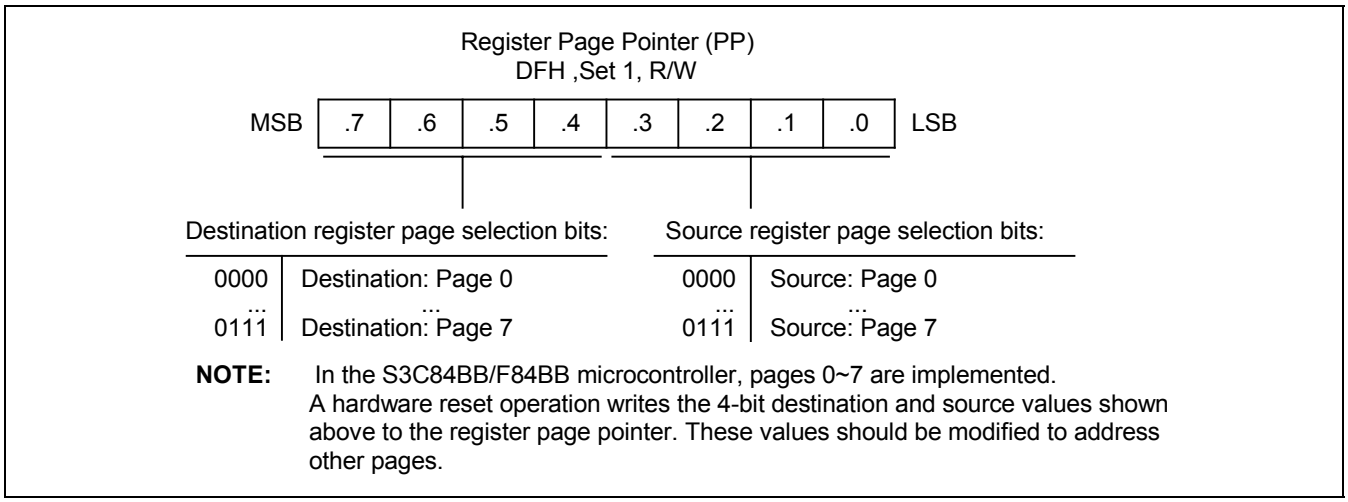


Figure 2-3. Register Page Pointer (PP)

☞PROGRAMMING TIP — Using the Page Pointer for RAM clear (Page 0, Page 1)

```
        LD      PP,#00H         ;  Destination ← 0, Source ← 0
        SRP     #0C0H
        LD      R0,#0FFH        ;  Page 0 RAM clear starts
RAMCL0  CLR     @R0
        DJNZ    R0,RAMCL0
        CLR     @R0             ;  R0 = 00H

        LD      PP,#10H         ;  Destination ← 1, Source ← 0
        LD      R0,#0FFH        ;  Page 1 RAM clear starts
RAMCL1  CLR     @R0
        DJNZ    R0,RAMCL1
        CLR     @R0             ;  R0 = 00H
```

NOTE: You should refer to page 6-39 and use DJNZ instruction properly when DJNZ instruction is used in your program.

REGISTER SET 1

The term *set 1* refers to the upper 64 bytes of the register file, locations C0H–FFH.

The upper 32-byte area of this 64-byte space (E0H–FFH) is expanded two 32-byte register banks, *bank 0* and *bank 1*. The set register bank instructions, SB0 or SB1, are used to address one bank or the other. A hardware reset operation always selects bank 0 addressing.

The upper two 32-byte areas (bank 0 and bank 1) of set 1 (E0H–FFH) contains 64 mapped system and peripheral control registers. The lower 32-byte area contains 16 system registers (D0H–DFH) and a 16-byte common working register area (C0H–CFH). You can use the common working register area as a “scratch” area for data operations being performed in other areas of the register file.

Registers in set 1 locations are directly accessible at all times using Register addressing mode. The 16-byte working register area can only be accessed using working register addressing (For more information about working register addressing, please refer to Chapter 3, “Addressing Modes.”)

REGISTER SET 2

The same 64-byte physical space that is used for set 1 locations C0H–FFH is logically duplicated to add another 64 bytes of register space. This expanded area of the register file is called set *2*. For the S3C84BB/F84BB, the set 2 address range (C0H–FFH) is accessible on pages 0-7.

The logical division of set 1 and set 2 is maintained by means of addressing mode restrictions. You can use only Register addressing mode to access set 1 locations. In order to access registers in set 2, you must use Register Indirect addressing mode or Indexed addressing mode.

The set 2 register area is commonly used for stack operations.

PRIME REGISTER SPACE

The lower 192 bytes (00H–BFH) of the S3C84BB/F84BB's eight 256-byte register pages is called *prime register area.* Prime registers can be accessed using any of the seven addressing modes (see Chapter 3, "Addressing Modes.")

The prime register area on page 0 is immediately addressable following a reset. In order to address prime registers on pages 0, or 1 you must set the register page pointer (PP) to the appropriate source and destination values.

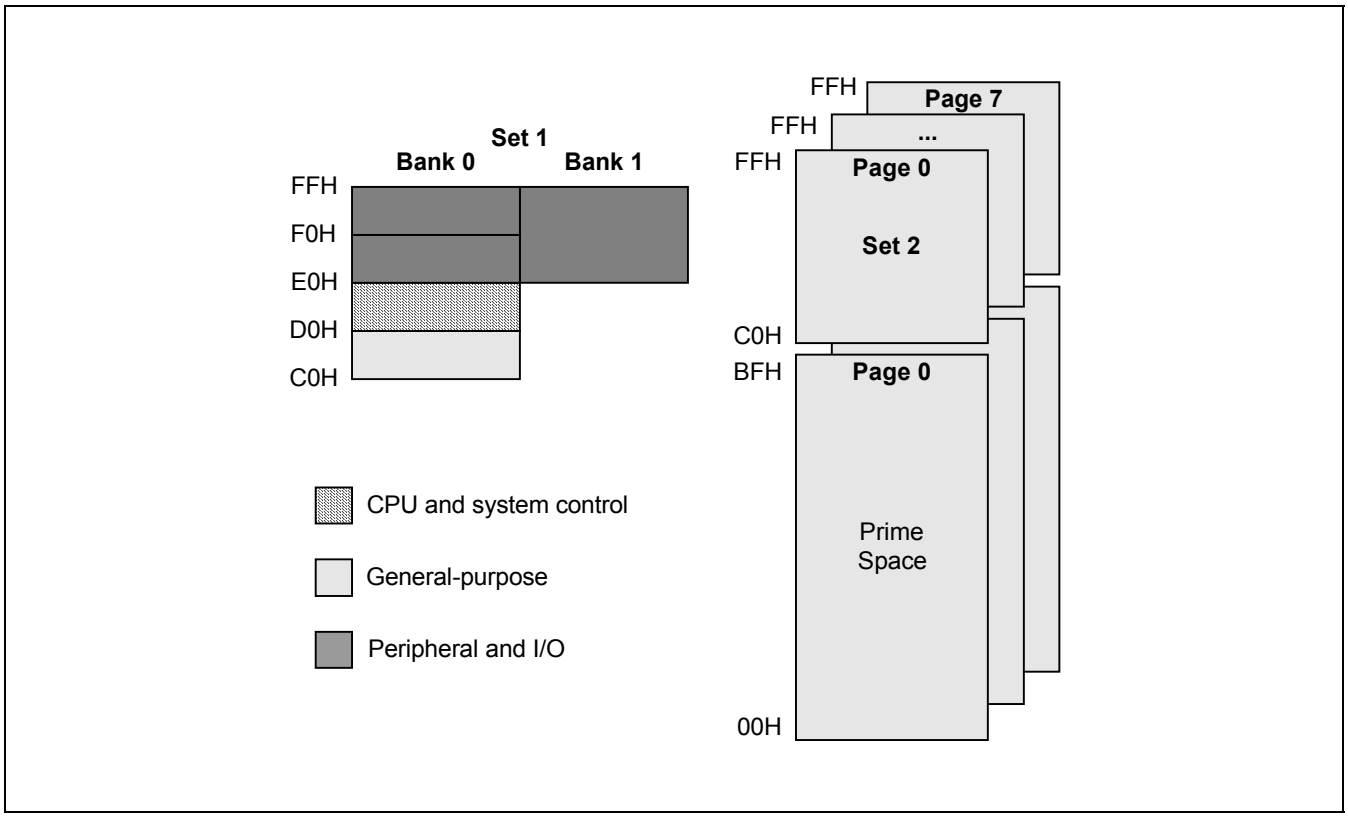


Figure 2-4. Set 1, Set 2, Prime Area Register

WORKING REGISTERS

Instructions can access specific 8-bit registers or 16-bit register pairs using either 4-bit or 8-bit address fields. When 4-bit working register addressing is used, the 256-byte register file can be seen by the programmer as one that consists of 32 8-byte register groups or "slices." Each slice comprises of eight 8-bit registers.

Using the two 8-bit register pointers, RP1 and RP0, two working register slices can be selected at any one time to form a 16-byte working register block. Using the register pointers, you can move this 16-byte register block anywhere in the addressable register file, except for the set 2 area.

The terms slice and block are used in this manual to help you visualize the size and relative locations of selected working register spaces:

— One working register *slice* is 8 bytes (eight 8-bit working registers, R0–R7 or R8–R15)

— One working register *block* is 16 bytes (sixteen 8-bit working registers, R0–R15)

All the registers in an 8-byte working register slice have the same binary value for their five most significant address bits. This makes it possible for each register pointer to point to one of the 24 slices in the register file other than set 2. The base addresses for the two selected 8-byte register slices are contained in register pointers RP0 and RP1.

After a reset, RP0 and RP1 always point to the 16-byte common area in set 1 (C0H–CFH).

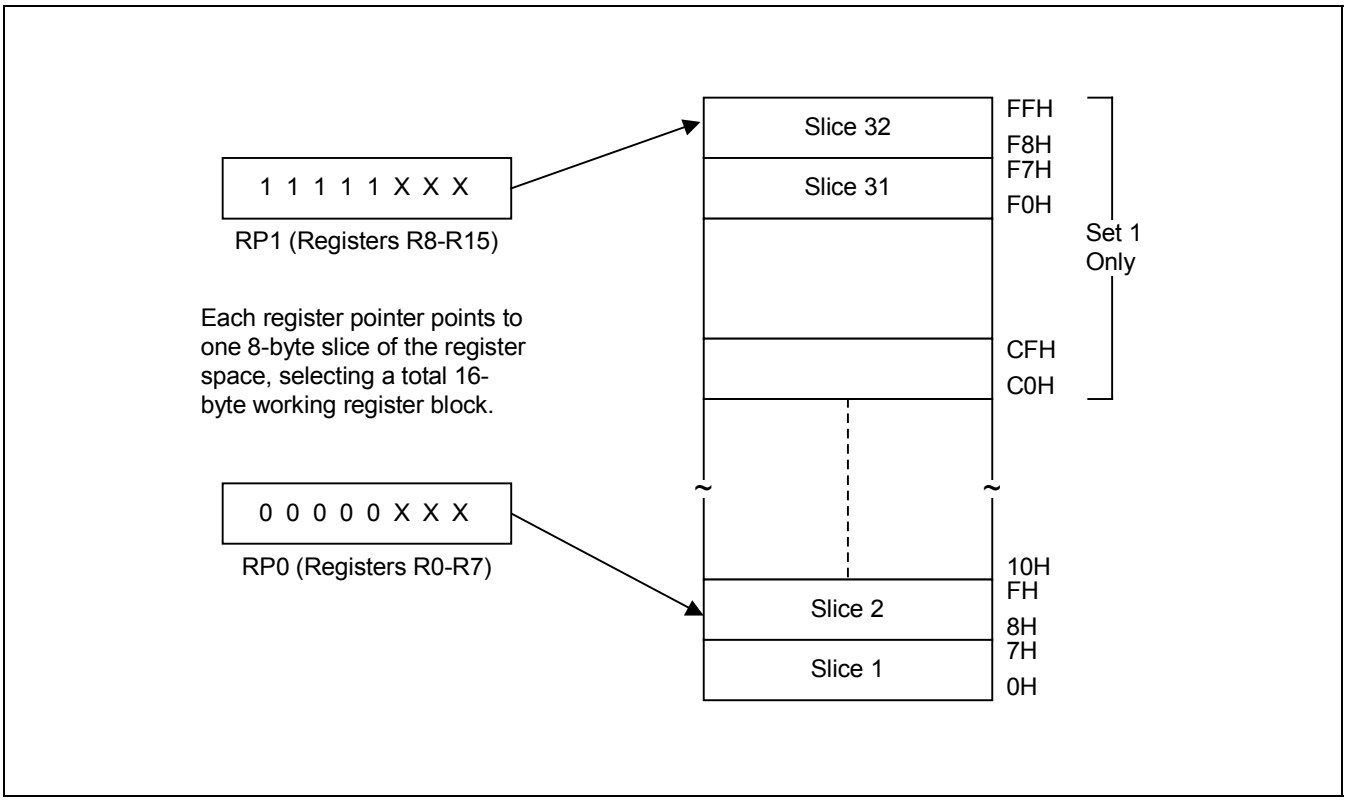


Figure 2-5. 8-Byte Working Register Areas (Slices)

USING THE REGISTER POINTERS

After a reset, RP# point to the working register common area: RP0 points to addresses C0H–C7H, and RP1 points to addresses C8H–CFH.

To change a register pointer value, you load a new value to RP0 and/or RP1 using an SRP or LD instruction. (see Figures 2-6 and 2-7).

With working register addressing, you can only access those two 8-bit slices of the register file that are currently pointed to by RP0 and RP1. You can not, however, use the register pointers to select a working register space in set 2, C0H–FFH, because these locations can be accessed only using the Indirect Register or Indexed addressing modes.

The selected 16-byte working register block usually consists of two contiguous 8-byte slices. As a general programming guideline, it is recommended that RP0 point to the "lower" slice and RP1 point to the "upper" slice (see Figure 2-6).

Because a register pointer can point to either of the two 8-byte slices in the working register block, you can flexibly define the working register area to support program requirements.

☞PROGRAMMING TIP — Setting the Register Pointers

```
SRP     #70H          ;  RP0 ← 70H, RP1 ← 78H
SRP1    #48H          ;  RP0 ← no change, RP1 ← 48H,
SRP0    #0A0H         ;  RP0 ← A0H, RP1 ← no change
CLR     RP0           ;  RP0 ← 00H, RP1 ← no change
LD      RP1,#0F8H     ;  RP0 ← no change, RP1 ← 0F8H
```

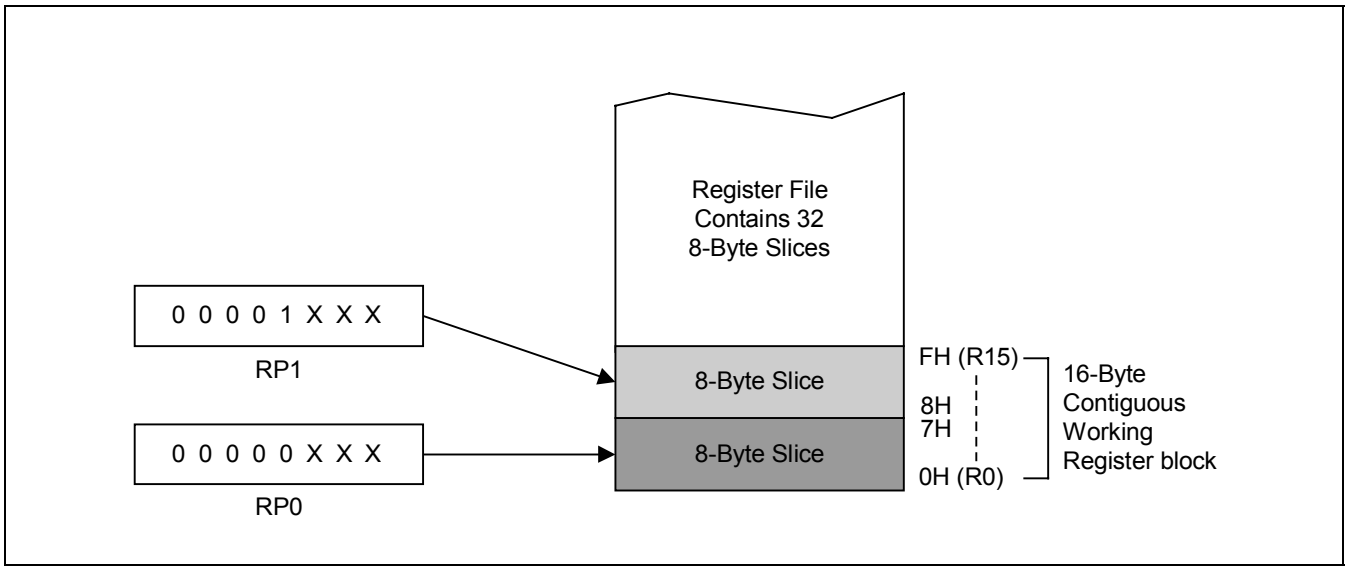


Figure 2-6. Contiguous 16-Byte Working Register Block

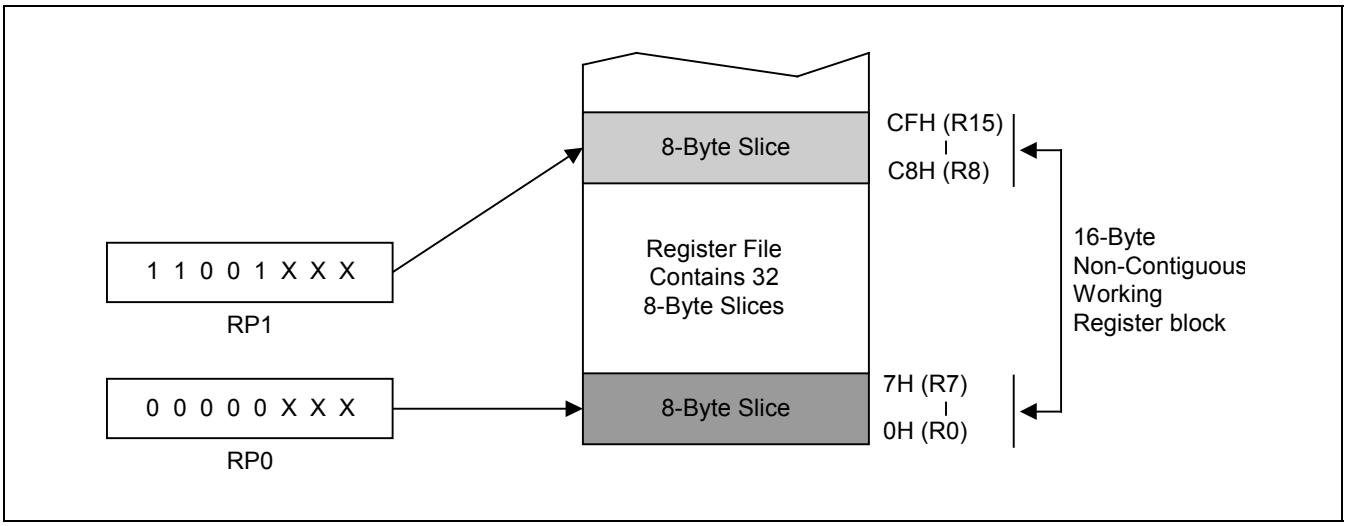


Figure 2-7. Non-Contiguous 16-Byte Working Register Block

☞PROGRAMMING TIP — Using the RPs to Calculate the Sum of a Series of Registers

Calculate the sum of registers 80H–85H using the register pointer. The register addresses from 80H through 85H contain the values 10H, 11H, 12H, 13H, 14H, and 15H, respectively:

```
SRP0    #80H        ;  RP0 ← 80H
ADD     R0,R1       ;  R0 ← R0 + R1
ADC     R0,R2       ;  R0 ← R0 + R2 + C
ADC     R0,R3       ;  R0 ← R0 + R3 + C
ADC     R0,R4       ;  R0 ← R0 + R4 + C
ADC     R0,R5       ;  R0 ← R0 + R5 + C
```

The sum of these six registers, 6FH, is located in the register R0 (80H). The instruction string used in this example takes 12 bytes of instruction code and its execution time is 36 cycles. If the register pointer is not used to calculate the sum of these registers, the following instruction sequence would have to be used:

```
ADD     80H,81H     ;  80H ← (80H) + (81H)
ADC     80H,82H     ;  80H ← (80H) + (82H) + C
ADC     80H,83H     ;  80H ← (80H) + (83H) + C
ADC     80H,84H     ;  80H ← (80H) + (84H) + C
ADC     80H,85H     ;  80H ← (80H) + (85H) + C
```

Now, the sum of the six registers is also located in register 80H. However, this instruction string takes 15 bytes of instruction code rather than 12 bytes, and its execution time is 50 cycles rather than 36 cycles.

REGISTER ADDRESSING

The S3C8-series register architecture provides an efficient method of working register addressing that takes full advantage of shorter instruction formats to reduce execution time.

With Register (R) addressing mode, in which the operand value is the content of a specific register or register pair, you can access any location in the register file except for set 2. With working register addressing, you use a register pointer to specify an 8-byte working register space in the register file and an 8-bit register within that space.

Registers are addressed either as a single 8-bit register or as a paired 16-bit register space. In a 16-bit register pair, the address of the first 8-bit register is always an even number and the address of the next register is always an odd number. The most significant byte of the 16-bit data is always stored in the even-numbered register, and the least significant byte is always stored in the next (+1) odd-numbered register.

Working register addressing differs from Register addressing as it uses a register pointer to identify a specific 8-byte working register space in the internal register file and a specific 8-bit register within that space.

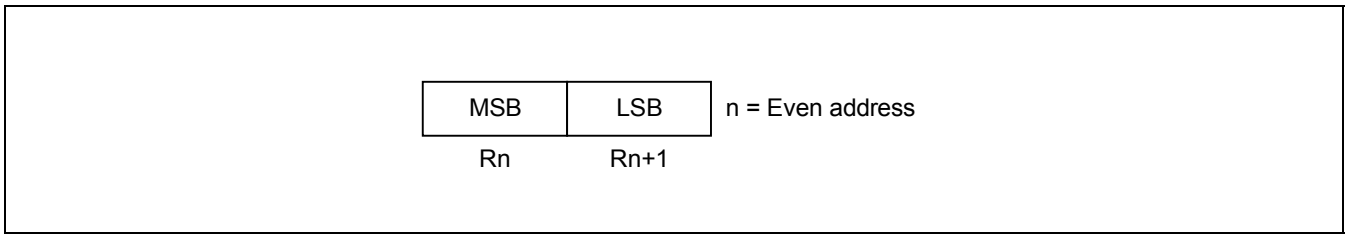


Figure 2-8. 16-Bit Register Pair

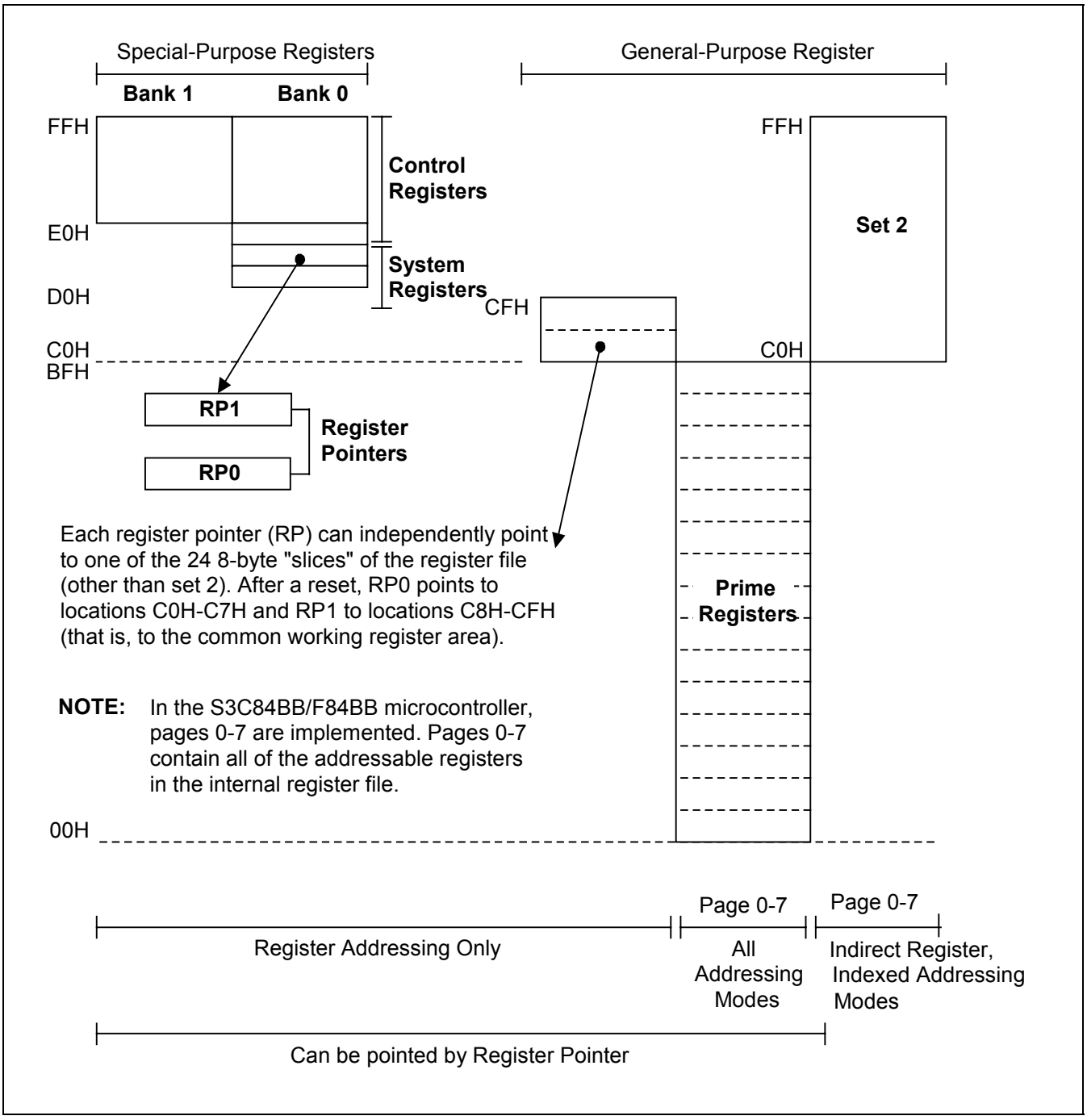


Figure 2-9. Register File Addressing

COMMON WORKING REGISTER AREA (C0H–CFH)

After a reset, register pointers RP0 and RP1 automatically select two 8-byte register slices in set 1, locations C0H–CFH, as the active 16-byte working register block:

RP0 → C0H–C7H

RP1 → C8H–CFH

This 16-byte address range is called *common area*. That is, locations in this area can be used as working registers by operations that address any location on any page in the register file. Typically, these working registers serve as temporary buffers for data operations between different pages.

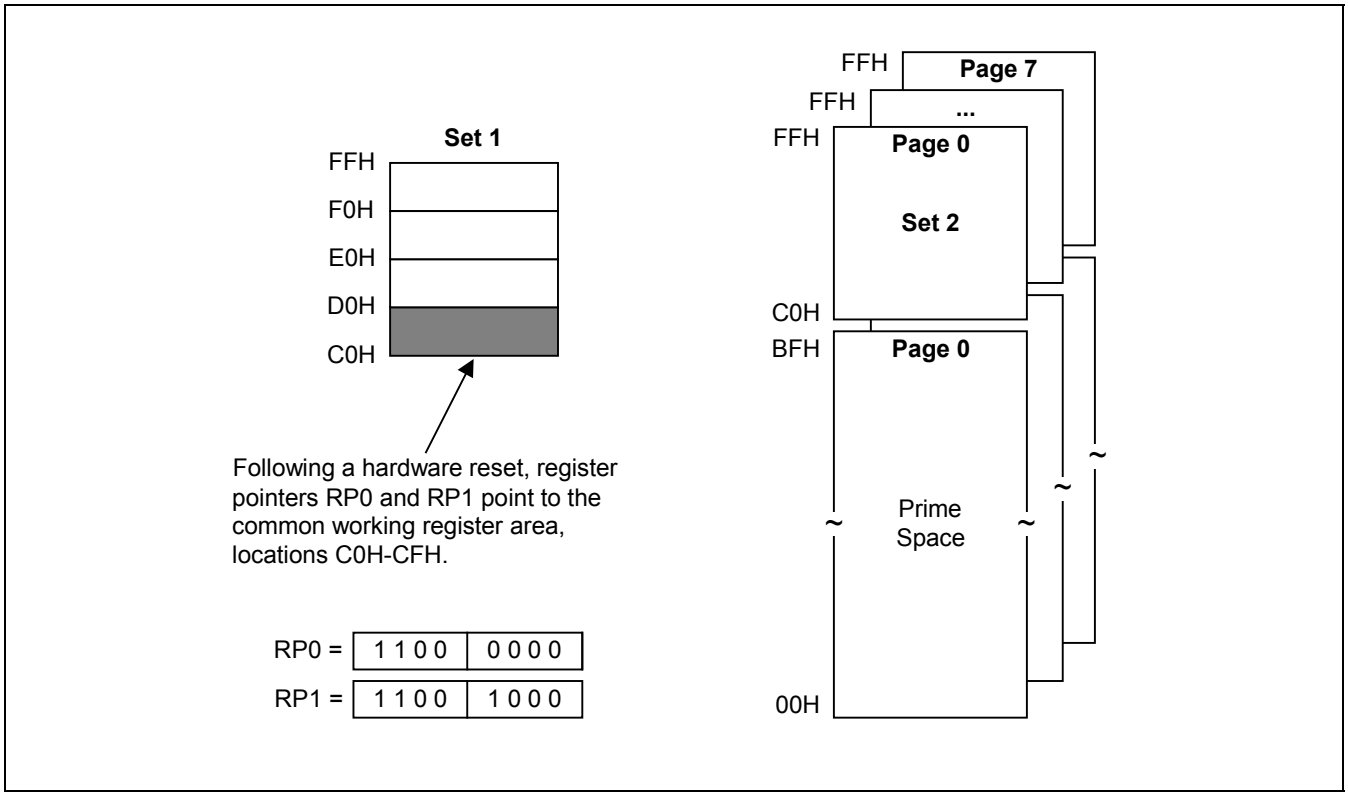


Figure 2-10. Common Working Register Area

☞PROGRAMMING TIP — Addressing the Common Working Register Area

As the following examples show, you should access working registers in the common area, locations C0H–CFH, using working register addressing mode only.

Examples 1:
```
LD      0C2H,40H        ;  Invalid addressing mode!
```
Use working register addressing instead:
```
SRP     #0C0H
LD      R2,40H          ;  R2 (C2H) ← the value in location 40H
```

Examples 2:
```
ADD     0C3H,#45H       ;  Invalid addressing mode!
```
Use working register addressing instead:
```
SRP     #0C0H
ADD     R3,#45H         ;  R3 (C3H) ← R3 + 45H
```

4-BIT WORKING REGISTER ADDRESSING

Each register pointer defines a movable 8-byte slice of working register space. The address information stored in a register pointer serves as an addressing "window" that makes it possible for instructions to access working registers very efficiently using short 4-bit addresses. When an instruction addresses a location in the selected working register area, the address bits are concatenated in the following way to form a complete 8-bit address:

— The high-order bit of the 4-bit address selects one of the register pointers ("0" selects RP0, "1" selects RP1).

— The five high-order bits in the register pointer select an 8-byte slice of the register space.

— The three low-order bits of the 4-bit address select one of the eight registers in the slice.

As shown in Figure 2-11, the result of this operation is that the five high-order bits from the register pointer are concatenated with the three low-order bits from the instruction address to form the complete address. As long as the address stored in the register pointer remains unchanged, the three bits from the address will always point to an address in the same 8-byte register slice.

Figure 2-12 shows a typical example of 4-bit working register addressing. The high-order bit of the instruction "INC R6" is "0", which selects RP0. The five high-order bits stored in RP0 (01110B) are concatenated with the three low-order bits of the instruction's 4-bit address (110B) to produce the register address 76H (01110110B).

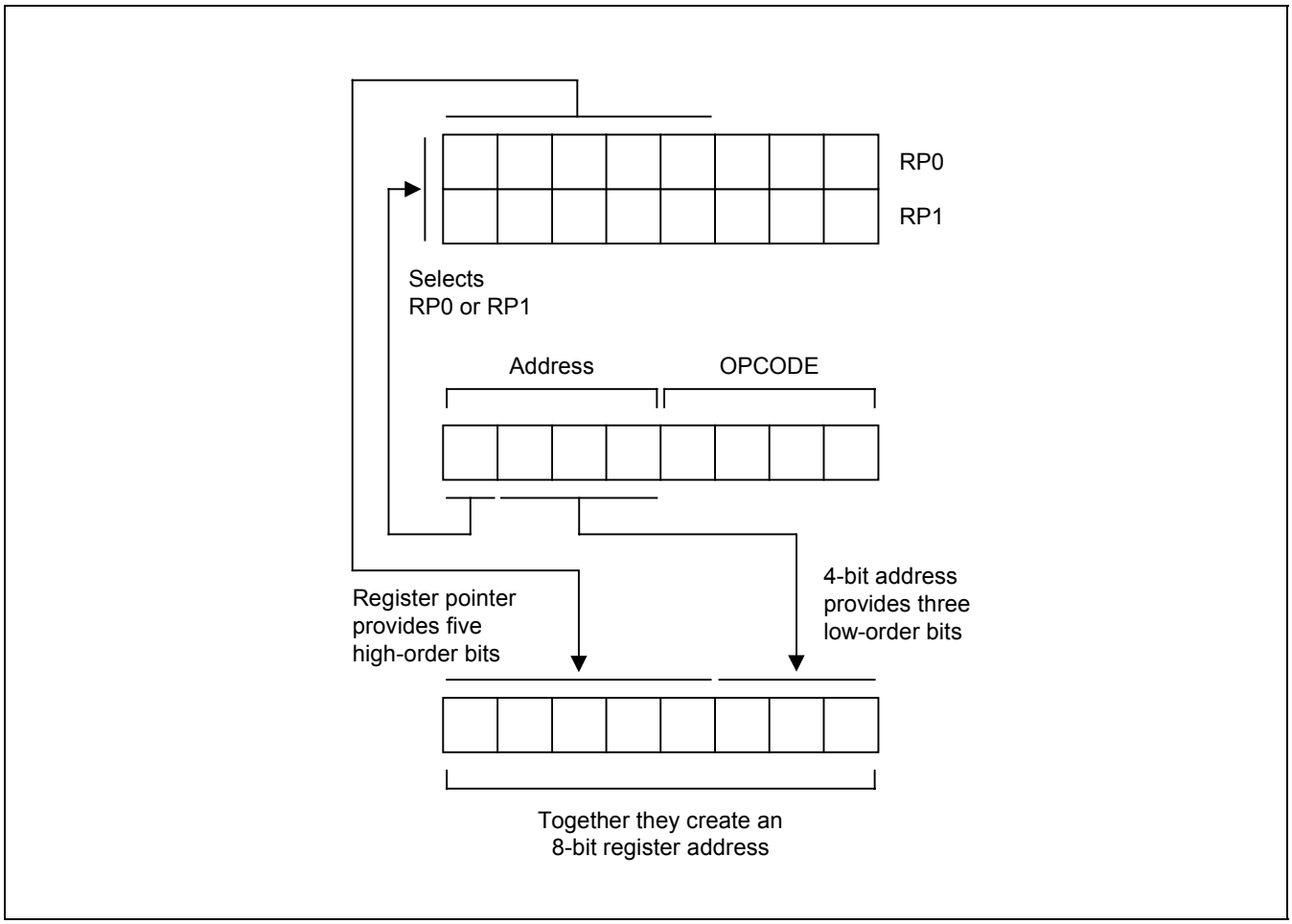


Figure 2-11. 4-Bit Working Register Addressing

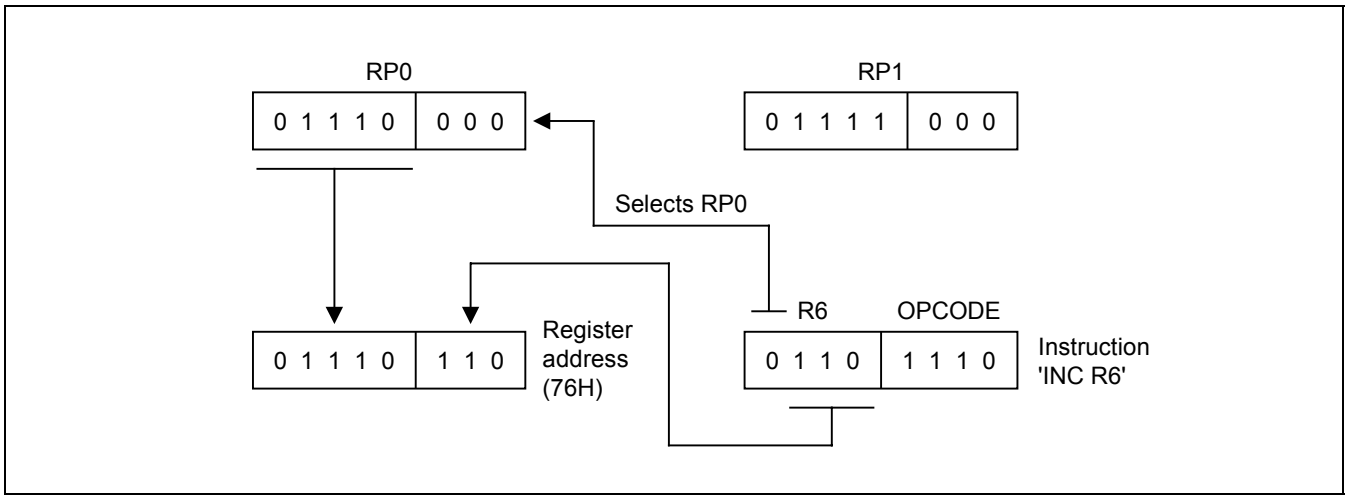


Figure 2-12. 4-Bit Working Register Addressing Example

8-BIT WORKING REGISTER ADDRESSING

You can also use 8-bit working register addressing to access registers in a selected working register area. To initiate 8-bit working register addressing, the upper four bits of the instruction address must contain the value "1100B." This 4-bit value (1100B) indicates that the remaining four bits have the same effect as 4-bit working register addressing.

As shown in Figure 2-13, the lower nibble of the 8-bit address is concatenated in much the same way as for 4-bit addressing. Bit 3 selects either RP0 or RP1, which then supplies the five high-order bits of the final address, the three low-order bits of the complete address are provided by the original instruction.

Figure 2-14 shows an example of 8-bit working register addressing. The four high-order bits of the instruction address (1100B) specify 8-bit working register addressing. Bit 3 ("1") selects RP1 and the five high-order bits in RP1 (10101B) become the five high-order bits of the register address. The three low-order bits of the register address (011) are provided by the three low-order bits of the 8-bit instruction address. The five address bits from RP1 and the three address bits from the instruction are concatenated to form the complete register address, 0ABH (10101011B).

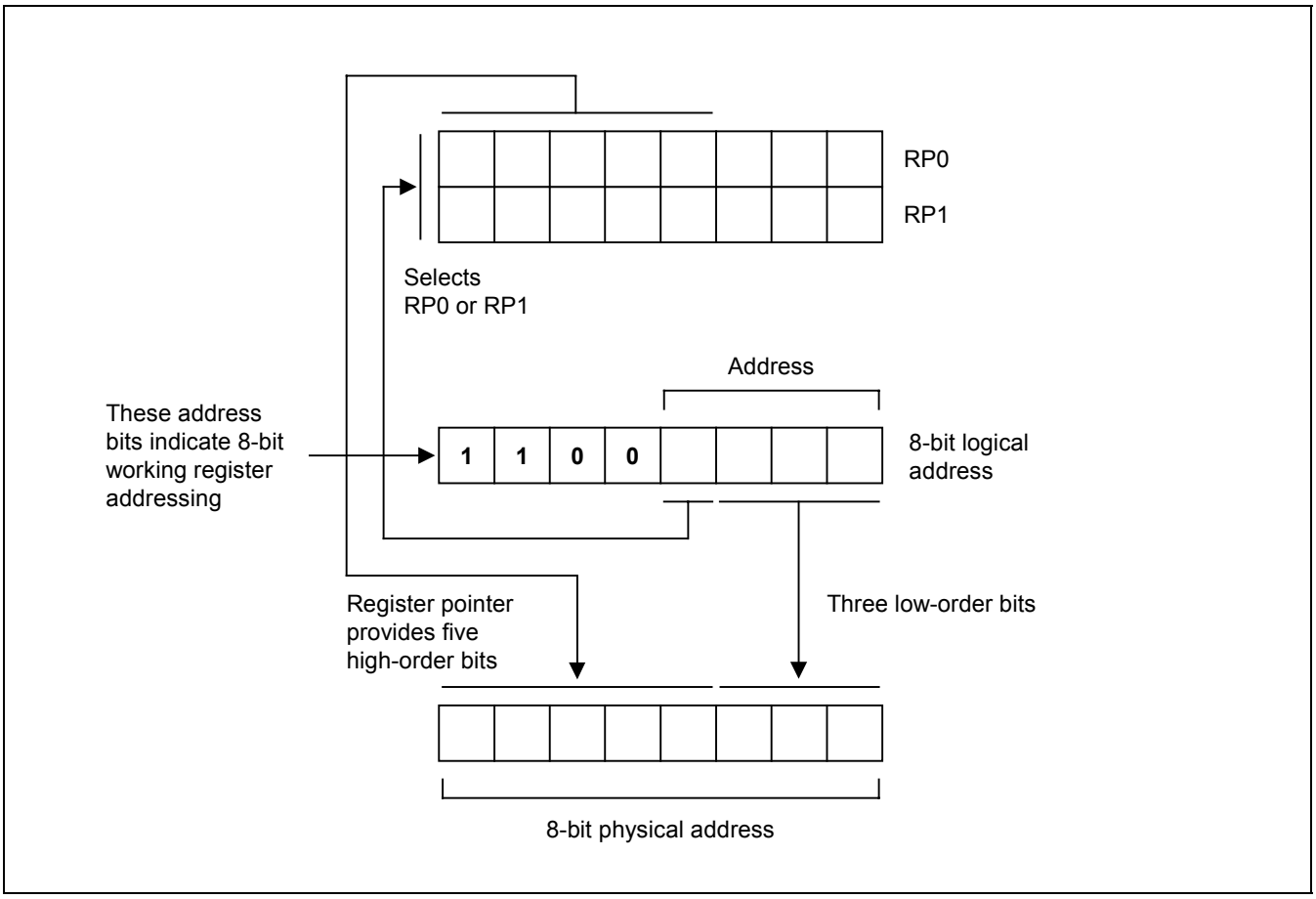


Figure 2-13. 8-Bit Working Register Addressing

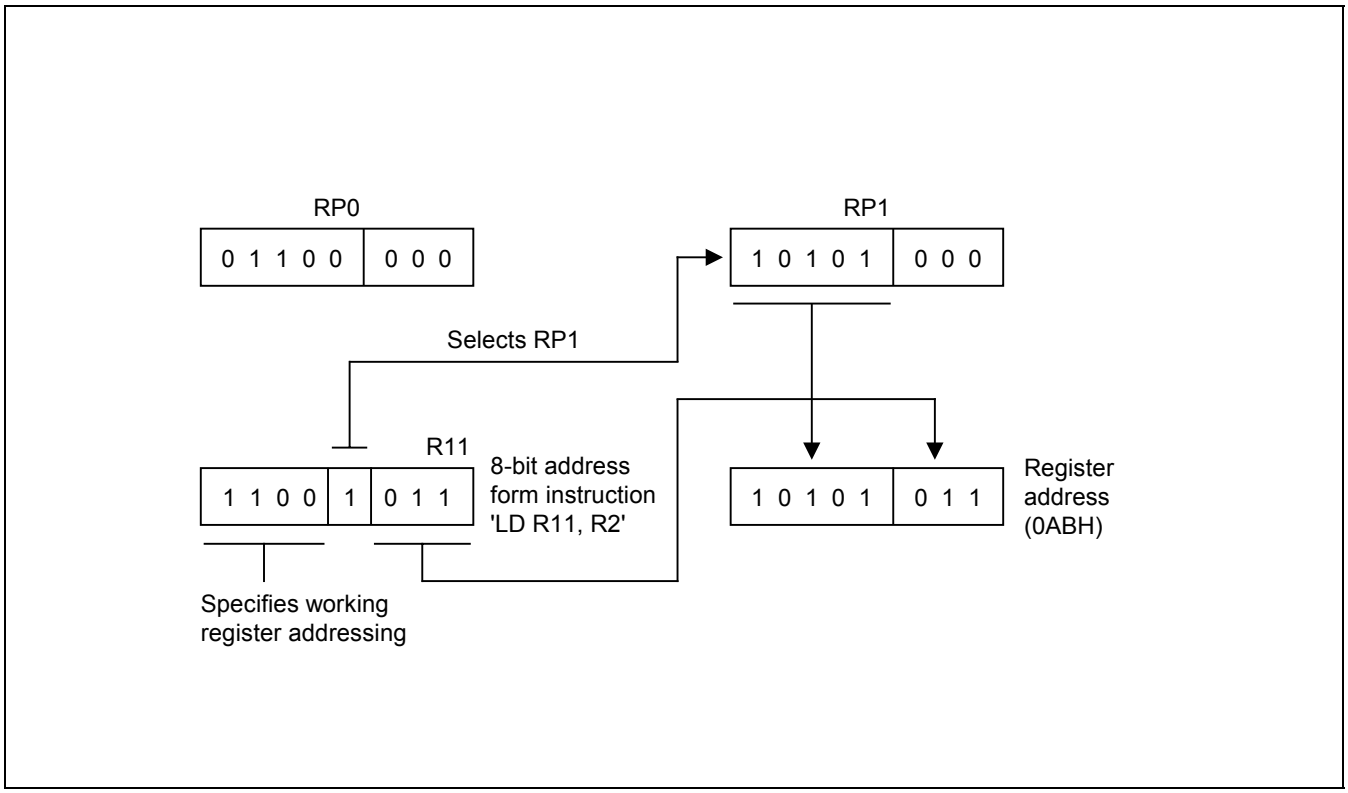


Figure 2-14. 8-Bit Working Register Addressing Example

SYSTEM AND USER STACK

The S3C8-series microcontrollers use the system stack for data storage, subroutine calls and returns. The PUSH and POP instructions are used to control system stack operations. The S3C84BB/F84BB architecture supports stack operations in the internal register file.

Stack Operations

Return addresses for procedure calls, interrupts, and data are stored on the stack. The contents of the PC are saved to stack by a CALL instruction and restored by the RET instruction. When an interrupt occurs, the contents of the PC and the FLAGS registers are pushed to the stack. The IRET instruction then pops these values back to their original locations. The stack address value is always decreased by one before a push operation and increased by one *after* a pop operation. The stack pointer (SP) always points to the stack frame stored on the top of the stack, as shown in Figure 2-15.

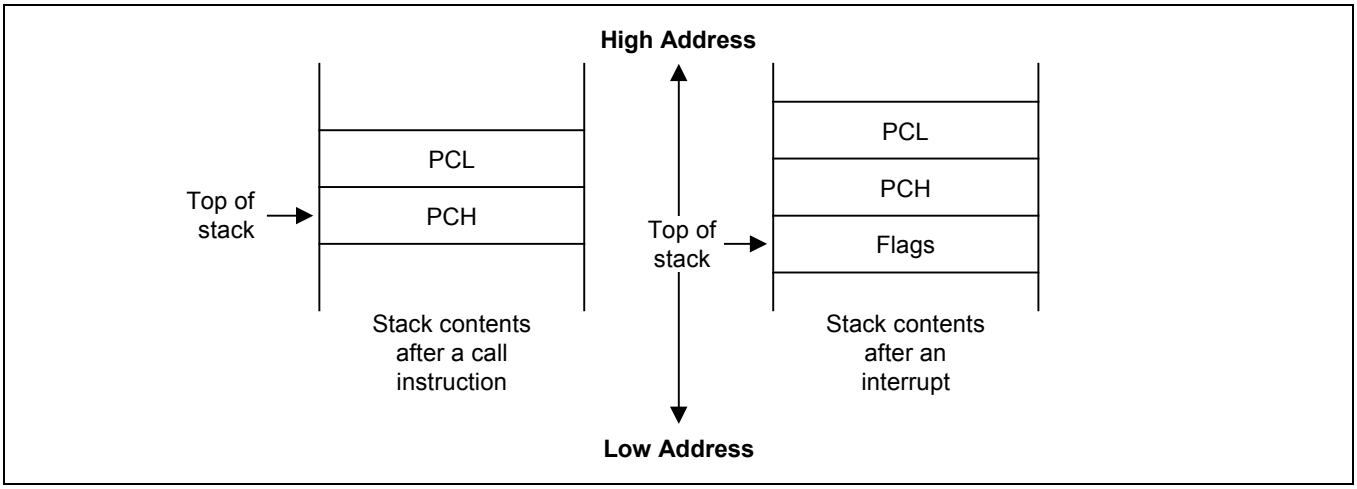


Figure 2-15. Stack Operations

User-Defined Stacks

You can freely define stacks in the internal register file as data storage locations. The instructions PUSHUI, PUSHUD, POPUI, and POPUD support user-defined stack operations.

Stack Pointers (SPL, SPH)

Register locations D8H and D9H contain the 16-bit stack pointer (SP) that is used for system stack operations. The most significant byte of the SP address, SP15–SP8, is stored in the SPH register (D8H), and the least significant byte, SP7–SP0, is stored in the SPL register (D9H). After a reset, the SP value is undetermined.

Because only internal memory space is implemented in the S3C84BB/F84BB, the SPL must be initialized to an 8-bit value in the range 00H–FFH. The SPH register is not needed and can be used as a general-purpose register, if necessary.

When the SPL register contains the only stack pointer value (that is, when it points to a system stack in the register file), you can use the SPH register as a general-purpose data register. However, if an overflow or underflow condition occurs as a result of increasing or decreasing the stack address value in the SPL register during normal stack operations, the value in the SPL register will overflow (or underflow) to the SPH register, overwriting any other data that is currently stored there. To avoid overwriting data in the SPH register, you can initialize the SPL value to "FFH" instead of "00H".

☞PROGRAMMING TIP — Standard Stack Operations Using PUSH and POP

The following example shows you how to perform stack operations in the internal register file using PUSH and POP instructions:

```
LD      SPL,#0FFH       ; SPL ← FFH
                        ; (Normally, the SPL is set to 0FFH by the initialization
                        ; routine)
•
•
•
PUSH    PP              ; Stack address 0FEH ← PP
PUSH    RP0             ; Stack address 0FDH ← RP0
PUSH    RP1             ; Stack address 0FCH ← RP1
PUSH    R3              ; Stack address 0FBH ← R3
•
•
•
POP     R3              ; R3 ← Stack address 0FBH
POP     RP1             ; RP1 ← Stack address 0FCH
POP     RP0             ; RP0 ← Stack address 0FDH
POP     PP              ; PP ← Stack address 0FEH
```

NOTES

3 ADDRESSING MODES

OVERVIEW

Instructions that are stored in program memory are fetched for execution using the program counter. Instructions indicate the operation to be performed and the data to be operated on. Addressing mode is the method used to determine the location of the data operand. The operands specified in SAM88RC instructions may be condition codes, immediate data, or a location in the register file, program memory, or data memory.

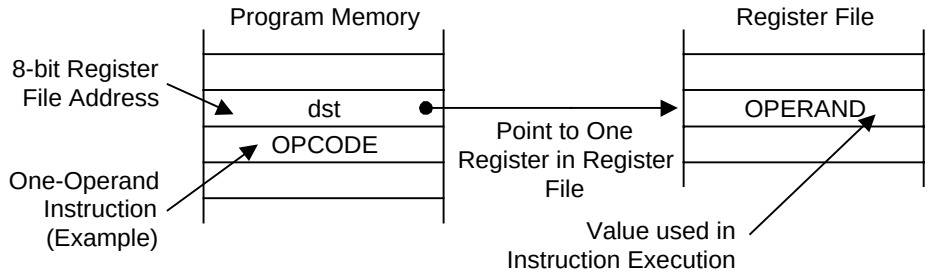
The S3C8-series instruction set supports seven explicit addressing modes. Not all of these addressing modes are available for each instruction. The seven addressing modes and their symbols are:

— Register (R)
— Indirect Register (IR)
— Indexed (X)
— Direct Address (DA)
— Indirect Address (IA)
— Relative Address (RA)
— Immediate (IM)

REGISTER ADDRESSING MODE (R)

In Register addressing mode (R), the operand value is the content of a specified register or register pair (see Figure 3-1).

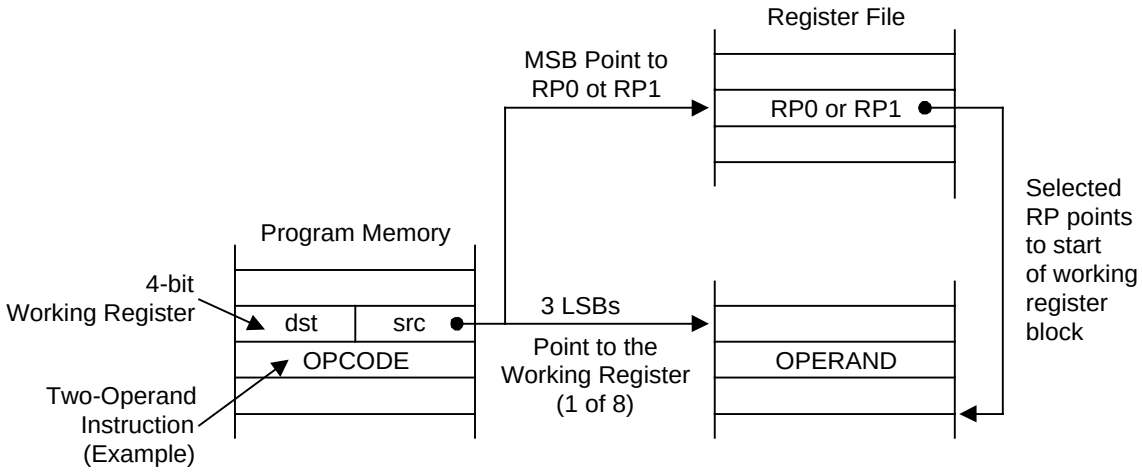
Working register addressing differs from Register addressing in that it uses a register pointer to specify an 8-byte working register space in the register file and an 8-bit register within that space (see Figure 3-2).



Sample Instruction:

```
DEC     CNTR            ;   Where CNTR is the label of an 8-bit register address
```

Figure 3-1. Register Addressing



Sample Instruction:

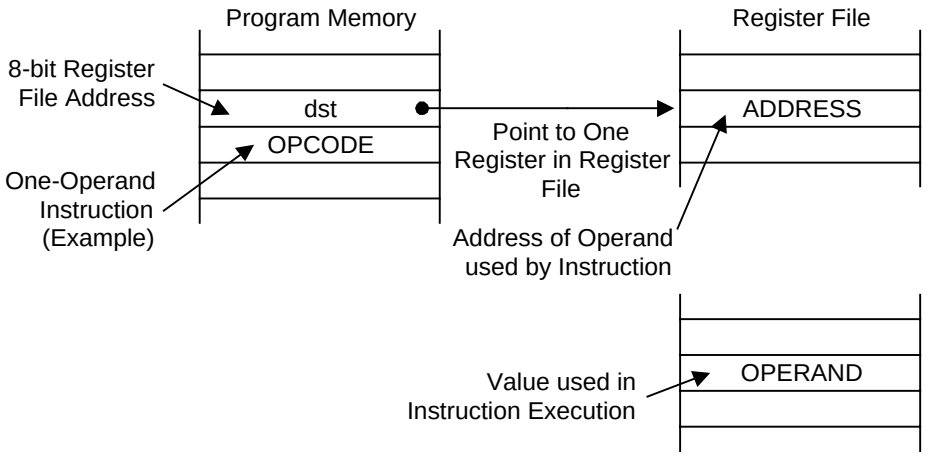
```
ADD     R1, R2          ;   Where R1 and R2 are registers in the currently
                            selected working register area.
```

Figure 3-2. Working Register Addressing

INDIRECT REGISTER ADDRESSING MODE (IR)

In Indirect Register (IR) addressing mode, the content of the specified register or register pair is the address of the operand. Depending on the instruction used, the actual address may point to a register in the register file, to program memory (ROM), or to an external memory space (see Figures 3-3 through 3-6).

You can use any 8-bit register to indirectly address another register. Any 16-bit register pair can be used to indirectly address another memory location. Please note, however, that you cannot access locations C0H–FFH in set 1 using the Indirect Register addressing mode.



Sample Instruction:

```
RL      @SHIFT          ;   Where SHIFT is the label of an 8-bit register address
```

Figure 3-3. Indirect Register Addressing to Register File

INDIRECT REGISTER ADDRESSING MODE (Continued)

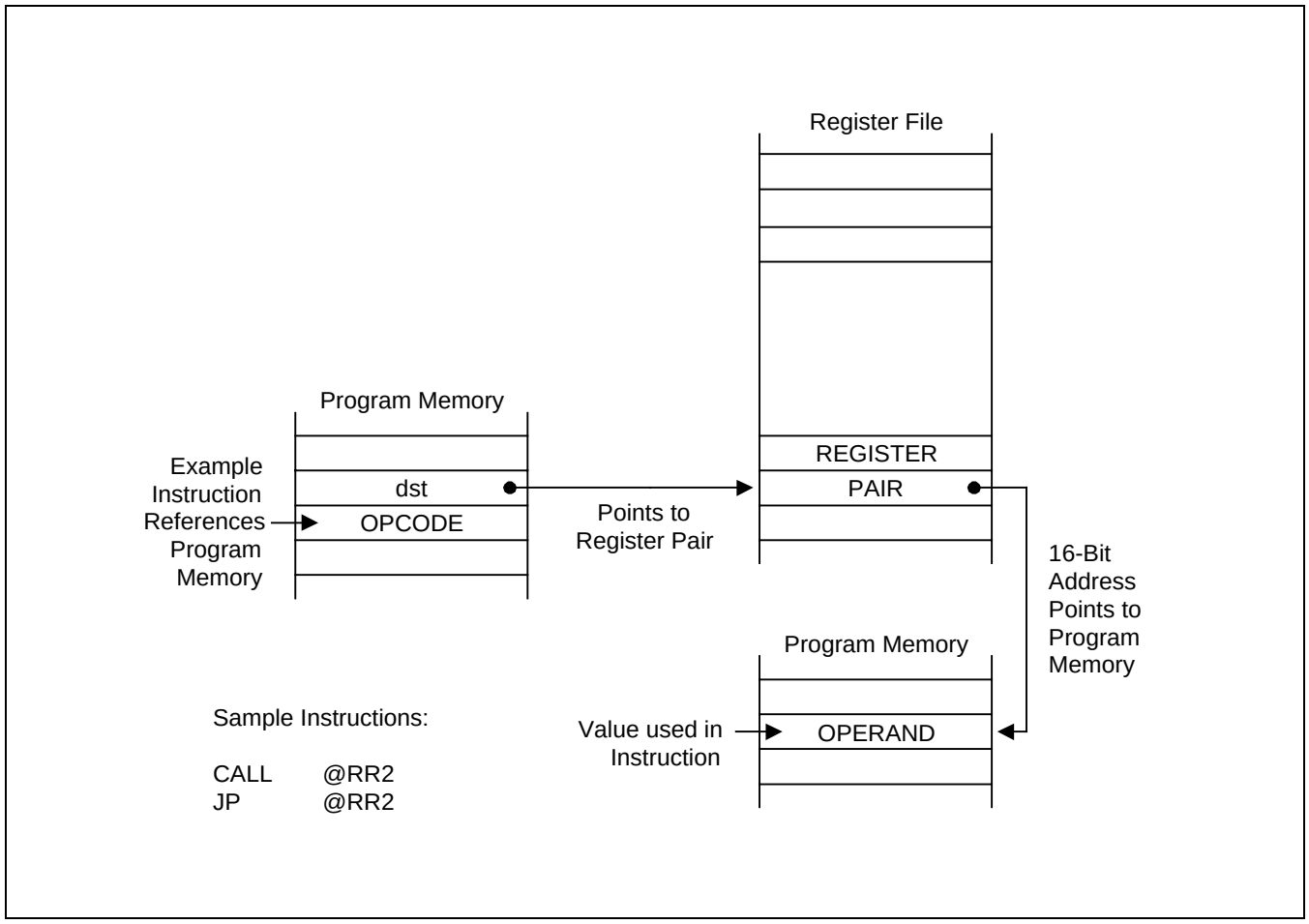


Figure 3-4. Indirect Register Addressing to Program Memory

INDIRECT REGISTER ADDRESSING MODE (Continued)

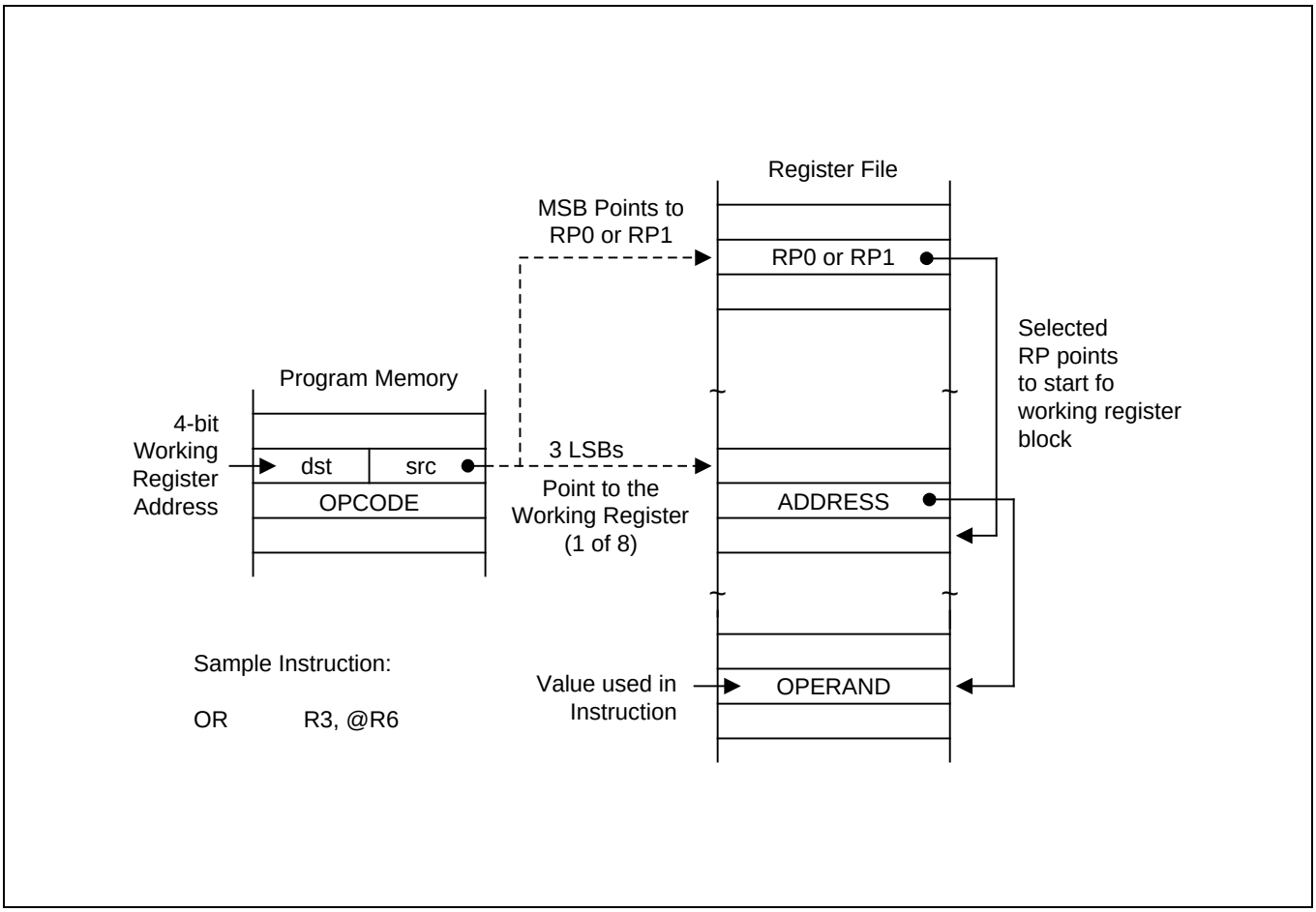
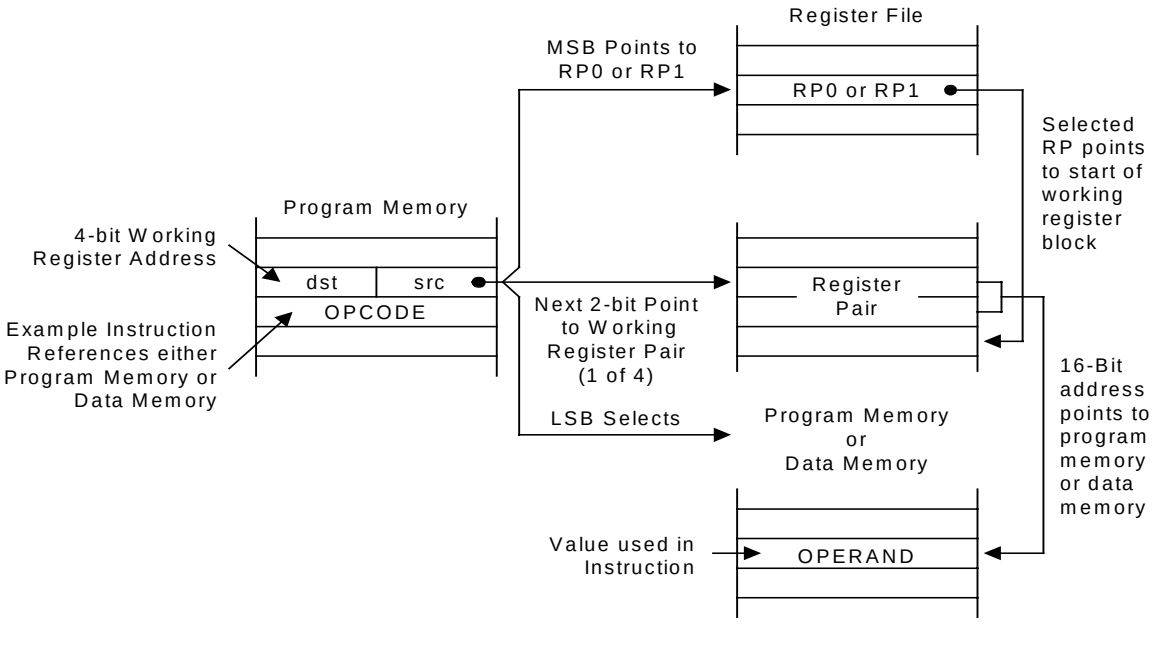


Figure 3-5. Indirect Working Register Addressing to Register File

INDIRECT REGISTER ADDRESSING MODE (Concluded)



Sample Instructions:

```
LDC     R5,@RR6 ;  Program memory access
LDE     R3,@RR14        ;  External data memory access
LDE     @RR4, R8        ;  External data memory access
```

Figure 3-6. Indirect Working Register Addressing to Program or Data Memory

INDEXED ADDRESSING MODE (X)

Indexed (X) addressing mode adds an offset value to a base address during instruction execution in order to calculate the effective operand address (see Figure 3-7). You can use Indexed addressing mode to access locations in the internal register file or in external memory. Please note, however, that you cannot access locations C0H–FFH in set 1 using indexed addressing mode.

In short offset Indexed addressing mode, the 8-bit displacement is treated as a signed integer in the range –128 to +127. This applies to external memory accesses only (see Figure 3-8.)

For register file addressing, an 8-bit base address provided by the instruction is added to an 8-bit offset contained in a working register. For external memory accesses, the base address is stored in the working register pair designated in the instruction. The 8-bit or 16-bit offset given in the instruction is then added to that base address (see Figure 3-9).

The only instruction that supports indexed addressing mode for the internal register file is the Load instruction (LD). The LDC and LDE instructions support indexed addressing mode for internal program memory and for external data memory, when implemented.

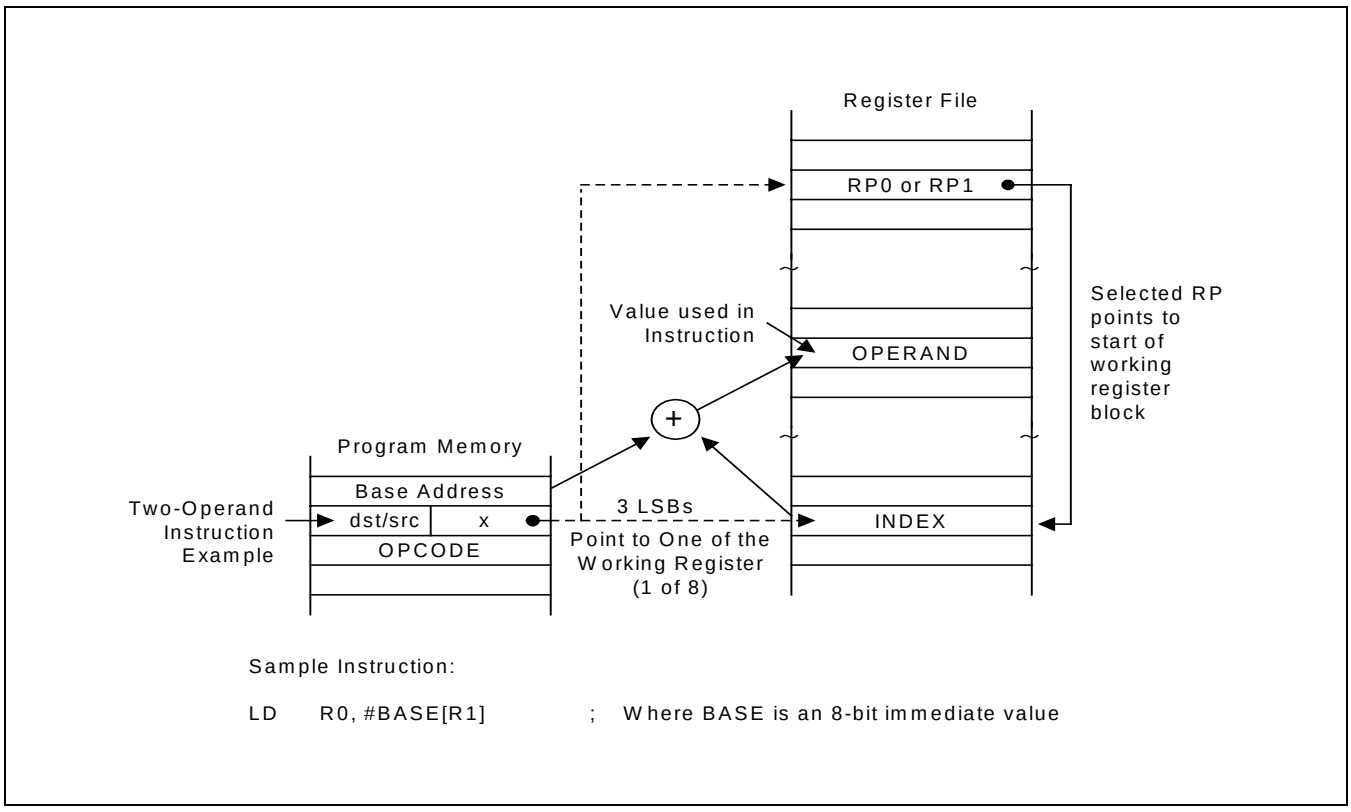
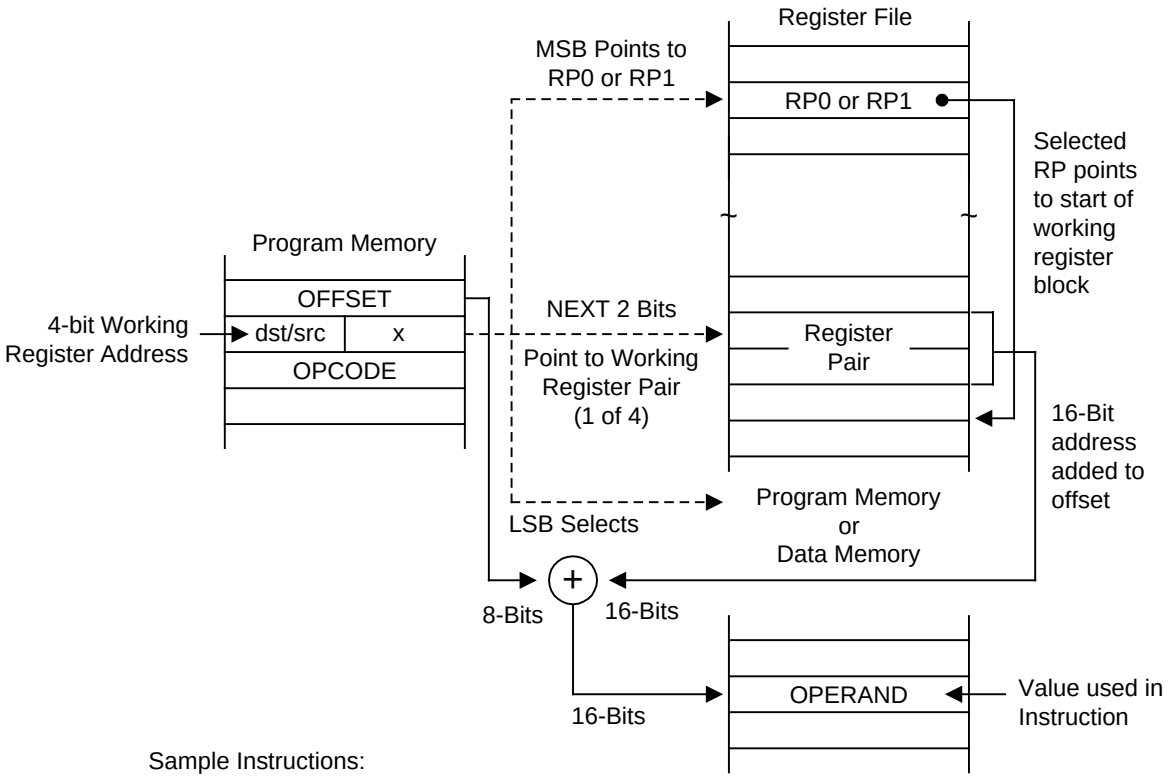


Figure 3-7. Indexed Addressing to Register File

INDEXED ADDRESSING MODE (Continued)

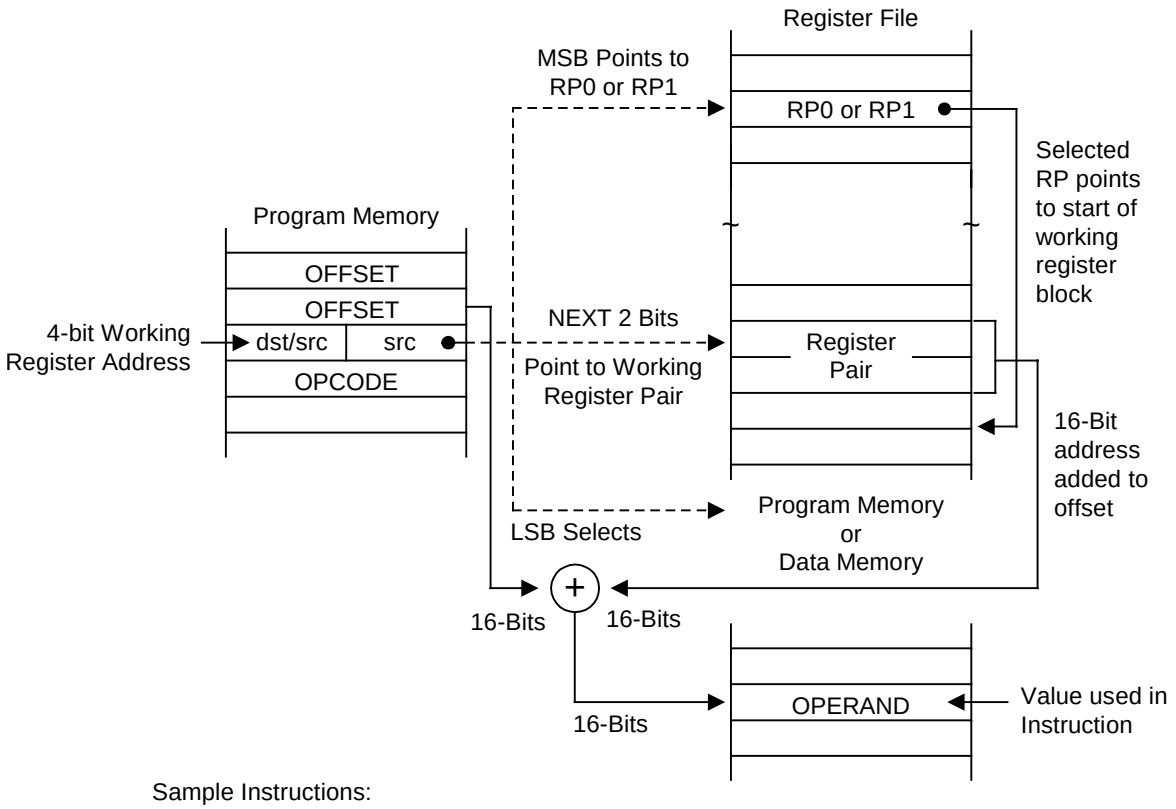


Sample Instructions:

```
LDC     R4, #04H[RR2]     ; The values in the program address (RR2 + 04H)
                          ; are loaded into register R4.
LDE     R4,#04H[RR2]      ; Identical operation to LDC example, except that
                          ; external program memory is accessed.
```

Figure 3-8. Indexed Addressing to Program or Data Memory with Short Offset

INDEXED ADDRESSING MODE (Continued)



Sample Instructions:

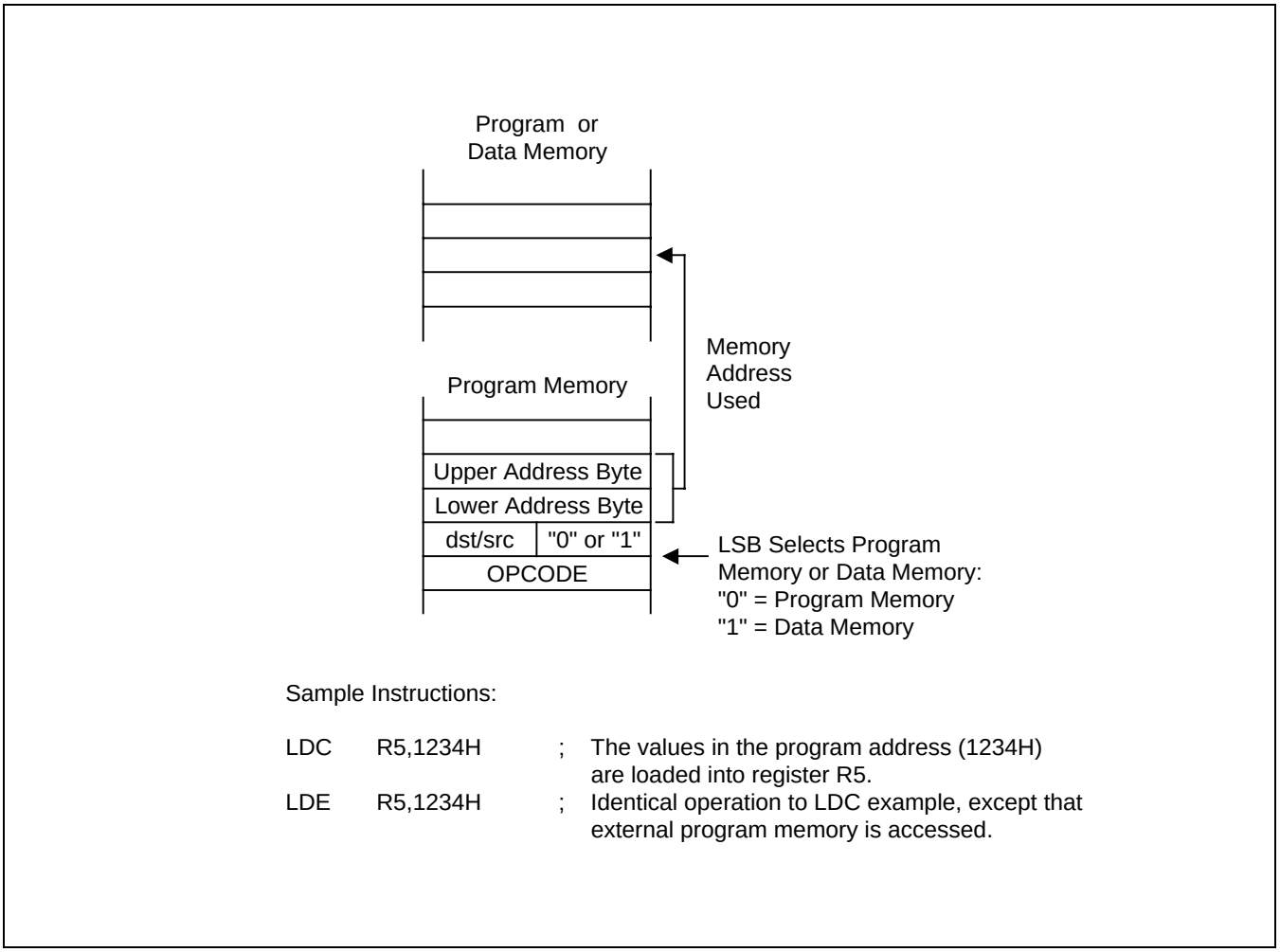
```
LDC     R4, #1000H[RR2]     ; The values in the program address (RR2 + 1000H)
                            ; are loaded into register R4.
LDE     R4,#1000H[RR2]      ; Identical operation to LDC example, except that
                            ; external program memory is accessed.
```

Figure 3-9. Indexed Addressing to Program or Data Memory

DIRECT ADDRESS MODE (DA)

In Direct Address (DA) mode, the instruction provides the operand's 16-bit memory address. Jump (JP) and Call (CALL) instructions use this addressing mode to specify the 16-bit destination address that is loaded into the PC whenever a JP or CALL instruction is executed.

The LDC and LDE instructions can use Direct Address mode to specify the source or destination address for Load operations to program memory (LDC) or to external data memory (LDE), if implemented.

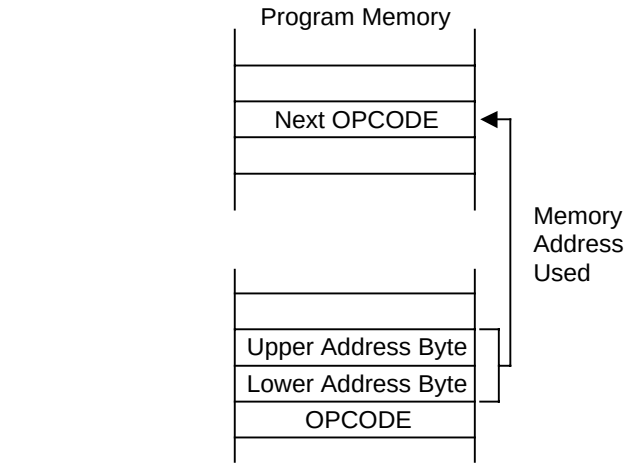


Sample Instructions:

```
LDC     R5,1234H    ;   The values in the program address (1234H)
                        are loaded into register R5.
LDE     R5,1234H    ;   Identical operation to LDC example, except that
                        external program memory is accessed.
```

Figure 3-10. Direct Addressing for Load Instructions

DIRECT ADDRESS MODE (Continued)



Sample Instructions:

```
JP      C,JOB1      ;  Where JOB1 is a 16-bit immediate address
CALL    DISPLAY     ;  Where DISPLAY is a 16-bit immediate address
```

Figure 3-11. Direct Addressing for Call and Jump Instructions

INDIRECT ADDRESS MODE (IA)

In Indirect Address (IA) mode, the instruction specifies an address located in the lowest 256 bytes of the program memory. The selected pair of memory locations contains the actual address of the next instruction to be executed. Only the CALL instruction can use the Indirect Address mode.

Because the Indirect Address mode assumes that the operand is located in the lowest 256 bytes of program memory, only an 8-bit address is supplied in the instruction; the upper bytes of the destination address are assumed to be all zeros.

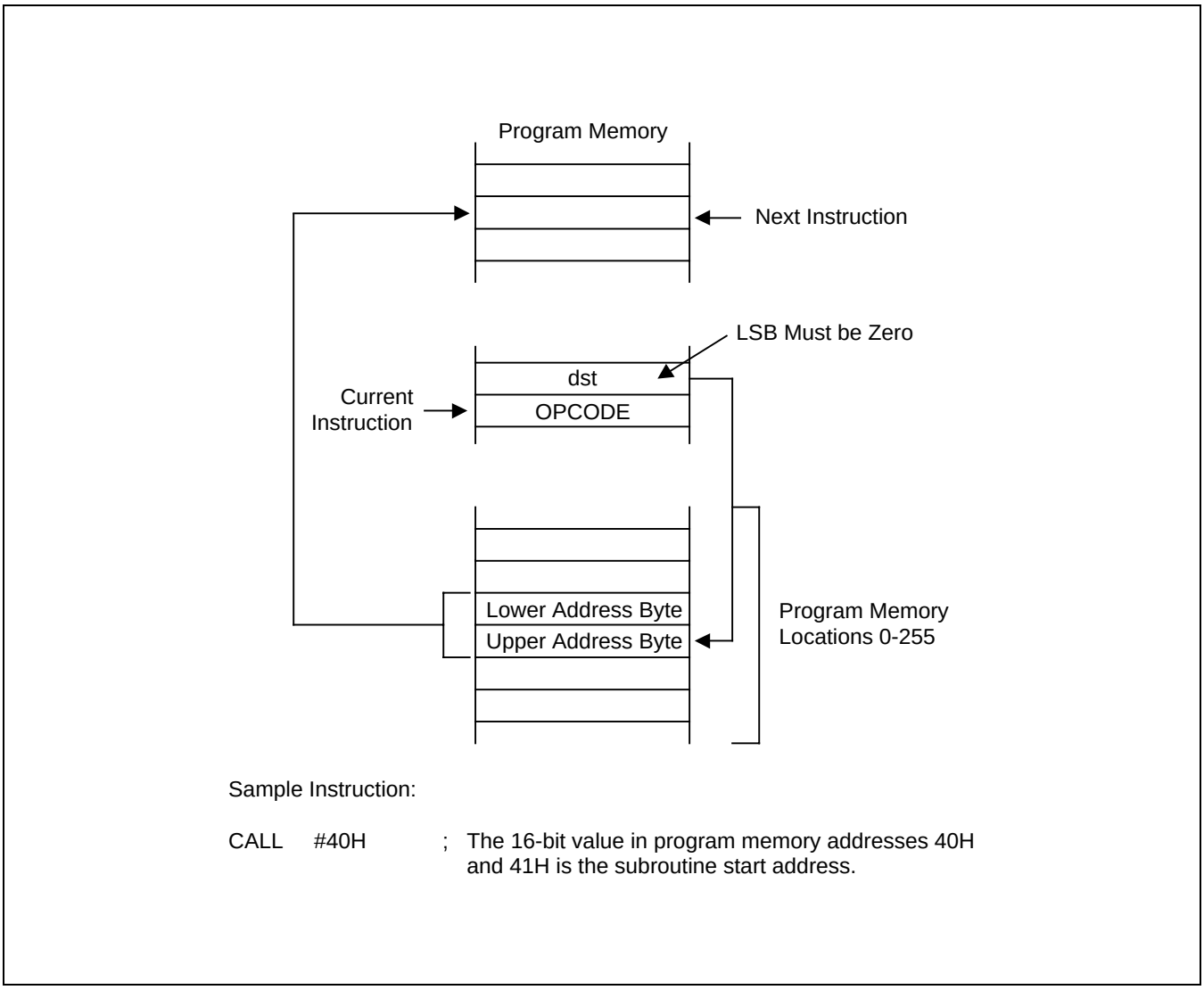
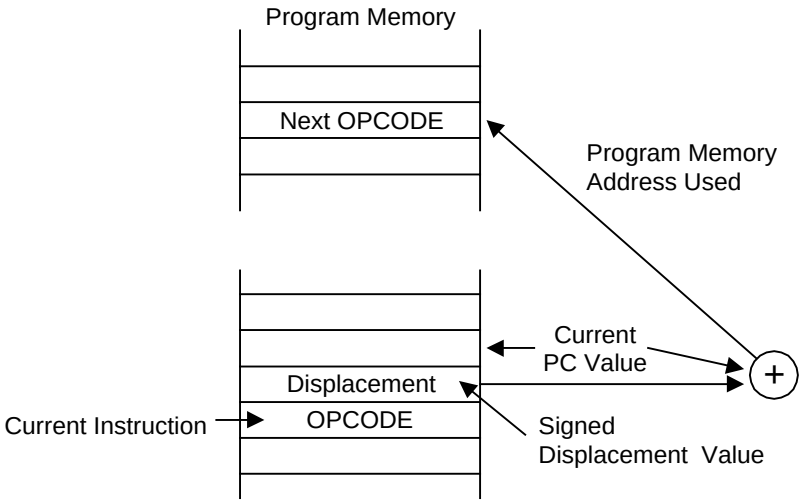


Figure 3-12. Indirect Addressing

RELATIVE ADDRESS MODE (RA)

In Relative Address (RA) mode, a twos-complement signed displacement between – 128 and + 127 is specified in the instruction. The displacement value is then added to the current PC value. The result is the address of the next instruction to be executed. Before this addition occurs, the PC contains the address of the instruction immediately following the current instruction.

Several program control instructions use the Relative Address mode to perform conditional jumps. The instructions that support RA addressing are BTJRF, BTJRT, DJNZ, CPIJE, CPIJNE, and JR.



Sample Instructions:

```
JR      ULT,$+OFFSET    ;   Where OFFSET is a value in the range +127 to -128
```

Figure 3-13. Relative Addressing

IMMEDIATE MODE (IM)

In Immediate (IM) addressing mode, the operand value used in the instruction is the value supplied in the operand field itself. The operand may be one byte or one word in length, depending on the instruction used. Immediate addressing mode is useful for loading constant values into registers.

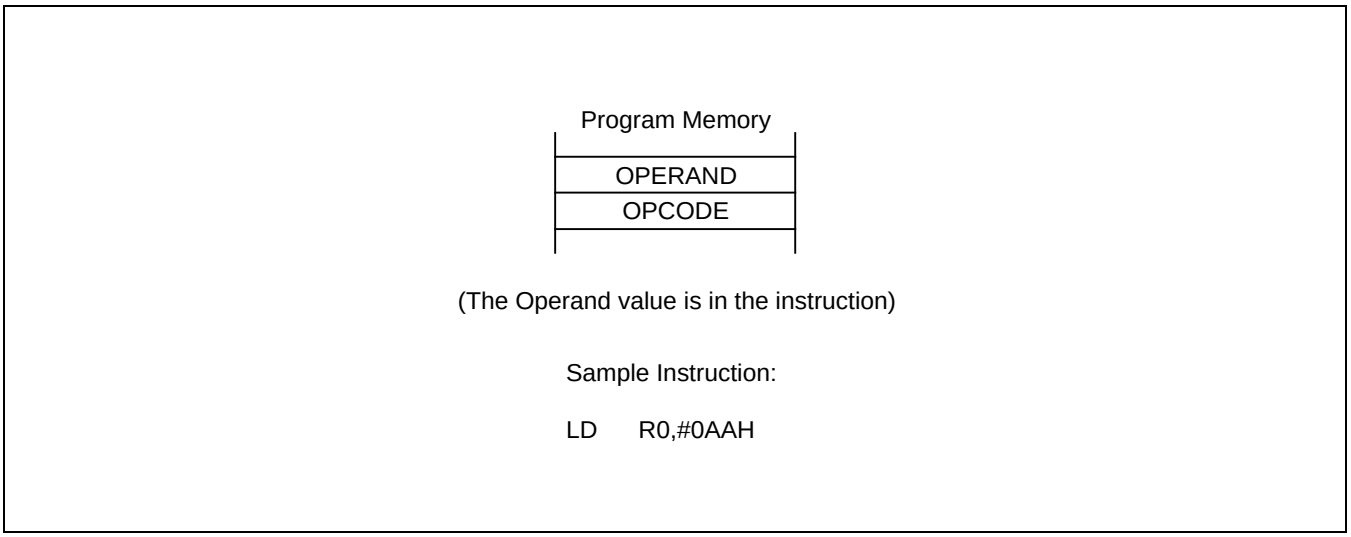


Figure 3-14. Immediate Addressing

4 CONTROL REGISTERS

OVERVIEW

Control register descriptions are arranged in alphabetical order according to register mnemonic. More detailed information about control registers is presented in the context of the specific peripheral hardware descriptions in Part II of this manual.

The locations and read/write characteristics of all mapped registers in the S3C84BB/F84BB register file are listed in Table 4-1. The hardware reset value for each mapped register is described in Chapter 8, "RESET and Power-Down."

Table 4-1. Set 1 Registers

Register Name	Mnemonic	Decimal	Hex	R/W
Timer B control register	TBCON	208	D0H	R/W
Timer B data register (high byte)	TBDATAH	209	D1H	R/W
Timer B data register (low byte)	TBDATAL	210	D2H	R/W
Basic timer control register	BTCON	211	D3H	R/W
Clock control register	CLKCON	212	D4H	R/W
System flags register	FLAGS	213	D5H	R/W
Register pointer 0	RP0	214	D6H	R/W
Register pointer 1	RP1	215	D7H	R/W
Stack pointer (high byte)	SPH	216	D8H	R/W
Stack pointer (low byte)	SPL	217	D9H	R/W
Instruction pointer (high byte)	IPH	218	DAH	R/W
Instruction pointer (low byte)	IPL	219	DBH	R/W
Interrupt request register	IRQ	220	DCH	R
Interrupt mask register	IMR	221	DDH	R/W
System mode register	SYM	222	DEH	R/W
Register page pointer	PP	223	DFH	R/W

Table 4-2. Set 1, Bank 0 Registers

Register Name	Mnemonic	Decimal	Hex	R/W
Port 0 data register	P0	224	E0H	R/W
Port 1 data register	P1	225	E1H	R/W
Port 2 data register	P2	226	E2H	R/W
Port 3 data register	P3	227	E3H	R/W
Port 4 data register	P4	228	E4H	R/W
Port 5 data register	P5	229	E5H	R/W
Port 6 data register	P6	230	E6H	R/W
Port 7 data register	P7	231	E7H	R/W
Port 8 data register	P8	232	E8H	R/W
Timer A/1 interrupt pending register	TINTPND	233	E9H	R/W
Timer A control register	TACON	234	EAH	R/W
Timer A data register	TADATA	235	EBH	R/W
Timer A counter register	TACNT	236	ECH	R
Port 8 control register (high byte)	P8CONH	237	EDH	R/W
Port 8 control register (low byte)	P8CONL	238	EEH	R/W
Port 8 interrupt/pending register	P8INTPND	239	EFH	R/W
Port 0 control register	P0CON	240	F0H	R/W
Port 1 control register	P1CON	241	F1H	R/W
Port 2 control register (high byte)	P2CONH	242	F2H	R/W
Port 2 control register (low byte)	P2CONL	243	F3H	R/W
Port 3 control register (high byte)	P3CONH	244	F4H	R/W
Port 3 control register (low byte)	P3CONL	245	F5H	R/W
Port 4 control register (high byte)	P4CONH	246	F6H	R/W
Port 4 control register (low byte)	P4CONL	247	F7H	R/W
Port 5 control register (high byte)	P5CONH	248	F8H	R/W
Port 5 control register (low byte)	P5CONL	249	F9H	R/W
Port 4 interrupt control register	P4INT	250	FAH	R/W
Port 4 interrupt/pending register	P4INTPND	251	FBH	R/W
Location FCH is factory use only				
Basic timer counter register	BTCNT	253	FDH	R
Location FEH is not mapped.				
Interrupt priority register	IPR	255	FFH	R/W

Table 4-3. Set 1, Bank 1 Registers

Register Name	Mnemonic	Decimal	Hex	R/W
SIO data register	SIODATA	224	E0H	R/W
SIO Control register	SIOCON	225	E1H	R/W
UART0 data register	UDATA0	226	E2H	R/W
UART0 control register	UARTCON0	227	E3H	R/W
UART0 baud rate data register	BRDATA0	228	E4H	R/W
UART0,1 pending register	UARTPND	229	E5H	R/W
Timer 1(0) data register (high byte)	T1DATAH0	230	E6H	R/W
Timer 1(0) data register (low byte)	T1DATAL0	231	E7H	R/W
Timer 1(1) data register (high byte)	T1DATAH1	232	E8H	R/W
Timer 1(1) data register (low byte)	T1DATAL1	233	E9H	R/W
Timer 1(0) control register	T1CON0	234	EAH	R/W
Timer 1(1) control register	T1CON1	235	EBH	R/W
Timer 1(0) counter register (high byte)	T1CNTH0	236	ECH	R
Timer 1(0) counter register (low byte)	T1CNTL0	237	EDH	R
Timer 1(1) counter register (high byte)	T1CNTH1	238	EEH	R
Timer 1(1) counter register (low byte)	T1CNTL1	239	EFH	R
Timer C(0) data register	TCDATA0	240	F0H	R/W
Timer C(1) data register	TCDATA1	241	F1H	R/W
Timer C(0) control register	TCCON0	242	F2H	R/W
Timer C(1) control register	TCCON1	243	F3H	R/W
SIO prescaler control register	SIOPS	244	F4H	R/W
Port 7 control register	P7CON	245	F5H	R/W
D/A converter data register	DADATA	246	F6H	R/W
A/D, D/A converter control register	ADACON	247	F7H	R/W
A/D converter data register (high byte)	ADDATAH	248	F8H	R
A/D converter data register (low byte)	ADDATAL	249	F9H	R
UART1 data register	UDATA1	250	FAH	R/W
UART1 control register	UARTCON1	251	FBH	R/W
UART1 baud rate data register	BRDATA1	252	FCH	R/W
Flash memory control register	FMCON	253	FDH	R/W
Pattern generation control register	PGCON	254	FEH	R/W
Pattern generation data register	PGDATA	255	FFH	R/W

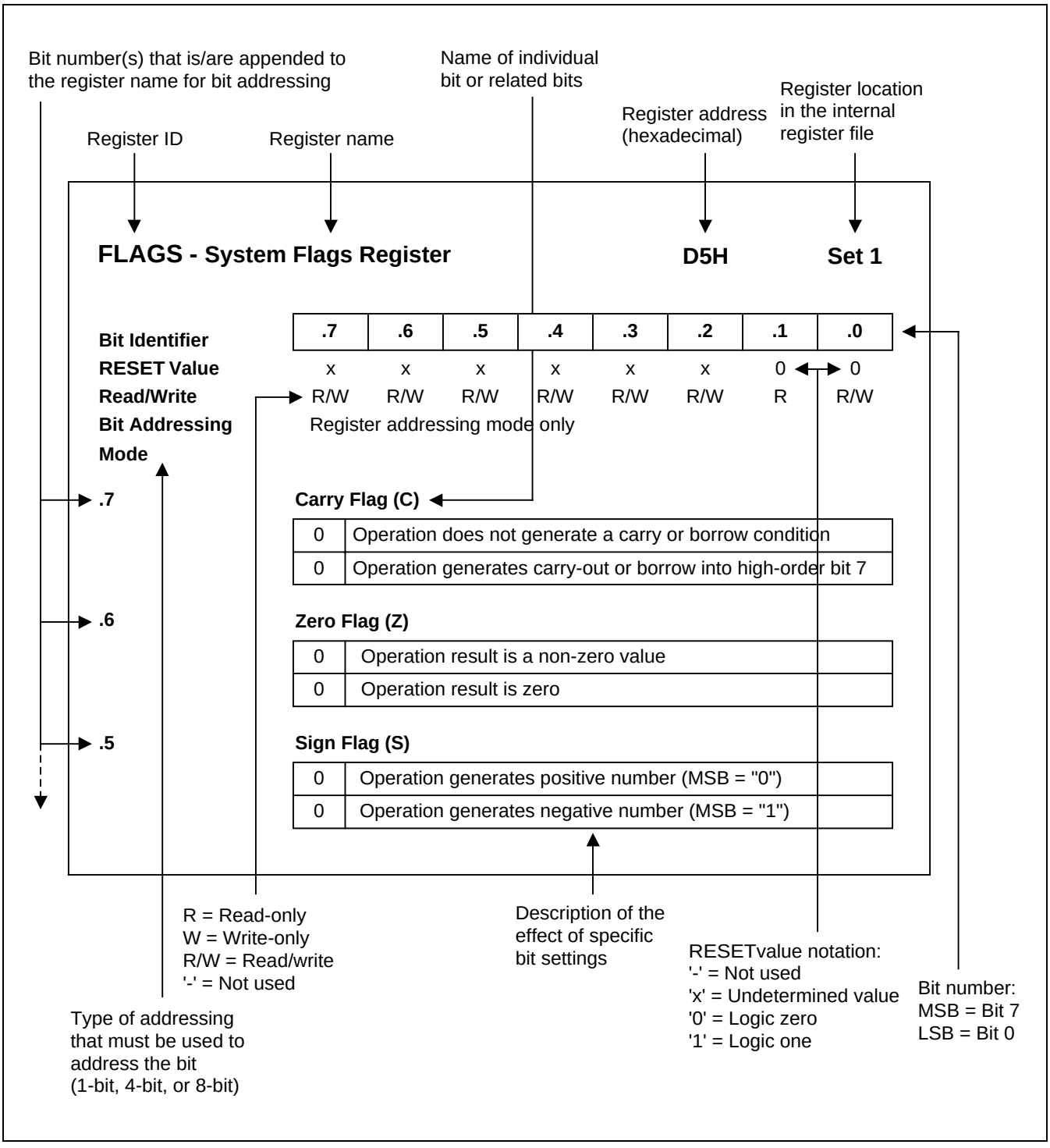


Figure 4-1. Register Description Format

ADACON — A/D, D/A Converter Control Register F7H Set 1, Bank 1

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R	R/W	R/W	R/W
Addressing Mode	Register addressing mode only							

.7 D/A Start Enable Bit

0	Disable operation
1	Start operation

.6-.4 A/D Input Pin Selection Bits

0	0	0	ADC0
0	0	1	ADC1
0	1	0	ADC2
0	1	1	ADC3
1	0	0	ADC4
1	0	1	ADC5
1	1	0	ADC6
1	1	1	ADC7

.3 End-of-Conversion Bit (Read-only)

0	A/D conversion opration is in progress
1	A/D conversion opration is complete

.2-.1 Clock Source Selection Bits

0	0	fxx/16
0	1	fxx/8
1	0	fxx/4
1	1	fxx

.0 A/D Start or Enable Bit

0	Disable operation
1	Start operation

BRDATA0 — UART0 Baud Rate Data Register E4H Set 1, Bank 1

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	1	1	1	1	1	1	1	1
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Addressing Mode	Register addressing mode only							

.7–.0	Baud Rate Data for UART0 (note) : fxx/(16 × (BRDATA + 1))

NOTE: Refer to UARTCON0 register.

BRDATA1 — UART1 Baud Rate Data Register FCH Set 1, Bank 1

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	1	1	1	1	1	1	1	1
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Addressing Mode	Register addressing mode only							

.7–.0	Baud Rate Data for UART1 (note) : fxx/(16 × (BRDATA + 1))

NOTE: Refer to UARTCON1 register.

BTCON — Basic Timer Control Register D3H Set 1

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Addressing Mode	Register addressing mode only							

.7-.4 Watchdog Timer Function Disable Code (for System Reset)

1	0	1	0	Disable watchdog timer function
Others				Enable watchdog timer function

.3-.2 Basic Timer Input Clock Selection Bits

0	0	fxx/4096 (3)
0	1	fxx/1024
1	0	fxx/128
1	1	fxx/16 (Not used)

.1 Basic Timer Counter Clear Bit (1)

0	No effect
1	Clear the basic timer counter value

.0 Clock Frequency Divider Clear Bit for Basic Timer (2)

0	No effect
1	Clear both clock frequency dividers

NOTES:

1. When you write a “1” to BTCON.1, the basic timer counter value is cleared to "00H". Immediately following the write operation, the BTCON.1 value is automatically cleared to “0”.
2. When you write a "1" to BTCON.0, the corresponding frequency divider is cleared to "00H". Immediately following the write operation, the BTCON.0 value is automatically cleared to "0".
3. The fxx is selected clock for system (main OSC. or sub OSC.).

CLKCON — System Clock Control Register D4H Set 1

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	0	0	0	0	0	0	0	0
Read/Write	–	–	–	R/W	R/W	–	–	–
Addressing Mode	Register addressing mode only							

.7-.5 Not used for the S3C84BB/F84BB (must keep always 0)

.4-.3 **CPU Clock (System Clock) Selection Bits (note)**

0	0	fxx/16
0	1	fxx/8
1	0	fxx/2
1	1	fxx/1 (non-divided)

.2-.0 Not used for the S3C84BB/F84BB (must keep always 0)

NOTE: After a reset, the slowest clock (divided by 16) is selected as the system clock. To select faster clock speeds, load the appropriate values to CLKCON.3 and CLKCON.4.

FLAGS — System Flags Register D5H Set 1

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	x	x	x	x	x	x	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W
Addressing Mode	Register addressing mode only							

.7 Carry Flag (C)

0	Operation does not generate a carry or underflow condition
1	Operation generates a carry-out or underflow into high-order bit 7

.6 Zero Flag (Z)

0	Operation result is a non-zero value
1	Operation result is zero

.5 Sign Flag (S)

0	Operation generates a positive number (MSB = "0")
1	Operation generates a negative number (MSB = "1")

.4 Overflow Flag (V)

0	Operation result is $\leq +127$ or ≥ -128
1	Operation result is $> +127$ or < -128

.3 Decimal Adjust Flag (D)

0	Add operation completed
1	Subtraction operation completed

.2 Half-Carry Flag (H)

0	No carry-out of bit 3 or no underflow into bit 3 by addition or subtraction
1	Addition generated carry-out of bit 3 or subtraction generated underflow into bit 3

.1 Fast Interrupt Status Flag (FIS)

0	Interrupt return (IRET) in progress (when read)
1	Fast interrupt service routine in progress (when read)

.0 Bank Address Selection Flag (BA)

0	Bank 0 is selected
1	Bank 1 is selected

FMCON — Flash Memory Control Register FDH Set 1, Bank1

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	0	0	0	0	0	0	0	0
Read/Write	R/W	–	–	–	–	–	R/W	R/W
Addressing Mode	Register addressing mode only							

.7 User Programming Serial Data Bit (FSDAT)

0	FSDA=Low (Logic 0)
1	FSDA=High (Logic 1)

.6-.2

Not used for the S3C84BB/F84BB (must keep always 0)

.1 User Programming Mode Status Bit (full-flash flag)

0	Not-user programming mode
1	User programming mode

.0 User Programming Serial Clock Bit (FSCLK)

0	FSCLK=Low (Logic 0)
1	FSCLK=High(Logic 1)

IMR — Interrupt Mask Register DDH Set 1

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	x	x	x	x	x	x	x	x
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Addressing Mode	Register addressing mode only							

.7 Interrupt Level 7 (IRQ7) Enable Bit

0	Disable (mask)
1	Enable (un-mask)

.6 Interrupt Level 6 (IRQ6) Enable Bit

0	Disable (mask)
1	Enable (un-mask)

.5 Interrupt Level 5 (IRQ5) Enable Bit

0	Disable (mask)
1	Enable (un-mask)

.4 Interrupt Level 4 (IRQ4) Enable Bit

0	Disable (mask)
1	Enable (un-mask)

.3 Interrupt Level 3 (IRQ3) Enable Bit

0	Disable (mask)
1	Enable (un-mask)

.2 Interrupt Level 2 (IRQ2) Enable Bit

0	Disable (mask)
1	Enable (un-mask)

.1 Interrupt Level 1 (IRQ1) Enable Bit

0	Disable (mask)
1	Enable (un-mask)

.0 Interrupt Level 0 (IRQ0) Enable Bit

0	Disable (mask)
1	Enable (un-mask)

NOTE: When an interrupt level is masked, any interrupt requests that may be issued are not recognized by the CPU.

IPH — Instruction Pointer (High Byte) DAH Set 1

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	x	x	x	x	x	x	x	x
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Addressing Mode	Register addressing mode only							

.7–.0 Instruction Pointer Address (High Byte)

The high-byte instruction pointer value is the upper eight bits of the 16-bit instruction pointer address (IP15–IP8). The lower byte of the IP address is located in the IPL register (DBH).

IPL — Instruction Pointer (Low Byte) DBH Set 1

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	x	x	x	x	x	x	x	x
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Addressing Mode	Register addressing mode only							

.7–.0 Instruction Pointer Address (Low Byte)

The low-byte instruction pointer value is the lower eight bits of the 16-bit instruction pointer address (IP7–IP0). The upper byte of the IP address is located in the IPH register (DAH).

IPR — Interrupt Priority Register FFH Set 1, Bank 0

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	x	x	x	x	x	x	x	x
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Addressing Mode	Register addressing mode only							

.7, .4, and .1 Priority Control Bits for Interrupt Groups A, B, and C

0	0	0	Group priority undefined
0	0	1	B > C > A
0	1	0	A > B > C
0	1	1	B > A > C
1	0	0	C > A > B
1	0	1	C > B > A
1	1	0	A > C > B
1	1	1	Group priority undefined

.6 Interrupt Subgroup C Priority Control Bit

0	IRQ6 > IRQ7
1	IRQ7 > IRQ6

.5 Interrupt Group C Priority Control Bit

0	IRQ5 > (IRQ6, IRQ7)
1	(IRQ6, IRQ7) > IRQ5

.3 Interrupt Subgroup B Priority Control Bit

0	IRQ3 > IRQ4
1	IRQ4 > IRQ3

.2 Interrupt Group B Priority Control Bit

0	IRQ2 > (IRQ3, IRQ4)
1	(IRQ3, IRQ4) > IRQ2

.0 Interrupt Group A Priority Control Bit

0	IRQ0 > IRQ1
1	IRQ1 > IRQ0

IRQ — Interrupt Request Register DCH Set 1

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	0	0	0	0	0	0	0	0
Read/Write	R	R	R	R	R	R	R	R
Addressing Mode	Register addressing mode only							

.7 Level 7 (IRQ7) Request Pending Bit

0	Not pending
1	Pending

.6 Level 6 (IRQ6) Request Pending Bit

0	Not pending
1	Pending

.5 Level 5 (IRQ5) Request Pending Bit

0	Not pending
1	Pending

.4 Level 4 (IRQ4) Request Pending Bit

0	Not pending
1	Pending

.3 Level 3 (IRQ3) Request Pending Bit

0	Not pending
1	Pending

.2 Level 2 (IRQ2) Request Pending Bit

0	Not pending
1	Pending

.1 Level 1 (IRQ1) Request Pending Bit

0	Not pending
1	Pending

.0 Level 0 (IRQ0) Request Pending Bit

0	Not pending
1	Pending

P0CON — Port 0 Control Register F0H Set 1, Bank 0

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Addressing Mode	Register addressing mode only							

.7-.6 **P0.7/P0.6/P0.5/P0.4**

0	0	Input mode
0	1	Input mode, pull-up
1	0	Push-pull output
1	1	Alternative function mode (PGOUT<7:4>)

.5–.4 **P0.3/P0.2**

0	0	Input mode
0	1	Input mode, pull-up
1	0	Push-pull output
1	1	Alternative function mode (PGOUT<3:2>)

.3–.2 **P0.1**

0	0	Input mode
0	1	Input mode, pull-up
1	0	Push-pull output
1	1	Alternative function mode (PGOUT<1>)

.1–.0 **P0.0**

0	0	Input mode
0	1	Input mode, pull-up
1	0	Push-pull output
1	1	Alternative function mode (PGOUT<0>)

P1CON — Port 1 Control Register F1H Set 1, Bank 0

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Addressing Mode	Register addressing mode only							

.7–.6 P1.7/P1.6

0	0	Input mode
0	1	Input mode, pull-up
1	x	Push-pull output

.5–.4 P1.5/P1.4

0	0	Input mode
0	1	Input mode, pull-up
1	x	Push-pull output

.3–.2 P1.3/P1.2

0	0	Input mode
0	1	Input mode, pull-up
1	x	Push-pull output

.1–.0 P1.1/P1.0

0	0	Input mode
0	1	Input mode, pull-up
1	x	Push-pull output

P2CONH — Port 2 Control Register (High Byte) F2H Set 1, Bank 0

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Addressing Mode	Register addressing mode only							

.7–.6 **P2.7/TAOUT**

0	0	Input mode
0	1	Input mode, pull-up
1	0	Push-pull output
1	1	Alternative output mode(TAOUT)

.5-.4 **P2.6/TACAP**

0	0	Input mode(TACAP)
0	1	Input mode, pull-up(TACAP)
1	0	Push-pull output
1	1	Alternative output mode(Not used)

.3–.2 **P2.5/TACK**

0	0	Input mode(TACK)
0	1	Input mode, pull-up(TACK)
1	0	Push-pull output
1	1	Alternative output mode(Not used)

.1–.0 **P2.4/ TBPWM**

0	0	Input mode
0	1	Input mode, pull-up
1	0	Push-pull output
1	1	Alternative output mode(TBPWM)

P2CONL — Port 2 Control Register (Low Byte) F3H Set 1, Bank 0

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Addressing Mode	Register addressing mode only							

.7-.6 **P2.3/DAOUT**

0	0	Input mode
0	1	Input mode, pull-up
1	0	Push-pull output
1	1	Alternative output mode (DAOUT)

.5-.4 **P2.2/SCK**

0	0	Input mode (SCK input)
0	1	Input mode, pull-up (SCK input)
1	0	Push-pull output
1	1	Alternative output mode (SCK output)

.3-.2 **P2.1/SI**

0	0	Input mode(SI)
0	1	Input mode, pull-up(SI)
1	0	Push-pull output
1	1	Alternative output mode(Not used)

.1-.0 **P2.0/SO**

0	0	Input mode
0	1	Input mode, pull-up
1	0	Push-pull output
1	1	Alternative output mode (SO)

P3CONH — Port 3 Control Register (High Byte) F4H Set 1, Bank 0

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Addressing Mode	Register addressing mode only							

.7–.6 **P3.7/TCOUT1**

0	0	Input mode
0	1	Input mode, pull-up
1	0	Push-pull output
1	1	Alternative output mode(TCOUT1)

.5-.4 **P3.6/TCOUT0**

0	0	Input mode
0	1	Input mode, pull-up
1	0	Push-pull output
1	1	Alternative output mode(TCOUT0)

.3–.2 **P3.5/ T1OUT1**

0	0	Input mode
0	1	Input mode, pull-up
1	0	Push-pull output
1	1	Alternative output mode(T1OUT1)

.1–.0 **P3.4/ T1OUT0**

0	0	Input mode
0	1	Input mode, pull-up
1	0	Push-pull output
1	1	Alternative output mode(T1OUT0)

P3CONL — Port 3 Control Register (Low Byte) F5H Set 1, Bank 0

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Addressing Mode	Register addressing mode only							

.7-.6 **P3.3/T1CAP1**

0	0	Input mode (T1CAP1)
0	1	Input mode, pull-up (T1CAP1)
1	x	Push-pull output

.5-.4 **P3.2/ T1CAP0**

0	0	Input mode (T1CAP0)
0	1	Input mode, pull-up (T1CAP0)
1	x	Push-pull output

.3-.3 **P3.1/T1CK1**

0	0	Input mode (T1CK1)
0	1	Input mode, pull-up (T1CK1)
1	x	Push-pull output

.1-.0 **P3.0/T1CK0**

0	0	Input mode (T1CK0)
0	1	Input mode, pull-up (T1CK0)
1	x	Push-pull output

P4CONH — Port 4 Control Register (High Byte) F6H Set 1, Bank 0

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Addressing Mode	Register addressing mode only							

.7–.6 **P4.7/INT7**

0	0	Input mode; falling edge interrupt
0	1	Input mode; rising edge interrupt
1	0	Input mode, pull-up; falling edge interrupt
1	1	Push-pull output

.5-.4 **P4.6/ INT6**

0	0	Input mode; falling edge interrupt
0	1	Input mode; rising edge interrupt
1	0	Input mode, pull-up; falling edge interrupt
1	1	Push-pull output

.3–.2 **P4.5/ INT5**

0	0	Input mode; falling edge interrupt
0	1	Input mode; rising edge interrupt
1	0	Input mode, pull-up; falling edge interrupt
1	1	Push-pull output

.1–.0 **P4.4/ INT4**

0	0	Input mode; falling edge interrupt
0	1	Input mode; rising edge interrupt
1	0	Input mode, pull-up; falling edge interrupt
1	1	Push-pull output

P4CONL — Port 4 Control Register (Low Byte) F7H Set 1, Bank 0

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Addressing Mode	Register addressing mode only							

.7-.6 **P4.3/INT3**

0	0	Input mode; falling edge interrupt
0	1	Input mode; rising edge interrupt
1	0	Input mode, pull-up; falling edge interrupt
1	1	Push-pull output

.5-.4 **P4.2/INT2**

0	0	Input mode; falling edge interrupt
0	1	Input mode; rising edge interrupt
1	0	Input mode, pull-up; falling edge interrupt
1	1	Push-pull output

.3-.2 **P4.1/INT1**

0	0	Input mode; falling edge interrupt
0	1	Input mode; rising edge interrupt
1	0	Input mode, pull-up; falling edge interrupt
1	1	Push-pull output

.1-.0 **P4.0/INT0**

0	0	Input mode; falling edge interrupt
0	1	Input mode; rising edge interrupt
1	0	Input mode, pull-up; falling edge interrupt
1	1	Push-pull output

P4INT — Port 4 Interrupt Control Register FAH Set 1, Bank 0

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Addressing Mode	Register addressing mode only							

.7 P4.7 External Interrupt (INT7) Enable Bit

0	Disable interrupt
1	Enable interrupt

.6 P4.6 External Interrupt (INT6) Enable Bit

0	Disable interrupt
1	Enable interrupt

.5 P4.5 External Interrupt (INT5) Enable Bit

0	Disable interrupt
1	Enable interrupt

.4 P4.4 External Interrupt (INT4) Enable Bit

0	Disable interrupt
1	Enable interrupt

.3 P4.3 External Interrupt (INT3) Enable Bit

0	Disable interrupt
1	Enable interrupt

.2 P4.2 External Interrupt (INT2) Enable Bit

0	Disable interrupt
1	Enable interrupt

.1 P4.1 External Interrupt (INT1) Enable Bit

0	Disable interrupt
1	Enable interrupt

.0 P4.0 External Interrupt (INT0) Enable Bit

0	Disable interrupt
1	Enable interrupt

P4INTPND — Port 4 Interrupt Pending Register FBH Set 1, Bank 0

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Addressing Mode	Register addressing mode only							

.7 P4.7/INT7 Interrupt Pending Bit

0	Interrupt request is not pending, pending bit clear when write 0
1	Interrupt request is pending

.6 P4.6/INT6 Interrupt Pending Bit

0	Interrupt request is not pending, pending bit clear when write 0
1	Interrupt request is pending

.5 P4.5/INT5 Interrupt Pending Bit

0	Interrupt request is not pending, pending bit clear when write 0
1	Interrupt request is pending

.4 P4.4/INT4 Interrupt Pending Bit

0	Interrupt request is not pending, pending bit clear when write 0
1	Interrupt request is pending

.3 P4.3/INT3 Interrupt Pending Bit

0	Interrupt request is not pending, pending bit clear when write 0
1	Interrupt request is pending

.2 P4.2/INT2 Interrupt Pending Bit

0	Interrupt request is not pending, pending bit clear when write 0
1	Interrupt request is pending

.1 P4.1/INT1 Interrupt Pending Bit

0	Interrupt request is not pending, pending bit clear when write 0
1	Interrupt request is pending

.0 P4.0/INT0 Interrupt Pending Bit

0	Interrupt request is not pending, pending bit clear when write 0
1	Interrupt request is pending

P5CONH — Port 5 Control Register (High Byte) F8H Set 1, Bank 0

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Addressing Mode	Register addressing mode only							

.7–.6 **P5.7**

0	0	Input mode
0	1	Input mode, pull-up
1	0	Push-pull output
1	1	Open-drain mode

.5-.4 **P5.6**

0	0	Input mode
0	1	Input mode, pull-up
1	0	Push-pull output
1	1	Open-drain mode

.3–.2 **P5.5**

0	0	Input mode
0	1	Input mode, pull-up
1	0	Push-pull output
1	1	Open-drain mode

.1–.0 **P5.4**

0	0	Input mode
0	1	Input mode, pull-up
1	0	Push-pull output
1	1	Open-drain mode

P5CONL — Port 5 Control Register (Low Byte) F9H Set 1, Bank 0

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Addressing Mode	Register addressing mode only							

.7-.6 P5.3/RxD0

0	0	Input mode (RxD0 input)
0	1	Input mode, pull-up mode (RxD0 input)
1	0	Push-pull output
1	1	Alternative output mode (RxD0 output)

.5-.4 P5.2/TxD0

0	0	Input mode
0	1	Input mode, pull-up mode
1	0	Push-pull output
1	1	Alternative output mode (TxD0 output)

.3-.2 P5.1/RxD1

0	0	Input mode (RxD1 input)
0	1	Input mode, pull-up mode (RxD1 input)
1	0	Push-pull output
1	1	Alternative output mode (RxD1 output)

.1-.0 P5.0/TxD1

0	0	Input mode
0	1	Input mode, pull-up mode
1	0	Push-pull output
1	1	Alternative output mode (TxD1 output)

P7CON — Port 7 Control Register F5H Set 1, Bank 1

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Addressing Mode	Register addressing mode only							

.7 P7.7/ADC7

0	Input mode
1	ADC input mode

.6 P7.6/ ADC6

0	Input mode
1	ADC input mode

.5 P7.5/ ADC5

0	Input mode
1	ADC input mode

.4 P7.4/ ADC4

0	Input mode
1	ADC input mode

.3 P7.3/ ADC3

0	Input mode
1	ADC input mode

.2 P7.2/ ADC2

0	Input mode
1	ADC input mode

.1 P7.1/ ADC1

0	Input mode
1	ADC input mode

.0 P7.0/ ADC0

0	Input mode
1	ADC input mode

P8CONH — Port 8 Control Register (High Byte) EDH Set 1, Bank 0

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	1	1	1	1	0	0	0	0
Read/Write	-–	–	–	–-	R/W	R/W	R/W	R/W
Addressing Mode	Register addressing mode only							

.7–.4

Not used for the S3C84BB/F84BB (must keep always 1)

.3–.2 **P8.5/ INT9**

0	0	Input mode; falling edge interrupt
0	1	Input mode; rising edge interrupt
1	0	Input mode, pull-up; falling edge interrupt
1	1	Push-pull output

.1–.0 **P8.4/ INT8**

0	0	Input mode; falling edge interrupt
0	1	Input mode; rising edge interrupt
1	0	Input mode, pull-up; falling edge interrupt
1	1	Push-pull output

P8CONL — Port 8 Control Register (Low Byte) EEH Set 1, Bank 0

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Addressing Mode	Register addressing mode only							

.7-.6 P8.3

0	0	Input mode
0	1	Input mode, pull-up
1	x	Push-pull output

.5-.4 P8.2

0	0	Input mode
0	1	Input mode, pull-up
1	x	Push-pull output

.3-.2 P8.1

0	0	Input mode
0	1	Input mode, pull-up
1	x	Push-pull output

.1-.0 P8.0

0	0	Input mode
0	1	Input mode, pull-up
1	x	Push-pull output

P8INTPND — Port 8 Interrupt Pending Register EFH Set 1, Bank 0

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	1	1	0	0	1	1	0	0
Read/Write	–	–	R/W	R/W	–	–	R/W	R/W
Addressing Mode	Register addressing mode only							

.7-.6 Not used for the S3C84BB/F84BB (must keep always 1)

.5 P8.5/INT9 Interrupt Pending Bit

0	Interrupt request is not pending, pending bit clear when write 0
1	Interrupt request is pending

.4 P8.4/INT8 Interrupt Pending Bit

0	Interrupt request is not pending, pending bit clear when write 0
1	Interrupt request is pending

.3-.2 Not used for the S3C84BB/F84BB (must keep always 1)

.1 P8.5/INT9 Interrupt Enable

0	Disable interrupt
1	Enable interrupt

.0 P8.4/INT8 Interrupt Enable

0	Disable interrupt
1	Enable interrupt

PGCON — Pattern Generation Control Register FEH Set 1, Bank 1

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	–	–	–	–	0	0	0	0
Read/Write	–	–	–	–	R/W	R/W	R/W	R/W
Addressing Mode	Register addressing mode only							

.7-.4

Not used for the S3C84BB/F84BB

.3 Software Trigger Start Bit

0	No effect
1	Software trigger start (will be automatically cleared)

.2 PG Operation Disable/Enable Selection Bit

0	PG operation disable
1	PG operation enable

.1-.0 PG Operation Trigger Mode Selection Bits

0	0	Timer A match siganal triggering
0	1	Timer B underflow siganal triggering
1	0	Timer 1(0) match siganal triggering
1	1	Software triggering mode

PP — Register Page Pointer DFH Set 1

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Addressing Mode	Register addressing mode only							

.7-.4 Destination Register Page Selection Bits

0	0	0	0	Destination: page 0
0	0	0	1	Destination: page 1
0	0	1	0	Destination: page 2
0	0	1	1	Destination: page 3
0	1	0	0	Destination: page 4
0	1	0	1	Destination: page 5
0	1	1	0	Destination: page 6
0	1	1	1	Destination: page 7

.3-.0 Source Register Page Selection Bits

0	0	0	0	Source: page 0
0	0	0	1	Source: page 1
0	0	1	0	Source: page 2
0	0	1	1	Source: page 3
0	1	0	0	Source: page 4
0	1	0	1	Source: page 5
0	1	1	0	Source: page 6
0	1	1	1	Source: page 7

NOTE: In the S3C84BB/F84BB microcontroller, the internal register file is configured as eight pages (Pages 0-7). The pages 0-1 are used for general-purpose register file, and page 2-7 is used for data register or general purpose registers.

RP0 — Register Pointer 0 D6H Set 1

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	1	1	0	0	0	–	–	–
Read/Write	R/W	R/W	R/W	R/W	R/W	–	–	–
Addressing Mode	Register addressing only							

.7-.3 Register Pointer 0 Address Value

Register pointer 0 can independently point to one of the 256-byte working register areas in the register file. Using the register pointers RP0 and RP1, you can select two 8-byte register slices at one time as active working register space. After a reset, RP0 points to address C0H in register set 1, selecting the 8-byte working register slice C0H–C7H.

.2-.0 Not used for the S3C84BB/F84BB

RP1 — Register Pointer 1 D7H Set 1

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	1	1	0	0	1	–	–	–
Read/Write	R/W	R/W	R/W	R/W	R/W	–	–	–
Addressing Mode	Register addressing only							

.7-.3 Register Pointer 1 Address Value

Register pointer 1 can independently point to one of the 256-byte working register areas in the register file. Using the register pointers RP0 and RP1, you can select two 8-byte register slices at one time as active working register space. After a reset, RP1 points to address C8H in register set 1, selecting the 8-byte working register slice C8H–CFH.

.2-.0 Not used for the S3C84BB/F84BB

SIOCON — SIO Control Register E1H Set 1, Bank 1

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Addressing Mode	Register addressing mode only							

.7 SIO Shift Clock Selection Bit

0	Internal clock (P.S clock)
1	External clock (SCK)

.6 Data Direction Control Bit

0	MSB first mode
1	LSB first mode

.5 SIO Mode Selection Bit

0	Receive only mode
1	Transmit/receive mode

.4 Shift Start Edge Selection Bit

0	Tx at falling edges, Rx at rising edges
1	Tx at rising edges, Rx at falling edges

.3 SIO Counter Clear and Shift Start Bit

0	No action
1	Clear 3-bit counter and start shifting (Auto-clear bit)

.2 SIO Shift Operation Enable Bit

0	Disable shifter and clock counter
1	Enable shifter and clock counter

.1 SIO Interrupt Enable Bit

0	Disable SIO interrupt
1	Enable SIO interrupt

.0 SIO Interrupt Pending Bit

0	No interrupt pending
0	*Clear pending condition (when write)*
1	Interrupt is pending

SIOPS — SIO Prescaler Register F4H Set 1, Bank 1

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Addressing Mode	Register addressing mode only							

.7–.0	Baud rate = Input clock (fxx)/[(SIOPS + 1) ×2] or SCK input clock

SPH — Stack Pointer (High Byte) D8H Set 1

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	x	x	x	x	x	x	x	x
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Addressing Mode	Register addressing mode only							

.7–.0 Stack Pointer Address (High Byte)

The high-byte stack pointer value is the upper eight bits of the 16-bit stack pointer address (SP15–SP8). The lower byte of the stack pointer value is located in register SPL (D9H). The SP value is undefined following a reset.

SPL — Stack Pointer (Low Byte) D9H Set 1

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	x	x	x	x	x	x	x	x
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Addressing Mode	Register addressing mode only							

.7–.0 Stack Pointer Address (Low Byte)

The low-byte stack pointer value is the lower eight bits of the 16-bit stack pointer address (SP7–SP0). The upper byte of the stack pointer value is located in register SPH (D8H). The SP value is undefined following a reset.

SYM — System Mode Register DEH Set 1

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	0	–	–	x	x	x	0	0
Read/Write	R/W	–	–	R/W	R/W	R/W	R/W	R/W
Addressing Mode	Register addressing mode only							

.7 Not used, But you must keep “0”

.6 and .5 Not used for S3C84BB/F84BB

.4–.2 Fast Interrupt Level Selection Bits

0	0	0	IRQ0
0	0	1	IRQ1
0	1	0	IRG2
0	1	1	IRQ3
1	0	0	IRQ4
1	0	1	IRQ5
1	1	0	IRQ6
1	1	1	IRQ7

.1 Fast Interrupt Enable Bit

0	Disable fast interrupt processing
1	Enable fast interrupt processing

.0 Global Interrupt Enable Bit (note)

0	Disable global interrupt processing
1	Enable global interrupt processing

NOTE: Following a reset, you enable global interrupt processing by executing an EI instruction (not by writing a "1" to SYM.0).

T1CON0 — Timer 1(0) Control Register EAH Set 1, Bank 1

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Addressing Mode	Register addressing mode only							

.7-.5 Timer 1 Input Clock Selection Bits

0	0	0	fxx/1024
0	0	1	fxx (Non-divide)
0	1	0	fxx/256
0	1	1	External clock falling edge
1	0	0	fxx/64
1	0	1	External clock rising edge
1	1	0	fxx/8
1	1	1	Counter stop

.4-.3 Timer 1 Operating Mode Selection Bits

0	0	Interval mode
0	1	Capture mode (Capture on rising edge, OVF can occur)
1	0	Capture mode (Capture on falling edge, OVF can occur)
1	1	PWM mode

.2 Timer 1 Counter Enable Bit

0	No effect
1	Clear the timer 1 counter (Auto-clear bit)

.1 Timer 1 Match/Capture Interrupt Enable Bit

0	Disable interrupt
1	Enable interrupt

.0 Timer 1 Overflow Interrupt Enable

0	Disable overflow interrupt
1	Enable overflow interrupt

T1CON1 — Timer 1(1) Control Register EBH Set 1, Bank 1

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Addressing Mode	Register addressing mode only							

.7-.5 Timer 1 Input Clock Selection Bits

0	0	0	fxx/1024
0	0	1	fxx (Non-divide)
0	1	0	fxx/256
0	1	1	External clock falling edge
1	0	0	fxx/64
1	0	1	External clock rising edge
1	1	0	fxx/8
1	1	1	Counter stop

.4-.3 Timer 1 Operating Mode Selection Bits

0	0	Interval mode
0	1	Capture mode (Capture on rising edge, OVF can occur)
1	0	Capture mode (Capture on falling edge, OVF can occur)
1	1	PWM mode

.2 Timer 1 Counter Enable Bit

0	No effect
1	Clear the timer 1 counter (Auto-clear bit)

.1 Timer 1 Match/Capture Interrupt Enable Bit

0	Disable interrupt
1	Enable interrupt

.0 Timer 1 Overflow Interrupt Enable

0	Disable overflow interrupt
1	Enable overflow interrupt

TACON — Timer A Control Register EAH Set 1, Bank 0

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	0	0	0	0	0	0	0	–
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	–
Addressing Mode	Register addressing mode only							

.7-.6 Timer A Input Clock Selection Bits

0	0	fxx/1024
0	1	fxx/256
1	0	fxx/64
1	1	External clock (TACK)

.5-.4 Timer A Operating Mode Selection Bits

0	0	Interval mode (TAOUT mode)
0	1	Capture mode (capture on rising edge, counter running, OVF can occur)
1	0	Capture mode (capture on falling edge, counter running, OVF can occur)
1	1	PWM mode (OVF interrupt can occur)

.3 Timer A Counter Clear Bit

0	No effect
1	Clear the timer A counter (After clearing, return to zero)

.2 Timer A Overflow Interrupt Enable Bit

0	Disable overflow interrupt
1	Enable overflow interrupt

.1 Timer A Match/Capture Interrupt Enable Bit

0	Disable interrupt
1	Enable interrupt

.0 Not used for the S3C84BB/F84BB

TBCON — Timer B Control Register D0H Set 1, Bank 0

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Addressing Mode	Register addressing mode only							

.7–.6 Timer B Input Clock Selection Bits

0	0	fxx
0	1	fxx/2
1	0	fxx/4
1	1	fxx/8

.5–.4 Timer B Interrupt Time Selection Bits

0	0	Elapsed time for low data value
0	1	Elapsed time for high data value
1	0	Elapsed time for low and high data values
1	1	Invalid setting

.3 Timer B Interrupt Enable Bit

0	Disable Interrupt
1	Enable Interrupt

.2 Timer B Start/Stop Bit

0	Stop timer B
1	Start timer B

.1 Timer B Mode Selection Bit

0	One-shot mode
1	Repeating mode

.0 Timer B Output flip-flop Control Bit

0	T-FF is low
1	T-FF is high

NOTE: fxx is selected clock for system.

TCCON0 — Timer C(0) Control Register F2H Set 1, Bank 1

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Addressing Mode	Register addressing mode only							

.7

Not used for the S3C84BB/F84BB (must keep always 0)

.6-.4 Timer C 3-bits Prescaler Bits

0	0	0	Non devided
0	0	1	Divided by 2
0	1	0	Divided by 3
0	1	1	Divided by 4
1	0	0	Divided by 5
1	0	1	Divided by 6
1	1	0	Divided by 7
1	1	1	Divided by 8

.3 Timer C Counter Clear Bit

0	No effect
1	Clear the timer C(0) counter (Auto-clear bit)

.2 Timer C Mode Selection Bit

0	fxx/1 & PWM mode
1	fxx/64 & interval mode

.1 Timer C Interrupt Enable Bit

0	Disable interrupt
1	Enable interrupt

.0 Timer C Pending Bit

0	No interrupt pending
0	*Clear pending bit when write*
1	Interrupt pending

TCCON1 — Timer C(1) Control Register F3H Set 1, Bank 1

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Addressing Mode	Register addressing mode only							

.7

Not used for the S3C84BB/F84BB (must keep always 0)

.6-.4 Timer C 3-bits Prescaler Bits

0	0	0	Non devided
0	0	1	Divided by 2
0	1	0	Divided by 3
0	1	1	Divided by 4
1	0	0	Divided by 5
1	0	1	Divided by 6
1	1	0	Divided by 7
1	1	1	Divided by 8

.3 Timer C Counter Clear Bit

0	No effect
1	Clear the timer C(1) counter (Auto-clear bit)

.2 Timer C Mode Selection Bit

0	fxx/1 & PWM mode
1	fxx/64 & interval mode

.1 Timer C Interrupt Enable Bit

0	Disable interrupt
1	Enable interrupt

.0 Timer C Pending Bit

0	No interrupt pending
0	*Clear pending bit when write*
1	Interrupt pending

TINTPND — Timer A,1 Interrupt Pending Register E9H Set 1, Bank 0

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	–	–	0	0	0	0	0	0
Read/Write	–	–	R/W	R/W	R/W	R/W	R/W	R/W
Addressing Mode	Register addressing mode only							

.7-.6 Not used for the S3C84BB/F84BB

.5 Timer 1(1) Overflow Interrupt Pending Bit

0	No interrupt pending
0	*Clear pending bit when write*
1	Interrupt pending

.4 Timer 1(1) Match/Capture Interrupt Pending Bit

0	No interrupt pending
0	*Clear pending bit when write*
1	Interrupt pending

.3 Timer 1(0) Overflow Interrupt Pending Bit

0	No interrupt pending
0	*Clear pending bit when write*
1	Interrupt pending

.2 Timer 1(0) Match/Capture Interrupt Pending Bit

0	No interrupt pending
0	*Clear pending bit when write*
1	Interrupt pending

.1 Timer A Overflow Interrupt Pending Bit

0	No interrupt pending
0	*Clear pending bit when write*
1	Interrupt pending

.0 Timer A Match/Capture Interrupt Pending Bit

0	No interrupt pending
0	*Clear pending bit when write*
1	Interrupt pending

UARTCON0 — UART0 Control Register E3H Set 1, Bank 1

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Addressing Mode	Register addressing mode only							

.7–.6 Operating mode and baud rate selection bits

0	0	Mode 0: SIO mode [fxx/(16 × (BRDATA0 + 1))]
0	1	Mode 1: 8-bit UART [fxx/(16 × (BRDATA0 + 1))]
1	0	Mode 2: 9-bit UART [fxx/16]
1	1	Mode 3: 9-bit UART [fxx/(16 × (BRDATA0 + 1))]

.5 Multiprocessor communication(1) enable bit (for modes 2 and 3 only)

0	Disable
1	Enable

.4 Serial data receive enable bit

0	Disable
1	Enable

.3 Location of the 9th data bit to be transmitted in UART mode 2 or 3 ("0" or "1")

.2 Location of the 9th data bit that was received in UART mode 2 or 3 ("0" or "1")

.1 Receive interrupt enable bit

0	Disable Receive interrupt
1	Enable Receive interrupt

.0 Transmit interrupt enable bit

0	Disable Transmit interrupt
1	Enable Transmit Interrupt

NOTES:

1. In mode 2 or 3, if the MCE (UARTCON.5) bit is set to "1", then the receive interrupt will not be activated if the received 9th data bit is "0". In mode 1, if MCE = "1", then the receive interrupt will not be activated if a valid stop bit was not received. In mode 0, the MCE(UARTCON.5) bit should be "0".
2. The descriptions for 8-bit and 9-bit UART mode do not include start and stop bits for serial data receive and transmit.

UARTCON1 — UART1 Control Register FBH Set 1, Bank 1

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	0	0	0	0	0	0	0	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Addressing Mode	Register addressing mode only							

.7–.6 Operating mode and baud rate selection bits

0	0	Mode 0: SIO mode [fxx/(16 × (BRDATA1 + 1))]
0	1	Mode 1: 8-bit UART [fxx/(16 × (BRDATA1 + 1))]
1	0	Mode 2: 9-bit UART [fxx/16]
1	1	Mode 3: 9-bit UART [fxx/(16 × (BRDATA1 + 1))]

.5 Multiprocessor communication(1) enable bit (for modes 2 and 3 only)

0	Disable
1	Enable

.4 Serial data receive enable bit

0	Disable
1	Enable

.3

Location of the 9th data bit to be transmitted in UART mode 2 or 3 ("0" or "1")

.2

Location of the 9th data bit that was received in UART mode 2 or 3 ("0" or "1")

.1 Receive interrupt enable bit

0	Disable Receive interrupt
1	Enable Receive interrupt

.0 Transmit interrupt enable bit

0	Disable Transmit interrupt
1	Enable Transmit Interrupt

NOTES:

1. In mode 2 or 3, if the MCE (UARTCON.5) bit is set to "1", then the receive interrupt will not be activated if the received 9th data bit is "0". In mode 1, if MCE = "1", then the receive interrupt will not be activated if a valid stop bit was not received. In mode 0, the MCE(UARTCON.5) bit should be "0".
2. The descriptions for 8-bit and 9-bit UART mode do not include start and stop bits for serial data receive and transmit.

UARTPND — UART0,1 Pending Register E5H Set 1, Bank 1

Bit Identifier	.7	.6	.5	.4	.3	.2	.1	.0
RESET Value	–	–	–	–	0	0	0	0
Read/Write	–	–	–	–	R/W	R/W	R/W	R/W
Addressing Mode	Register addressing mode only							

.7–.4

Not used for S3C84BB/F84BB

.3 UART1 receive interrupt pending flag

0	Not pending
0	*Clear pending bit (when write)*
1	Interrupt pending

.2 UART1 transmit interrupt pending flag

0	Not pending
0	*Clear pending bit (when write)*
1	Interrupt pending

.1 UART0 receive interrupt pending flag

0	Not pending
0	*Clear pending bit (when write)*
1	Interrupt pending

.0 UART0 transmit interrupt pending flag

0	Not pending
0	*Clear pending bit (when write)*
1	Interrupt pending

NOTES:

1. In order to clear a data transmit or receive interrupt pending flag, you must write a "0" to the appropriate pending bit.
2. To avoid programming errors, we recommend using load instruction (except for LDB), when manipulating UARTPND values.

5 INTERRUPT STRUCTURE

OVERVIEW

The S3C8-series interrupt structure has three basic components: levels, vectors, and sources. The SAM8 CPU recognizes up to eight interrupt levels and supports up to 128 interrupt vectors. When a specific interrupt level has more than one vector address, the vector priorities are established in hardware. A vector address can be assigned to one or more sources.

Levels

Interrupt levels are the main unit for interrupt priority assignment and recognition. All peripherals and I/O blocks can issue interrupt requests. In other words, peripheral and I/O operations are interrupt-driven. There are eight possible interrupt levels: IRQ0–IRQ7, also called level 0–level 7. Each interrupt level directly corresponds to an interrupt request number (IRQn). The total number of interrupt levels used in the interrupt structure varies from device to device. The S3C84BB/F84BB interrupt structure recognizes eight interrupt levels.

The interrupt level numbers 0 through 7 do not necessarily indicate the relative priority of the levels. They are just identifiers for the interrupt levels that are recognized by the CPU. The relative priority of different interrupt levels is determined by settings in the interrupt priority register, IPR. Interrupt group and subgroup logic controlled by IPR settings lets you define more complex priority relationships between different levels.

Vectors

Each interrupt level can have one or more interrupt vectors, or it may have no vector address assigned at all. The maximum number of vectors that can be supported for a given level is 128 (The actual number of vectors used for S3C8-series devices is always much smaller). If an interrupt level has more than one vector address, the vector priorities are set in hardware. S3C84BB/F84BB uses twenty four vectors.

Sources

A source is any peripheral that generates an interrupt. A source can be an external pin or a counter overflow. Each vector can have several interrupt sources. In the S3C84BB/F84BB interrupt structure, there are twenty four possible interrupt sources.

When a service routine starts, the respective pending bit should be either cleared automatically by hardware or cleared "manually" by program software. The characteristics of the source's pending mechanism determine which method would be used to clear its respective pending bit.

INTERRUPT TYPES

The three components of the S3C8 interrupt structure described before — levels, vectors, and sources — are combined to determine the interrupt structure of an individual device and to make full use of its available interrupt logic. There are three possible combinations of interrupt structure components, called interrupt types 1, 2, and 3. The types differ in the number of vectors and interrupt sources assigned to each level (see Figure 5-1):

Type 1: One level (IRQn) + one vector (V_1) + one source (S_1)

Type 2: One level (IRQn) + one vector (V_1) + multiple sources ($S_1 – S_n$)

Type 3: One level (IRQn) + multiple vectors ($V_1 – V_n$) + multiple sources ($S_1 – S_n$, $S_{n+1} – S_{n+m}$)

In the S3C84BB/F84BB microcontroller, two interrupt types are implemented.

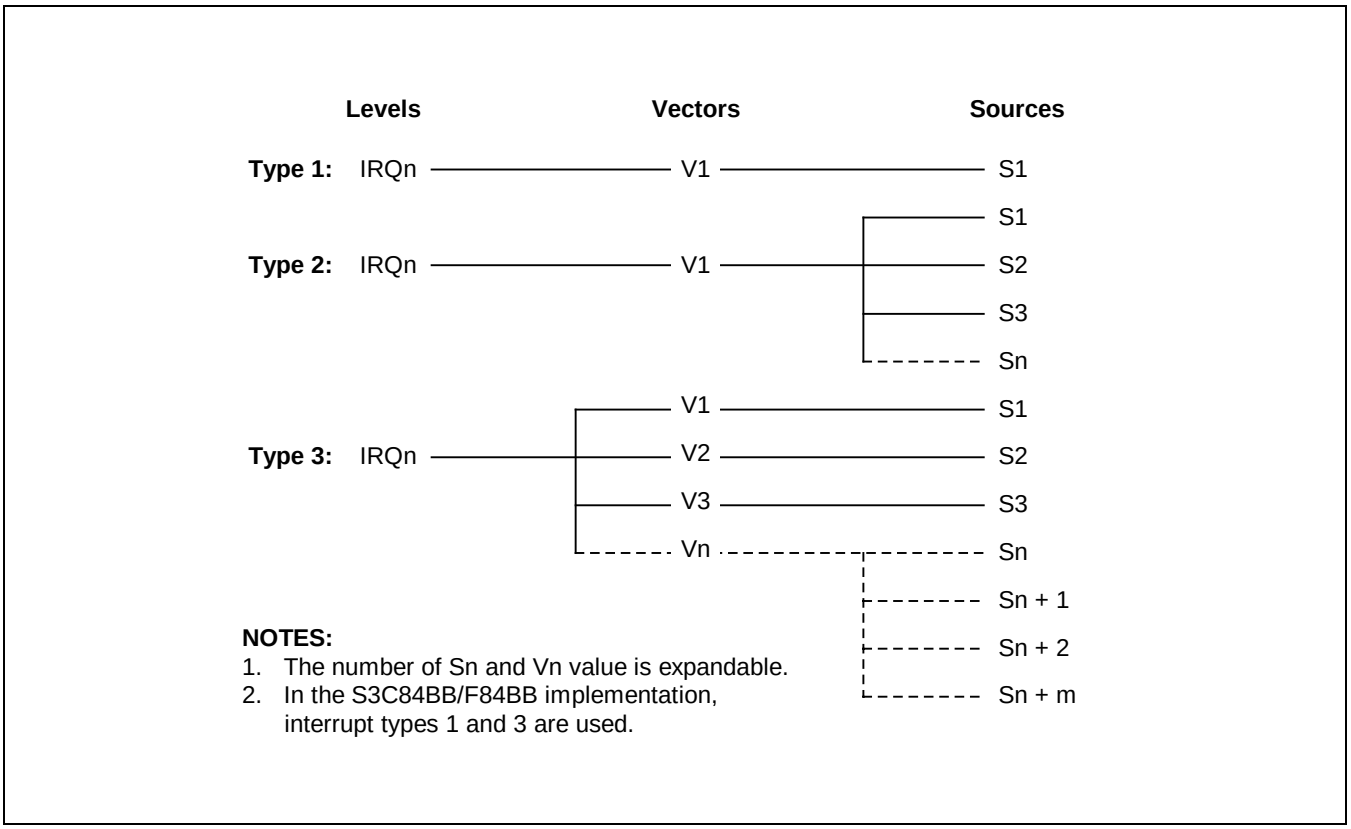


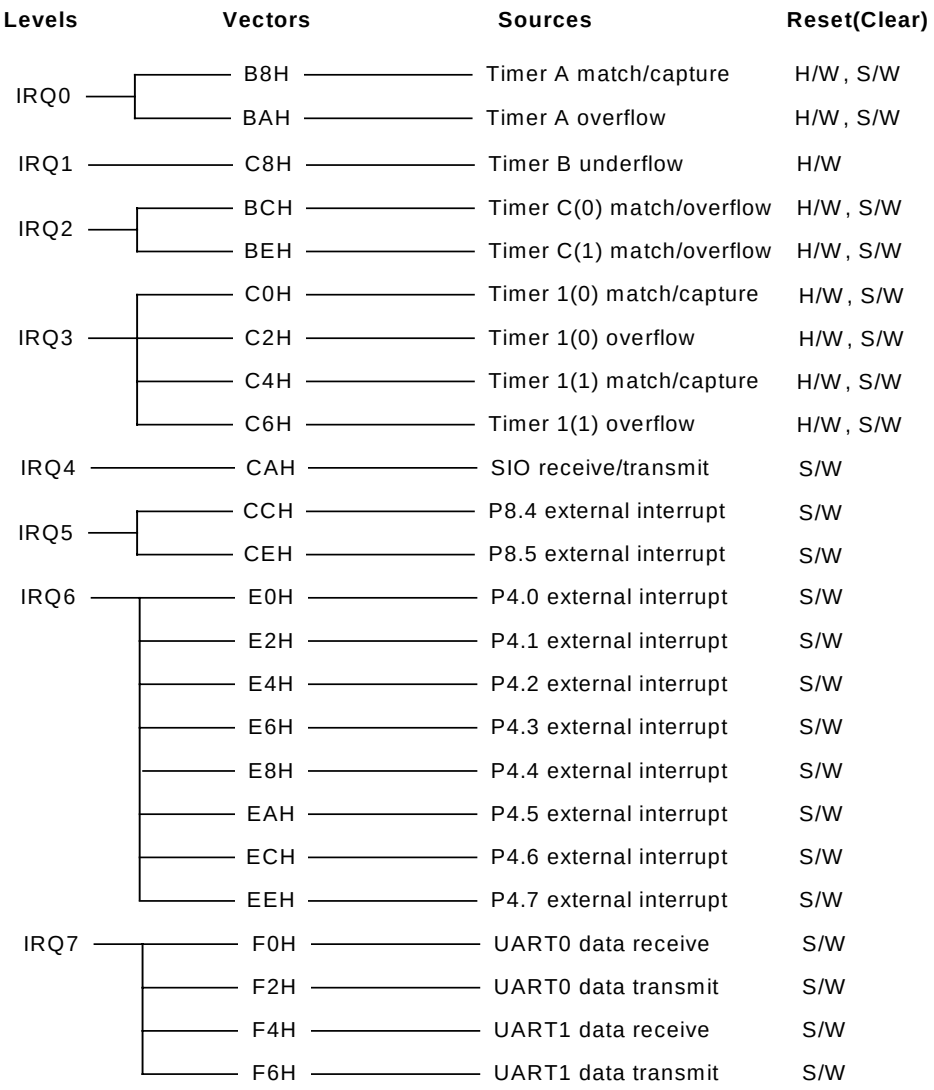
Figure 5-1. S3C8-Series Interrupt Types

S3C84BB/F84BB INTERRUPT STRUCTURE

The S3C84BB/F84BB microcontroller supports twenty four interrupt sources. All twenty four of the interrupt sources have a corresponding interrupt vector address. Eight interrupt levels are recognized by the CPU in this device-specific interrupt structure, as shown in Figure 5-2.

When multiple interrupt levels are active, the interrupt priority register (IPR) determines the order in which contending interrupts are to be serviced. If multiple interrupts occur within the same interrupt level, the interrupt with the lowest vector address is usually processed first (The relative priorities of multiple interrupts within a single level are fixed in hardware).

When the CPU grants an interrupt request, interrupt processing starts. All other interrupts are disabled and the program counter value and status flags are pushed to stack. The starting address of the service routine is fetched from the appropriate vector address (plus the next 8-bit value to concatenate the full 16-bit address) and the service routine is executed.



NOTES:

1. Within a given interrupt level, the lower vector address has high priority. For example, B8H has higher priority than BAH within the level IRQ0 the priorities within each level are set at the factory.
2. External interrupts are triggered by a rising or falling edge, depending on the corresponding control register setting.

Figure 5-2. S3C84BB/F84BB Interrupt Structure

INTERRUPT VECTOR ADDRESSES

All interrupt vector addresses for the S3C84BB/F84BB interrupt structure are stored in the vector address area of the internal 64-Kbyte ROM, 0H–FFFFH (see Figure 5-3).

You can allocate unused locations in the vector address area as normal program memory. If you do so, please be careful not to overwrite any of the stored vector addresses (Table 5-1 lists all vector addresses).

The program reset address in the ROM is 0100H.

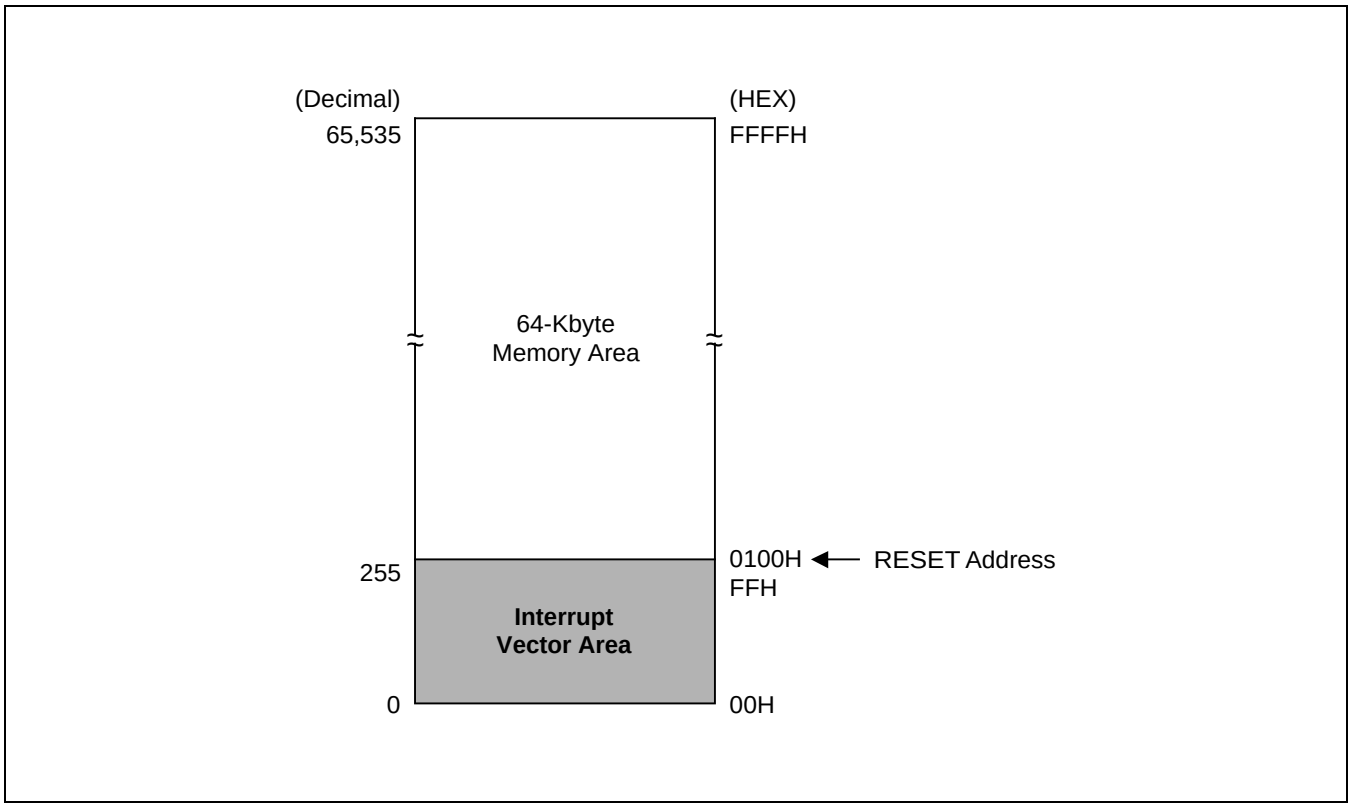


Figure 5-3. ROM Vector Address Area

Table 5-1. Interrupt Vectors

Vector Address		Interrupt Source	Request		Reset/Clear	
Decimal Value	Hex Value		Interrupt Level	Priority in Level	H/W	S/W
256	100H	Basic timer(WDT) overflow	RESETB	-	√	
246	F6H	UART1 transmit	IRQ7	3		√
244	F4H	UART1 receive		2		√
242	F2H	UART0 transmit		1		√
240	F0H	UART0 receive		0		√
238	EEH	P4.7 external interrupt	IRQ6	7		√
236	ECH	P4.6 external interrupt		6		√
234	EAH	P4.5 external interrupt		5		√
232	E8H	P4.4 external interrupt		4		√
230	E6H	P4.3 external interrupt		3		√
228	E4H	P4.2 external interrupt		2		√
226	E2H	P4.1 external interrupt		1		√
224	E0H	P4.0 external interrupt		0		√
206	CEH	P8.5 external interrupt	IRQ5	1		√
204	CCH	P8.4 external interrupt		0		√
202	CAH	SIO receive/transmit	IRQ4	-		√
198	C6H	Timer 1(1) overflow	IRQ3	3	√	√
196	C4H	Timer 1(1) match/capture		2	√	√
194	C2H	Timer 1(0) overflow		1	√	√
192	C0H	Timer 1(0) match/capture		0	√	√
190	BEH	Timer C(1) match/overflow	IRQ2	1	√	√
188	BCH	Timer C(0) match/overflow		0	√	√
200	C8H	Timer B underflow	IRQ1	-	√	
186	BAH	Timer A overflow	IRQ0	1	√	√
184	B8H	Timer A match/capture		0	√	√

NOTES:

1. Interrupt priorities are identified in inverse order: "0" is the highest priority, "1" is the next highest, and so on.
2. If two or more interrupts within the same level contend, the interrupt with the lowest vector address usually has priority over one with a higher vector address. The priorities within a given level are fixed in hardware.

ENABLE/DISABLE INTERRUPT INSTRUCTIONS (EI, DI)

Executing the Enable Interrupts (EI) instruction globally enables the interrupt structure. All interrupts are then serviced as they occur according to the established priorities.

NOTE

The system initialization routine executed after a reset must always contain an EI instruction to globally enable the interrupt structure.

During the normal operation, you can execute the DI (Disable Interrupt) instruction at any time to globally disable interrupt processing. The EI and DI instructions change the value of bit 0 in the SYM register.

SYSTEM-LEVEL INTERRUPT CONTROL REGISTERS

In addition to the control registers for specific interrupt sources, four system-level registers control interrupt processing:

— The interrupt mask register, IMR, enables (un-masks) or disables (masks) interrupt levels.

— The interrupt priority register, IPR, controls the relative priorities of interrupt levels.

— The interrupt request register, IRQ, contains interrupt pending flags for each interrupt level (as opposed to each interrupt source).

— The system mode register, SYM, enables or disables global interrupt processing (SYM settings also enable fast interrupts and control the activity of external interface, if implemented).

Table 5-2. Interrupt Control Register Overview

Control Register	ID	R/W	Function Description
Interrupt mask register	IMR	R/W	Bit settings in the IMR register enable or disable interrupt processing for each of the eight interrupt levels: IRQ0–IRQ7.
Interrupt priority register	IPR	R/W	Controls the relative processing priorities of the interrupt levels. The seven levels of S3C84BB/F84BB are organized into three groups: A, B, and C. Group A is IRQ0 and IRQ1, group B is IRQ2, IRQ3 and IRQ4, and group C is IRQ5, IRQ6, and IRQ7.
Interrupt request register	IRQ	R	This register contains a request pending bit for each interrupt level.
System mode register	SYM	R/W	This register enables/disables fast interrupt processing, dynamic global interrupt processing, and external interface control (An external memory interface is not implemented in the S3C84BB/F84BB microcontroller).

NOTE: Before IMR register is changed to any value, all interrupts must be disable.
Using DI instruction is recommended.

INTERRUPT PROCESSING CONTROL POINTS

Interrupt processing can therefore be controlled in two ways: globally or by specific interrupt level and source. The system-level control points in the interrupt structure are:

— Global interrupt enable and disable (by EI and DI instructions or by direct manipulation of SYM.0)

— Interrupt level enable/disable settings (IMR register)

— Interrupt level priority settings (IPR register)

— Interrupt source enable/disable settings in the corresponding peripheral control registers

NOTE

When writing an application program that handles interrupt processing, be sure to include the necessary register file address (register pointer) information.

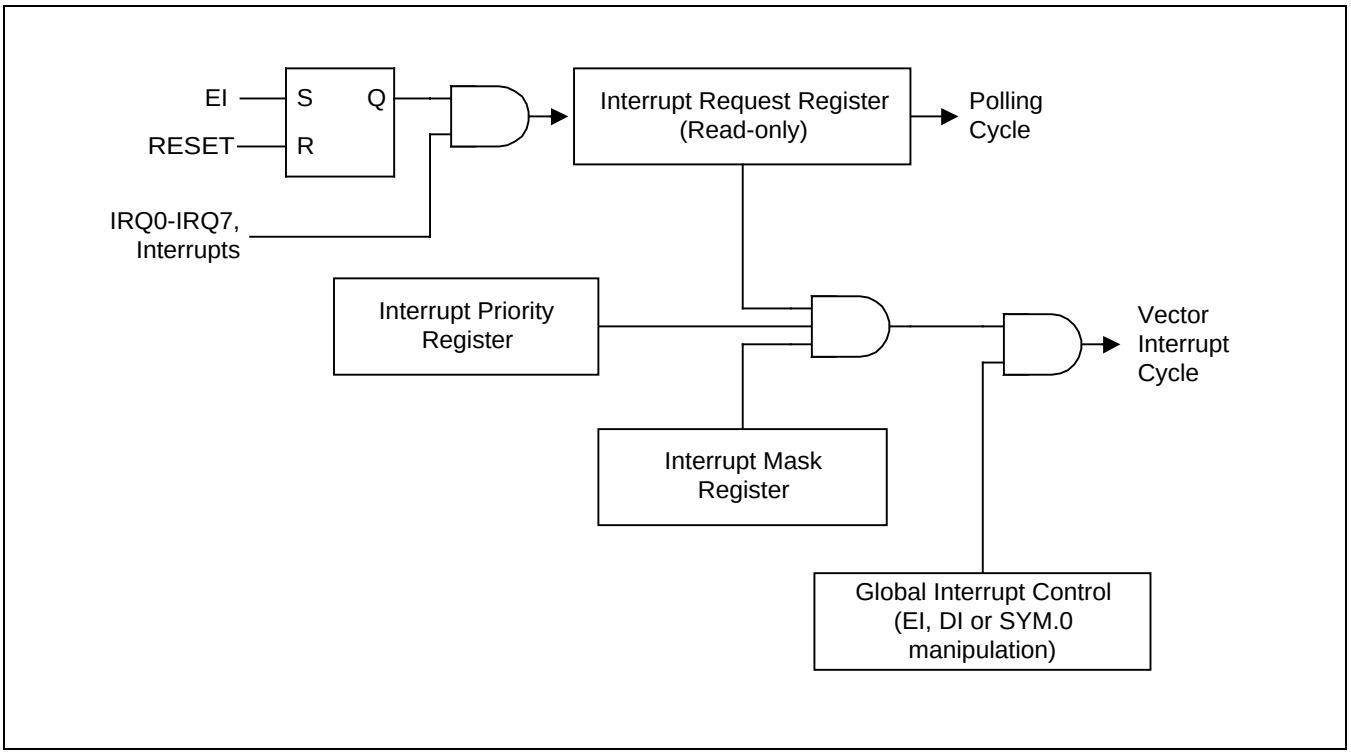


Figure 5-4. Interrupt Function Diagram

PERIPHERAL INTERRUPT CONTROL REGISTERS

For each interrupt source there is one or more corresponding peripheral control registers that let you control the interrupt generated by the related peripheral (see Table 5-3).

Table 5-3. Interrupt Source Control and Data Registers

Interrupt Source	Interrupt Level	Register(s)	Location(s) in Set 1
Timer A overflow	IRQ0	TINTPND	E9H, bank 0
Timer A match/capture		TACON	EAH, bank 0
		TADATA	EBH, bank 0
		TACNT	ECH, bank 0
Timer B underflow	IRQ1	TBCON	D0H, bank 0
		TBDATAH, TBDATAL	D1H, D2H, bank 0
Timer C(0) match/overflow	IRQ2	TCCON0	F2H, bank 1
Timer C(1) match/overflow		TCCON1	F3H, bank 1
		TCDATA0	F0H, bank 1
		TCDATA1	F1H, bank 1
Timer1(0) match/capture	IRQ3	T1DATAH0,T1DATAL0	E6H,E7H, bank 1
Timer1(0) overflow		T1DATAH1,T1DATAL1	E8H,E9H, bank 1
Timer1(1)match/capture		T1CON0, T1CON1	EAH,EBH, bank1
Timer1(1)overflow		T1CNTH0, T1CNTL0	ECH, EDH, bank1
		T1CNTH1, T1CNTL1	EEH, EFH, bank1
SIO receive/transmit	IRQ4	SIOCON, SIODATA	E1H,E0H, bank1
P8.5 external interrupt	IRQ5	P8CONH,P8CONL	EDH,EEH, bank0
P8.4 external interrupt		P8INTPND	EFH, bank0
P4.7 external interrupt	IRQ6	P4CONH	F6H, bank 0
P4.6 external interrupt		P4CONL	F7H, bank 0
P4.5 external interrupt		P4INT	FAH, bank 0
P4.4 external interrupt		P4INTPND	FBH, bank 0
P4.3 external interrupt			
P4.2 external interrupt			
P4.1 external interrupt			
P4.0 external interrupt			
UART0 receive/transmit	IRQ7	UARTCON0	E3H, bank 1
UART1 receive/transmit		UARTCON1	FBH, bank 1
		UDATA0, UDATA1	E2H, FAH, bank 1
		UARTPND	E5H, bank 1

SYSTEM MODE REGISTER (SYM)

The system mode register, SYM (set 1, DEH), is used to globally enable and disable interrupt processing (see Figure 5-5).

A reset clears SYM.0 to "0".

The instructions EI and DI enable and disable global interrupt processing, respectively, by modifying the bit 0 value of the SYM register. In order to enable interrupt processing an Enable Interrupt (EI) instruction must be included in the initialization routine, which follows a reset operation. Although you can manipulate SYM.0 directly to enable and disable interrupts during the normal operation, it is recommended to use the EI and DI instructions for this purpose.

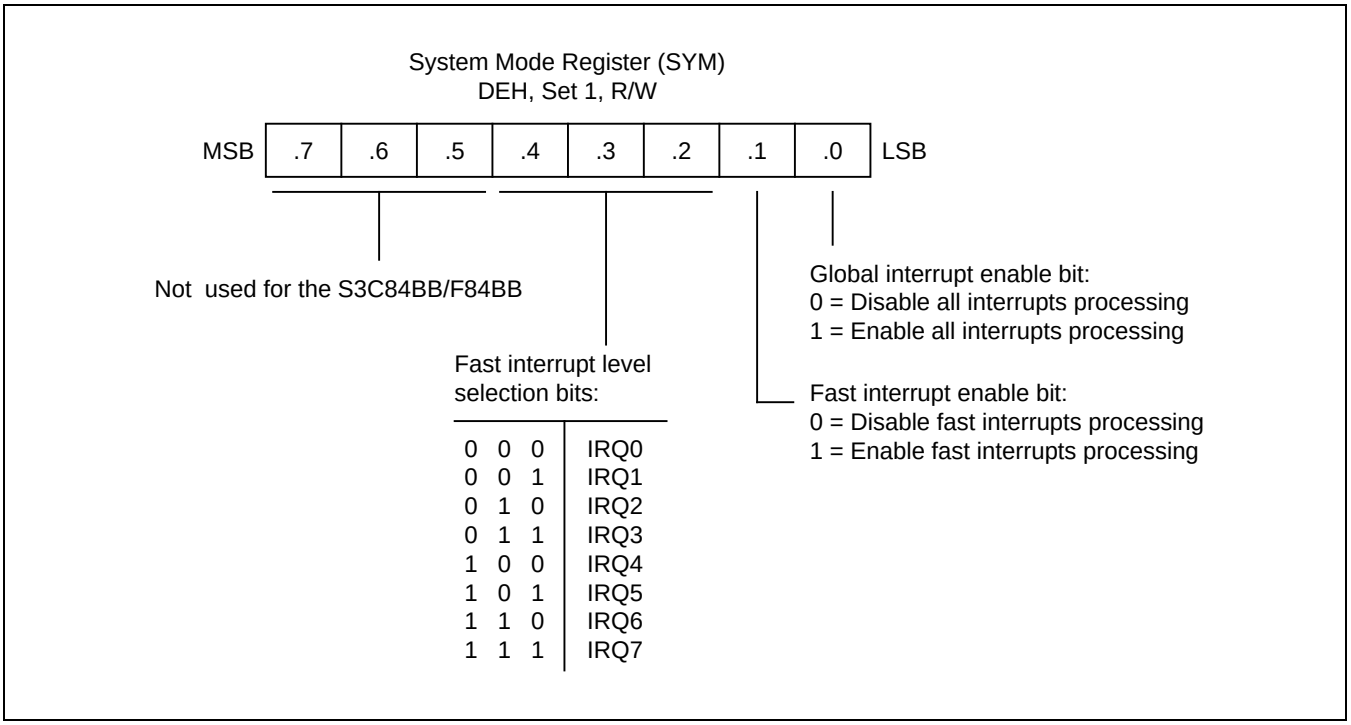


Figure 5-5. System Mode Register (SYM)

INTERRUPT MASK REGISTER (IMR)

The interrupt mask register, IMR (set 1, DDH) is used to enable or disable interrupt processing for individual interrupt levels. After a reset, all IMR bit values are undetermined and must therefore be written to their required settings by the initialization routine.

Each IMR bit corresponds to a specific interrupt level: bit 1 to IRQ1, bit 2 to IRQ2, and so on. When the IMR bit of an interrupt level is cleared to "0", interrupt processing for that level is disabled (masked). When you set a level's IMR bit to "1", interrupt processing for the level is enabled (not masked).

The IMR register is mapped to register location DDH in set 1. Bit values can be read and written by instructions using the Register addressing mode.

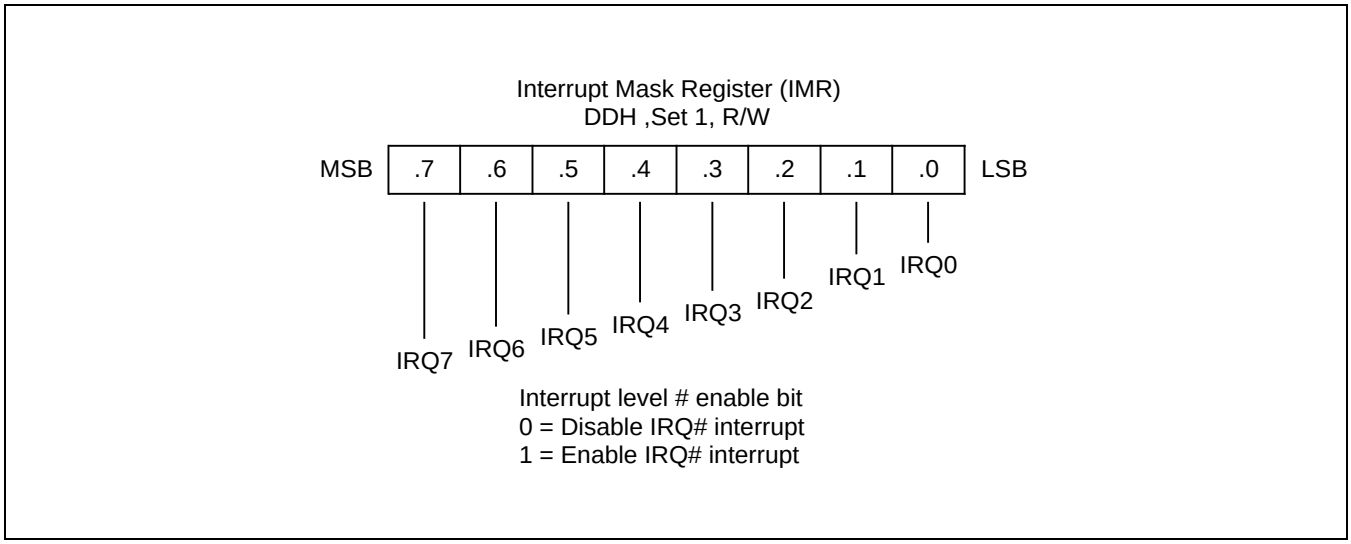


Figure 5-6. Interrupt Mask Register (IMR)

INTERRUPT PRIORITY REGISTER (IPR)

The interrupt priority register, IPR (set 1, bank 0, FFH), is used to set the relative priorities of the interrupt levels in the microcontroller's interrupt structure. After a reset, all IPR bit values are undetermined and must therefore be written to their required settings by the initialization routine.

When more than one interrupt sources are active, the source with the highest priority level is serviced first. If two sources belong to the same interrupt level, the source with the lower vector address usually has the priority (This priority is fixed in hardware).

To support programming of the relative interrupt level priorities, they are organized into groups and subgroups by the interrupt logic. Please note that these groups (and subgroups) are used only by IPR logic for the IPR register priority definitions (see Figure 5-7):

Group A IRQ0, IRQ1

Group B IRQ2, IRQ3, IRQ4

Group C IRQ5, IRQ6, IRQ7

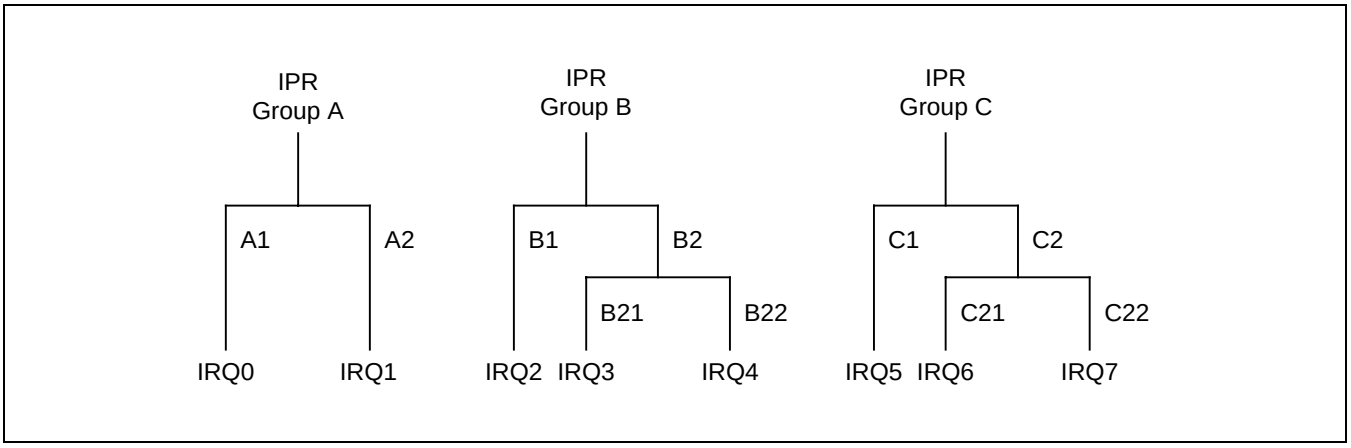


Figure 5-7. Interrupt Request Priority Groups

As you can see in Figure 5-8, IPR.7, IPR.4, and IPR.1 control the relative priority of interrupt groups A, B, and C. For example, the setting "001B" for these bits would select the group relationship B > C > A. The setting "101B" would select the relationship C > B > A.

The functions of the other IPR bit settings are as follows:

— IPR.5 controls the relative priorities of group C interrupts.

— Interrupt group C includes a subgroup that has an additional priority relationship among the interrupt levels 5, 6, and 7. IPR.6 defines the subgroup C relationship. IPR.5 controls the interrupt group C.

— IPR.0 controls the relative priority setting of IRQ0 and IRQ1 interrupts.

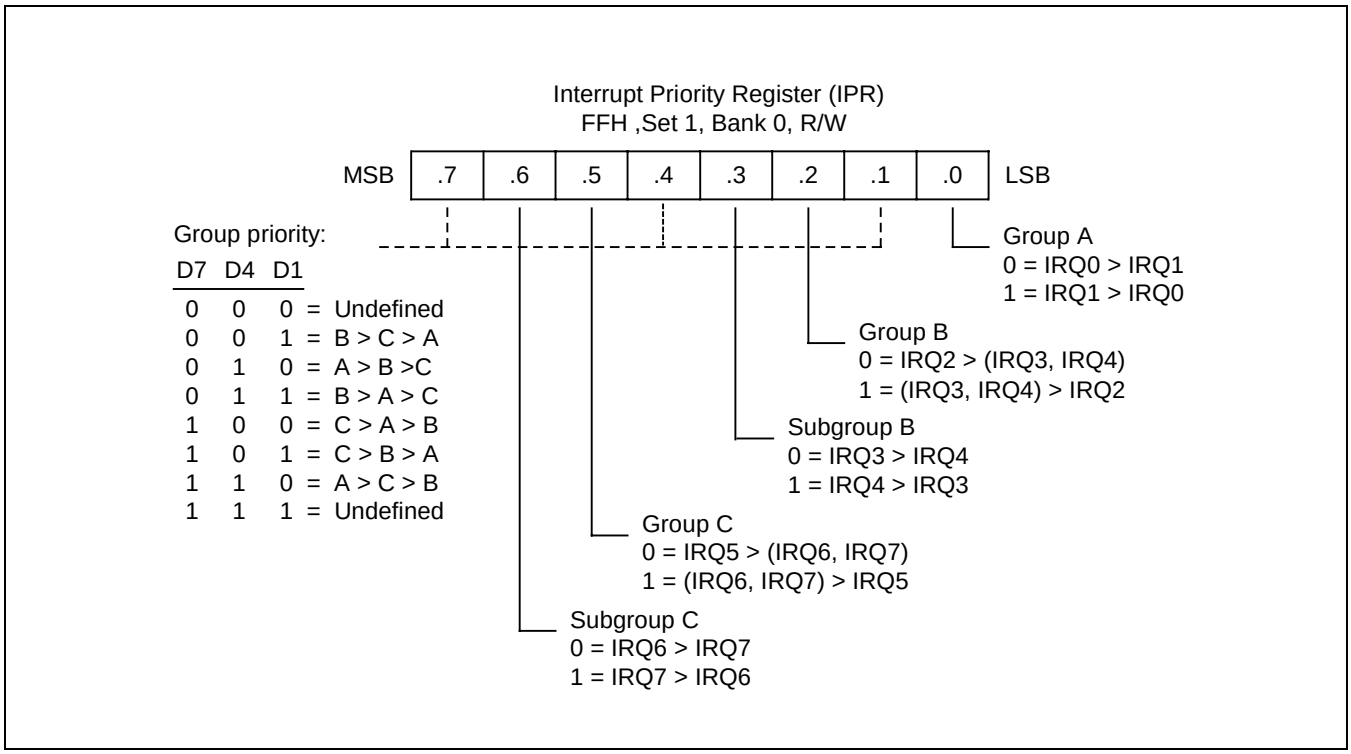


Figure 5-8. Interrupt Priority Register (IPR)

INTERRUPT REQUEST REGISTER (IRQ)

You can poll bit values in the interrupt request register, IRQ (set 1, DCH), to monitor interrupt request status for all levels in the microcontroller's interrupt structure. Each bit corresponds to the interrupt level of the same number: bit 0 to IRQ0, bit 1 to IRQ1, and so on. A "0" indicates that no interrupt request is currently being issued for that level. A "1" indicates that an interrupt request has been generated for that level.

IRQ bit values are read-only addressable using Register addressing mode. You can read (test) the contents of the IRQ register at any time using bit or byte addressing to determine the current interrupt request status of specific interrupt levels. After a reset, all IRQ status bits are cleared to "0".

You can poll IRQ register values even if a DI instruction has been executed (that is, if global interrupt processing is disabled). If an interrupt occurs while the interrupt structure is disabled, the CPU will not service it. You can, however, still detect the interrupt request by polling the IRQ register. In this way, you can determine which events occurred while the interrupt structure was globally disabled.

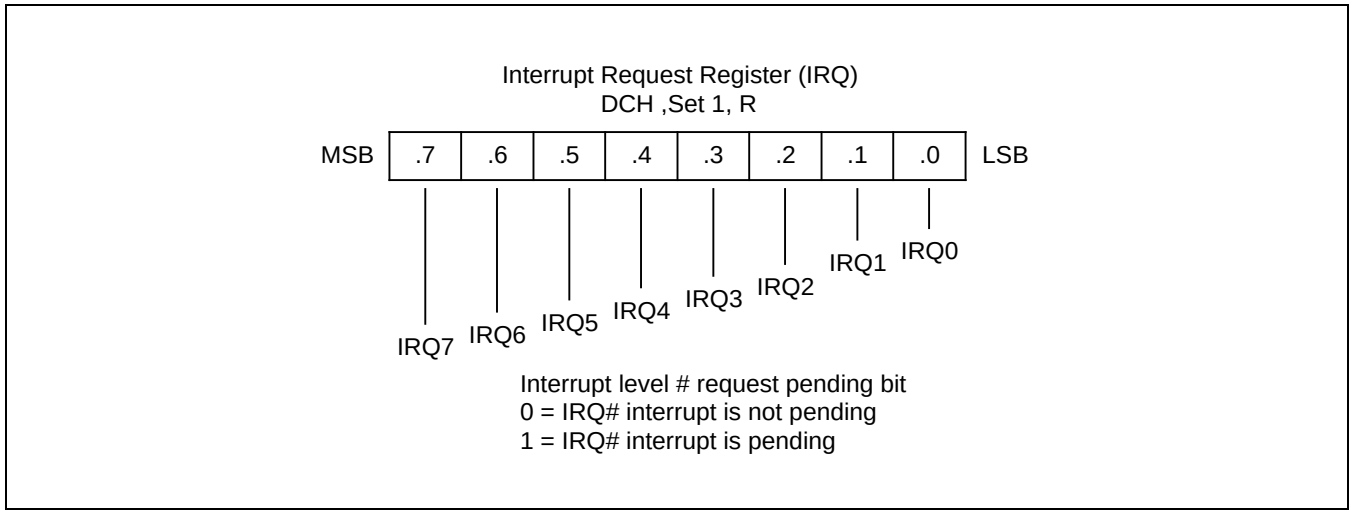


Figure 5-9. Interrupt Request Register (IRQ)

INTERRUPT PENDING FUNCTION TYPES

Overview

There are two types of interrupt pending bits: one type that is automatically cleared by hardware after the interrupt service routine is acknowledged and executed; the other that must be cleared in the interrupt service routine.

Pending Bits Cleared Automatically by Hardware

For interrupt pending bits that are cleared automatically by hardware, interrupt logic sets the corresponding pending bit to "1" when a request occurs. It then issues an IRQ pulse to inform the CPU that an interrupt is waiting to be serviced. The CPU acknowledges the interrupt source by sending an IACK, executes the service routine, and clears the pending bit to "0". This type of pending bit is not mapped and cannot, therefore, be read or written by application software.

In the S3C84BB/F84BB interrupt structure, the timer B underflow interrupt (IRQ1) belongs to this category of interrupts in which pending condition is cleared automatically by hardware.

Pending Bits Cleared by the Service Routine

The second type of pending bit is the one that should be cleared by program software. The service routine must clear the appropriate pending bit before a return-from-interrupt subroutine (IRET) occurs. To do this, a "0" must be written to the corresponding pending bit location in the source's mode or control register.

In the S3C84BB/F84BB interrupt structure, pending conditions for IRQ4, IRQ5, IRQ6, and IRQ7 must be cleared in the interrupt service routine.

INTERRUPT SOURCE POLLING SEQUENCE

The interrupt request polling and servicing sequence is as follows:

1. A source generates an interrupt request by setting the interrupt request bit to "1".
2. The CPU polling procedure identifies a pending condition for that source.
3. The CPU checks the source's interrupt level.
4. The CPU generates an interrupt acknowledge signal.
5. Interrupt logic determines the interrupt's vector address.
6. The service routine starts and the source's pending bit is cleared to "0" (by hardware or by software).
7. The CPU continues polling for interrupt requests.

INTERRUPT SERVICE ROUTINES

Before an interrupt request is serviced, the following conditions must be met:

— Interrupt processing must be globally enabled (EI, SYM.0 = "1")
— The interrupt level must be enabled (IMR register)
— The interrupt level must have the highest priority if more than one level is currently requesting service
— The interrupt must be enabled at the interrupt's source (peripheral control register)

When all the above conditions are met, the interrupt request is acknowledged at the end of the instruction cycle. The CPU then initiates an interrupt machine cycle that completes the following processing sequence:

1. Reset (clear to "0") the interrupt enable bit in the SYM register (SYM.0) to disable all subsequent interrupts.
2. Save the program counter (PC) and status flags to the system stack.
3. Branch to the interrupt vector to fetch the address of the service routine.
4. Pass control to the interrupt service routine.

When the interrupt service routine is completed, the CPU issues an Interrupt Return (IRET). The IRET restores the PC and status flags, setting SYM.0 to "1". It allows the CPU to process the next interrupt request.

GENERATING INTERRUPT VECTOR ADDRESSES

The interrupt vector area in the ROM (00H–FFH) contains the addresses of interrupt service routines that correspond to each level in the interrupt structure. Vectored interrupt processing follows this sequence:

1. Push the program counter's low-byte value to the stack.
2. Push the program counter's high-byte value to the stack.
3. Push the FLAG register values to the stack.
4. Fetch the service routine's high-byte address from the vector location.
5. Fetch the service routine's low-byte address from the vector location.
6. Branch to the service routine specified by the concatenated 16-bit vector address.

NOTE

A 16-bit vector address always begins at an even-numbered ROM address within the range of 00H–FFH.

NESTING OF VECTORED INTERRUPTS

It is possible to nest a higher-priority interrupt request while a lower-priority request is being serviced. To do this, you must follow these steps:

1. Push the current 8-bit interrupt mask register (IMR) value to the stack (PUSH IMR).
2. Load the IMR register with a new mask value that enables only the higher priority interrupt.
3. Execute an EI instruction to enable interrupt processing (a higher priority interrupt will be processed if it occurs).
4. When the lower-priority interrupt service routine ends, restore the IMR to its original value by returning the previous mask value from the stack (POP IMR).
5. Execute an IRET.

Depending on the application, you may be able to simplify the procedure above to some extent.

NOTES

6 INSTRUCTION SET

OVERVIEW

The instruction set is specifically designed to support large register files that are typical of most S3C8-series microcontrollers. There are 78 instructions. The powerful data manipulation capabilities and features of the instruction set include:

— A full complement of 8-bit arithmetic and logic operations, including multiply and divide
— No special I/O instructions (I/O control/data registers are mapped directly into the register file)
— Decimal adjustment included in binary-coded decimal (BCD) operations
— 16-bit (word) data can be incremented and decremented
— Flexible instructions for bit addressing, rotate, and shift operations

DATA TYPES

The CPU performs operations on bits, bytes, BCD digits, and two-byte words. Bits in the register file can be set, cleared, complemented, and tested. Bits within a byte are numbered from 7 to 0, where bit 0 is the least significant (right-most) bit.

REGISTER ADDRESSING

To access an individual register, an 8-bit address in the range 0–255 or the 4-bit address of a working register is specified. Paired registers can be used to construct 16-bit data, 16-bit program memory or data memory addresses. For detailed information about register addressing, please refer to Chapter 2, "Address Spaces."

ADDRESSING MODES

There are seven explicit addressing modes: Register (R), Indirect Register (IR), Indexed (X), Direct (DA), Relative (RA), Immediate (IM), and Indirect (IA). For detailed descriptions of these addressing modes, please refer to Chapter 3, "Addressing Modes."

Table 6-1. Instruction Group Summary

Mnemonic	Operands	Instruction
Load Instructions		
CLR	dst	Clear
LD	dst,src	Load
LDB	dst,src	Load bit
LDE	dst,src	Load external data memory
LDC	dst,src	Load program memory
LDED	dst,src	Load external data memory and decrement
LDCD	dst,src	Load program memory and decrement
LDEI	dst,src	Load external data memory and increment
LDCI	dst,src	Load program memory and increment
LDEPD	dst,src	Load external data memory with pre-decrement
LDCPD	dst,src	Load program memory with pre-decrement
LDEPI	dst,src	Load external data memory with pre-increment
LDCPI	dst,src	Load program memory with pre-increment
LDW	dst,src	Load word
POP	dst	Pop from stack
POPUD	dst,src	Pop user stack (decrementing)
POPUI	dst,src	Pop user stack (incrementing)
PUSH	src	Push to stack
PUSHUD	dst,src	Push user stack (decrementing)
PUSHUI	dst,src	Push user stack (incrementing)

NOTE: LDE, LDED, LDEI, LDEPP, and LDEPI instructions can be used to read/write the data from the 64-Kbyte data memory.

Table 6-1. Instruction Group Summary (Continued)

Mnemonic	Operands	Instruction
Arithmetic Instructions		
ADC	dst,src	Add with carry
ADD	dst,src	Add
CP	dst,src	Compare
DA	dst	Decimal adjust
DEC	dst	Decrement
DECW	dst	Decrement word
DIV	dst,src	Divide
INC	dst	Increment
INCW	dst	Increment word
MULT	dst,src	Multiply
SBC	dst,src	Subtract with carry
SUB	dst,src	Subtract
Logic Instructions		
AND	dst,src	Logical AND
COM	dst	Complement
OR	dst,src	Logical OR
XOR	dst,src	Logical exclusive OR

Table 6-1. Instruction Group Summary (Continued)

Mnemonic	Operands	Instruction
Program Control Instructions		
BTJRF	dst,src	Bit test and jump relative on false
BTJRT	dst,src	Bit test and jump relative on true
CALL	dst	Call procedure
CPIJE	dst,src	Compare, increment and jump on equal
CPIJNE	dst,src	Compare, increment and jump on non-equal
DJNZ	r,dst	Decrement register and jump on non-zero
ENTER		Enter
EXIT		Exit
IRET		Interrupt return
JP	cc,dst	Jump on condition code
JP	dst	Jump unconditional
JR	cc,dst	Jump relative on condition code
NEXT		Next
RET		Return
WFI		Wait for interrupt
Bit Manipulation Instructions		
BAND	dst,src	Bit AND
BCP	dst,src	Bit compare
BITC	dst	Bit complement
BITR	dst	Bit reset
BITS	dst	Bit set
BOR	dst,src	Bit OR
BXOR	dst,src	Bit XOR
TCM	dst,src	Test complement under mask
TM	dst,src	Test under mask

Table 6-1. Instruction Group Summary (Concluded)

Mnemonic	Operands	Instruction
Rotate and Shift Instructions		
RL	dst	Rotate left
RLC	dst	Rotate left through carry
RR	dst	Rotate right
RRC	dst	Rotate right through carry
SRA	dst	Shift right arithmetic
SWAP	dst	Swap nibbles
CPU Control Instructions		
CCF		Complement carry flag
DI		Disable interrupts
EI		Enable interrupts
IDLE		Enter Idle mode
NOP		No operation
RCF		Reset carry flag
SB0		Set bank 0
SB1		Set bank 1
SCF		Set carry flag
SRP	src	Set register pointers
SRP0	src	Set register pointer 0
SRP1	src	Set register pointer 1
STOP		Enter Stop mode

FLAGS REGISTER (FLAGS)

The flags register FLAGS contains eight bits which describe the current status of CPU operations. Four of these bits, FLAGS.7–FLAGS.4, can be tested and used with conditional jump instructions. Two other flag bits, FLAGS.3 and FLAGS.2, are used for BCD arithmetic.

The FLAGS register also contains a bit to indicate the status of fast interrupt processing (FLAGS.1) and a bank address status bit (FLAGS.0) to indicate whether register bank 0 or bank 1 is currently being addressed.

FLAGS register can be set or reset by instructions as long as its outcome does not affect the flags, such as, Load instruction. Logical and Arithmetic instructions such as, AND, OR, XOR, ADD, and SUB can affect the Flags register. For example, the AND instruction updates the Zero, Sign and Overflow flags based on the outcome of the AND instruction. If the AND instruction uses the Flags register as the destination, then two write will simultaneously occur to the Flags register producing an unpredictable result.

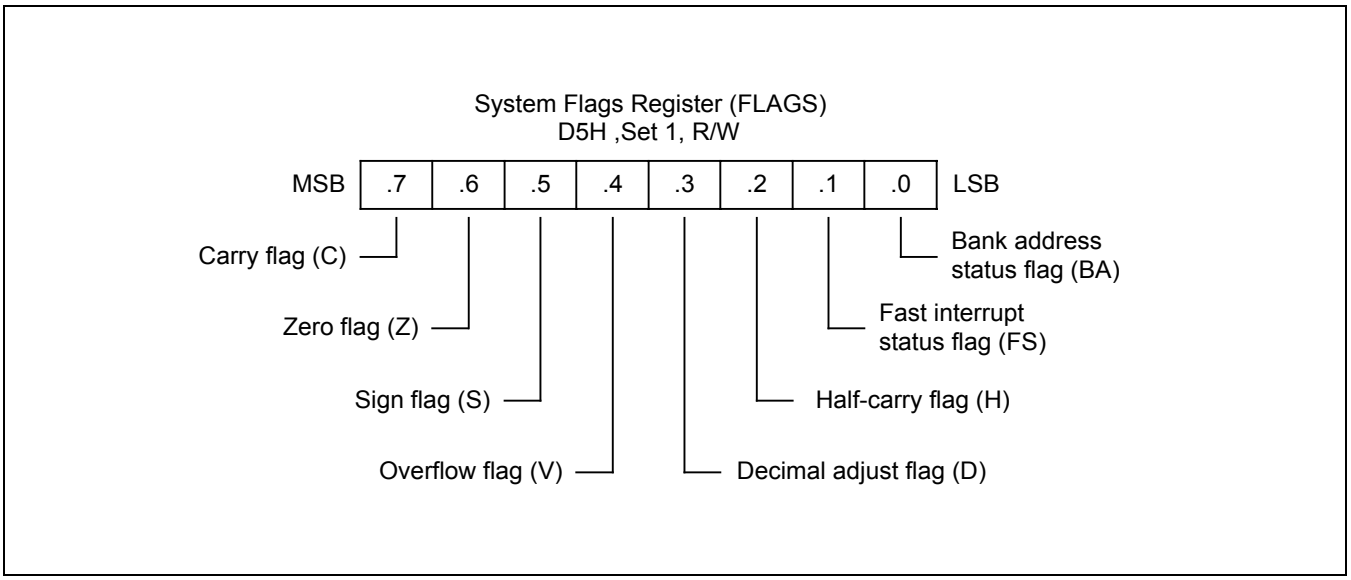


Figure 6-1. System Flags Register (FLAGS)

FLAG DESCRIPTIONS

C Carry Flag (FLAGS.7)

The C flag is set to "1" if the result from an arithmetic operation generates a carry-out from or a borrow to the bit 7 position (MSB). After rotate and shift operations have been performed, it contains the last value shifted out of the specified register. Program instructions can set, clear, or complement the carry flag.

Z Zero Flag (FLAGS.6)

For arithmetic and logic operations, the Z flag is set to "1" if the result of the operation is zero. In operations that test register bits, and in shift and rotate operations, the Z flag is set to "1" if the result is logic zero.

S Sign Flag (FLAGS.5)

Following arithmetic, logic, rotate, or shift operations, the sign bit identifies the state of the MSB of the result. A logic zero indicates a positive number and a logic one indicates a negative number.

V Overflow Flag (FLAGS.4)

The V flag is set to "1" when the result of a two's-complement operation is greater than + 127 or less than – 128. It is cleared to "0" after a logic operation has been performed.

D Decimal Adjust Flag (FLAGS.3)

The DA bit is used to specify what type of instruction was executed last during BCD operations so that a subsequent decimal adjust operation can execute correctly. The DA bit is not usually accessed by programmers, and it cannot be addressed as a test condition.

H Half-Carry Flag (FLAGS.2)

The H bit is set to "1" whenever an addition generates a carry-out of bit 3, or when a subtraction borrows out of bit 4. It is used by the Decimal Adjust (DA) instruction to convert the binary result of a previous addition or subtraction into the correct decimal (BCD) result. The H flag is normally not accessed directly by a program.

FIS Fast Interrupt Status Flag (FLAGS.1)

The FIS bit is set during a fast interrupt cycle and reset during the IRET following interrupt servicing. When set, it inhibits all interrupts and causes the fast interrupt return to be executed when the IRET instruction is executed.

BA Bank Address Flag (FLAGS.0)

The BA flag indicates which register bank in the set 1 area of the internal register file is currently selected, bank 0 or bank 1. The BA flag is cleared to "0" (select bank 0) when the SB0 instruction is executed and is set to "1" (select bank 1) when the SB1 instruction is executed.

INSTRUCTION SET NOTATION

Table 6-2. Flag Notation Conventions

Flag	Description
C	Carry flag
Z	Zero flag
S	Sign flag
V	Overflow flag
D	Decimal-adjust flag
H	Half-carry flag
0	Cleared to logic zero
1	Set to logic one
*	Set or cleared according to operation
–	Value is unaffected
x	Value is undefined

Table 6-3. Instruction Set Symbols

Symbol	Description
dst	Destination operand
src	Source operand
@	Indirect register address prefix
PC	Program counter
IP	Instruction pointer
FLAGS	Flags register (D5H)
RP	Register pointer
#	Immediate operand or register address prefix
H	Hexadecimal number suffix
D	Decimal number suffix
B	Binary number suffix
opc	Opcode

Table 6-4. Instruction Notation Conventions

Notation	Description	Actual Operand Range
cc	Condition code	See list of condition codes in Table 6-6.
r	Working register only	Rn (n = 0–15)
rb	Bit (b) of working register	Rn.b (n = 0–15, b = 0–7)
r0	Bit 0 (LSB) of working register	Rn (n = 0–15)
rr	Working register pair	RRp (p = 0, 2, 4, ..., 14)
R	Register or working register	reg or Rn (reg = 0–255, n = 0–15)
Rb	Bit "b" of register or working register	reg.b (reg = 0–255, b = 0–7)
RR	Register pair or working register pair	reg or RRp (reg = 0–254, even number only, where p = 0, 2, ..., 14)
IA	Indirect addressing mode	addr (addr = 0–254, even number only)
Ir	Indirect working register only	@Rn (n = 0–15)
IR	Indirect register or indirect working register	@Rn or @reg (reg = 0–255, n = 0–15)
Irr	Indirect working register pair only	@RRp (p = 0, 2, ..., 14)
IRR	Indirect register pair or indirect working register pair	@RRp or @reg (reg = 0–254, even only, where p = 0, 2, ..., 14)
X	Indexed addressing mode	#reg[Rn] (reg = 0–255, n = 0–15)
XS	Indexed (short offset) addressing mode	#addr[RRp] (addr = range –128 to +127, where p = 0, 2, ..., 14)
XL	Indexed (long offset) addressing mode	#addr [RRp] (addr = range 0–65535, where p = 2, ..., 14)
DA	Direct addressing mode	addr (addr = range 0–65535)
RA	Relative addressing mode	addr (addr = a number from +127 to –128 that is an offset relative to the address of the next instruction)
IM	Immediate addressing mode	#data (data = 0–255)
IML	Immediate (long) addressing mode	#data (data = 0–65535)

Table 6-5. OPCODE Quick Reference

OPCODE MAP									
		LOWER NIBBLE (HEX)							
	–	0	1	2	3	4	5	6	7
U	0	DEC R1	DEC IR1	ADD r1,r2	ADD r1,Ir2	ADD R2,R1	ADD IR2,R1	ADD R1,IM	BOR r0–Rb
P	1	RLC R1	RLC IR1	ADC r1,r2	ADC r1,Ir2	ADC R2,R1	ADC IR2,R1	ADC R1,IM	BCP r1.b, R2
P	2	INC R1	INC IR1	SUB r1,r2	SUB r1,Ir2	SUB R2,R1	SUB IR2,R1	SUB R1,IM	BXOR r0–Rb
E	3	JP IRR1	SRP/0/1 IM	SBC r1,r2	SBC r1,Ir2	SBC R2,R1	SBC IR2,R1	SBC R1,IM	BTJR r2.b, RA
R	4	DA R1	DA IR1	OR r1,r2	OR r1,Ir2	OR R2,R1	OR IR2,R1	OR R1,IM	LDB r0–Rb
	5	POP R1	POP IR1	AND r1,r2	AND r1,Ir2	AND R2,R1	AND IR2,R1	AND R1,IM	BITC r1.b
N	6	COM R1	COM IR1	TCM r1,r2	TCM r1,Ir2	TCM R2,R1	TCM IR2,R1	TCM R1,IM	BAND r0–Rb
I	7	PUSH R2	PUSH IR2	TM r1,r2	TM r1,Ir2	TM R2,R1	TM IR2,R1	TM R1,IM	BIT r1.b
B	8	DECW RR1	DECW IR1	PUSHUD IR1,R2	PUSHUI IR1,R2	MULT R2,RR1	MULT IR2,RR1	MULT IM,RR1	LD r1, x, r2
B	9	RL R1	RL IR1	POPUD IR2,R1	POPUI IR2,R1	DIV R2,RR1	DIV IR2,RR1	DIV IM,RR1	LD r2, x, r1
L	A	INCW RR1	INCW IR1	CP r1,r2	CP r1,Ir2	CP R2,R1	CP IR2,R1	CP R1,IM	LDC r1, Irr2, xL
E	B	CLR R1	CLR IR1	XOR r1,r2	XOR r1,Ir2	XOR R2,R1	XOR IR2,R1	XOR R1,IM	LDC r2, Irr2, xL
	C	RRC R1	RRC IR1	CPIJE Ir,r2,RA	LDC r1,Irr2	LDW RR2,RR1	LDW IR2,RR1	LDW RR1,IML	LD r1, Ir2
H	D	SRA R1	SRA IR1	CPIJNE Irr,r2,RA	LDC r2,Irr1	CALL IA1		LD IR1,IM	LD Ir1, r2
E	E	RR R1	RR IR1	LDCD r1,Irr2	LDCI r1,Irr2	LD R2,R1	LD R2,IR1	LD R1,IM	LDC r1, Irr2, xs
X	F	SWAP R1	SWAP IR1	LDCPD r2,Irr1	LDCPI r2,Irr1	CALL IRR1	LD IR2,R1	CALL DA1	LDC r2, Irr1, xs

Table 6-5. OPCODE Quick Reference (Continued)

OPCODE MAP									
LOWER NIBBLE (HEX)									
	–	8	9	A	B	C	D	E	F
U	0	LD r1,R2	LD r2,R1	DJNZ r1,RA	JR cc,RA	LD r1,IM	JP cc,DA	INC r1	NEXT
P	1	↓	↓	↓	↓	↓	↓	↓	ENTER
P	2								EXIT
E	3								WFI
R	4								SB0
	5								SB1
N	6								IDLE
I	7	↓	↓	↓	↓	↓	↓	↓	STOP
B	8								DI
B	9								EI
L	A								RET
E	B								IRET
	C								RCF
H	D	↓	↓	↓	↓	↓	↓	↓	SCF
E	E								CCF
X	F	LD r1,R2	LD r2,R1	DJNZ r1,RA	JR cc,RA	LD r1,IM	JP cc,DA	INC r1	NOP

CONDITION CODES

The opcode of a conditional jump always contains a 4-bit field called the condition code (cc). This specifies under which conditions it is to execute the jump. For example, a conditional jump with the condition code for "equal" after a compare operation only jumps if the two operands are equal. Condition codes are listed in Table 6-6.

The carry (C), zero (Z), sign (S), and overflow (V) flags are used to control the operation of conditional jump instructions.

Table 6-6. Condition Codes

Binary	Mnemonic	Description	Flags Set
0000	F	Always false	–
1000	T	Always true	–
0111 (1)	C	Carry	C = 1
1111 (1)	NC	No carry	C = 0
0110 (1)	Z	Zero	Z = 1
1110 (1)	NZ	Not zero	Z = 0
1101	PL	Plus	S = 0
0101	MI	Minus	S = 1
0100	OV	Overflow	V = 1
1100	NOV	No overflow	V = 0
0110 (1)	EQ	Equal	Z = 1
1110 (1)	NE	Not equal	Z = 0
1001	GE	Greater than or equal	(S XOR V) = 0
0001	LT	Less than	(S XOR V) = 1
1010	GT	Greater than	(Z OR (S XOR V)) = 0
0010	LE	Less than or equal	(Z OR (S XOR V)) = 1
1111 (1)	UGE	Unsigned greater than or equal	C = 0
0111 (1)	ULT	Unsigned less than	C = 1
1011	UGT	Unsigned greater than	(C = 0 AND Z = 0) = 1
0011	ULE	Unsigned less than or equal	(C OR Z) = 1

NOTES:

1. It indicate condition codes which are related to two different mnemonics but which test the same flag. For example, Z and EQ are both true if the zero flag (Z) is set, but after an ADD instruction, Z would probably be used. Following a CP instruction, you would probably want to use the instruction EQ.
2. For operations using unsigned numbers, the special condition codes UGE, ULT, UGT, and ULE must be used.

INSTRUCTION DESCRIPTIONS

This Chapter contains detailed information and programming examples for each instruction in the S3C8-series instruction set. Information is arranged in a consistent format for improved readability and for quick reference. The following information is included in each instruction description:

— Instruction name (mnemonic)
— Full instruction name
— Source/destination format of the instruction operand
— Shorthand notation of the instruction's operation
— Textual description of the instruction's effect
— Flag settings that may be affected by the instruction
— Detailed description of the instruction's format, execution time, and addressing mode(s)
— Programming example(s) explaining how to use the instruction

ADC — Add with Carry

ADC dst,src

Operation: dst ← dst + src + c

The source operand, along with the carry flag setting, is added to the destination operand and the sum is stored in the destination. The contents of the source are unaffected. Two's-complement addition is performed. In multiple-precision arithmetic, this instruction lets the carry value from the addition of low-order operands be carried into the addition of high-order operands.

Flags:

C: Set if there is a carry from the most significant bit of the result; cleared otherwise.

Z: Set if the result is "0"; cleared otherwise.

S: Set if the result is negative; cleared otherwise.

V: Set if arithmetic overflow occurs, that is, if both operands are of the same sign and the result is of the opposite sign; cleared otherwise.

D: Always cleared to "0".

H: Set if there is a carry from the most significant bit of the low-order four bits of the result; cleared otherwise.

Format:

			Bytes	Cycles	Opcode (Hex)	Addr Mode dst	Addr Mode src
opc	dst | src		2	4	12	r	r
				6	13	r	lr
opc	src	dst	3	6	14	R	R
					15	R	IR
opc	dst	src	3	6	16	R	IM

Examples: Given: R1 = 10H, R2 = 03H, C flag = "1", register 01H = 20H, register 02H = 03H, and register 03H = 0AH:

```
ADC     R1,R2       →     R1 = 14H, R2 = 03H
ADC     R1,@R2      →     R1 = 1BH, R2 = 03H
ADC     01H,02H     →     Register 01H = 24H, register 02H = 03H
ADC     01H,@02H    →     Register 01H = 2BH, register 02H = 03H
ADC     01H,#11H    →     Register 01H = 32H
```

In the first example, the destination register R1 contains the value 10H, the carry flag is set to "1" and the source working register R2 contains the value 03H. The statement "ADC R1,R2" adds 03H and the carry flag value ("1") to the destination value 10H, leaving 14H in the register R1.

ADD — Add

ADD dst,src

Operation: dst ← dst + src

The source operand is added to the destination operand and the sum is stored in the destination. The contents of the source are unaffected. Two's-complement addition is performed.

Flags:

C: Set if there is a carry from the most significant bit of the result; cleared otherwise.

Z: Set if the result is "0"; cleared otherwise.

S: Set if the result is negative; cleared otherwise.

V: Set if arithmetic overflow occurred, that is, if both operands are of the same sign and the result is of the opposite sign; cleared otherwise.

D: Always cleared to "0".

H: Set if a carry from the low-order nibble occurred.

Format:

	Bytes	Cycles	Opcode (Hex)	Addr Mode dst	src
opc | dst | src	2	4	02	r	r
		6	03	r	lr
opc | src | dst	3	6	04	R	R
			05	R	IR
opc | dst | src	3	6	06	R	IM

Examples: Given: R1 = 12H, R2 = 03H, register 01H = 21H, register 02H = 03H, register 03H = 0AH:

ADD	R1,R2	→	R1 = 15H, R2 = 03H
ADD	R1,@R2	→	R1 = 1CH, R2 = 03H
ADD	01H,02H	→	Register 01H = 24H, register 02H = 03H
ADD	01H,@02H	→	Register 01H = 2BH, register 02H = 03H
ADD	01H,#25H	→	Register 01H = 46H

In the first example, the destination working register R1 contains 12H and the source working register R2 contains 03H. The statement "ADD R1,R2" adds 03H to 12H, leaving the value 15H in the register R1.

AND — Logical AND

AND dst,src

Operation: dst ← dst AND src

The source operand is logically ANDed with the destination operand. The result is stored in the destination. The AND operation causes a "1" bit to be stored whenever the corresponding bits in the two operands are both logic ones; otherwise a "0" bit value is stored. The contents of the source are unaffected.

Flags:

C: Unaffected.

Z: Set if the result is "0"; cleared otherwise.

S: Set if the result bit 7 is set; cleared otherwise.

V: Always cleared to "0".

D: Unaffected.

H: Unaffected.

Format:

			Bytes	Cycles	Opcode (Hex)	Addr Mode dst	Addr Mode src
opc	dst | src		2	4	52	r	r
				6	53	r	lr
opc	src	dst	3	6	54	R	R
					55	R	IR
opc	dst	src	3	6	56	R	IM

Examples: Given: R1 = 12H, R2 = 03H, register 01H = 21H, register 02H = 03H, register 03H = 0AH:

AND	R1,R2	→	R1 = 02H, R2 = 03H
AND	R1,@R2	→	R1 = 02H, R2 = 03H
AND	01H,02H	→	Register 01H = 01H, register 02H = 03H
AND	01H,@02H	→	Register 01H = 00H, register 02H = 03H
AND	01H,#25H	→	Register 01H = 21H

In the first example, the destination working register R1 contains the value 12H and the source working register R2 contains 03H. The statement "AND R1,R2" logically ANDs the source operand 03H with the destination operand value 12H, leaving the value 02H in the register R1.

BAND — Bit AND

BAND dst,src.b

BAND dst.b,src

Operation: dst(0) ← dst(0) AND src(b)

or

dst(b) ← dst(b) AND src(0)

The specified bit of the source (or the destination) is logically ANDed with the zero bit (LSB) of the destination (or the source). The resultant bit is stored in the specified bit of the destination. No other bits of the destination are affected. The source is unaffected.

Flags:

C: Unaffected.

Z: Set if the result is "0"; cleared otherwise.

S: Cleared to "0".

V: Undefined.

D: Unaffected.

H: Unaffected.

Format:

			Bytes	Cycles	Opcode (Hex)	Addr Mode dst	src
opc	dst | b | 0	src	3	6	67	r0	Rb
opc	src | b | 1	dst	3	6	67	Rb	r0

NOTE: In the second byte of the 3-byte instruction formats, the destination (or the source) address is four bits, the bit address "b" is three bits, and the LSB address value is one bit in length.

Examples: Given: R1 = 07H and register 01H = 05H:

```
BAND    R1,01H.1    →    R1 = 06H, register 01H = 05H
BAND    01H.1,R1    →    Register 01H = 05H, R1 = 07H
```

In the first example, the source register 01H contains the value 05H (00000101B) and the destination working register R1 contains 07H (00000111B). The statement "BAND R1,01H.1" ANDs the bit 1 value of the source register ("0") with the bit 0 value of the register R1 (destination), leaving the value 06H (00000110B) in the register R1.

BCP — Bit Compare

BCP dst,src.b

Operation: dst(0) – src(b)

The specified bit of the source is compared to (subtracted from) bit zero (LSB) of the destination. The zero flag is set if the bits are the same; otherwise it is cleared. The contents of both operands are unaffected by the comparison.

Flags:

C: Unaffected.

Z: Set if the two bits are the same; cleared otherwise.

S: Cleared to "0".

V: Undefined.

D: Unaffected.

H: Unaffected.

Format:

			Bytes	Cycles	Opcode (Hex)	Addr Mode dst	src
opc	dst | b | 0	src	3	6	17	r0	Rb

NOTE: In the second byte of the instruction format, the destination address is four bits, the bit address "0" is three bits, and the LSB address value is one bit in length.

Example: Given: R1 = 07H and register 01H = 01H:

BCP R1,01H.1 → R1 = 07H, register 01H = 01H

If the destination working register R1 contains the value 07H (00000111B) and the source register 01H contains the value 01H (00000001B), the statement "BCP R1,01H.1" compares bit one of the source register (01H) and bit zero of the destination register (R1). Because the bit values are not identical, the zero flag bit (Z) is cleared in the FLAGS register (0D5H).

BITC — Bit Complement

BITC dst.b

Operation: dst(b) ← NOT dst(b)

This instruction complements the specified bit within the destination without affecting any other bit in the destination.

Flags:
C: Unaffected.
Z: Set if the result is "0"; cleared otherwise.
S: Cleared to "0".
V: Undefined.
D: Unaffected.
H: Unaffected.

Format:

	Bytes	Cycles	Opcode (Hex)	Addr Mode dst
opc | dst | b | 0	2	4	57	rb

NOTE: In the second byte of the instruction format, the destination address is four bits, the bit address “b” is three bits, and the LSB address value is one bit in length.

Example: Given: R1 = 07H

BITC R1.1 → R1 = 05H

If the working register R1 contains the value 07H (00000111B), the statement "BITC R1.1" complements bit one of the destination and leaves the value 05H (00000101B) in the register R1. Because the result of the complement is not "0", the zero flag (Z) in the FLAGS register (0D5H) is cleared.

BITR — Bit Reset

BITR dst.b

Operation: dst(b) ← 0

The BITR instruction clears the specified bit within the destination without affecting any other bit in the destination.

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)	Addr Mode dst
opc | dst | b | 0	2	4	77	rb

NOTE: In the second byte of the instruction format, the destination address is four bits, the bit address “0” is three bits, and the LSB address value is one bit in length.

Example: Given: R1 = 07H:

BITR R1.1 → R1 = 05H

If the value of the working register R1 is 07H (00000111B), the statement "BITR R1.1" clears bit one of the destination register R1, leaving the value 05H (00000101B).

BITS — Bit Set

BITS dst.b

Operation: dst(b) ← 1

The BITS instruction sets the specified bit within the destination without affecting any other bit in the destination.

Flags: No flags are affected.

Format:

		Bytes	Cycles	Opcode (Hex)	Addr Mode dst
opc	dst | b | 1	2	4	77	rb

NOTE: In the second byte of the instruction format, the destination address is four bits, the bit address “b” is three bits, and the LSB address value is one bit in length.

Example: Given: R1 = 07H:

BITS R1.3 → R1 = 0FH

If the working register R1 contains the value 07H (00000111B), the statement "BITS R1.3" sets bit three of the destination register R1 to "1", leaving the value 0FH (00001111B).

BOR — Bit OR

BOR dst,src.b

BOR dst.b,src

Operation: dst(0) ← dst(0) OR src(b)

or

dst(b) ← dst(b) OR src(0)

The specified bit of the source (or the destination) is logically ORed with bit zero (LSB) of the destination (or the source). The resulting bit value is stored in the specified bit of the destination. No other bits of the destination are affected. The source is unaffected.

Flags: **C:** Unaffected.

Z: Set if the result is "0"; cleared otherwise.

S: Cleared to "0".

V: Undefined.

D: Unaffected.

H: Unaffected.

Format:

			Bytes	Cycles	Opcode (Hex)	Addr Mode dst	Addr Mode src
opc	dst | b | 0	src	3	6	07	r0	Rb
opc	src | b | 1	dst	3	6	07	Rb	r0

NOTE: In the second byte of the 3-byte instruction format, the destination (or the source) address is four bits, the bit address “b” is three bits, and the LSB address value is one bit.

Examples: Given: R1 = 07H and register 01H = 03H:

```
BOR     R1, 01H.1     →     R1 = 07H, register 01H = 03H
BOR     01H.2, R1     →     Register 01H = 07H, R1 = 07H
```

In the first example, the destination working register R1 contains the value 07H (00000111B) and the source register 01H the value 03H (00000011B). The statement "BOR R1,01H.1" logically ORs bit one of the register 01H (source) with bit zero of R1 (destination). This leaves the same value (07H) in the working register R1.

In the second example, the destination register 01H contains the value 03H (00000011B) and the source working register R1 the value 07H (00000111B). The statement "BOR 01H.2,R1" logically ORs bit two of the register 01H (destination) with bit zero of R1 (source). This leaves the value 07H in the register 01H.

BTJRF — Bit Test, Jump Relative on False

BTJRF dst,src.b

Operation: If src(b) is a "0", then PC ← PC + dst

The specified bit within the source operand is tested. If it is a "0", the relative address is added to the program counter and control passes to the statement whose address is currently in the program counter. Otherwise, the instruction following the BTJRF instruction is executed.

Flags: No flags are affected.

Format:

(note)			Bytes	Cycles	Opcode (Hex)	Addr Mode dst	src
opc	src | b | 0	dst	3	10	37	RA	rb

NOTE: In the second byte of the instruction format, the source address is four bits, the bit address "b" is three bits, and the LSB address value is one bit in length.

Example: Given: R1 = 07H:

BTJRF SKIP,R1.3 → PC jumps to SKIP location

If the working register R1 contains the value 07H (00000111B), the statement "BTJRF SKIP,R1.3" tests bit 3. Because it is "0", the relative address is added to the PC and the PC jumps to the memory location pointed to by the SKIP (Remember that the memory location must be within the allowed range of + 127 to – 128).

BTJRT — Bit Test, Jump Relative on True

BTJRT dst,src.b

Operation: If src(b) is a "1", then PC ← PC + dst

The specified bit within the source operand is tested. If it is a "1", the relative address is added to the program counter and control passes to the statement whose address is now in the PC. Otherwise, the instruction following the BTJRT instruction is executed.

Flags: No flags are affected.

Format:

(note)			Bytes	Cycles	Opcode (Hex)	Addr Mode dst	Addr Mode src
opc	src | b | 1	dst	3	10	37	RA	rb

NOTE: In the second byte of the instruction format, the source address is four bits, the bit address "b" is three bits, and the LSB address value is one bit in length.

Example: Given: R1 = 07H:

```
BTJRT    SKIP,R1.1
```

If the working register R1 contains the value 07H (00000111B), the statement "BTJRT SKIP,R1.1" tests bit one in the source register (R1). Because it is a "1", the relative address is added to the PC and the PC jumps to the memory location pointed to by the SKIP.

Remember that the memory location addressed by the BTJRT instruction must be within the allowed range of + 127 to – 128.

BXOR — Bit XOR

BXOR dst,src.b

BXOR dst.b,src

Operation: dst(0) ← dst(0) XOR src(b)
or
dst(b) ← dst(b) XOR src(0)

The specified bit of the source (or the destination) is logically exclusive-ORed with bit zero (LSB) of the destination (or the source). The result bit is stored in the specified bit of the destination. No other bits of the destination are affected. The source is unaffected.

Flags:
C: Unaffected.
Z: Set if the result is "0"; cleared otherwise.
S: Cleared to "0".
V: Undefined.
D: Unaffected.
H: Unaffected.

Format:

			Bytes	Cycles	Opcode (Hex)	Addr Mode dst	Addr Mode src
opc	dst | b | 0	src	3	6	27	r0	Rb
opc	src | b | 1	dst	3	6	27	Rb	r0

NOTE: In the second byte of the 3-byte instruction format, the destination (or the source) address is four bits, the bit address "b" is three bits, and the LSB address value is one bit in length.

Examples: Given: R1 = 07H (00000111B) and register 01H = 03H (00000011B):

```
BXOR    R1,01H.1    →    R1 = 06H, register 01H = 03H
BXOR    01H.2,R1    →    Register 01H = 07H, R1 = 07H
```

In the first example, the destination working register R1 has the value 07H (00000111B) and the source register 01H has the value 03H (00000011B). The statement "BXOR R1,01H.1" exclusive-ORs bit one of the register 01H (the source) with bit zero of R1 (the destination). The result bit value is stored in bit zero of R1, changing its value from 07H to 06H. The value of the source register 01H is unaffected.

CALL — Call Procedure

CALL dst

Operation: SP ← SP–1
@SP ← PCL
SP ← SP–1
@SP ← PCH
PC ← dst

The contents of the program counter are pushed onto the top of the stack. The program counter value used is the address of the first instruction following the CALL instruction. The specified destination address is then loaded into the program counter and points to the first instruction of a procedure. At the end of the procedure the return instruction (RET) can be used to return to the original program flow. RET pops the top of the stack back into the program counter.

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)	Addr Mode dst
opc | dst	3	14	F6	DA
opc | dst	2	12	F4	IRR
opc | dst	2	14	D4	IA

Examples: Given: R0 = 35H, R1 = 21H, PC = 1A47H, and SP = 0002H:

CALL	3521H	→	SP = 0000H (Memory locations 0000H = 1AH, 0001H = 4AH, where, 4AH is the address that follows the instruction.)
CALL	@RR0	→	SP = 0000H (0000H = 1AH, 0001H = 49H)
CALL	#40H	→	SP = 0000H (0000H = 1AH, 0001H = 49H)

In the first example, if the program counter value is 1A47H and the stack pointer contains the value 0002H, the statement "CALL 3521H" pushes the current PC value onto the top of the stack. The stack pointer now points to the memory location 0000H. The PC is then loaded with the value 3521H, the address of the first instruction in the program sequence to be executed.

If the contents of the program counter and the stack pointer are the same as in the first example, the statement "CALL @RR0" produces the same result except that the 49H is stored in stack location 0001H (because the two-byte instruction format was used). The PC is then loaded with the value 3521H, the address of the first instruction in the program sequence to be executed. Assuming that the contents of the program counter and the stack pointer are the same as in the first example, if the program address 0040H contains 35H and the program address 0041H contains 21H, the statement "CALL #40H" produces the same result as in the second example.

CCF — Complement Carry Flag

CCF

Operation: C ← NOT C

The carry flag (C) is complemented. If C = "1", the value of the carry flag is changed to logic zero. If C = "0", the value of the carry flag is changed to logic one.

Flags: **C:** Complemented.

No other flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)
opc	1	4	EF

Example: Given: The carry flag = "0":

CCF

If the carry flag = "0", the CCF instruction complements it in the FLAGS register (0D5H), changing its value from logic zero to logic one.

CLR — Clear

CLR dst

Operation: dst ← "0"

The destination location is cleared to "0".

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)	Addr Mode dst
opc dst	2	4	B0	R
		4	B1	IR

Examples: Given: Register 00H = 4FH, register 01H = 02H, and register 02H = 5EH:

```
CLR   00H    →   Register 00H = 00H
CLR   @01H   →   Register 01H = 02H, register 02H = 00H
```

In Register (R) addressing mode, the statement "CLR 00H" clears the destination register 00H value to 00H.

In the second example, the statement "CLR @01H" uses Indirect Register (IR) addressing mode to clear the 02H register value to 00H.

COM — Complement

COM dst

Operation: dst ← NOT dst

The contents of the destination location are complemented (one's complement). All "1s" are changed to "0s", and vice-versa.

Flags:

C: Unaffected.

Z: Set if the result is "0"; cleared otherwise.

S: Set if the result bit 7 is set; cleared otherwise.

V: Always reset to "0".

D: Unaffected.

H: Unaffected.

Format:

		Bytes	Cycles	Opcode (Hex)	Addr Mode dst
opc	dst	2	4	60	R
			4	61	IR

Examples: Given: R1 = 07H and register 07H = 0F1H:

```
COM     R1      →     R1 = 0F8H
COM     @R1     →     R1 = 07H, register 07H = 0EH
```

In the first example, the destination working register R1 contains the value 07H (00000111B). The statement "COM R1" complements all the bits in R1: all logic ones are changed to logic zeros, and logic zeros to logic ones, leaving the value 0F8H (11111000B).

In the second example, Indirect Register (IR) addressing mode is used to complement the value of the destination register 07H (11110001B), leaving the new value 0EH (00001110B).

CP — Compare

CP dst,src

Operation: dst–src

The source operand is compared to (subtracted from) the destination operand, and the appropriate flags are set accordingly. The contents of both operands are unaffected by the comparison.

Flags:

C: Set if a "borrow" occurred (src > dst); cleared otherwise.

Z: Set if the result is "0"; cleared otherwise.

S: Set if the result is negative; cleared otherwise.

V: Set if arithmetic overflow occurred; cleared otherwise.

D: Unaffected.

H: Unaffected.

Format:

			Bytes	Cycles	Opcode (Hex)	Addr Mode dst	Addr Mode src
opc	dst | src		2	4	A2	r	r
				6	A3	r	lr
opc	src	dst	3	6	A4	R	R
				6	A5	R	IR
opc	dst	src	3	6	A6	R	IM

Examples: 1. Given: R1 = 02H and R2 = 03H:

```
CP      R1,R2       →       Set the C and S flags
```

The destination working register R1 contains the value 02H and the source register R2 contains the value 03H. The statement "CP R1,R2" subtracts the R2 value (source/subtrahend) from the R1 value (destination/minuend). Because a "borrow" occurs and the difference is negative, the C and the S flag values are "1".

2. Given: R1 = 05H and R2 = 0AH:

```
CP      R1,R2
JP      UGE,SKIP
INC     R1
SKIP    LD  R3,R1
```

In this example, the destination working register R1 contains the value 05H which is less than the contents of the source working register R2 (0AH). The statement "CP R1,R2" generates C = "1" and the JP instruction does not jump to the SKIP location. After the statement "LD R3,R1" executes, the value 06H remains in the working register R3.

CPIJE — Compare, Increment, and Jump on Equal

CPIJE dst,src,RA

Operation: If dst–src = "0", PC ← PC + RA
Ir ← Ir + 1

The source operand is compared to (subtracted from) the destination operand. If the result is "0", the relative address is added to the program counter and control passes to the statement whose address is now in the program counter. Otherwise, the instruction immediately following the CPIJE instruction is executed. In either case, the source pointer is incremented by one before the next instruction is executed.

Flags: No flags are affected.

Format:

				Bytes	Cycles	Opcode (Hex)	Addr Mode dst	Addr Mode src
opc	src	dst	RA	3	12	C2	r	Ir

Example: Given: R1 = 02H, R2 = 03H, and register 03H = 02H:

CPIJE R1,@R2,SKIP → R2 = 04H, PC jumps to SKIP location

In this example, the working register R1 contains the value 02H, the working register R2 the value 03H, and the register 03 contains 02H. The statement "CPIJE R1,@R2,SKIP" compares the @R2 value 02H (00000010B) to 02H (00000010B). Because the result of the comparison is *equal*, the relative address is added to the PC and the PC then jumps to the memory location pointed to by SKIP. The source register (R2) is incremented by one, leaving a value of 04H.

Remember that the memory location addressed by the CPIJE instruction must be within the allowed range of + 127 to – 128.

CPIJNE — Compare, Increment, and Jump on Non-Equal

CPIJNE dst,src,RA

Operation: If dst–src ≠ "0", PC ← PC + RA
Ir ← Ir + 1

The source operand is compared to (subtracted from) the destination operand. If the result is not "0", the relative address is added to the program counter and control passes to the statement whose address is now in the program counter. Otherwise the instruction following the CPIJNE instruction is executed. In either case the source pointer is incremented by one before the next instruction.

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)	Addr Mode dst	Addr Mode src
opc | src | dst | RA	3	12	D2	r	Ir

Example: Given: R1 = 02H, R2 = 03H, and register 03H = 04H:

CPIJNE R1,@R2,SKIP → R2 = 04H, PC jumps to SKIP location

The working register R1 contains the value 02H, the working register R2 (the source pointer) the value 03H, and the general register 03 the value 04H. The statement "CPIJNE R1,@R2,SKIP" subtracts 04H (00000100B) from 02H (00000010B). Because the result of the comparison is *non-equal*, the relative address is added to the PC and the PC then jumps to the memory location pointed to by SKIP. The source pointer register (R2) is also incremented by one, leaving a value of 04H.

Remember that the memory location addressed by the CPIJNE instruction must be within the allowed range of + 127 to – 128.

DA — Decimal Adjust

DA dst

Operation: dst ← DA dst

The destination operand is adjusted to form two 4-bit BCD digits following an addition or subtraction operation. For addition (ADD, ADC) or subtraction (SUB, SBC), the following table indicates the operation performed (The operation is undefined if the destination operand is not the result of a valid addition or subtraction of BCD digits):

Instruction	Carry Before DA	Bits 4–7 Value (Hex)	H Flag Before DA	Bits 0–3 Value (Hex)	Number Added to Byte	Carry After DA
	0	0–9	0	0–9	00	0
	0	0–8	0	A–F	06	0
	0	0–9	1	0–3	06	0
ADD	0	A–F	0	0–9	60	1
ADC	0	9–F	0	A–F	66	1
	0	A–F	1	0–3	66	1
	1	0–2	0	0–9	60	1
	1	0–2	0	A–F	66	1
	1	0–3	1	0–3	66	1
	0	0–9	0	0–9	00 = – 00	0
SUB	0	0–8	1	6–F	FA = – 06	0
SBC	1	7–F	0	0–9	A0 = – 60	1
	1	6–F	1	6–F	9A = – 66	1

Flags:
C: Set if there was a carry from the most significant bit; cleared otherwise (see table).
Z: Set if result is "0"; cleared otherwise.
S: Set if result bit 7 is set; cleared otherwise.
V: Undefined.
D: Unaffected.
H: Unaffected.

Format:

opc	dst

Bytes	Cycles	Opcode (Hex)	Addr Mode dst
2	4	40	R
	4	41	IR

DA — Decimal Adjust

DA (Continued)

Example: Given: The working register R0 contains the value 15 (BCD), the working register R1 contains 27 (BCD), and the address 27H contains 46 (BCD):

```
ADD     R1,R0       ;       C ← "0", H ← "0", Bits 4–7 = 3, bits 0–3 = C, R1 ← 3CH
DA      R1          ;       R1 ← 3CH + 06
```

If an addition is performed using the BCD values 15 and 27, the result should be 42. The sum is incorrect, however, when the binary representations are added in the destination location using the standard binary arithmetic:

```
  0 0 0 1   0 1 0 1       15
+ 0 0 1 0   0 1 1 1       27
  0 0 1 1   1 1 0 0   =3CH
```

The DA instruction adjusts this result so that the correct BCD representation is obtained:

```
  0 0 1 1   1 1 0 0
+ 0 0 0 0   0 1 1 0
  0 1 0 0   0 0 1 0   =42
```

Assuming the same values given above, the statements

```
SUB     27H,R0      ;       C ← "0", H ← "0", Bits 4–7 = 3, bits 0–3 = 1
DA      @R1         ;       @R1 ← 31–0
```

leave the value 31 (BCD) in the address 27H (@R1).

DEC — Decrement

DEC dst

Operation: dst ← dst–1

The contents of the destination operand are decremented by one.

Flags:
C: Unaffected.
Z: Set if the result is "0"; cleared otherwise.
S: Set if result is negative; cleared otherwise.
V: Set if arithmetic overflow occurred; cleared otherwise.
D: Unaffected.
H: Unaffected.

Format:

		Bytes	Cycles	Opcode (Hex)	Addr Mode dst
opc	dst	2	4	00	R
			4	01	IR

Examples: Given: R1 = 03H and register 03H = 10H:

```
DEC     R1      →     R1 = 02H
DEC     @R1     →     Register 03H = 0FH
```

In the first example, if the working register R1 contains the value 03H, the statement "DEC R1" decrements the hexadecimal value by one, leaving the value 02H. In the second example, the statement "DEC @R1" decrements the value 10H contained in the destination register 03H by one, leaving the value 0FH.

DECW — Decrement Word

DECW dst

Operation: dst ← dst – 1

The contents of the destination location (which must be an even address) and the operand following that location are treated as a single 16-bit value that is decremented by one.

Flags:

C: Unaffected.

Z: Set if the result is "0"; cleared otherwise.

S: Set if the result is negative; cleared otherwise.

V: Set if arithmetic overflow occurred; cleared otherwise.

D: Unaffected.

H: Unaffected.

Format:

		Bytes	Cycles	Opcode (Hex)	Addr Mode dst
opc	dst	2	8	80	RR
			8	81	IR

Examples: Given: R0 = 12H, R1 = 34H, R2 = 30H, register 30H = 0FH, and register 31H = 21H:

```
DECW    RR0     →    R0 = 12H, R1 = 33H
DECW    @R2     →    Register 30H = 0FH, register 31H = 20H
```

In the first example, the destination register R0 contains the value 12H and the register R1 the value 34H. The statement "DECW RR0" addresses R0 and the following operand R1 as a 16-bit word and decrements the value of R1 by one, leaving the value 33H.

NOTE: A system malfunction may occur if you use a Zero flag (FLAGS.6) result together with a DECW instruction. To avoid this problem, it is recommended to use DECW as shown in the following example.

```
LOOP    DECW    RR0
LD      R2,R1
OR      R2,R0
JR      NZ,LOOP
```

DI — Disable Interrupts

DI

Operation: SYM (0) ← 0

Bit zero of the system mode control register, SYM.0, is cleared to "0", globally disabling all interrupt processing. Interrupt requests will continue to set their respective interrupt pending bits, but the CPU will not service them while interrupt processing is disabled.

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)
opc	1	4	8F

Example: Given: SYM = 01H:

DI

If the value of the SYM register is 01H, the statement "DI" leaves the new value 00H in the register and clears SYM.0 to "0", disabling interrupt processing.

DIV — Divide (Unsigned)

DIV dst,src

Operation: dst ÷ src
dst (UPPER) ← REMAINDER
dst (LOWER) ← QUOTIENT

The destination operand (16 bits) is divided by the source operand (8 bits). The quotient (8 bits) is stored in the lower half of the destination. The remainder (8 bits) is stored in the upper half of the destination. When the quotient is ≥ 2^8, the numbers stored in the upper and lower halves of the destination for quotient and remainder are incorrect. Both operands are treated as unsigned integers.

Flags:
C: Set if the V flag is set and the quotient is between 2^8 and 2^9 –1; cleared otherwise.
Z: Set if the divisor or the quotient = "0"; cleared otherwise.
S: Set if MSB of the quotient = "1"; cleared otherwise.
V: Set if the quotient is ≥ 2^8 or if the divisor = "0"; cleared otherwise.
D: Unaffected.
H: Unaffected.

Format:

opc	src	dst

Bytes	Cycles	Opcode (Hex)	Addr Mode dst	src
3	2^6/10 *	94	RR	R
	2^6/10 *	95	RR	IR
	2^6/10 *	96	RR	IM

* Execution takes 10 cycles if the divide-by-zero is attempted, otherwise, it takes 2^6 cycles.

Examples: Given: R0 = 10H, R1 = 03H, R2 = 40H, register 40H = 80H:

DIV RR0,R2 → R0 = 03H, R1 = 40H
DIV RR0,@R2 → R0 = 03H, R1 = 20H
DIV RR0,#20H → R0 = 03H, R1 = 80H

In the first example, the destination working register pair RR0 contains the values 10H (R0) and 03H (R1), and the register R2 contains the value 40H. The statement "DIV RR0,R2" divides the 16-bit RR0 value by the 8-bit value of the R2 (source) register. After the DIV instruction, R0 contains the value 03H and R1 contains 40H. The 8-bit remainder is stored in the upper half of the destination register RR0 (R0) and the quotient in the lower half (R1).

DJNZ — Decrement and Jump if Non-Zero

DJNZ r,dst

Operation: r ← r – 1
If r ≠ 0, PC ← PC + dst

The working register being used as a counter is decremented. If the contents of the register are not logic zero after decrementing, the relative address is added to the program counter and control passes to the statement whose address is now in the PC. The range of the relative address is + 127 to – 128, and the original value of the PC is taken to be the address of the instruction byte following the DJNZ statement.

NOTE: In case of using DJNZ instruction, the working register being used as a counter should be set at the one of location 0C0H to 0CFH with SRP, SRP0 or SRP1 instruction.

Flags: No flags are affected.

Format:

		Bytes	Cycles	Opcode (Hex)	Addr Mode dst
r | opc	dst	2	8 (jump taken)	rA	RA
			8 (no jump)	r = 0 to F	

Example: Given: R1 = 02H and LOOP is the label of a relative address:

```
SRP     #0C0H
DJNZ    R1,LOOP
```

DJNZ is typically used to control a "loop" of instructions. In many cases, a label is used as the destination operand instead of a numeric relative address value. In the example, the working register R1 contains the value 02H, and LOOP is the label for a relative address.

The statement "DJNZ R1, LOOP" decrements the register R1 by one, leaving the value 01H. Because the contents of R1 after the decrement are non-zero, the jump is taken to the relative address specified by the LOOP label.

EI — Enable Interrupts

EI

Operation: SYM (0) ← 1

The EI instruction sets bit zero of the system mode register, SYM.0 to "1". This allows interrupts to be serviced as they occur (assuming they have the highest priority). If an interrupt's pending bit was set while interrupt processing was disabled (by executing a DI instruction), it will be serviced when the EI instruction is executed.

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)
opc	1	4	9F

Example: Given: SYM = 00H:

EI

If the SYM register contains the value 00H, that is, if interrupts are currently disabled, the statement "EI" sets the SYM register to 01H, enabling all interrupts. (SYM.0 is the enable bit for global interrupt processing.)

ENTER — Enter

ENTER

Operation: SP ← SP – 2
@SP ← IP
IP ← PC
PC ← @IP
IP ← IP + 2

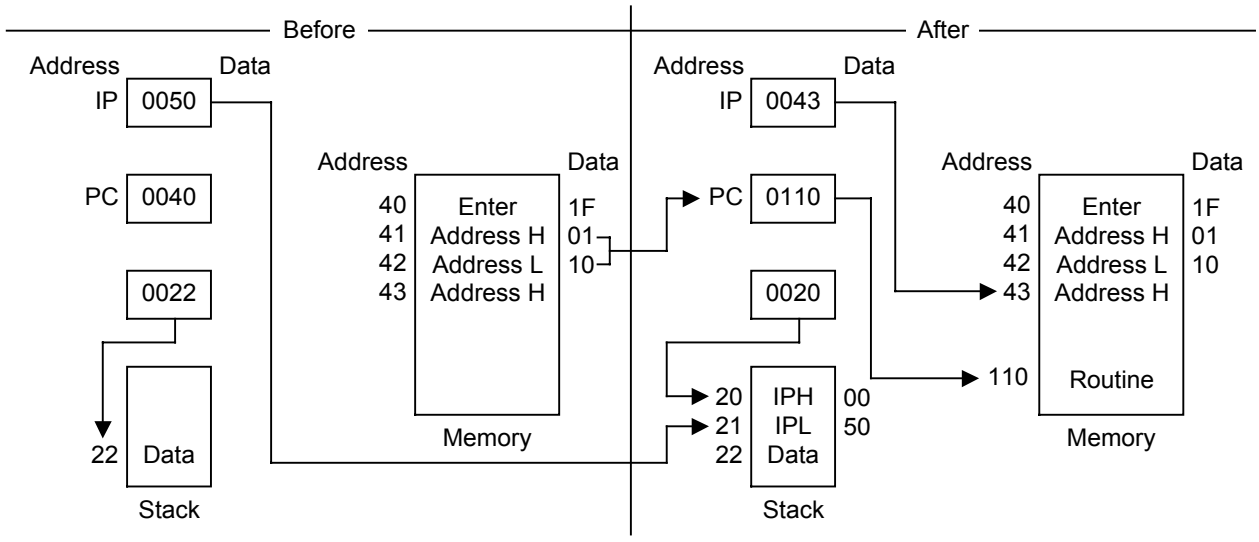
This instruction is useful when implementing threaded-code languages. The contents of the instruction pointer are pushed to the stack. The program counter (PC) value is then written to the instruction pointer. The program memory word that is pointed to by the instruction pointer is loaded into the PC, and the instruction pointer is incremented by two.

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)
opc	1	14	1F

Example: The diagram below shows an example of how to use an ENTER statement.



EXIT — Exit

EXIT

Operation: IP ← @SP
SP ← SP + 2
PC ← @IP
IP ← IP + 2

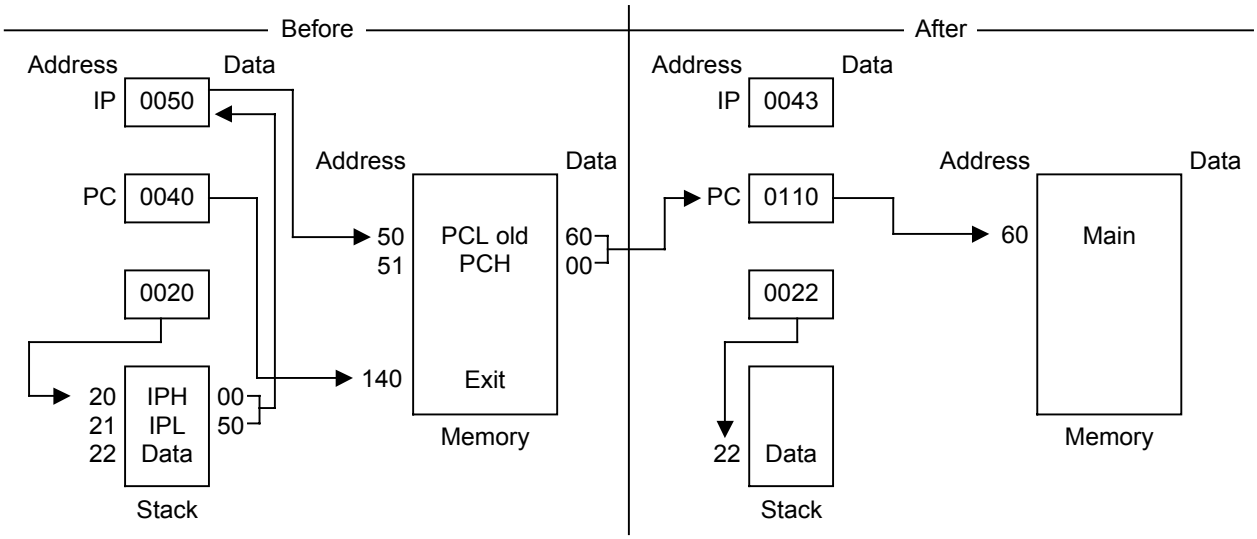
This instruction is useful when implementing threaded-code languages. The stack value is popped and loaded into the instruction pointer. The program memory word that is pointed to by the instruction pointer is then loaded into the program counter, and the instruction pointer is incremented by two.

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)
opc	1	16	2F

Example: The diagram below shows an example of how to use an EXIT statement.



IDLE — Idle Operation

IDLE

Operation: (See description)

The IDLE instruction stops the CPU clock while allowing the system clock oscillation to continue. Idle mode can be released by an interrupt request (IRQ) or an external reset operation.

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)	Addr Mode dst	src
opc	1	4	6F	–	–

Example: The instruction **IDLE** stops the CPU clock but it does not stop the system clock.

INC — Increment

INC dst

Operation: dst ← dst + 1

The contents of the destination operand are incremented by one.

Flags:
C: Unaffected.
Z: Set if the result is "0"; cleared otherwise.
S: Set if the result is negative; cleared otherwise.
V: Set if arithmetic overflow occurred; cleared otherwise.
D: Unaffected.
H: Unaffected.

Format:

	Bytes	Cycles	Opcode (Hex)	Addr Mode dst
dst | opc	1	4	rE	r
			r = 0 to F	
opc dst	2	4	20	R
		4	21	IR

Examples: Given: R0 = 1BH, register 00H = 0CH, and register 1BH = 0FH:

```
INC R0      → R0 = 1CH
INC 00H     → Register 00H = 0DH
INC @R0     → R0 = 1BH, register 01H = 10H
```

In the first example, if the destination working register R0 contains the value 1BH, the statement "INC R0" leaves the value 1CH in that same register.

The second example shows the effect an INC instruction has on the register at the location 00H, assuming that it contains the value 0CH.

In the third example, INC is used in Indirect Register (IR) addressing mode to increment the value of the register 1BH from 0FH to 10H.

INCW — Increment Word

INCW dst

Operation: dst ← dst + 1

The contents of the destination (which must be an even address) and the byte following that location are treated as a single 16-bit value that is incremented by one.

Flags:

C: Unaffected.

Z: Set if the result is "0"; cleared otherwise.

S: Set if the result is negative; cleared otherwise.

V: Set if arithmetic overflow occurred; cleared otherwise.

D: Unaffected.

H: Unaffected.

Format:

		Bytes	Cycles	Opcode (Hex)	Addr Mode dst
opc	dst	2	8	A0	RR
			8	A1	IR

Examples: Given: R0 = 1AH, R1 = 02H, register 02H = 0FH, and register 03H = 0FFH:

```
INCW    RR0     →    R0 = 1AH, R1 = 03H
INCW    @R1     →    Register 02H = 10H, register 03H = 00H
```

In the first example, the working register pair RR0 contains the value 1AH in the register R0 and 02H in the register R1. The statement "INCW RR0" increments the 16-bit destination by one, leaving the value 03H in the register R1. In the second example, the statement "INCW @R1" uses Indirect Register (IR) addressing mode to increment the contents of the general register 03H from 0FFH to 00H and the register 02H from 0FH to 10H.

NOTE: A system malfunction may occur if you use a Zero (Z) flag (FLAGS.6) result together with an INCW instruction. To avoid this problem, it is recommended to use the INCW instruction as shown in the following example:

```
LOOP:   INCW    RR0
        LD      R2,R1
        OR      R2,R0
        JR      NZ,LOOP
```

IRET — Interrupt Return

IRET IRET (Normal) iRET (Fast)

Operation:

IRET (Normal)	iRET (Fast)
FLAGS ← @SP	PC ↔ IP
SP ← SP + 1	FLAGS ← FLAGS'
PC ← @SP	FIS ← 0
SP ← SP + 2	
SYM(0) ← 1	

This instruction is used at the end of an interrupt service routine. It restores the flag register and the program counter. It also re-enables global interrupts. A "normal IRET" is executed only if the fast interrupt status bit (FIS, bit one of the FLAGS register, 0D5H) is cleared (= "0"). If a fast interrupt occurred, IRET clears the FIS bit that was set at the beginning of the service routine.

Flags: All flags are restored to their original settings (that is, the settings before the interrupt occurred).

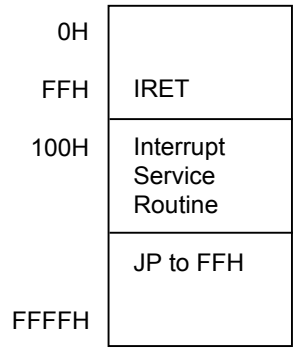
Format:

IRET (Normal)	**Bytes**	**Cycles**	**Opcode (Hex)**
opc	1	12	BF

IRET (Fast)	Bytes	Cycles	Opcode (Hex)
opc	1	6	BF

Example: In the figure below, the instruction pointer is initially loaded with 100H in the main program before interrupt are enabled. When an interrupt occurs, the program counter and the instruction pointer are swapped. This causes the PC to jump to the address 100H and the IP to keep the return address. The last instruction in the service routine is normally a jump to IRET at the address FFH.

This loads the instruction pointer with 100H "again" and causes the program counter to jump back to the main program. Now, the next interrupt can occur and the IP is still correct at 100H.



NOTE: In the fast interrupt example above, if the last instruction is not a jump to IRET, you must pay attention to the order of the last tow instruction. The IRET cannot be immediately proceeded by an instruction which clears the interrupt status (as with a reset of the IPR register).

JP — JUMP

JP cc,dst (Conditional)

JP dst (Unconditional)

Operation: If cc is true, PC ← dst

The conditional JUMP instruction transfers program control to the destination address if the condition specified by the condition code (cc) is true, otherwise, the instruction following the JP instruction is executed. The unconditional JP simply replaces the contents of the PC with the contents of the specified register pair. Control then passes to the statement addressed by the PC.

Flags: No flags are affected.

Format: (1)

(2)		Bytes	Cycles	Opcode (Hex)	Addr Mode dst
cc | opc	dst	3	8	ccD cc = 0 to F	DA
opc	dst	2	8	30	IRR

NOTES:

1. The 3-byte format is used for a conditional jump and the 2-byte format for an unconditional jump.
2. In the first byte of the 3-byte instruction format (conditional jump), the condition code and the OPCODE are both four bits.

Examples: Given: The carry flag (C) = "1", register 00 = 01H, and register 01 = 20H

```
JP  C,LABEL_W      →    LABEL_W = 1000H, PC = 1000H
JP  @00H           →    PC = 0120H
```

The first example shows a conditional JP. Assuming that the carry flag is set to "1", the statement "JP C,LABEL_W" replaces the contents of the PC with the value 1000H and transfers control to that location. Had the carry flag not been set, control would then have passed to the statement immediately following the JP instruction.

The second example shows an unconditional JP. The statement "JP @00" replaces the contents of the PC with the contents of the register pair 00H and 01H, leaving the value 0120H.

JR — Jump Relative

JR cc,dst

Operation: If cc is true, PC ← PC + dst

If the condition specified by the condition code (cc) is true, the relative address is added to the program counter and control passes to the statement whose address is now in the program counter, otherwise, the instruction following the JR instruction is executed. (See the list of condition codes at the beginning of this chapter).

The range of the relative address is +127, –128, and the original value of the program counter is taken to be the address of the first instruction byte following the JR statement.

Flags: No flags are affected.

Format:

(note)	Bytes	Cycles	Opcode (Hex)	Addr Mode dst
cc | opc | dst	2	6	ccB cc = 0 to F	RA

NOTE: In the first byte of the two-byte instruction format, the condition code and the opcode are each four bits in length.

Example: Given: The carry flag = "1" and LABEL_X = 1FF7H:

JR C,LABEL_X → PC = 1FF7H

If the carry flag is set (that is, if the condition code is “true”), the statement "JR C,LABEL_X" will pass control to the statement whose address is currently in the program counter. Otherwise, the program instruction following the JR will be executed.

LD — LOAD

LD dst,src

Operation: dst ← src

The contents of the source are loaded into the destination. The source's contents are unaffected.

Flags: No flags are affected.

Format:

			Bytes	Cycles	Opcode (Hex)	Addr Mode dst	src
dst | opc	src		2	4	rC	r	IM
				4	r8	r	R
src | opc	dst		2	4	r9	R	r
					r = 0 to F		
opc	dst | src		2	4	C7	r	Ir
				4	D7	Ir	r
opc	src	dst	3	6	E4	R	R
				6	E5	R	IR
opc	dst	src	3	6	E6	R	IM
				6	D6	IR	IM
opc	src	dst	3	6	F5	IR	R
opc	dst | src	x	3	6	87	r	x [r]
opc	src | dst	x	3	6	97	x [r]	r

LD — Load

LD (Continued)

Examples: Given: R0 = 01H, R1 = 0AH, register 00H = 01H, register 01H = 20H, register 02H = 02H, LOOP = 30H, and register 3AH = 0FFH:

LD R0,#10H	→	R0 = 10H
LD R0,01H	→	R0 = 20H, register 01H = 20H
LD 01H,R0	→	Register 01H = 01H, R0 = 01H
LD R1,@R0	→	R1 = 20H, R0 = 01H
LD @R0,R1	→	R0 = 01H, R1 = 0AH, register 01H = 0AH
LD 00H,01H	→	Register 00H = 20H, register 01H = 20H
LD 02H,@00H	→	Register 02H = 20H, register 00H = 01H
LD 00H,#0AH	→	Register 00H = 0AH
LD @00H,#10H	→	Register 00H = 01H, register 01H = 10H
LD @00H,02H	→	Register 00H = 01H, register 01H = 02, register 02H = 02H
LD R0,#LOOP[R1]	→	R0 = 0FFH, R1 = 0AH
LD #LOOP[R0],R1	→	Register 31H = 0AH, R0 = 01H, R1 = 0AH

LDB — Load Bit

LDB dst,src.b

LDB dst.b,src

Operation: dst(0) ← src(b)
or
dst(b) ← src(0)

The specified bit of the source is loaded into bit zero (LSB) of the destination, or bit zero of the source is loaded into the specified bit of the destination. No other bits of the destination are affected. The source is unaffected.

Flags: No flags are affected.

Format:

			Bytes	Cycles	Opcode (Hex)	Addr Mode dst	Addr Mode src
opc	dst | b | 0	src	3	6	47	r0	Rb
opc	src | b | 1	dst	3	6	47	Rb	r0

NOTE: In the second byte of the instruction format, the destination (or the source) address is four bits, the bit address "b" is three bits, and the LSB address value is one bit in length.

Examples: Given: R0 = 06H and general register 00H = 05H:

```
LDB     R0,00H.2    →    R0 = 07H, register 00H = 05H
LDB     00H.0,R0    →    R0 = 06H, register 00H = 04H
```

In the first example, the destination working register R0 contains the value 06H and the source general register 00H the value 05H. The statement "LD R0,00H.2" loads the bit two value of the 00H register into bit zero of the R0 register, leaving the value 07H in the register R0.

In the second example, 00H is the destination register. The statement "LD 00H.0,R0" loads bit zero of the register R0 to the specified bit (bit zero) of the destination register, leaving 04H in the general register 00H.

LDC/LDE — Load Memory

LDC dst,src

LDE dst,src

Operation: dst ← src

This instruction loads a byte from program or data memory into a working register or vice-versa. The source values are unaffected. LDC refers to program memory and LDE to data memory. The assembler makes "Irr" or "rr" values an even number for program memory and an odd number for data memory.

Flags: No flags are affected.

Format:

					Bytes	Cycles	Opcode (Hex)	Addr Mode dst	Addr Mode src
1.	opc	dst | src			2	10	C3	r	Irr
2.	opc	src | dst			2	10	D3	Irr	r
3.	opc	dst | src	XS		3	12	E7	r	XS [rr]
4.	opc	src | dst	XS		3	12	F7	XS [rr]	r
5.	opc	dst | src	XL_L	XL_H	4	14	A7	r	XL [rr]
6.	opc	src | dst	XL_L	XL_H	4	14	B7	XL [rr]	r
7.	opc	dst | 0000	DA_L	DA_H	4	14	A7	r	DA
8.	opc	src | 0000	DA_L	DA_H	4	14	B7	DA	r
9.	opc	dst | 0001	DA_L	DA_H	4	14	A7	r	DA
10.	opc	src | 0001	DA_L	DA_H	4	14	B7	DA	r

NOTES:

1. The source (src) or the working register pair [rr] for formats 5 and 6 cannot use the register pair 0–1.
2. For the formats 3 and 4, the destination "XS [rr]" and the source address "XS [rr]" are both one byte.
3. For the formats 5 and 6, the destination "XL [rr] and the source address "XL [rr]" are both two bytes.
4. The DA and the r source values for the formats 7 and 8 are used to address program memory. The second set of values, used in the formats 9 and 10, are used to address data memory.
5. LDE instruction can be used to read/write the data of 64-Kbyte data memory.

LDC/LDE — Load Memory

LDC/LDE (Continued)

Examples: Given: R0 = 11H, R1 = 34H, R2 = 01H, R3 = 04H; Program memory locations 0103H = 4FH, 0104H = 1A, 0105H = 6DH, and 1104H = 88H.
External data memory locations 0103H = 5FH, 0104H = 2AH, 0105H = 7DH, and 1104H = 98H:

```
LDC     R0,@RR2          ; R0 ← contents of program memory location 0104H;
                         ; R0 = 1AH, R2 = 01H, R3 = 04H
LDE     R0,@RR2          ; R0 ← contents of external data memory location
                           0104H;
                         ; R0 = 2AH, R2 = 01H, R3 = 04H
LDC     @RR2,R0          ; 11H (contents of R0) is loaded into program memory
                         ; location 0104H (RR2); R0, R2, R3 → no change
LDE     @RR2,R0          ; 11H (contents of R0) is loaded into external data
                           memory
                         ; location 0104H (RR2); R0, R2, R3 → no change
LDC     R0,#01H[RR2]     ; R0 ← contents of program memory location 0105H
                         ; (01H + RR2); R0 = 6DH, R2 = 01H, R3 = 04H
LDE     R0,#01H[RR2]     ; R0 ← contents of external data memory location
                           0105H
                         ; (01H + RR2); R0 = 7DH, R2 = 01H, R3 = 04H
LDC     #01H[RR2],R0     ; 11H (contents of R0) is loaded into program memory
                           location
                         ; 0105H (01H + 0104H)
LDE     #01H[RR2],R0     ; 11H (contents of R0) is loaded into external data
                           memory
                         ; location 0105H (01H + 0104H)
LDC     R0,#1000H[RR2]   ; R0 ← contents of program memory location 1104H
                         ; (1000H + 0104H); R0 = 88H, R2 = 01H, R3 = 04H
LDE     R0,#1000H[RR2]   ; R0 ← contents of external data memory location
                           1104H
                         ; (1000H + 0104H); R0 = 98H, R2 = 01H, R3 = 04H
LDC     R0,1104H         ; R0 ← contents of program memory location 1104H
                         ; R0 = 88H
LDE     R0,1104H         ; R0 ← contents of external data memory location
                           1104H;
                         ; R0 = 98H
LDC     1105H,R0         ; 11H (contents of R0) is loaded into program memory
                           location
                         ; 1105H; (1105H) ← 11H
LDE     1105H,R0         ; 11H (contents of R0) is loaded into external data
                           memory
                         ; location 1105H; (1105H) ← 11H
```

NOTE: The LDC and the LDE instructions are not supported by masked ROM type devices.

LDCD/LDED — Load Memory and Decrement

LDCD dst,src

LDED dst,src

Operation: dst ← src
rr ← rr – 1

These instructions are used for user stacks or block transfers of data from program or data memory to the register file. The address of the memory location is specified by a working register pair. The contents of the source location are loaded into the destination location. The memory address is then decremented. The contents of the source are unaffected.

LDCD refers to program memory and LDED refers to external data memory. The assembler makes "Irr" an even number for program memory and an odd number for data memory.

Flags: No flags are affected.

Format:

		Bytes	Cycles	Opcode (Hex)	Addr Mode dst	src
opc	dst | src	2	10	E2	r	Irr

Examples: Given: R6 = 10H, R7 = 33H, R8 = 12H, program memory location 1033H = 0CDH, and external data memory location 1033H = 0DDH:

```
LDCD    R8,@RR6     ; 0CDH (contents of program memory location 1033H) is loaded
                    ; into R8 and RR6 is decremented by one;
                    ; R8 = 0CDH, R6 = 10H, R7 = 32H (RR6 ← RR6 – 1)
LDED    R8,@RR6     ; 0DDH (contents of data memory location 1033H) is loaded
                    ; into R8 and RR6 is decremented by one
                    (RR6 ← RR6 – 1);
                    ; R8 = 0DDH, R6 = 10H, R7 = 32H
```

NOTE: LDED instruction can be used to read/write the data of 64-Kbyte data memory.

LDCI/LDEI — Load Memory and Increment

LDCI dst,src

LDEI dst,src

Operation: dst ← src
rr ← rr + 1

These instructions are used for user stacks or block transfers of data from program or data memory to the register file. The address of the memory location is specified by a working register pair. The contents of the source location are loaded into the destination location. The memory address is then incremented automatically. The contents of the source are unaffected.
LDCI refers to program memory and LDEI refers to external data memory. The assembler makes "Irr" an even number for program memory and an odd number for data memory.

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)	Addr Mode dst	src
opc | dst | src	2	10	E3	r	Irr

Examples: Given: R6 = 10H, R7 = 33H, R8 = 12H, program memory locations 1033H = 0CDH and 1034H = 0C5H; external data memory locations 1033H = 0DDH and 1034H = 0D5H:

```
LDCI    R8,@RR6     ; 0CDH (contents of program memory location 1033H) is loaded
                    ; into R8 and RR6 is incremented by one (RR6 ← RR6 + 1);
                    ; R8 = 0CDH, R6 = 10H, R7 = 34H
LDEI    R8,@RR6     ; 0DDH (contents of data memory location 1033H) is loaded
                    ; into R8 and RR6 is incremented by one (RR6 ← RR6 + 1);
                    ; R8 = 0DDH, R6 = 10H, R7 = 34H
```

NOTE: LDEI instruction can be used to read/write the data of 64-Kbyte data memory.

`

LDCPD/LDEPD — Load Memory with Pre-Decrement

LDCPD dst,src

LDEPD dst,src

Operation: rr ← rr – 1
dst ← src

These instructions are used for block transfers of data from program or data memory to the register file. The address of the memory location is specified by a working register pair and is first decremented. The contents of the source location are then loaded into the destination location. The contents of the source are unaffected.

LDCPD refers to program memory and LDEPD refers to external data memory. The assembler makes "Irr" an even number for program memory and an odd number for external data memory.

Flags: No flags are affected.

Format:

		Bytes	Cycles	Opcode (Hex)	Addr Mode dst	src
opc	src | dst	2	14	F2	Irr	r

Examples: Given: R0 = 77H, R6 = 30H, and R7 = 00H:

```
LDCPD   @RR6,R0     ; (RR6 ← RR6 – 1)
                    ; 77H (the contents of R0) is loaded into program memory
                    ; location 2FFFH (3000H – 1H);
                    ; R0 = 77H, R6 = 2FH, R7 = 0FFH
LDEPD   @RR6,R0     ; (RR6 ← RR6 – 1)
                    ; 77H (the contents of R0) is loaded into external data
memory
                    ; location 2FFFH (3000H – 1H);
```

NOTE: LDEPD instruction can be used to read/write the data of 64-Kbyte data memory.

LDCPI/LDEPI — Load Memory with Pre-Increment

LDCPI dst,src

LDEPI dst,src

Operation: rr ← rr + 1
dst ← src

These instructions are used for block transfers of data from program or data memory to the register file. The address of the memory location is specified by a working register pair and is first incremented. The contents of the source location are loaded into the destination location. The contents of the source are unaffected.

LDCPI refers to program memory and LDEPI refers to external data memory. The assembler makes "Irr" an even number for program memory and an odd number for data memory.

Flags: No flags are affected.

Format:

		Bytes	Cycles	Opcode (Hex)	Addr Mode dst	src
opc	src | dst	2	14	F3	Irr	r

Examples: Given: R0 = 7FH, R6 = 21H, and R7 = 0FFH:

```
LDCPI   @RR6,R0     ; (RR6 ← bRR6 + 1)
                    ; 7FH (the contents of R0) is loaded into program memory
                    ; location 2200H (21FFH + 1H);
                    ; R0 = 7FH, R6 = 22H, R7 = 00H
LDEPI   @RR6,R0     ; (RR6 ← bRR6 + 1)
                    ; 7FH (the contents of R0) is loaded into external data
memory
                    ; location 2200H (21FFH + 1H);
                    ; R0 = 7FH, R6 = 22H, R7 = 00H
```

NOTE: LDEPI instruction can be used to read/write the data of 64-Kbyte data memory.

LDW — Load Word

LDW dst,src

Operation: dst ← src

The contents of the source (a word) are loaded into the destination. The contents of the source are unaffected.

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)	Addr Mode dst	Addr Mode src
opc | src | dst	3	8	C4	RR	RR
		8	C5	RR	IR
opc | dst | src	4	8	C6	RR	IML

Examples: Given: R4 = 06H, R5 = 1CH, R6 = 05H, R7 = 02H, register 00H = 1AH, register 01H = 02H, register 02H = 03H,and register 03H = 0FH

```
LDW     RR6,RR4     →   R6 = 06H, R7 = 1CH, R4 = 06H, R5 = 1CH
LDW     00H,02H     →   Register 00H = 03H, register 01H = 0FH,
                        register 02H = 03H, register 03H = 0FH
LDW     RR2,@R7     →   R2 = 03H, R3 = 0FH,
LDW     04H,@01H    →   Register 04H = 03H, register 05H = 0FH
LDW     RR6,#1234H  →   R6 = 12H, R7 = 34H
LDW     02H,#0FEDH  →   Register 02H = 0FH, register 03H = 0EDH
```

In the second example, please note that the statement "LDW 00H,02H" loads the contents of the source word 02H and 03H into the destination word 00H and 01H. This leaves the value 03H in the general register 00H and the value 0FH in the register 01H.

Other examples show how to use the LDW instruction with various addressing modes and formats.

MULT — Multiply (Unsigned)

MULT dst,src

Operation: dst ← dst × src

The 8-bit destination operand (the even numbered register of the register pair) is multiplied by the source operand (8 bits) and the product (16 bits) is stored in the register pair specified by the destination address. Both operands are treated as unsigned integers.

Flags:

C: Set if the result is > 255; cleared otherwise.

Z: Set if the result is "0"; cleared otherwise.

S: Set if MSB of the result is a "1"; cleared otherwise.

V: Cleared.

D: Unaffected.

H: Unaffected.

Format:

			Bytes	Cycles	Opcode (Hex)	Addr Mode dst	src
opc	src	dst	3	22	84	RR	R
				22	85	RR	IR
				22	86	RR	IM

Examples: Given: Register 00H = 20H, register 01H = 03H, register 02H = 09H, register 03H = 06H:

```
MULT    00H, 02H    →    Register 00H = 01H, register 01H = 20H,
                         register 02H = 09H
MULT    00H, @01H   →    Register 00H = 00H, register 01H = 0C0H
MULT    00H, #30H   →    Register 00H = 06H, register 01H = 00H
```

In the first example, the statement "MULT 00H,02H" multiplies the 8-bit destination operand (in the register 00H of the register pair 00H, 01H) by the source register 02H operand (09H). The 16-bit product, 0120H, is stored in the register pair 00H, 01H.

NEXT — Next

NEXT

Operation: PC ← @IP
IP ← IP + 2

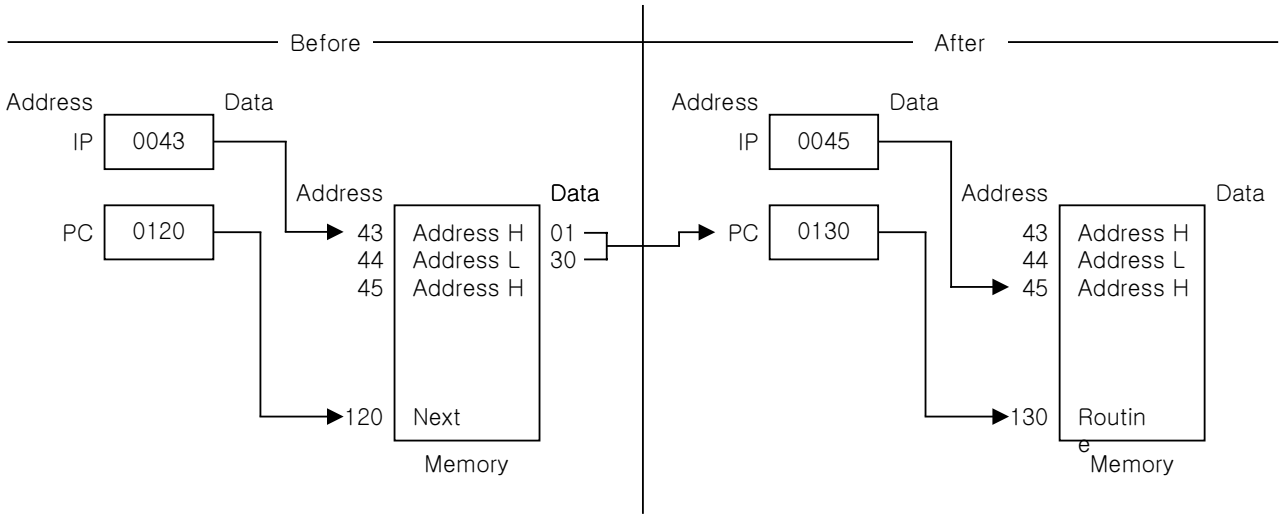
The NEXT instruction is useful when implementing threaded-code languages. The program memory word that is pointed to by the instruction pointer is loaded into the program counter. The instruction pointer is then incremented by two.

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)
opc	1	10	0F

Example: The following diagram shows an example of how to use the NEXT instruction.



NOP — No Operation

NOP

Operation: No action is performed when the CPU executes this instruction. Typically, one or more NOPs are executed in sequence in order to affect a timing delay of variable duration.

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)
opc	1	4	FF

Example: When the instruction NOP is executed in a program, no operation occurs. Instead, there happens a delay in instruction execution time which is of approximately one machine cycle per each **NOP** instruction encountered.

OR — Logical OR

OR dst,src

Operation: dst ← dst OR src

The source operand is logically ORed with the destination operand and the result is stored in the destination. The contents of the source are unaffected. The OR operation results in a "1" being stored whenever either of the corresponding bits in the two operands is a "1", otherwise, a "0" is stored.

Flags:

C: Unaffected.

Z: Set if the result is "0"; cleared otherwise.

S: Set if the result bit 7 is set; cleared otherwise.

V: Always cleared to "0".

D: Unaffected.

H: Unaffected.

Format:

			Bytes	Cycles	Opcode (Hex)	Addr Mode dst	src
opc	dst | src		2	4	42	r	r
				6	43	r	lr
opc	src	dst	3	6	44	R	R
				6	45	R	IR
opc	dst	src	3	6	46	R	IM

Examples: Given: R0 = 15H, R1 = 2AH, R2 = 01H, register 00H = 08H, register 01H = 37H, and register 08H = 8AH

OR	R0,R1	→	R0 = 3FH, R1 = 2AH
OR	R0,@R2	→	R0 = 37H, R2 = 01H, register 01H = 37H
OR	00H,01H	→	Register 00H = 3FH, register 01H = 37H
OR	01H,@00H	→	Register 00H = 08H, register 01H = 0BFH
OR	00H,#02H	→	Register 00H = 0AH

In the first example, if the working register R0 contains the value 15H and the register R1 the value 2AH, the statement "OR R0,R1" logical-ORs the R0 and R1 register contents and stores the result (3FH) in the destination register R0.

Other examples show the use of the logical OR instruction with various addressing modes and formats.

POP — Pop from Stack

POP dst

Operation: dst ← @SP
SP ← SP + 1

The contents of the location addressed by the stack pointer are loaded into the destination. The stack pointer is then incremented by one.

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)	Addr Mode dst
opc | dst	2	8	50	R
		8	51	IR

Examples: Given: Register 00H = 01H, register 01H = 1BH, SPH (0D8H) = 00H, SPL (0D9H) = 0FBH, and stack register 0FBH = 55H:

```
POP   00H    →   Register 00H = 55H, SP = 00FCH
POP   @00H   →   Register 00H = 01H, register 01H = 55H, SP = 00FCH
```

In the first example, the general register 00H contains the value 01H. The statement "POP 00H" loads the contents of the location 00FBH (55H) into the destination register 00H and then increments the stack pointer by one. The register 00H then contains the value 55H and the SP points to the location 00FCH.

POPUD — Pop User Stack (Decrementing)

POPUD dst,src

Operation: dst ← src
IR ← IR – 1

This instruction is used for user-defined stacks in the register file. The contents of the register file location addressed by the user stack pointer are loaded into the destination. The user stack pointer is then decremented.

Flags: No flags are affected.

Format:

			Bytes	Cycles	Opcode (Hex)	Addr Mode dst	Addr Mode src
opc	src	dst	3	8	92	R	IR

Example: Given: Register 00H = 42H (user stack pointer register), register 42H = 6FH, and register 02H = 70H:

POPUD 02H,@00H → Register 00H = 41H, register 02H = 6FH, register 42H = 6FH

If the general register 00H contains the value 42H and the register 42H the value 6FH, the statement "POPUD 02H,@00H" loads the contents of the register 42H into the destination register. The user stack pointer is then decremented by one, leaving the value 41H.

POPUI — Pop User Stack (Incrementing)

POPUI dst,src

Operation: dst ← src
IR ← IR + 1

The POPUI instruction is used for user-defined stacks in the register file. The contents of the register file location addressed by the user stack pointer are loaded into the destination. The user stack pointer is then incremented.

Flags: No flags are affected.

Format:

			Bytes	Cycles	Opcode (Hex)	Addr Mode dst	Addr Mode src
opc	src	dst	3	8	93	R	IR

Example: Given: Register 00H = 01H and register 01H = 70H:

POPUI 02H,@00H → Register 00H = 02H, register 01H = 70H, register 02H = 70H

If the general register 00H contains the value 01H and the register 01H the value 70H, the statement "POPUI 02H,@00H" loads the value 70H into the destination general register 02H. The user stack pointer (the register 00H) is then incremented by one, changing its value from 01H to 02H.

PUSH — Push to Stack

PUSH src

Operation: SP ← SP – 1
@SP ← src

A PUSH instruction decrements the stack pointer value and loads the contents of the source (src) into the location addressed by the decremented stack pointer. The operation then adds the new value to the top of the stack.

Flags: No flags are affected.

Format:

		Bytes	Cycles	Opcode (Hex)	Addr Mode dst
opc	src	2	8 (internal clock)	70	R
			8 (external clock)		
			8 (internal clock)		
			8 (external clock)	71	IR

Examples: Given: Register 40H = 4FH, register 4FH = 0AAH, SPH = 00H, and SPL = 00H:

PUSH 40H → Register 40H = 4FH, stack register 0FFH = 4FH, SPH = 0FFH, SPL = 0FFH

PUSH @40H → Register 40H = 4FH, register 4FH = 0AAH, stack register 0FFH = 0AAH, SPH = 0FFH, SPL = 0FFH

In the first example, if the stack pointer contains the value 0000H, and the general register 40H the value 4FH, the statement "PUSH 40H" decrements the stack pointer from 0000 to 0FFFFH. It then loads the contents of the register 40H into the location 0FFFFH and adds this new value to the top of the stack.

PUSHUD — Push User Stack (Decrementing)

PUSHUD dst,src

Operation: IR ← IR – 1
dst ← src

This instruction is used to address user-defined stacks in the register file. PUSHUD decrements the user stack pointer and loads the contents of the source into the register addressed by the decremented stack pointer.

Flags: No flags are affected.

Format:

			Bytes	Cycles	Opcode (Hex)	Addr Mode dst	src
opc	dst	src	3	8	82	IR	R

Example: Given: Register 00H = 03H, register 01H = 05H, and register 02H = 1AH:

PUSHUD @00H,01H → Register 00H = 02H, register 01H = 05H, register 02H = 05H

If the user stack pointer (the register 00H, for example) contains the value 03H, the statement "PUSHUD @00H,01H" decrements the user stack pointer by one, leaving the value 02H. The 01H register value, 05H, is then loaded into the register addressed by the decremented user stack pointer.

PUSHUI — Push User Stack (Incrementing)

PUSHUI dst,src

Operation: IR ← IR + 1
dst ← src

This instruction is used for user-defined stacks in the register file. PUSHUI increments the user stack pointer and then loads the contents of the source into the register location addressed by the incremented user stack pointer.

Flags: No flags are affected.

Format:

			Bytes	Cycles	Opcode (Hex)	Addr Mode dst	Addr Mode src
opc	dst	src	3	8	83	IR	R

Example: Given: Register 00H = 03H, register 01H = 05H, and register 04H = 2AH:

PUSHUI @00H,01H → Register 00H = 04H, register 01H = 05H, register 04H = 05H

If the user stack pointer (the register 00H, for example) contains the value 03H, the statement "PUSHUI @00H,01H" increments the user stack pointer by one, leaving the value 04H. The 01H register value, 05H, is then loaded into the location addressed by the incremented user stack pointer.

RCF — Reset Carry Flag

RCF RCF

Operation: C ← 0

The carry flag is cleared to logic zero, regardless of its previous value.

Flags: **C:** Cleared to "0".

No other flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)
opc	1	4	CF

Example: Given: C = "1" or "0":

The instruction RCF clears the carry flag (C) to logic zero.

RET — Return

RET

Operation: PC ← @SP
SP ← SP + 2

The RET instruction is normally used to return to the previously executed procedure at the end of the procedure entered by a CALL instruction. The contents of the location addressed by the stack pointer are popped into the program counter. The next statement to be executed is the one that is addressed by the new program counter value.

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)
opc	1	10	AF

Example: Given: SP = 00FCH, (SP) = 101AH, and PC = 1234:

RET → PC = 101AH, SP = 00FEH

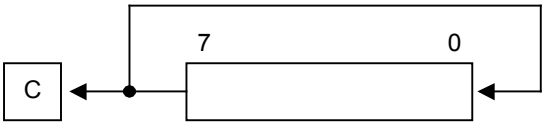
The RET instruction pops the contents of the stack pointer location 00FCH (10H) into the high byte of the program counter. The stack pointer then pops the value in the location 00FEH (1AH) into the PC's low byte and the instruction at the location 101AH is executed. The stack pointer now points to the memory location 00FEH.

RL — Rotate Left

RL dst

Operation: C ← dst (7)
dst (0) ← dst (7)
dst (n + 1) ← dst (n), n = 0–6

The contents of the destination operand are rotated left one bit position. The initial value of bit 7 is moved to the bit zero (LSB) position and also replaces the carry flag, as shown in the figure below.



Flags:
C: Set if the bit rotated from the most significant bit position (bit 7) was "1".
Z: Set if the result is "0"; cleared otherwise.
S: Set if the result bit 7 is set; cleared otherwise.
V: Set if arithmetic overflow occurred; cleared otherwise.
D: Unaffected.
H: Unaffected.

Format:

	Bytes	Cycles	Opcode (Hex)	Addr Mode dst
opc | dst	2	4	90	R
		4	91	IR

Examples: Given: Register 00H = 0AAH, register 01H = 02H and register 02H = 17H:

RL 00H → Register 00H = 55H, C = "1"
RL @01H → Register 01H = 02H, register 02H = 2EH, C = "0"

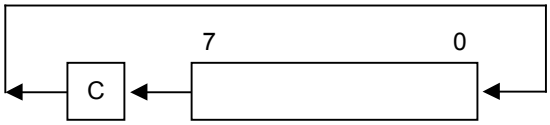
In the first example, if the general register 00H contains the value 0AAH (10101010B), the statement "RL 00H" rotates the 0AAH value left one bit position, leaving the new value 55H (01010101B) and setting the carry (C) and the overflow (V) flags.

RLC — Rotate Left through Carry

RLC dst

Operation: dst (0) ← C
C ← dst (7)
dst (n + 1) ← dst (n), n = 0–6

The contents of the destination operand with the carry flag are rotated left one bit position. The initial value of bit 7 replaces the carry flag (C), and the initial value of the carry flag replaces bit zero.



Flags:
C: Set if the bit rotated from the most significant bit position (bit 7) was "1".
Z: Set if the result is "0"; cleared otherwise.
S: Set if the result bit 7 is set; cleared otherwise.
V: Set if arithmetic overflow occurred, that is, if the sign of the destination is changed during the rotation; cleared otherwise.
D: Unaffected.
H: Unaffected.

Format:

	Bytes	Cycles	Opcode (Hex)	Addr Mode dst
opc dst	2	4	10	R
		4	11	IR

Examples: Given: Register 00H = 0AAH, register 01H = 02H, and register 02H = 17H, C = "0":

```
RLC   00H    →   Register 00H = 54H, C = "1"
RLC   @01H   →   Register 01H = 02H, register 02H = 2EH, C = "0"
```

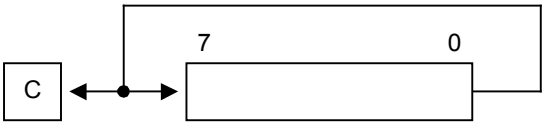
In the first example, if the general register 00H has the value 0AAH (10101010B), the statement "RLC 00H" rotates 0AAH one bit position to the left. The initial value of bit 7 sets the carry flag and the initial value of the C flag replaces bit zero of the register 00H, leaving the value 55H (01010101B). The MSB of the register 00H resets the carry flag to "1" and sets the overflow flag.

RR — Rotate Right

RR dst

Operation: C ← dst (0)
dst (7) ← dst (0)
dst (n) ← dst (n + 1), n = 0–6

The contents of the destination operand are rotated right one bit position. The initial value of bit zero (LSB) is moved to bit 7 (MSB) and also replaces the carry flag (C).



Flags:
C: Set if the bit rotated from the least significant bit position (bit zero) was "1".
Z: Set if the result is "0"; cleared otherwise.
S: Set if the result bit 7 is set; cleared otherwise.
V: Set if arithmetic overflow occurred, that is, if the sign of the destination is changed during the rotation; cleared otherwise.
D: Unaffected.
H: Unaffected.

Format:

	Bytes	Cycles	Opcode (Hex)	Addr Mode dst
opc dst	2	4	E0	R
		4	E1	IR

Examples: Given: Register 00H = 31H, register 01H = 02H, and register 02H = 17H:

```
RR    00H    →    Register 00H = 98H, C = "1"
RR    @01H   →    Register 01H = 02H, register 02H = 8BH, C = "1"
```

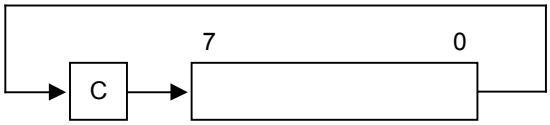
In the first example, if the general register 00H contains the value 31H (00110001B), the statement "RR 00H" rotates this value one bit position to the right. The initial value of bit zero is moved to bit 7, leaving the new value 98H (10011000B) in the destination register. The initial bit zero also resets the C flag to "1" and the sign flag and the overflow flag are also set to "1".

RRC — Rotate Right through Carry

RRC dst

Operation: dst (7) ← C
C ← dst (0)
dst (n) ← dst (n + 1), n = 0–6

The contents of the destination operand and the carry flag are rotated right one bit position. The initial value of bit zero (LSB) replaces the carry flag, and the initial value of the carry flag replaces bit 7 (MSB).



Flags:
C: Set if the bit rotated from the least significant bit position (bit zero) was "1".
Z: Set if the result is "0" cleared otherwise.
S: Set if the result bit 7 is set; cleared otherwise.
V: Set if arithmetic overflow occurred, that is, if the sign of the destination is changed during the rotation; cleared otherwise.
D: Unaffected.
H: Unaffected.

Format:

	Bytes	Cycles	Opcode (Hex)	Addr Mode dst
opc dst	2	4	C0	R
		4	C1	IR

Examples: Given: Register 00H = 55H, register 01H = 02H, register 02H = 17H, and C = "0":

```
RRC     00H     →     Register 00H = 2AH, C = "1"
RRC     @01H    →     Register 01H = 02H, register 02H = 0BH, C = "1"
```

In the first example, if the general register 00H contains the value 55H (01010101B), the statement "RRC 00H" rotates this value one bit position to the right. The initial value of bit zero ("1") replaces the carry flag and the initial value of the C flag ("1") replaces bit 7. This leaves the new value 2AH (00101010B) in the destination register 00H. The sign flag and the overflow flag are both cleared to "0".

SB0 — Select Bank 0

SB0

Operation: BANK ← 0

The SB0 instruction clears the bank address flag in the FLAGS register (FLAGS.0) to logic zero, selecting the bank 0 register addressing in the set 1 area of the register file.

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)
opc	1	4	4F

Example: The statement **SB0** clears FLAGS.0 to "0", selecting the bank 0 register addressing.

SB1 — Select Bank 1

SB1

Operation: BANK ← 1

The SB1 instruction sets the bank address flag in the FLAGS register (FLAGS.0) to logic one, selecting the bank 1 register addressing in the set 1 area of the register file.

NOTE: Bank 1 is not implemented in some KS88-series microcontrollers.

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)
opc	1	4	5F

Example: The statement **SB1** sets FLAGS.0 to “1”, selecting the bank 1 register addressing (if bank 1 is implemented in the microcontroller’s internla register file).

SBC — Subtract with Carry

SBC dst,src

Operation: dst ← dst – src – c

The source operand, along with the current value of the carry flag, is subtracted from the destination operand and the result is stored in the destination. The contents of the source are unaffected. Subtraction is performed by adding the two's-complement of the source operand to the destination operand. In multiple precision arithmetic, this instruction permits the carry ("borrow") from the subtraction of the low-order operands to be subtracted from the subtraction of high-order operands.

Flags:

C: Set if a borrow occurred (src > dst); cleared otherwise.

Z: Set if the result is "0"; cleared otherwise.

S: Set if the result is negative; cleared otherwise.

V: Set if arithmetic overflow occurred, that is, if the operands were of opposite sign and the sign of the result is the same as the sign of the source; cleared otherwise.

D: Always set to "1".

H: Cleared if there is a carry from the most significant bit of the low-order four bits of the result; set otherwise, indicating a "borrow"

Format:

			Bytes	Cycles	Opcode (Hex)	Addr Mode dst	Addr Mode src
opc	dst | src		2	4	32	r	r
				6	33	r	lr
opc	src	dst	3	6	34	R	R
				6	35	R	IR
opc	dst	src	3	6	36	R	IM

Examples: Given: R1 = 10H, R2 = 03H, C = "1", register 01H = 20H, register 02H = 03H, and register 03H = 0AH:

SBC	R1,R2	→	R1 = 0CH, R2 = 03H
SBC	R1,@R2	→	R1 = 05H, R2 = 03H, register 03H = 0AH
SBC	01H,02H	→	Register 01H = 1CH, register 02H = 03H
SBC	01H,@02H	→	Register 01H = 15H, register 02H = 03H, register 03H = 0AH
SBC	01H,#8AH	→	Register 01H = 95H; C, S, and V = "1"

In the first example, if the working register R1 contains the value 10H and the register R2 the value 03H, the statement "SBC R1,R2" subtracts the source value (03H) and the C flag value ("1") from the destination (10H) and then stores the result (0CH) in the register R1.

SCF — Set Carry Flag

SCF

Operation: C ← 1

The carry flag (C) is set to logic one, regardless of its previous value.

Flags: C: Set to "1".

No other flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)
opc	1	4	DF

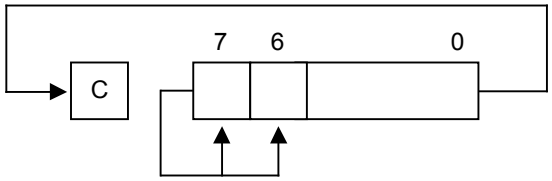
Example: The statement **SCF** sets the carry flag to “1”.

SRA — Shift Right Arithmetic

SRA dst

Operation: dst (7) ← dst (7)
C ← dst (0)
dst (n) ← dst (n + 1), n = 0–6

An arithmetic shift-right of one bit position is performed on the destination operand. Bit zero (the LSB) replaces the carry flag. The value of bit 7 (the sign bit) is unchanged and is shifted into the bit position 6.



Flags:
C: Set if the bit shifted from the LSB position (bit zero) was "1".
Z: Set if the result is "0"; cleared otherwise.
S: Set if the result is negative; cleared otherwise.
V: Always cleared to "0".
D: Unaffected.
H: Unaffected.

Format:

	Bytes	Cycles	Opcode (Hex)	Addr Mode dst
opc dst	2	4	D0	R
		4	D1	IR

Examples: Given: Register 00H = 9AH, register 02H = 03H, register 03H = 0BCH, and C = "1":

SRA 00H → Register 00H = 0CD, C = "0"
SRA @02H → Register 02H = 03H, register 03H = 0DEH, C = "0"

In the first example, if the general register 00H contains the value 9AH (10011010B), the statement "SRA 00H" shifts the bit values in the register 00H right one bit position. Bit zero ("0") clears the C flag and bit 7 ("1") is then shifted into the bit 6 position (bit 7 remains unchanged). This leaves the value 0CDH (11001101B) in the destination register 00H.

SRP/SRP0/SRP1 — Set Register Pointer

SRP src

SRP0 src

SRP1 src

Operation:

If src (1) = 1 and src (0) = 0 then: RP0 (3–7) ← src (3–7)
If src (1) = 0 and src (0) = 1 then: RP1 (3–7) ← src (3–7)
If src (1) = 0 and src (0) = 0 then: RP0 (4–7) ← src (4–7),
RP0 (3) ← 0
RP1 (4–7) ← src (4–7),
RP1 (3) ← 1

The source data bits one and zero (LSB) determine whether to write one or both of the register pointers, RP0 and RP1. Bits 3–7 of the selected register pointer are written unless both register pointers are selected. RP0.3 is then cleared to logic zero and RP1.3 is set to logic one.

Flags: No flags are affected.

Format:

		Bytes	Cycles	Opcode (Hex)	Addr Mode src
opc	src	2	4	31	IM

Examples: The statement **SRP #40H** sets the register pointer 0 (RP0) at the location 0D6H to 40H and the register pointer 1 (RP1) at the location 0D7H to 48 H.

The statement "SRP0 #50H" would set RP0 to 50H, and the statement "SRP1 #68H" would set RP1 to 68H.

NOTE: Before execute the STOP instruction, You must set the STPCON register as “10100101b”. Otherwise the STOP instruction will not execute.

STOP — Stop Operation

STOP

Operation: The STOP instruction stops the both the CPU clock and system clock and causes the microcontroller to enter Stop mode. During Stop mode, the contents of on-chip CPU registers, peripheral registers, and I/O port control and data registers are retained. Stop mode can be released by an external reset operation or by external interrupts. For the reset operation, the RESET pin must be held to Low level until the required oscillation stabilization interval has elapsed.

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)	Addr Mode dst	src
opc	1	4	7F	–	–

Example: The statement **STOP** halts all microcontroller operations.

SUB — Subtract

SUB dst,src

Operation: dst ← dst – src

The source operand is subtracted from the destination operand and the result is stored in the destination. The contents of the source are unaffected. Subtraction is performed by adding the two's complement of the source operand to the destination operand.

Flags:

C: Set if a "borrow" occurred; cleared otherwise.

Z: Set if the result is "0"; cleared otherwise.

S: Set if the result is negative; cleared otherwise.

V: Set if arithmetic overflow occurred, that is, if the operands were of opposite signs and the sign of the result is of the same as the sign of the source operand; cleared otherwise.

D: Always set to "1".

H: Cleared if there is a carry from the most significant bit of the low-order four bits of the result; set otherwise indicating a “borrow”.

Format:

			Bytes	Cycles	Opcode (Hex)	Addr Mode dst	Addr Mode src
opc	dst | src		2	4	22	r	r
				6	23	r	lr
opc	src	dst	3	6	24	R	R
				6	25	R	IR
opc	dst	src	3	6	26	R	IM

Examples: Given: R1 = 12H, R2 = 03H, register 01H = 21H, register 02H = 03H, register 03H = 0AH:

SUB	R1,R2	→	R1 = 0FH, R2 = 03H
SUB	R1,@R2	→	R1 = 08H, R2 = 03H
SUB	01H,02H	→	Register 01H = 1EH, register 02H = 03H
SUB	01H,@02H	→	Register 01H = 17H, register 02H = 03H
SUB	01H,#90H	→	Register 01H = 91H; C, S, and V = "1"
SUB	01H,#65H	→	Register 01H = 0BCH; C and S = "1", V = "0"

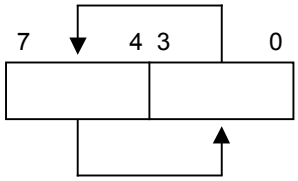
In the first example, if he working register R1 contains the value 12H and if the register R2 contains the value 03H, the statement "SUB R1,R2" subtracts the source value (03H) from the destination value (12H) and stores the result (0FH) in the destination register R1.

SWAP — Swap Nibbles

SWAP dst

Operation: dst (0 – 3) ↔ dst (4 – 7)

The contents of the lower four bits and the upper four bits of the destination operand are swapped.



Flags:

C: Undefined.

Z: Set if the result is "0"; cleared otherwise.

S: Set if the result bit 7 is set; cleared otherwise.

V: Undefined.

D: Unaffected.

H: Unaffected.

Format:

	Bytes	Cycles	Opcode (Hex)	Addr Mode dst
opc dst	2	4	F0	R
		4	F1	IR

Examples: Given: Register 00H = 3EH, register 02H = 03H, and register 03H = 0A4H:

```
SWAP    00H     →    Register 00H = 0E3H
SWAP    @02H    →    Register 02H = 03H, register 03H = 4AH
```

In the first example, if the general register 00H contains the value 3EH (00111110B), the statement "SWAP 00H" swaps the lower and the upper four bits (nibbles) in the 00H register, leaving the value 0E3H (11100011B).

TCM — Test Complement under Mask

TCM dst,src

Operation: (NOT dst) AND src

This instruction tests selected bits in the destination operand for a logic one value. The bits to be tested are specified by setting a "1" bit in the corresponding position of the source operand (mask). The TCM statement complements the destination operand, which is then ANDed with the source mask. The zero (Z) flag can then be checked to determine the result. The destination and the source operands are unaffected.

Flags:

C: Unaffected.

Z: Set if the result is "0"; cleared otherwise.

S: Set if the result bit 7 is set; cleared otherwise.

V: Always cleared to "0".

D: Unaffected.

H: Unaffected.

Format:

			Bytes	Cycles	Opcode (Hex)	Addr Mode dst	Addr Mode src
opc	dst | src		2	4	62	r	r
				6	63	r	lr
opc	src	dst	3	6	64	R	R
				6	65	R	IR
opc	dst	src	3	6	66	R	IM

Examples: Given: R0 = 0C7H, R1 = 02H, R2 = 12H, register 00H = 2BH, register 01H = 02H, and register 02H = 23H:

```
TCM     R0,R1       →   R0 = 0C7H, R1 = 02H, Z = "1"
TCM     R0,@R1      →   R0 = 0C7H, R1 = 02H, register 02H = 23H, Z = "0"
TCM     00H,01H     →   Register 00H = 2BH, register 01H = 02H, Z = "1"
TCM     00H,@01H    →   Register 00H = 2BH, register 01H = 02H,
                        register 02H = 23H, Z = "1"
TCM     00H,#34     →   Register 00H = 2BH, Z = "0"
```

In the first example, if the working register R0 contains the value 0C7H (11000111B) and the register R1 the value 02H (00000010B), the statement "TCM R0,R1" tests bit one in the destination register for a "1" value. Because the mask value corresponds to the test bit, the Z flag is set to logic one and can be tested to determine the result of the TCM operation.

TM — Test under Mask

TM dst,src

Operation: dst AND src

This instruction tests selected bits in the destination operand for a logic zero value. The bits to be tested are specified by setting a "1" bit in the corresponding position of the source operand (mask), which is ANDed with the destination operand. The zero (Z) flag can then be checked to determine the result. The destination and the source operands are unaffected.

Flags:

C: Unaffected.

Z: Set if the result is "0"; cleared otherwise.

S: Set if the result bit 7 is set; cleared otherwise.

V: Always reset to "0".

D: Unaffected.

H: Unaffected.

Format:

			Bytes	Cycles	Opcode (Hex)	Addr Mode dst	Addr Mode src
opc	dst | src		2	4	72	r	r
				6	73	r	lr
opc	src	dst	3	6	74	R	R
				6	75	R	IR
opc	dst	src	3	6	76	R	IM

Examples: Given: R0 = 0C7H, R1 = 02H, R2 = 18H, register 00H = 2BH, register 01H = 02H, and register 02H = 23H:

TM	R0,R1	→	R0 = 0C7H, R1 = 02H, Z = "0"
TM	R0,@R1	→	R0 = 0C7H, R1 = 02H, register 02H = 23H, Z = "0"
TM	00H,01H	→	Register 00H = 2BH, register 01H = 02H, Z = "0"
TM	00H,@01H	→	Register 00H = 2BH, register 01H = 02H, register 02H = 23H, Z = "0"
TM	00H,#54H	→	Register 00H = 2BH, Z = "1"

In the first example, if the working register R0 contains the value 0C7H (11000111B) and the register R1 the value 02H (00000010B), the statement "TM R0,R1" tests bit one in the destination register for a "0" value. Because the mask value does not match the test bit, the Z flag is cleared to logic zero and can be tested to determine the result of the TM operation.

WFI — Wait for Interrupt

WFI

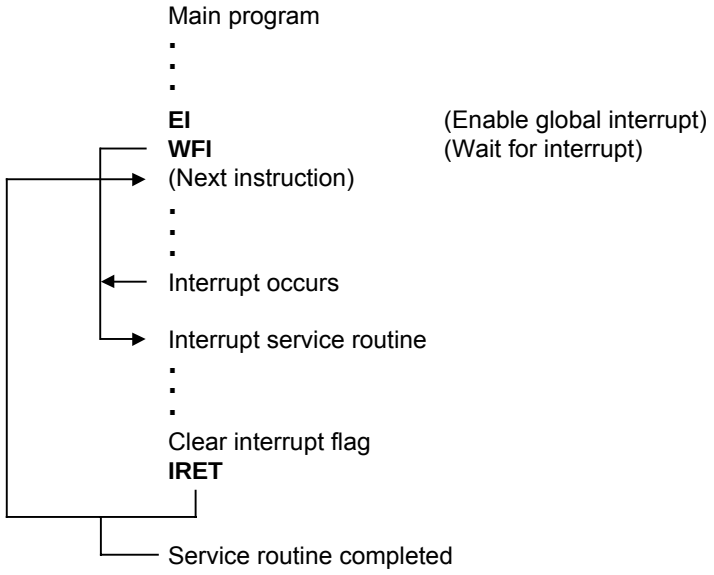
Operation: The CPU is effectively halted before an interrupt occurs, except that DMA transfers can still take place during this wait state. The WFI status can be released by an internal interrupt, including a fast interrupt.

Flags: No flags are affected.

Format:

	Bytes	Cycles	Opcode (Hex)
opc	1	4n (n = 1, 2, 3, …)	3F

Example: The following sample program structure shows the sequence of operations that follow a "WFI" statement:



XOR — Logical Exclusive OR

XOR dst,src

Operation: dst ← dst XOR src

The source operand is logically exclusive-ORed with the destination operand and the result is stored in the destination. The exclusive-OR operation results in a "1" bit being stored whenever the corresponding bits in the operands are different. Otherwise, a "0" bit is stored.

Flags:

C: Unaffected.

Z: Set if the result is "0"; cleared otherwise.

S: Set if the result bit 7 is set; cleared otherwise.

V: Always reset to "0".

D: Unaffected.

H: Unaffected.

Format:

			Bytes	Cycles	Opcode (Hex)	Addr Mode dst	src
opc	dst | src		2	4	B2	r	r
				6	B3	r	lr
opc	src	dst	3	6	B4	R	R
				6	B5	R	IR
opc	dst	src	3	6	B6	R	IM

Examples: Given: R0 = 0C7H, R1 = 02H, R2 = 18H, register 00H = 2BH, register 01H = 02H, and register 02H = 23H:

XOR	R0,R1	→	R0 = 0C5H, R1 = 02H
XOR	R0,@R1	→	R0 = 0E4H, R1 = 02H, register 02H = 23H
XOR	00H,01H	→	Register 00H = 29H, register 01H = 02H
XOR	00H,@01H	→	Register 00H = 08H, register 01H = 02H, register 02H = 23H
XOR	00H,#54H	→	Register 00H = 7FH

In the first example, if the working register R0 contains the value 0C7H and if the register R1 contains the value 02H, the statement "XOR R0,R1" logically exclusive-ORs the R1 value with the R0 value and stores the result (0C5H) in the destination register R0.

NOTES

7 CLOCK CIRCUIT

OVERVIEW

The clock frequency generated for the S3C84BB/F84BB by an external crystal can range from 1 MHz to 12 MHz. The maximum CPU clock frequency is 12 MHz. The X_{IN} and X_{OUT} pins connect the external oscillator or clock source to the on-chip clock circuit.

SYSTEM CLOCK CIRCUIT

The system clock circuit has the following components:

— External crystal or ceramic resonator oscillation source (or an external clock source)
— Oscillator stop and wake-up functions
— Programmable frequency divider for the CPU clock (fxx divided by 1, 2, 8, or 16)
— System clock control register, CLKCON

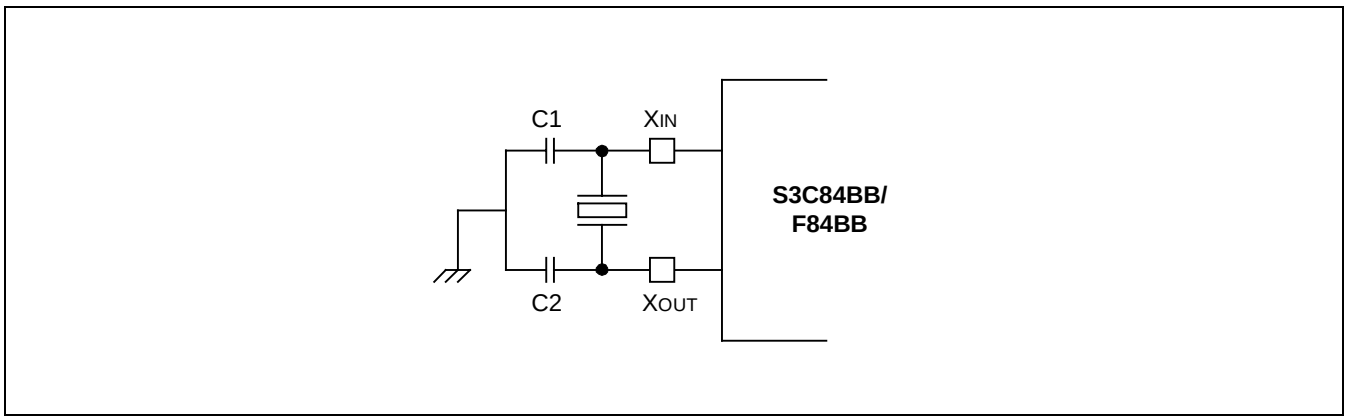


Figure 7-1. Main Oscillator Circuit (Crystal or Ceramic Oscillator)

CLOCK STATUS DURING POWER-DOWN MODES

The two power-down modes, Stop mode and Idle mode, affect the system clock as follows:

— In Stop mode, the main oscillator is halted. Stop mode is released, and the oscillator started, by a reset operation or an external interrupt (with RC delay noise filter), and can be released by internal interrupt too when the sub-system oscillator is running and watch timer is operating with sub-system clock.

— In Idle mode, the internal clock signal is gated to the CPU, but not to interrupt structure, timers and timer/counters. Idle mode is released by a reset or by an external or internal interrupt.

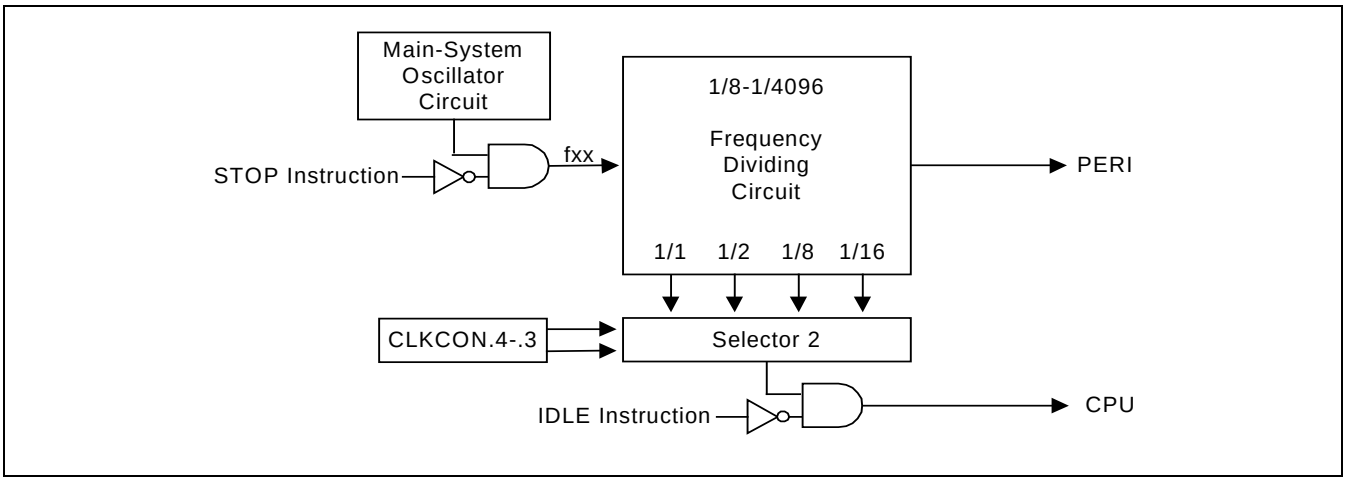


Figure 7-2. System Clock Circuit Diagram

SYSTEM CLOCK CONTROL REGISTER (CLKCON)

The system clock control register, CLKCON, is located in the bank 0 of set 1, address D4H. It is read/write addressable and has the following functions:

— Oscillator frequency divide-by value

After the main oscillator is activated, and the fxx/16 (the slowest clock speed) is selected as the CPU clock. If necessary, you can then increase the CPU clock speed to fxx/8, fxx/2, or fxx/1.

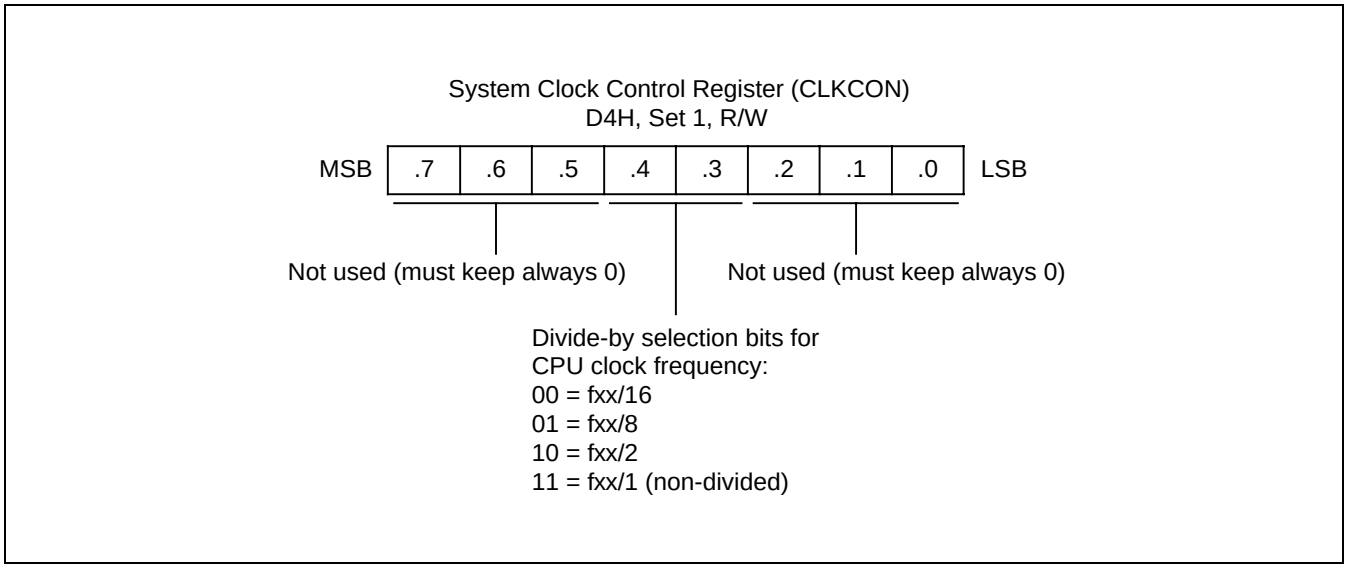


Figure 7-3. System Clock Control Register (CLKCON)

NOTES

8 RESET and POWER-DOWN

SYSTEM RESET

OVERVIEW

During a power-on reset, the voltage at V_{DD} goes to High level and the RESET pin is forced to Low level. The RESET signal is input through a Schmitt trigger circuit where it is then synchronized with the CPU clock. This procedure brings S3C84BB/F84BB into a known operating status.

To allow time for internal CPU clock oscillation to stabilize, the RESET pin must be held to Low level for a minimum time interval after the power supply comes within tolerance. The minimum required oscillation stabilization time for a reset operation is 1 millisecond.

Whenever a reset occurs during normal operation (that is, when both V_{DD} and RESET are High level), the RESET pin is forced Low and the reset operation starts. All system and peripheral control registers are then reset to their default hardware values.

In summary, the following sequence of events occurs during a reset operation:

— Interrupt is disabled.

— The watchdog function (basic timer) is enabled.

— Ports 0-8 are set to input mode.

— Peripheral control and data registers are disabled and reset to their default hardware values.

— The program counter (PC) is loaded with the program reset address in the ROM, 0100H.

— When the programmed oscillation stabilization time interval has elapsed, the instruction stored in ROM location 0100H (and 0101H) is fetched and executed.

NORMAL MODE RESET OPERATION

In normal (masked ROM) mode, the Test pin is tied to V_{SS}. A reset enables access to the 64-Kbyte on-chip ROM.

NOTE

To program the duration of the oscillation stabilization interval, you make the appropriate settings to the basic timer control register, BTCON, *before* entering Stop mode. Also, if you do not want to use the basic timer watchdog function (which causes a system reset if a basic timer counter overflow occurs), you can disable it by writing '1010B' to the upper nibble of BTCON.

HARDWARE RESET VALUES

Table 8-1, 8-2, 8-3 list the reset values for CPU and system registers, peripheral control registers, and peripheral data registers following a reset operation. The following notation is used to represent reset values:

— A "1" or a "0" shows the reset bit value as logic one or logic zero, respectively.

— An "x" means that the bit value is undefined after a reset.

— A dash ("–") means that the bit is either not used or not mapped, but read 0 is the bit value.

Table 8-1. S3C84BB/F84BB Set 1 Register Values after RESET

Register Name	Mnemonic	Address		Bit Values After RESET							
		Dec	Hex	7	6	5	4	3	2	1	0
Timer B control register	TBCON	208	D0H	0	0	0	0	0	0	0	0
Timer B data register (high byte)	TBDATAH	209	D1H	1	1	1	1	1	1	1	1
Timer B data register (low byte)	TBDATAL	210	D2H	1	1	1	1	1	1	1	1
Basic timer control register	BTCON	211	D3H	0	0	0	0	0	0	0	0
Clock Control register	CLKCON	212	D4H	0	0	0	0	0	0	0	0
System flags register	FLAGS	213	D5H	x	x	x	x	x	x	0	0
Register pointer 0	RP0	214	D6H	1	1	0	0	0	–	–	–
Register pointer 1	RP1	215	D7H	1	1	0	0	1	–	–	–
Stack pointer (high byte)	SPH	216	D8H	x	x	x	x	x	x	x	x
Stack pointer (low byte)	SPL	217	D9H	x	x	x	x	x	x	x	x
Instruction pointer (high byte)	IPH	218	DAH	x	x	x	x	x	x	x	x
Instruction pointer (low byte)	IPL	219	DBH	x	x	x	x	x	x	x	x
Interrupt request register	IRQ	220	DCH	0	0	0	0	0	0	0	0
Interrupt mask register	IMR	221	DDH	x	x	x	x	x	x	x	x
System mode register	SYM	222	DEH	0	–	–	x	x	x	0	0
Register page pointer	PP	223	DFH	0	0	0	0	0	0	0	0

Table 8-2. S3C84BB/F84BB Set 1, Bank 0 Register Values after RESET

Register Name	Mnemonic	Address		Bit Values After Reset							
		Dec	Hex	7	6	5	4	3	2	1	0
Port 0 data register	P0	224	E0H	0	0	0	0	0	0	0	0
Port 1 data register	P1	225	E1H	0	0	0	0	0	0	0	0
Port 2 data register	P2	226	E2H	0	0	0	0	0	0	0	0
Port 3 data register	P3	227	E3H	0	0	0	0	0	0	0	0
Port 4 data register	P4	228	E4H	0	0	0	0	0	0	0	0
Port 5 data register	P5	229	E5H	0	0	0	0	0	0	0	0
Port 6 data register	P6	230	E6H	0	0	0	0	0	0	0	0
Port 7 data register	P7	231	E7H	0	0	0	0	0	0	0	0
Port 8 data register	P8	232	E8H	0	0	0	0	0	0	0	0
Timer A/1 interrupt pending register	TINTPND	233	E9H	–	–	0	0	0	0	0	0
Timer A control register	TACON	234	EAH	0	0	0	0	0	0	0	–
Timer A data register	TADATA	235	EBH	1	1	1	1	1	1	1	1
Timer A counter register	TACNT	236	ECH	0	0	0	0	0	0	0	0
Port 8 control register (high byte)	P8CONH	237	EDH	1	1	1	1	0	0	0	0
Port 8 control register (low byte)	P8CONL	238	EEH	0	0	0	0	0	0	0	0
Port 8 interrupt/pending register	P8INTPND	239	EFH	1	1	0	0	1	1	0	0
Port 0 control register	P0CON	240	F0H	0	0	0	0	0	0	0	0
Port 1 control register	P1CON	241	F1H	0	0	0	0	0	0	0	0
Port 2 control register (high byte)	P2CONH	242	F2H	0	0	0	0	0	0	0	0
Port 2 control register (low byte)	P2CONL	243	F3H	0	0	0	0	0	0	0	0
Port 3 control register (high byte)	P3CONH	244	F4H	0	0	0	0	0	0	0	0
Port 3 control register (low byte)	P3CONL	245	F5H	0	0	0	0	0	0	0	0
Port 4 control register (high byte)	P4CONH	246	F6H	0	0	0	0	0	0	0	0
Port 4 control register (low byte)	P4CONL	247	F7H	0	0	0	0	0	0	0	0
Port 5 control register (high byte)	P5CONH	248	F8H	0	0	0	0	0	0	0	0
Port 5 control register (low byte)	P5CONL	249	F9H	0	0	0	0	0	0	0	0
Port 4 interrupt control register	P4INT	250	FAH	0	0	0	0	0	0	0	0
Port 4 interrupt/pending register	P4INTPND	251	FBH	0	0	0	0	0	0	0	0
Location FCH is factory use only											
Basic timer counter register	BTCNT	253	FDH	0	0	0	0	0	0	0	0
Location FEH is not mapped											
Interrupt priority register	IPR	255	FFH	x	x	x	x	x	x	x	x

Table 8-3. S3C84BB/F84BB Set 1, Bank 1 Register Values after RESET

Register Name	Mnemonic	Address		Bit Values After Reset							
		Dec	Hex	7	6	5	4	3	2	1	0
SIO data register	SIODATA	224	E0H	0	0	0	0	0	0	0	0
SIO Control register	SIOCON	225	E1H	0	0	0	0	0	0	0	0
UART0 data register	UDATA0	226	E2H	1	1	1	1	1	1	1	1
UART0 control register	UARTCON0	227	E3H	0	0	0	0	0	0	0	0
UART0 baud rate data register	BRDATA0	228	E4H	1	1	1	1	1	1	1	1
UART0,1 pending register	UARTPND	229	E5H	-	-	-	-	0	0	0	0
Timer 1(0) data register (high byte)	T1DATAH0	230	E6H	1	1	1	1	1	1	1	1
Timer 1(0) data register (low byte)	T1DATAL0	231	E7H	1	1	1	1	1	1	1	1
Timer 1(1) data register (high byte)	T1DATAH1	232	E8H	1	1	1	1	1	1	1	1
Timer 1(1) data register (low byte)	T1DATAL1	233	E9H	1	1	1	1	1	1	1	1
Timer 1(0) control register	T1CON0	234	EAH	0	0	0	0	0	0	0	0
Timer 1(1) control register	T1CON1	235	EBH	0	0	0	0	0	0	0	0
Timer 1(0) counter register(high byte)	T1CNTH0	236	ECH	0	0	0	0	0	0	0	0
Timer 1(0) counter register(low byte)	T1CNTL0	237	EDH	0	0	0	0	0	0	0	0
Timer 1(1) counter register(high byte)	T1CNTH1	238	EEH	0	0	0	0	0	0	0	0
Timer 1(1) counter register(low byte)	T1CNTL1	239	EFH	0	0	0	0	0	0	0	0
Timer C(0) data register	TCDATA0	240	F0H	1	1	1	1	1	1	1	1
Timer C(1) data register	TCDATA1	241	F1H	1	1	1	1	1	1	1	1
Timer C(0) control register	TCCON0	242	F2H	0	0	0	0	0	0	0	0
Timer C(1) control register	TCCON1	243	F3H	0	0	0	0	0	0	0	0
SIO prescaler control register	SIOPS	244	F4H	0	0	0	0	0	0	0	0
Port 7 control register	P7CON	245	F5H	0	0	0	0	0	0	0	0
D/A converter data register	DADATA	246	F6H	0	0	0	0	0	0	0	0
A/D, D/A converter control register	ADACON	247	F7H	0	0	0	0	0	0	0	0
A/D converter data register(high byte)	ADDATAH	248	F8H	0	0	0	0	0	0	0	0
A/D converter data register(low byte)	ADDATAL	249	F9H	0	0	0	0	0	0	0	0
UART1 data register	UDATA1	250	FAH	1	1	1	1	1	1	1	1
UART1 control register	UARTCON1	251	FBH	0	0	0	0	0	0	0	0
UART1 baud rate data register	BRDATA1	252	FCH	1	1	1	1	1	1	1	1
Flash memory control register	FMCON	253	FDH	0	0	0	0	0	0	0	0
Pattern generation control register	PGCON	254	FEH	–	–	–	–	0	0	0	0
Pattern generation data register	PGDATA	255	FFH	0	0	0	0	0	0	0	0

POWER-DOWN MODES

STOP MODE

Stop mode is invoked by the instruction STOP (opcode 7FH). In Stop mode, the operation of the CPU and all peripherals is halted. That is, the on-chip main oscillator stops and the supply current is reduced to less than 3 μA. All system functions stop when the clock "freezes," but data stored in the internal register file is retained. Stop mode can be released in one of two ways: by a reset or by interrupts.

NOTE

Do not use stop mode if you are using an external clock source because X_{IN} input must be restricted internally to V_{SS} to reduce current leakage.

Using RESET to Release Stop Mode

Stop mode is released when the RESET signal is released and returns to high level: all system and peripheral control registers are reset to their default hardware values and the contents of all data registers are retained. A reset operation automatically selects a slow clock (1/16) because CLKCON.3 and CLKCON.4 are cleared to '00B'. After the programmed oscillation stabilization interval has elapsed, the CPU starts the system initialization routine by fetching the program instruction stored in ROM location 0100H (and 0101H).

Using an External Interrupt to Release Stop Mode

External interrupts with an RC-delay noise filter circuit can be used to release Stop mode. Which interrupt you can use to release Stop mode in a given situation depends on the microcontroller's current internal operating mode. The external interrupts in the S3F84BB interrupt structure that can be used to release Stop mode are:

— External interrupts P4.0/INT0-P4.7/INT7, P8.4/INT8 and P8.5/INT9

Please note the following conditions for Stop mode release:

— If you release Stop mode using an external interrupt, the current values in system and peripheral control registers are unchanged.

— If you use an external interrupt for Stop mode release, you can also program the duration of the oscillation stabilization interval. To do this, you must make the appropriate control and clock settings *before* entering Stop mode.

— When the Stop mode is released by external interrupt, the CLKCON.4 and CLKCON.3 bit-pair setting remains unchanged and the currently selected clock value is used.

— The external interrupt is serviced when the Stop mode release occurs. Following the IRET from the service routine, the instruction immediately following the one that initiated Stop mode is executed.

Using an internal Interrupt to Release Stop Mode

Activate any enabled interrupt, causing stop mode to be released. Other things are same as using external interrupt.

IDLE MODE

Idle mode is invoked by the instruction IDLE (opcode 6FH). In idle mode, CPU operations are halted while some peripherals remain active. During idle mode, the internal clock signal is gated away from the CPU, but all peripherals timers remain active. Port pins retain the mode (input or output) they had at the time idle mode was entered.

There are two ways to release idle mode:

1. Execute a reset. All system and peripheral control registers are reset to their default values and the contents of all data registers are retained. The reset automatically selects the slow clock fxx/16 because CLKCON.4 and CLKCON.3 are cleared to '00B'. If interrupts are masked, a reset is the only way to release idle mode.
2. Activate any enabled interrupt, causing idle mode to be released. When you use an interrupt to release idle mode, the CLKCON.4 and CLKCON.3 register values remain unchanged, and the currently selected clock value is used. The interrupt is then serviced. When the return-from-interrupt (IRET) occurs, the instruction immediately following the one that initiated idle mode is executed.

9 I/O PORTS

OVERVIEW

The S3C84BB/F84BB microcontroller has nine bit-programmable I/O ports, P0-P8. The port 8 are 6-bit ports and the others are 8-bit ports. This gives a total of 70 I/O pins. Each port can be flexibly configured to meet application design requirements. The CPU accesses ports by directly writing or reading port registers. No special I/O instructions are required.

Table 9-1 gives you a general overview of the S3C84BB/F84BB I/O port functions.

Table 9-1. S3C84BB/F84BB Port Configuration Overview

Port	Configuration Options
0	Bit programmable port; input or output mode selected by software; input or push-pull output. Software assignable pull-up. Alternately, P0.0-P0.7 can be used as the PG output port (PG0-PG7).
1	Bit programmable port; input or output mode selected by software; input or push-pull output. Software assignable pull-up.
2	Bit programmable port; input or output mode selected by software; input or push-pull output. Software assignable pull-up. Alternately, P2.0~P2.7 can be used as I/O for TIMERA, TIMERB, DAC, SIO
3	Bit programmable port; input or output mode selected by software; input or push-pull output. Software assignable pull-up. Alternately, P3.0~P3.7 can be used as I/O for TIMERC0/C1, TIMER10/11
4	Bit programmable port; input or output mode selected by software; input or push-pull output. Software assignable pull-up. P4.0-P4.7 can alternately be used as inputs for external interrupts INT0-INT7, respectively (with noise filters and interrupt controller)
5	Bit programmable port; input or output mode selected by software; input or push-pull output. Software assignable pull-up. Alternately, P5.0~P5.3 can be used as I/O for serial port UART0, UART1, respectively.
6	N-channel, open-drain output only port.
7	General-purpose digital input ports. Alternatively used as analog input pins for A/D converter modules.
8	Bit programmable port; input or output mode selected by software; input or push-pull output. Software assignable pull-up. P8.4, P8.5 can alternately be used as inputs for external interrupts INT8, INT9, respectively (with noise filters and interrupt controller)

PORT DATA REGISTERS

Table 9-2 gives you an overview of the register locations of all five S3C84BB/F84BB I/O port data registers. Data registers for ports 0, 1, 2, 3, 4, 5, 6, 7 and 8 have the general format shown in Table 9-2.

Table 9-2. Port Data Register Summary

Register Name	Mnemonic	Decimal	Hex	Location	R/W
Port 0 data register	P0	224	E0H	Set 1, Bank 0	R/W
Port 1 data register	P1	225	E1H	Set 1, Bank 0	R/W
Port 2 data register	P2	226	E2H	Set 1, Bank 0	R/W
Port 3 data register	P3	227	E3H	Set 1, Bank 0	R/W
Port 4 data register	P4	228	E4H	Set 1, Bank 0	R/W
Port 5 data register	P5	229	E5H	Set 1, Bank 0	R/W
Port 6 data register	P6	230	E6H	Set 1, Bank 0	R/W
Port 7 data register	P7	231	E7H	Set 1, Bank 0	R/W
Port 8 data register	P8	232	E8H	Set 1, Bank 0	R/W

PORT 0

Port 0 is an 8-bit I/O Port that you can use two ways:

— General-purpose I/O

— Alternative function: PGOUT7-PGOUT0

Port 0 is accessed directly by writing or reading the port 0 data register, P0 at location E0H in set 1, bank 0.

Port 0 Control Register (P0CON)

Port 0 pins are configured individually by bit-pair settings in one control registers located in set 1, bank 0: P0CON (F0H).

When programming the port, please remember that any alternative peripheral I/O function you configure using the port 0 control registers must also be enabled in the associated peripheral module.

Port 0 Control Register (P0CON)
F0H, Set 1, Bank 0, R/W

MSB	.7	.6	.5	.4	.3	.2	.1	.0	LSB

P0.7/P0.6/ P0.5/P0.4/ PGOUT[7:4] | P0.3/P0.2/ PGOUT[3:2] | P0.1/ PGOUT[1] | P0.0/ PGOUT[0]

.7 .6 bit/P0.7/P0.6/P0.5/P0.4

00	Input mode
01	Input mode, pull-up
10	Push-pull output
11	Alternative function mode(PGOUT[7:4])

.5 .4 bit/P0.3/P0.2

00	Input mode
01	Input mode, pull-up
10	Push-pull output
11	Alternative function mode(PGOUT[3:2])

.3 .2 bit/P0.1

00	Input mode
01	Input mode, pull-up
10	Push-pull output
11	Alternative function mode(PGOUT[1])

.1 .0 bit/P0.0

00	Input mode
01	Input mode, pull-up
10	Push-pull output
11	Alternative function mode(PGOUT[0])

Figure 9-1. Port 0 Control Register (P0CON)

PORT 1

Port 1 is an 8-bit I/O Port that you can use one ways:

— General-purpose I/O

Port 1 is accessed directly by writing or reading the port 1 data register, P1 at location E1H in set 1, bank 0.

Port 1 Control Register (P1CON)

Port 1 pins are configured individually by bit-pair settings in one control registers located in set 1, bank 0: P1CON (F1H).

When programming the port, please remember that any alternative peripheral I/O function you configure using the port 1 control registers must also be enabled in the associated peripheral module.

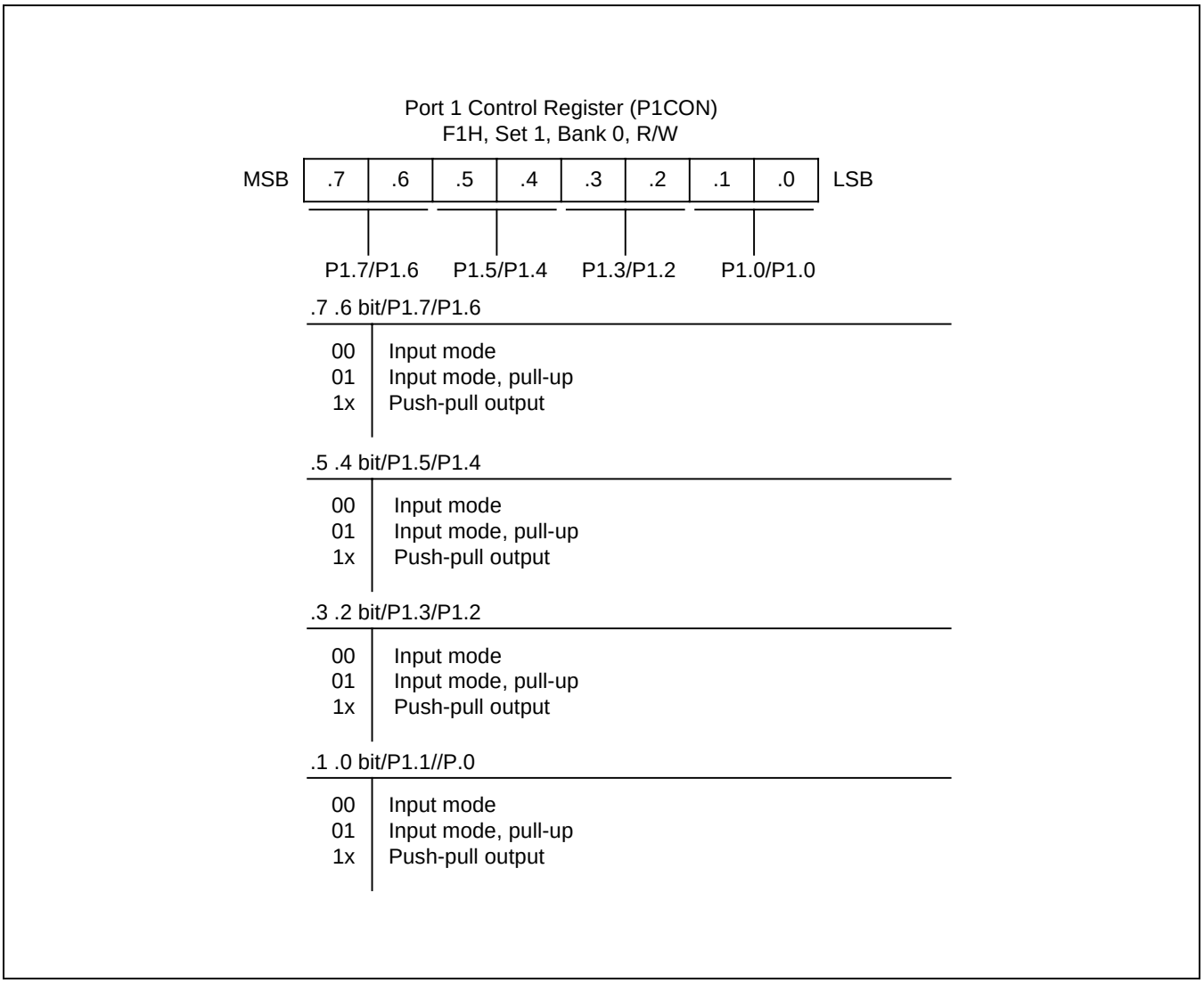


Figure 9-2. Port 1 Control Register (P1CON)

PORT 2

Port 2 is an 8-bit I/O port with individually configurable pins. Port 2 pins are accessed directly by writing or reading the port 2 data register, P2 at location E2H in set 1, bank 0. P2.0–P2.7 can serve as inputs, outputs (push pull) or you can configure the following alternative functions:

— Low-byte pins (P2.0-P2.3): DAOUT, SCK, SI, SO

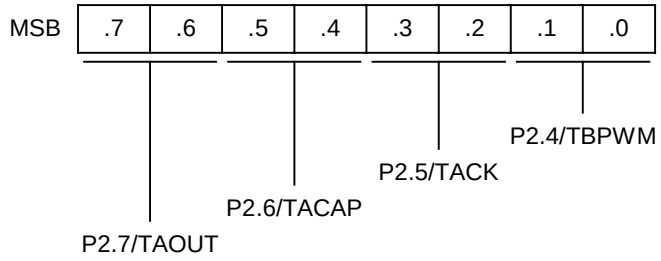
— High-byte pins (P2.4-P2.7): TAOUT, TACAP, TACK, TBPWM

Port 2 Control Register (P2CONH, P2CONL)

Port 2 has two 8-bit control registers: P2CONH for P2.4–P2.7 and P2CONL for P2.0–P2.3. A reset clears the P2CONH and P2CONL registers to “00H”, configuring all pins to input mode. You use control registers settings to select input or output mode (push-pull) and enable the alternative functions.

When programming the port, please remember that any alternative peripheral I/O function you configure using the port 2 control registers must also be enabled in the associated peripheral module.

Port 2 Control Register, High Byte (P2CONH)
F2H, Set 1, Bank 0, R/W



.7 .6 bit/P2.7/TAOUT

00	Input mode
01	Input mode, pull-up
10	Push-pull output
11	Alternative output mode(TAOUT)

.5 .4 bit/P2.6/TACAP

00	Input mode(TACAP)
01	Input mode, pull-up(TACAP)
10	Push-pull output
11	Alternative output mode(Not used)

.3 .2 bit/P2.5/TACK

00	Input mode(TACK)
01	Input mode, pull-up(TACK)
10	Push-pull output
11	Alternative output mode(Not used)

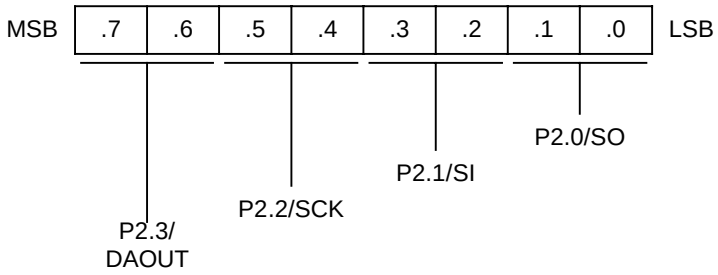
.1 .0 bit/P2.4/TBPWM

00	Input mode
01	Input mode, pull-up
10	Push-pull output
11	Alternative output mode(TBPWM)

NOTE: When use this port 2, user must be care of the pull-up resistance status.

Figure 9-3. Port 2 High-Byte Control Register (P2CONH)

Port 2 Control Register, Low Byte (P2CONL)
F3H, Set 1, Bank 0, R/W



.7 .6 bit/P2.3/DAOUT

00	Input mode
01	Input mode, pull-up
10	Push-pull output
11	Alternative output mode(DAOUT)

.5 .4 bit/P2.2/SCK

00	Input mode(SCK input)
01	Input mode, pull-up(SCK input)
10	Push-pull output
11	Alternative output mode(SCK output)

.3 .2 bit/P2.1/SI

00	Input mode(SI)
01	Input mode, pull-up(SI)
10	Push-pull output
11	Alternative output mode(Not used)

.1 .0 bit/P2.0/SO

00	Input mode
01	Input mode, pull-up
10	Push-pull output
11	Alternative output mode(SO)

NOTE: When use this port 2, user must be care of the pull-up resistance status.

Figure 9-4. Port 2 Low-Byte Control Register (P2CONL)

PORT 3

Port 3 is an 8-bit I/O port that can be used for general-purpose I/O. The pins are accessed directly by writing or reading the port 3 data register, P3 at location E3H in set 1, bank 0. P3.7–P3.0 can serve as inputs, outputs (push pull) or you can configure the following alternative functions:

— Low-byte pins (P3.0-P3.3): T1CAP1, T1CAP0, T1CK1, T1CK0

— High-byte pins (P3.4-P3.7): TCOUT1, TCOUT0, T1OUT1, T1OUT0

To individually configure the port 3 pins P3.0–P3.7, you make bit-pair settings in two control registers located in set 1, bank 0: P3CONL (low byte, F5H) and P3CONH (high byte, F4H).

Port 3 Control Registers (P3CONH, P3CONL)

Two 8-bit control registers are used to configure port 3 pins: P3CONL (F5H, set 1, Bank 0) for pins P3.0–P3.3 and P3CONH (F4H, set 1, Bank 0) for pins P3.4–P3.7. Each byte contains four bit-pairs and each bit-pair configures one pin of port 3.

Port 3 Control Register, High Byte (P3CONH)
F4H, Set 1, Bank 0, R/W

MSB	.7	.6	.5	.4	.3	.2	.1	.0	LSB

P3.7/TCOUT1
P3.6/TCOUT0
P3.5/T1OUT1
P3.4/T1OUT0

.7 .6 bit : P3.7/TCOUT1

00	Input mode
01	Input mode, pull-up
10	Push-pull output
11	Alternative function(TCOUT1)

.5 .4 bit : P3.6/TCOUT0

00	Input mode
01	Input mode, pull-up
10	Push-pull output
11	Alternative function(TCOUT0)

.3 .2 bit : P3.5/T1OUT1

00	Input mode
01	Input mode, pull-up
10	Push-pull outputt
11	Alternative function(T1OUT1)

.1 .0 bit : P3.4/T1OUT0

00	Input mode
01	Input mode, pull-up
10	Push-pull outputt
11	Alternative function(T1OUT0)

Figure 9-5. Port 3 High-Byte Control Register (P3CONH)

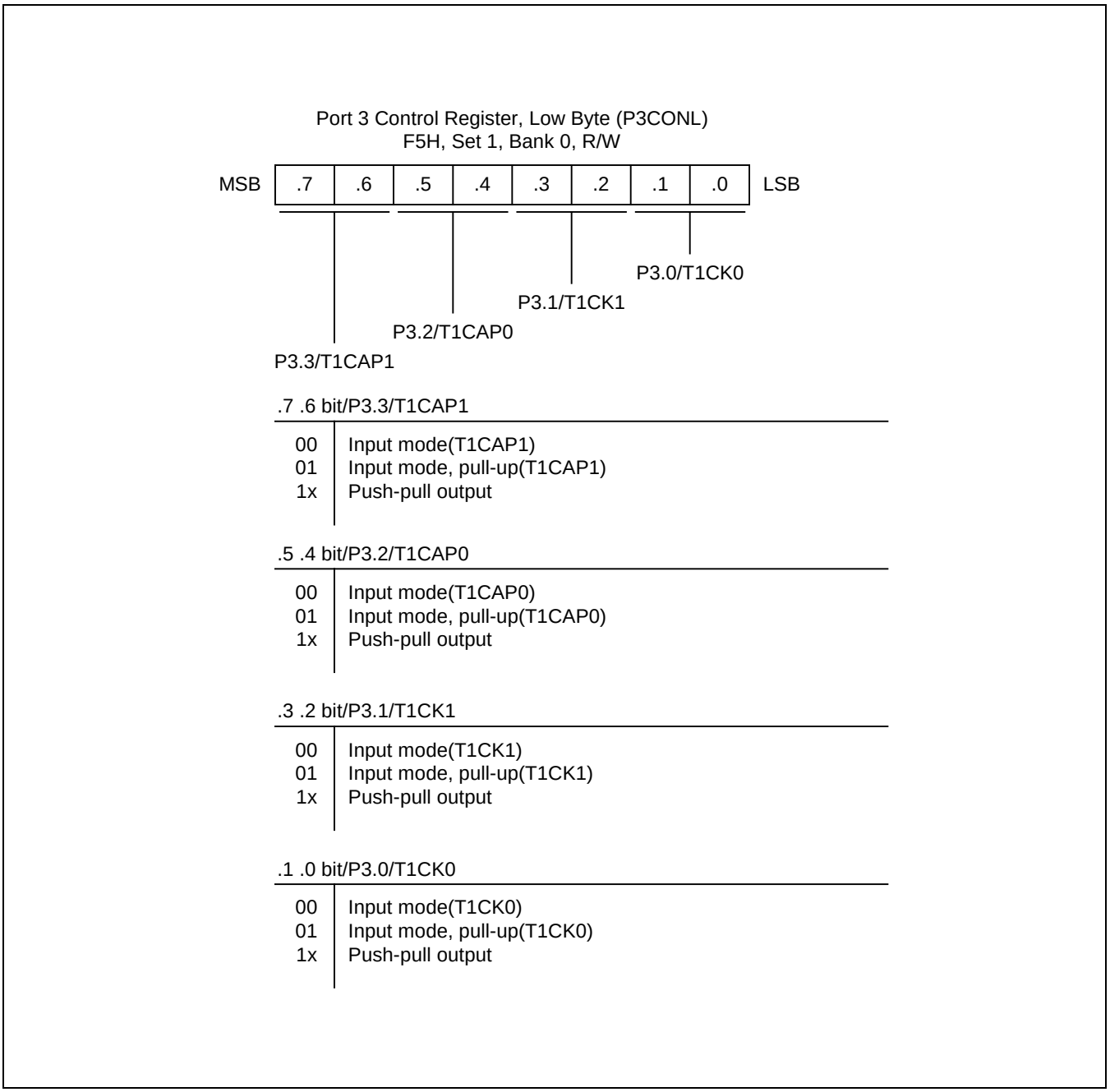


Figure 9-6. Port 3 Low-Byte Control Register (P3CONL)

PORT 4

Port 4 is an 8-bit I/O Port that you can use two ways:

— General-purpose I/O

— External interrupt inputs for INT0-INT7

Port 4 is accessed directly by writing or reading the port 4 data register, P4 at location E4H in set 1, bank 0.

Port 4 Control Register (P4CONH, P4CONL)

Port 4 pins are configured individually by bit-pair settings in two control registers located in set 1, bank 0: P4CONL (low byte, F7H) and P4CONH (high byte, F6H).

When you select output mode, a push-pull circuit is configured. In input mode, three different selections are available:

— Schmitt trigger input with interrupt generation on falling signal edges.

— Schmitt trigger input with interrupt generation on rising signal edges.

— Schmitt trigger input with pull-up resistor and interrupt generation on falling signal edges.

Port 4 Interrupt Enable and Pending Registers (P4INT, P4INTPND)

To process external interrupts at the port 4 pins, two additional control registers are provided: the port 4 interrupt enable register P4INT (FAH, set 1, bank 0) and the port 4 interrupt pending register P4INTPND (FBH, set 1, bank 0).

The port 4 interrupt pending register P4INTPND lets you check for interrupt pending conditions and clear the pending condition when the interrupt service routine has been initiated. The application program detects interrupt requests by polling the P4INTPND register at regular intervals.

When the interrupt enable bit of any port 4 pin is “1”, a rising or falling signal edge at that pin will generate an interrupt request. The corresponding P4INTPND bit is then automatically set to “1” and the IRQ level goes low to signal the CPU that an interrupt request is waiting. When the CPU acknowledges the interrupt request, application software must clear the pending condition by writing a “0” to the corresponding P4INTPND bit.

Port 4 Control Register, High Byte (P4CONH)
F6H, Set 1, Bank 0, R/W

MSB	.7	.6	.5	.4	.3	.2	.1	.0	LSB
	P4.7/ INT7		P4.6/ INT6		P4.5/ INT5		P4.4/ INT4		

.7 .6 bit/P4.7INT7

00	Input mode; (falling edge interrupt)
01	Input mode; (rising edge interrupt)
10	Input mode, pull-up; (falling edge interrupt)
11	Push-pull output

.5 .4 bit/P4.6/INT6

00	Input mode; (falling edge interrupt)
01	Input mode; (rising edge interrupt)
10	Input mode, pull-up; (falling edge interrupt)
11	Push-pull output

.3 .2 bit/P4.5/INT5

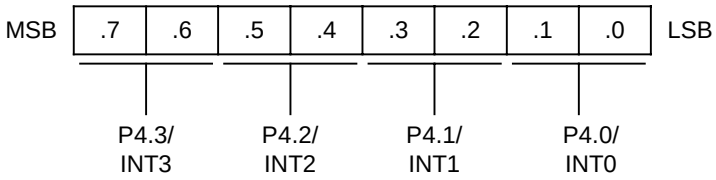
00	Input mode; (falling edge interrupt)
01	Input mode; (rising edge interrupt)
10	Input mode, pull-up; (falling edge interrupt)
11	Push-pull output

.1 .0 bit/P4.4/INT4

00	Input mode; (falling edge interrupt)
01	Input mode; (rising edge interrupt)
10	Input mode, pull-up; (falling edge interrupt)
11	Push-pull output

Figure 9-7. Port 4 High-Byte Control Register (P4CONH)

Port 4 Control Register, Low Byte (P4CONL)
F7H, Set 1, Bank 0, R/W



.7 .6 bit/P4.3/INT3

00	Input mode; (falling edge interrupt)
01	Input mode; (rising edge interrupt)
10	Input mode, pull-up; (falling edge interrupt)
11	Push-pull output

.5 .4 bit/P4.2/INT2

00	Input mode; (falling edge interrupt)
01	Input mode; (rising edge interrupt)
10	Input mode, pull-up; (falling edge interrupt)
11	Push-pull output

.3 .2 bit/P4.1/INT1

00	Input mode; (falling edge interrupt)
01	Input mode; (rising edge interrupt)
10	Input mode, pull-up; (falling edge interrupt)
11	Push-pull output

.1 .0 bit/P4.0/INT0

00	Input mode; (falling edge interrupt)
01	Input mode; (rising edge interrupt)
10	Input mode, pull-up; (falling edge interrupt)
11	Push-pull output

Figure 9-8. Port 4 Low-Byte Control Register (P4CONL)

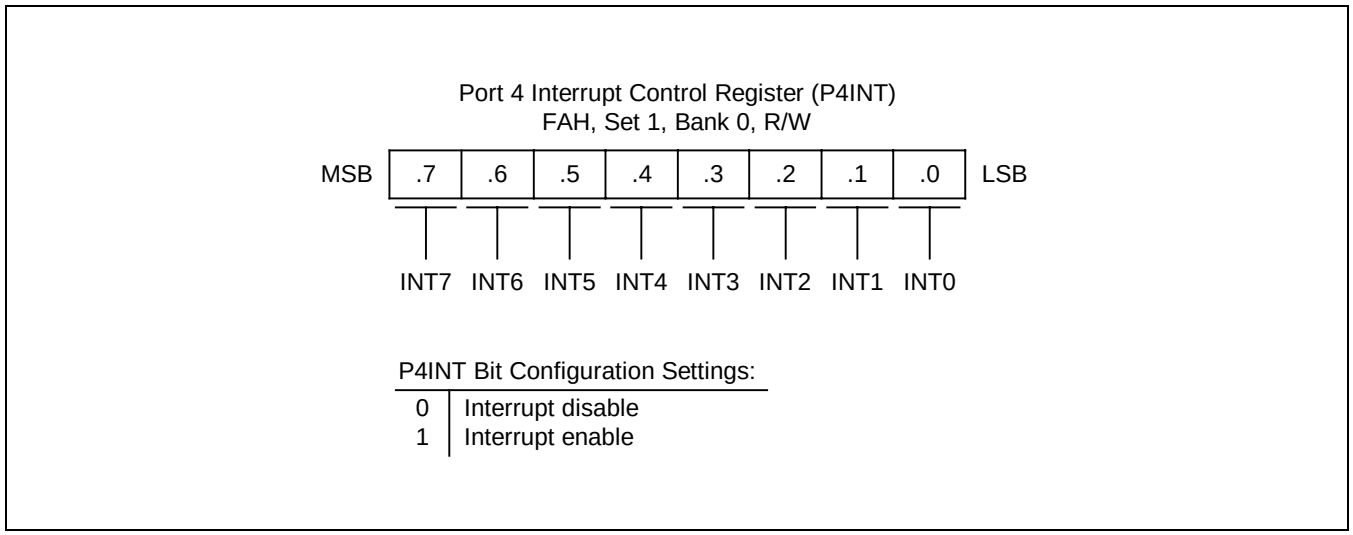


Figure 9-9. Port 4 Interrupt Control Register (P4INT)

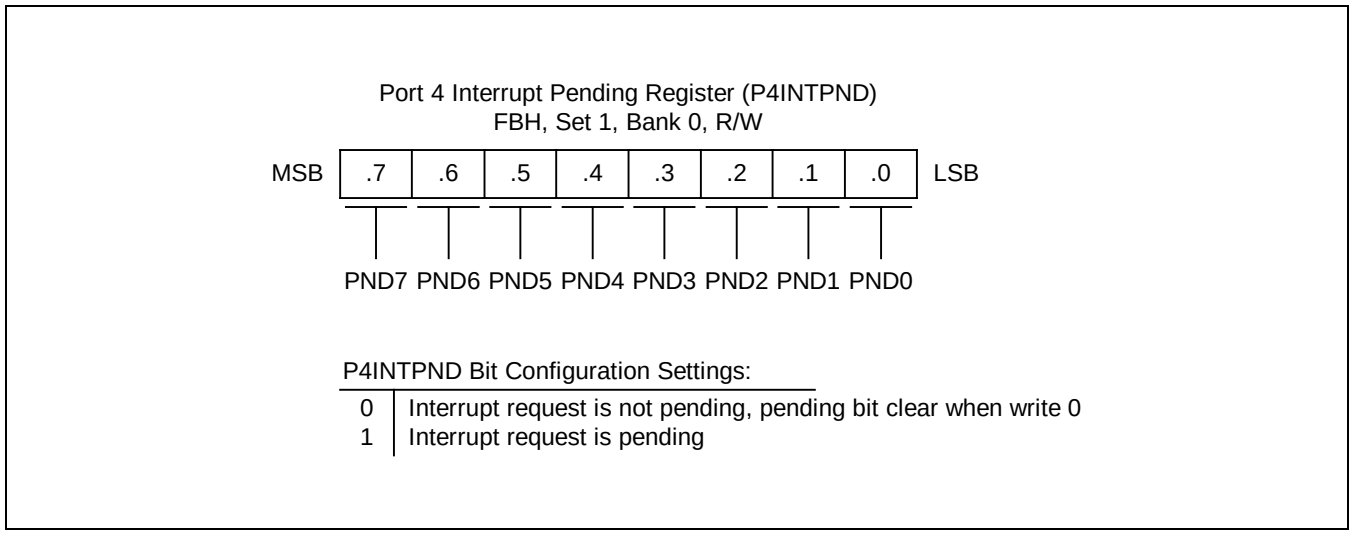


Figure 9-10. Port 4 Interrupt Pending Register (P4INTPND)

PORT 5

Port 5 is an 8-bit I/O port with individually configurable pins. Port 5 pins are accessed directly by writing or reading the port 5 data register, P5 at location E5H in set 1, bank 0. P5.7–P5.4 can serve as inputs, outputs (push pull or open-drain). P5.3–P5.0 can serve as inputs, outputs (push pull) or you can configure the following alternative functions:

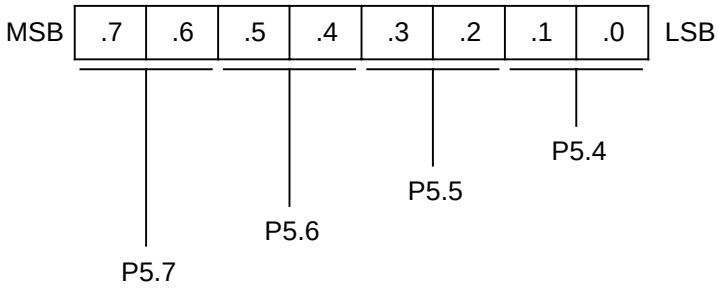
— Low-byte pins (P5.3-P5.0): RxD0, TxD0, RxD1, TxD1

Port 5 Control Register (P5CONH, P5CONL)

Port 5 has two 8-bit control registers: P5CONH for P5.4–P5.7 and P5CONL for P5.0–P5.3. A reset clears the P5CONH and P5CONL registers to “00H”, configuring all pins to input mode. You use control registers settings to select input or output mode (push-pull, open-drain) and enable the alternative functions.

When programming the port, please remember that any alternative peripheral I/O function you configure using the port 5 control registers must also be enabled in the associated peripheral module.

Port 5 Control Register, High Byte (P5CONH)
F8H, Set 1, Bank 0, R/W



.7 .6 bit : P5.7

00	Input mode
01	Input mode, pull-up
10	Push-pull output
11	Open-drain mode

.5 .4 bit : P5.6

00	Input mode
01	Input mode, pull-up
10	Push-pull output
11	Open-drain mode

.3 .2 bit : P5.5

00	Input mode
01	Input mode, pull-up
10	Push-pull output
11	Open-drain mode

.1 .0 bit : P5.4

00	Input mode
01	Input mode, pull-up
10	Push-pull output
11	Open-drain mode

Figure 9-11. Port 5 High-Byte Control Register (P5CONH)

Port 5 Control Register, Low Byte (P5CONL)
F9H, Set 1, Bank 0, R/W

MSB	.7	.6	.5	.4	.3	.2	.1	.0	LSB

P5.3/ RxD0 — P5.2/ TxD0 — P5.1/ RxD1 — P5.0/ TxD1

.7 .6 bit/P5.3/RxD0

00	Input mode(RxD0 input)
01	Input mode, pull-up(RxD0 input)
10	Push-pull output
11	Alternative output mode(RxD0 output)

.5 .4 bit/P5.2/TxD0

00	Input mode
01	Input mode, pull-up
10	Push-pull output
11	Alternative output mode(TxD0 output)

.3 .2 bit/P5.1/RxD1

00	Input mode(RxD1 input)
01	Input mode, pull-up(RxD1 input)
10	Push-pull output
11	Alternative output mode(RxD1 output)

.1 .0 bit/P5.0/TxD1

00	Input mode
01	Input mode, pull-up
10	Push-pull output
11	Alternative output mode(TxD1 output)

Figure 9-12. Port 5 Low-Byte Control Register (P5CONL)

PORT 6

Port 6 is an 8-bit open drain output only port pins. Port 6 pins are accessed directly by writing the port6 data register, P6 at location E6H in set 1, bank 0.

PORT 7

Port 7 is an 8-bit Input port that you can use two ways:

— General-purpose Input

— Alternative function: ADC0-ADC7 input

Port 7 is accessed directly by reading the port 7 data register, P7 at location E7H in set 1, bank 0.

Port 7 Control Register (P7CON)

Port 7 pins are configured individually by bit-pair settings in one control registers located in set 1, bank 1: P7CON (F5H).

When programming the port, please remember that any alternative peripheral I function you configure using the port 7 control registers must also be enabled in the associated peripheral module.

Port 7 Control Register (P7CON)
F5H, Set 1, Bank 1, R/W

MSB	.7	.6	.5	.4	.3	.2	.1	.0	LSB
	P7.7/ ADC7	P7.6/ ADC6	P7.5/ ADC5	P7.4/ ADC4	P7.3/ ADC3	P7.2/ ADC2	P7.1/ ADC1	P7.0/ ADC0	

.7 bit : P7.7/ADC7

0	Input mode
1	ADC input mode

.6 bit : P7.6/ADC6

0	Input mode
1	ADC input mode

.5 bit : P7.5/ADC5

0	Input mode
1	ADC input mode

.4 bit : P7.4/ADC4

0	Input mode
1	ADC input mode

.3 bit : P7.3/ADC3

0	Input mode
1	ADC input mode

.2 bit : P7.2/ADC2

0	Input mode
1	ADC input mode

.1 bit : P7.1/ADC1

0	Input mode
1	ADC input mode

.0 bit : P7.0/ADC0

0	Input mode
1	ADC input mode

Figure 9-13. Port 7 Control Register (P7CON)

PORT 8

Port 8 is an 8-bit I/O Port that you can use two ways:

— General-purpose I/O

— External interrupt inputs for INT8-INT9

Port 8 is accessed directly by writing or reading the port 8 data register, P8 at location E8H in set 1, bank 0.

Port 8 Control Register (P8CONH, P8CONL)

Port 8 pins are configured individually by bit-pair settings in two control registers located in set 1, bank 0: P8CONL (low byte, EEH) and P8CONH (high byte, EDH).

When you select output mode, a push-pull circuit is configured. In input mode, three different selections are available:

— Schmitt trigger input with interrupt generation on falling signal edges.

— Schmitt trigger input with interrupt generation on rising signal edges.

— Schmitt trigger input with pull-up resistor and interrupt generation on falling signal edges.

Port 8 Interrupt Enable and Pending Registers (P8INTPND)

To process external interrupts at the port 8 pins, one additional control register is provided: the port 8 interrupt enable register P8INTPND (EFH, set 1, bank 0).

The port 8 interrupt pending register P8INTPND lets you check for interrupt pending conditions and clear the pending condition when the interrupt service routine has been initiated. The application program detects interrupt requests by polling the P8INTPND register at regular intervals.

When the interrupt enable bit of any port 8 pin is “1”, a rising or falling signal edge at that pin will generate an interrupt request. The corresponding P8INTPND bit is then automatically set to “1” and the IRQ level goes low to signal the CPU that an interrupt request is waiting. When the CPU acknowledges the interrupt request, application software must the clear the pending condition by writing a “0” to the corresponding P8INTPND bit.

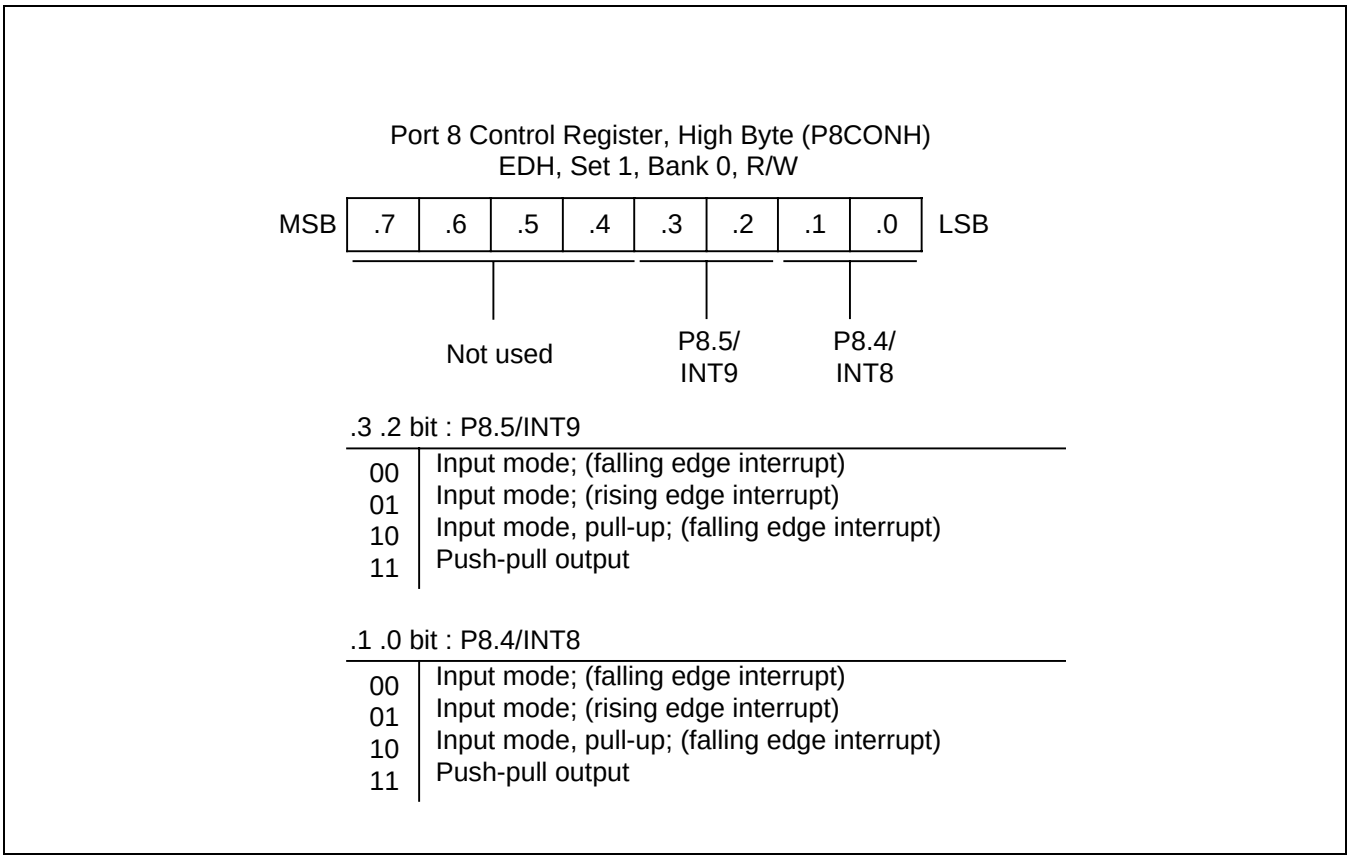


Figure 9-14. Port 8 High-Byte Control Register (P8CONH)

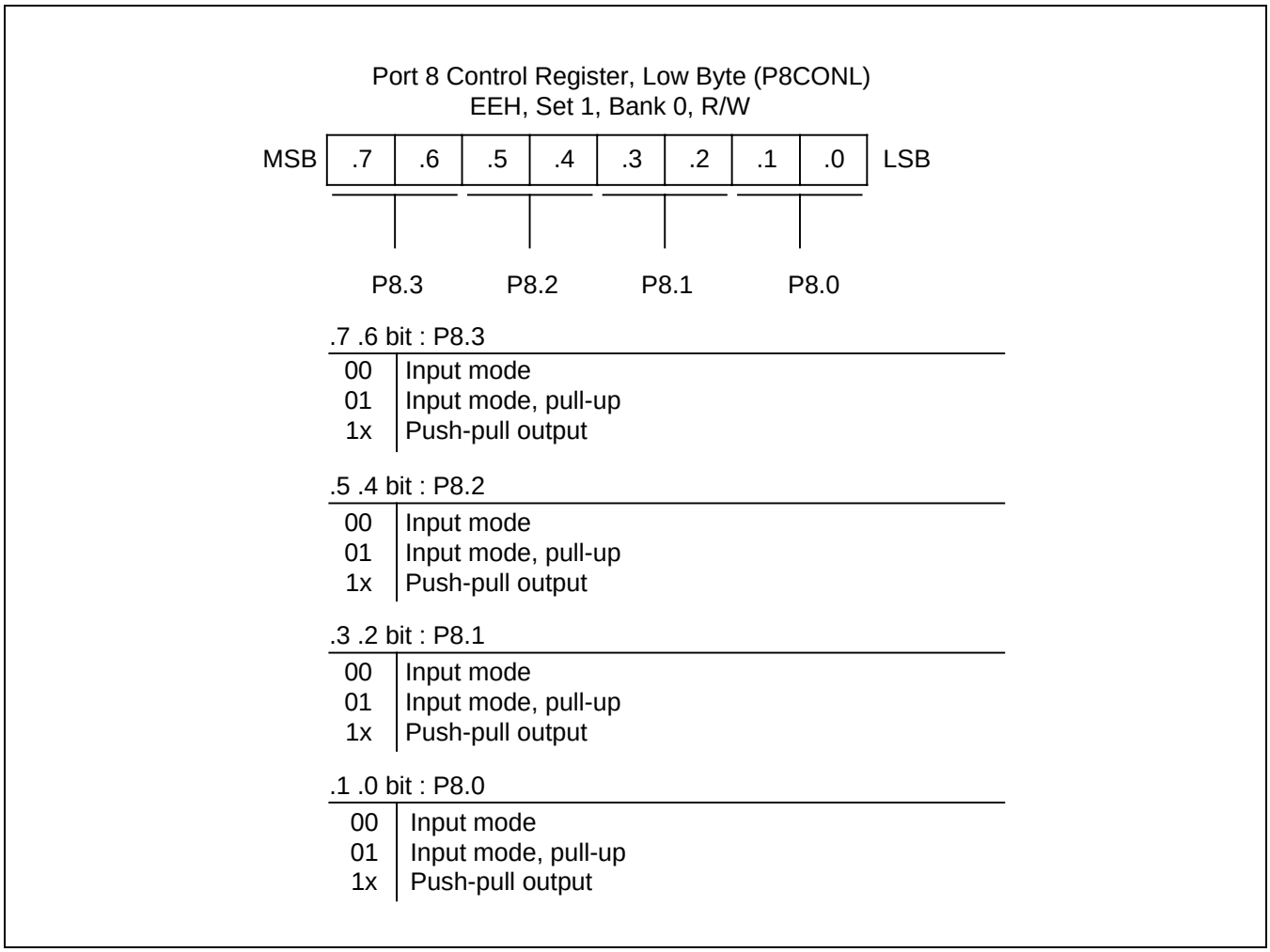
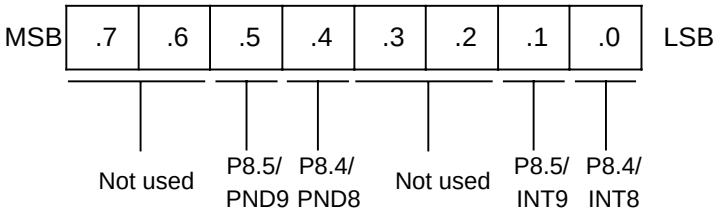


Figure 9-15. Port 8 Low-Byte Control Register (P8CONL)

Port 8 Interrupt Pending Register (P8INTPND)
EFH, Set 1, Bank 0, R/W



.5 bit : P8.5/PND9

0	Interrupt request is not pending, pending bit clear when write 0
1	Interrupt request is pending

.4 bit : P8.4/PND8

0	Interrupt request is not pending, pending bit clear when write 0
1	Interrupt request is pending

.1 bit : P8.5/INT9

0	Disable interrupt
1	Enable interrupt

.0 bit : P8.4/INT8

0	Disable interrupt
1	Enable interrupt

Figure 9-16. Port 8 Interrupt Pending Register (P8INTPND)

10 BASIC TIMER

OVERVIEW

BASIC TIMER (BT)

You can use the basic timer (BT) in two different ways:

— As a watchdog timer to provide an automatic reset mechanism in the event of a system malfunction.

— To signal the end of the required oscillation stabilization interval after a reset or a Stop mode release.

The functional components of the basic timer block are:

— Clock frequency divider (fxx divided by 4096, 1024 or 128) with multiplexer

— 8-bit basic timer counter, BTCNT (set 1, bank 0, FDH, read-only)

— Basic timer control register, BTCON (set 1, D3H, read/write)

BASIC TIMER CONTROL REGISTER (BTCON)

The basic timer control register, BTCON, is used to select the input clock frequency, to clear the basic timer counter and frequency dividers, and to enable or disable the watchdog timer function. It is located in set 1, address D3H, and is read/write addressable using register addressing mode.

A reset clears BTCON to '00H'. This enables the watchdog function and selects a basic timer clock frequency of $f_{XX}/4096$. To disable the watchdog function, write the signature code '1010B' to the basic timer register control bits BTCON.7–BTCON.4.

The 8-bit basic timer counter, BTCNT (set 1, bank 0, FDH), can be cleared at any time during normal operation by writing a "1" to BTCON.1. To clear the frequency dividers, write a "1" to BTCON.0.

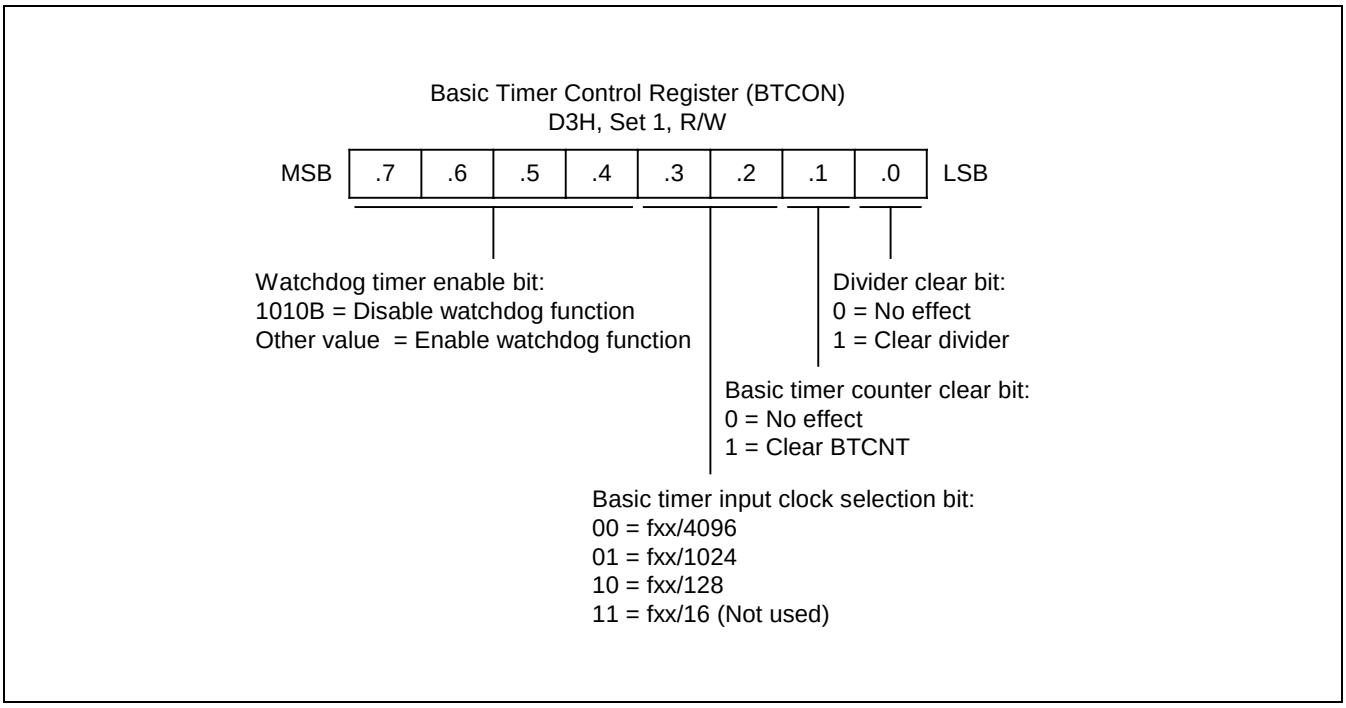


Figure 10-1. Basic Timer Control Register (BTCON)

BASIC TIMER FUNCTION DESCRIPTION

Watchdog Timer Function

You can program the basic timer overflow signal (BTOVF) to generate a reset by setting BTCON.7–BTCON.4 to any value other than "1010B". (The "1010B" value disables the watchdog function.) A reset clears BTCON to "00H", automatically enabling the watchdog timer function. A reset also selects the CPU clock (as determined by the current CLKCON register setting), divided by 4096, as the BT clock.

The MCU is reset whenever a basic timer counter overflow occurs, During normal operation, the application program must prevent the overflow, and the accompanying reset operation, from occurring, To do this, the BTCNT value must be cleared (by writing a “1” to BTCON.1) at regular intervals.

If a system malfunction occurs due to circuit noise or some other error condition, the BT counter clear operation will not be executed and a basic timer overflow will occur, initiating a reset. In other words, during the normal operation, the basic timer overflow loop (a bit 7 overflow of the 8-bit basic timer counter, BTCNT) is always broken by a BTCNT clear instruction. If a malfunction does occur, a reset is triggered automatically.

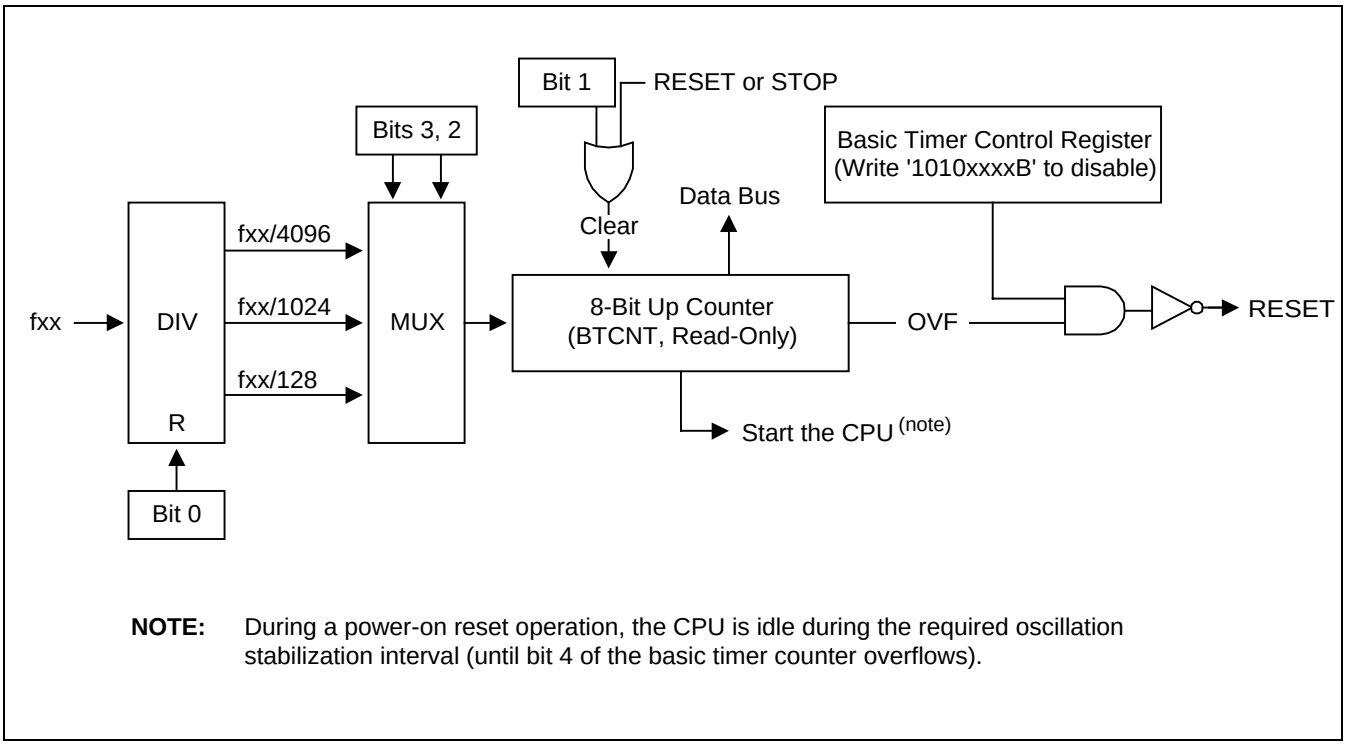
Oscillation Stabilization Interval Timer Function

You can also use the basic timer to program a specific oscillation stabilization interval following a reset or when Stop mode has been released by an external interrupt.

In Stop mode, whenever a reset or an external interrupt occurs, the oscillator starts. The BTCNT value then starts increasing at the rate of fxx/4096 (for reset), or at the rate of the preset clock source (for an external interrupt). When BTCNT.4 overflows, a signal is generated to indicate that the stabilization interval has elapsed and to gate the clock signal off to the CPU so that it can resume normal operation.

In summary, the following events occur when stop mode is released:

1. During stop mode, a power-on reset or an interrupt occurs to trigger the Stop mode release and oscillation starts.
2. If a power-on reset occurred, the basic timer counter will increase at the rate of fxx/4096. If an interrupt is used to release stop mode, the BTCNT value increases at the rate of the preset clock source.
3. Clock oscillation stabilization interval begins and continues until bit 4 of the basic timer counter overflows.
4. When a BTCNT.4 overflow occurs, normal CPU operation resumes.



NOTE: During a power-on reset operation, the CPU is idle during the required oscillation stabilization interval (until bit 4 of the basic timer counter overflows).

Figure 10-2. Basic Timer Block Diagram

11 8-BIT TIMER A/B/C(0/1)

8-BIT TIMER A

OVERVIEW

The 8-bit timer A is an 8-bit general-purpose timer/counter. Timer A has three operating modes, you can select one of them using the appropriate TACON setting:

— Interval timer mode (Toggle output at TAOUT pin)
— Capture input mode with a rising or falling edge trigger at the TACAP pin
— PWM mode (TAPWM); PWM output shares its output port with TAOUT pin

Timer A has the following functional components:

— Clock frequency divider (fxx divided by 1024, 256, or 64) with multiplexer
— External clock input pin (TACK)
— 8-bit counter (TACNT), 8-bit comparator, and 8-bit reference data register (TADATA)
— I/O pins for capture input (TACAP) or PWM or match output (TAPWM, TAOUT)
— Timer A overflow interrupt (IRQ0, vector BAH) and match/capture interrupt (IRQ0, vector B8H) generation
— Timer A control register, TACON (set 1, bank0, EAH, read/write)

FUNCTION DESCRIPTION

Timer A Interrupts (IRQ0, Vectors B8H and BAH)

The timer A module can generate two interrupts: the timer A overflow interrupt (TAOVF), and the timer A match/ capture interrupt (TAINT). TAOVF is interrupt level IRQ0, vector BAH. TAINT also belongs to interrupt level IRQ0, but is assigned the separate vector address, B8H.

A timer A overflow interrupt pending condition is automatically cleared by hardware when it has been serviced. A timer A match/capture interrupt, TAINT pending condition is also cleared by hardware when it has been serviced.

Interval Timer Function

The timer A module can generate an interrupt: the timer A match interrupt (TAINT). TAINT belongs to interrupt level IRQ0, and is assigned the separate vector address, B8H.

When timer A match interrupt occurs and is serviced by the CPU, the pending condition is cleared automatically by hardware.

In interval timer mode, a match signal is generated and TAOUT is toggled when the counter value is identical to the value written to the TA reference data register, TADATA. The match signal generates a timer A match interrupt (TAINT, vector B8H) and clears the counter.

If, for example, you write the value 10H to TADATA and 0AH to TACON, the counter will increment until it reaches 10H. At this point, the TA interrupt request is generated, the counter value is reset, and counting resumes.

Pulse Width Modulation Mode

Pulse width modulation (PWM) mode lets you program the width (duration) of the pulse that is output at the TAPWM pin. As in interval timer mode, a match signal is generated when the counter value is identical to the value written to the timer A data register. In PWM mode, however, the match signal does not clear the counter. Instead, it runs continuously, overflowing at FFH, and then continues incrementing from 00H.

Although timer A overflow interrupt is occurred, this interrupt is not typically used in PWM-type applications. Instead, the pulse at the TAPWM pin is held to Low level as long as the reference data value is *less than or equal to* (≤) the counter value and then the pulse is held to High level for as long as the data value is *greater than* (>) the counter value. One pulse width is equal to $t_{CLK} \bullet 256$.

Capture Mode

In capture mode, a signal edge that is detected at the TACAP pin opens a gate and loads the current counter value into the TA data register. You can select rising or falling edges to trigger this operation.

Timer A also gives you capture input source: the signal edge at the TACAP pin. You select the capture input by setting the value of the timer A capture input selection bit in the port 2 control register, P2CONH, (set 1, bank 0, F2H). When P2CONH.5.4 is 00, the TACAP input or normal input is selected. When P2CONH.5.4 is set to 10, normal output is selected.

Both kinds of timer A interrupts can be used in capture mode: the timer A overflow interrupt is generated whenever a counter overflow occurs; the timer A match/capture interrupt is generated whenever the counter value is loaded into the TA data register.

By reading the captured data value in TADATA, and assuming a specific value for the timer A clock frequency, you can calculate the pulse width (duration) of the signal that is being input at the TACAP pin.

TIMER A CONTROL REGISTER (TACON)

You use the timer A control register, TACON, to

— Select the timer A operating mode (interval timer, capture mode, or PWM mode)

— Select the timer A input clock frequency

— Clear the timer A counter, TACNT

— Enable the timer A overflow interrupt or timer A match/capture interrupt

— Clear timer A match/capture interrupt pending conditions

TACON is located in set 1, Bank 0 at address EAH, and is read/write addressable using Register addressing mode.

A reset clears TACON to '00H'. This sets timer A to normal interval timer mode, selects an input clock frequency of fxx/1024, and disables all timer A interrupts. You can clear the timer A counter at any time during normal operation by writing a "1" to TACON.3.

The timer A overflow interrupt (TAOVF) is interrupt level IRQ0 and has the vector address BAH. When a timer A overflow interrupt occurs and is serviced by the CPU, the pending condition is cleared automatically by hardware. To enable the timer A match/capture interrupt (IRQ0, vector B8H), you must write TACON.1 to "1". To generate the exact time interval, you should write “1” to TACON.3 and “0” to TINTPND.0, which cleared counter and interrupt pending bit.

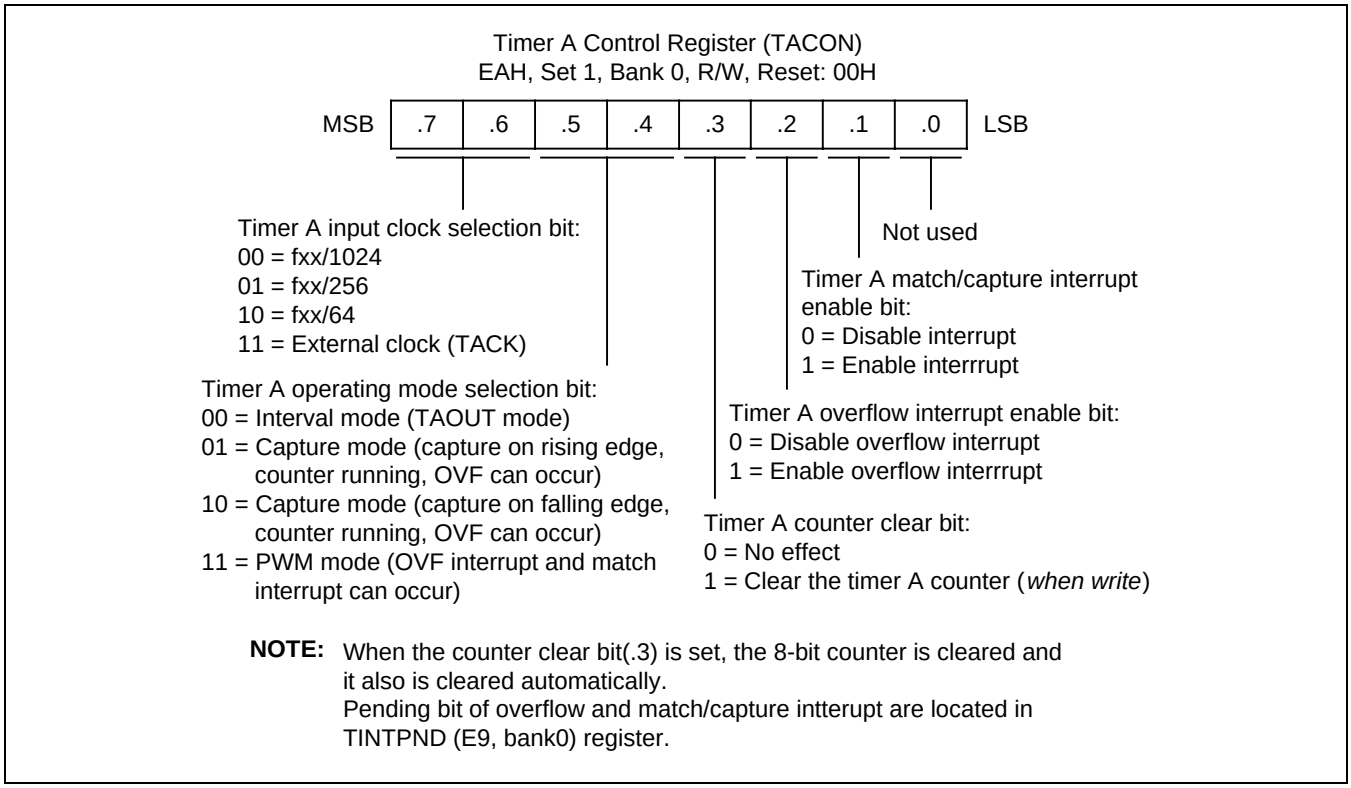
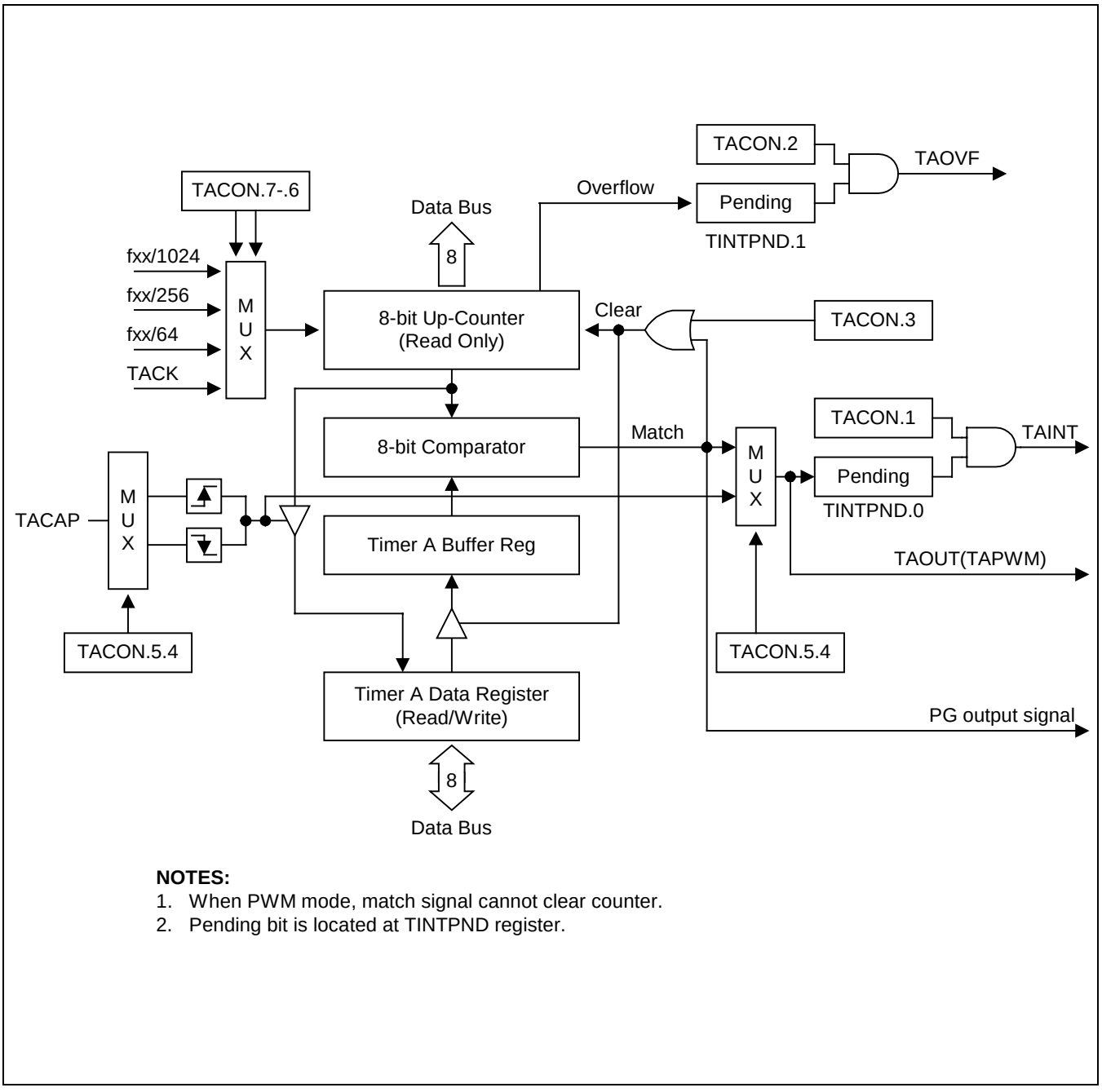


Figure 11-1. Timer A Control Register (TACON)

BLOCK DIAGRAM



NOTES:
1. When PWM mode, match signal cannot clear counter.
2. Pending bit is located at TINTPND register.

Figure 11-2. Timer A Functional Block Diagram

8-BIT TIMER B

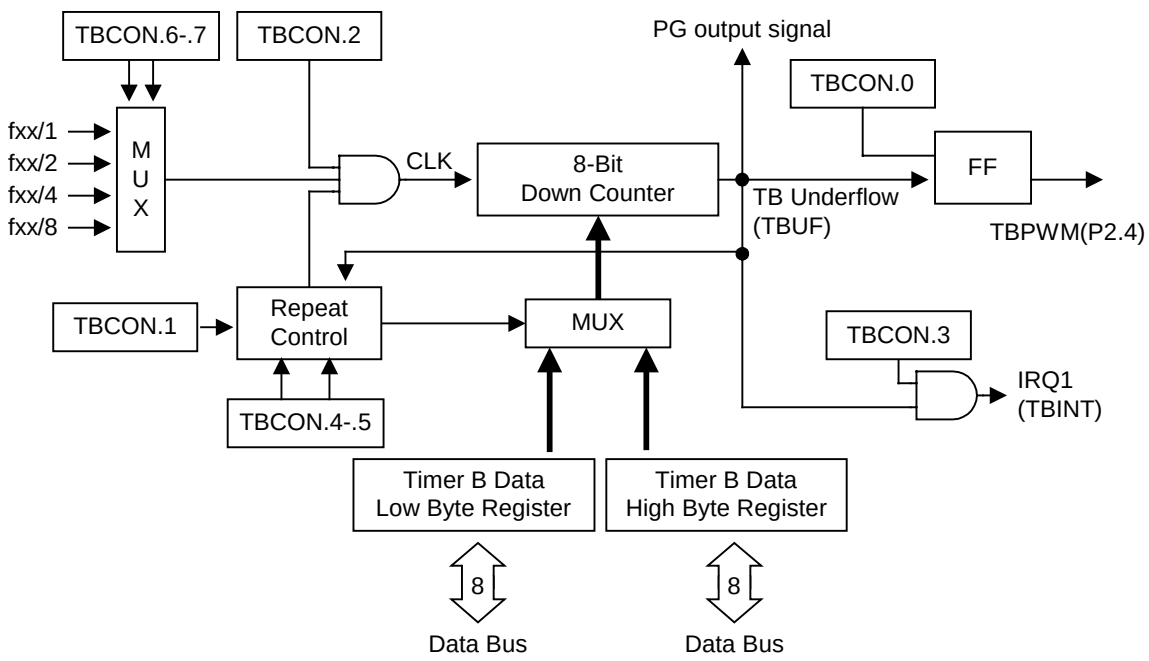
OVERVIEW

The S3C84BB/F84BB micro-controller has an 8-bit counter called timer B. Timer B, which can be used to generate the carrier frequency of a remote controller signal. Pending bit of timer B is cleared automatically by hardware.

Timer B has two functions:

— As a normal interval timer, generating a timer B interrupt at programmed time intervals.
— To generate a programmable carrier pulse for a remote control signal at P2.4.

BLOCK DIAGRAM



NOTE: In case of setting TBCON.5-.4 at '10', the value of the TBDATAL register is loaded into the 8-bit counter when the operation of the timer B starts. And then if a underflow occurs in the counter, the value of the TBDATAH register is loaded with the value of the 8-bit counter. However, if the next borrow occurs, the value of the TBDATAL register is loaded with the value of the 8-bit counter. To output TBPWM as carrier wave, you have to set P2CONH.1-.0 as "11".

Figure 11-3. Timer B Functional Block Diagram

TIMER B CONTROL REGISTER (TBCON)

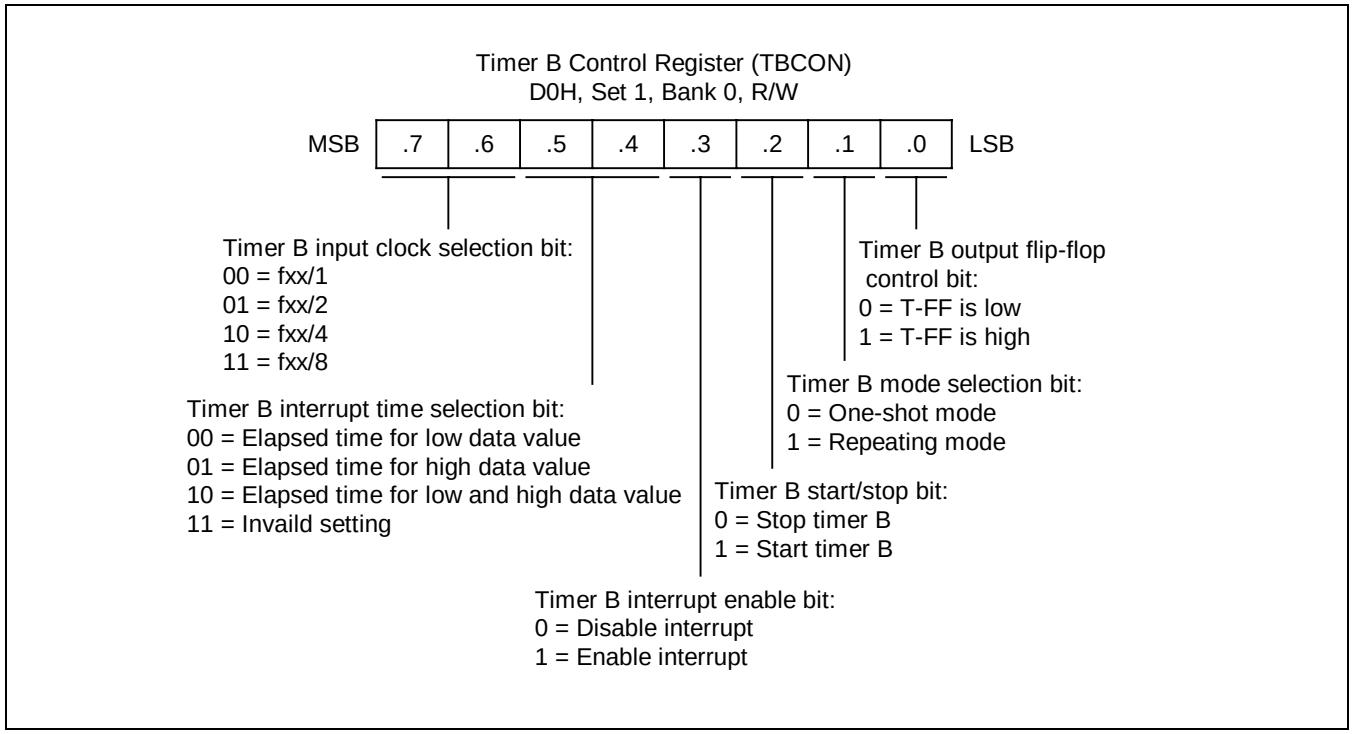


Figure 11-4. Timer B Control Register (TBCON)

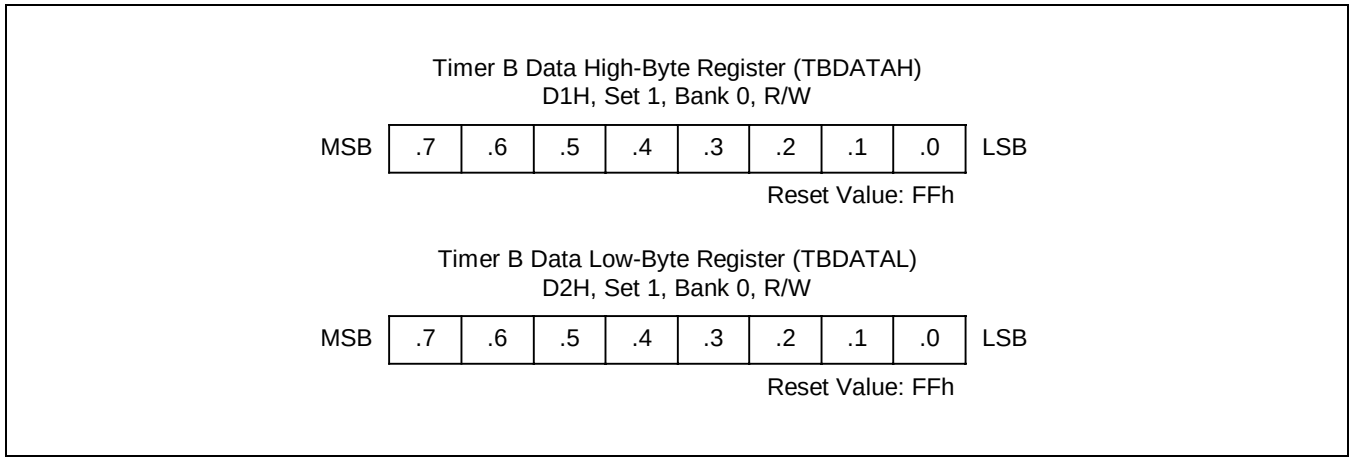
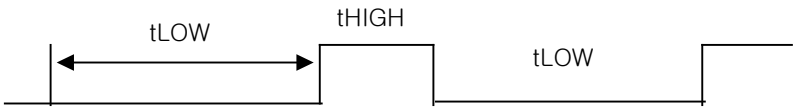


Figure 11-5. Timer B Data Registers (TBDATAH, TBDATAL)

TIMER B PULSE WIDTH CALCULATIONS



To generate the above repeated waveform consisted of low period time, t_{LOW}, and high period time, t_{HIGH}.

When T-FF = 0,

t_{LOW} = (TBDATAL + 1) x 1/fx, 0H < TBDATAL < 100H, where fx = The selected clock.

t_{HIGH} = (TBDATAH + 1) x 1/fx, 0H < TBDATAH < 100H, where fx = The selected clock.

When T-FF = 1,

t_{LOW} = (TBDATAH + 1) x 1/fx, 0H < TBDATAH < 100H, where fx = The selected clock.

t_{HIGH} = (TBDATAL + 1) x 1/fx, 0H < TBDATAL < 100H, where fx = The selected clock.

To make t_{LOW} = 24 us and t_{HIGH} = 15 us. f_{OSC} = 4 MHz, fx = 4 MHz/4 = 1 MHz

When T-FF = 0,

t_{LOW} = 24 us = (TBDATAL + 1) /fx = (TBDATAL + 1) x 1us, TBDATAL = 23.

t_{HIGH} = 15 us = (TBDATAH + 1) /fx = (TBDATAH + 1) x 1us, TBDATAH = 14.

When T-FF = 1,

t_{HIGH} = 15 us = (TBDATAL + 1) /fx = (TBDATAL + 1) x 1us, TBDATAL = 14.

t_{LOW} = 24 us = (TBDATAH + 1) /fx = (TBDATAH + 1) x 1us, TBDATAH = 23.

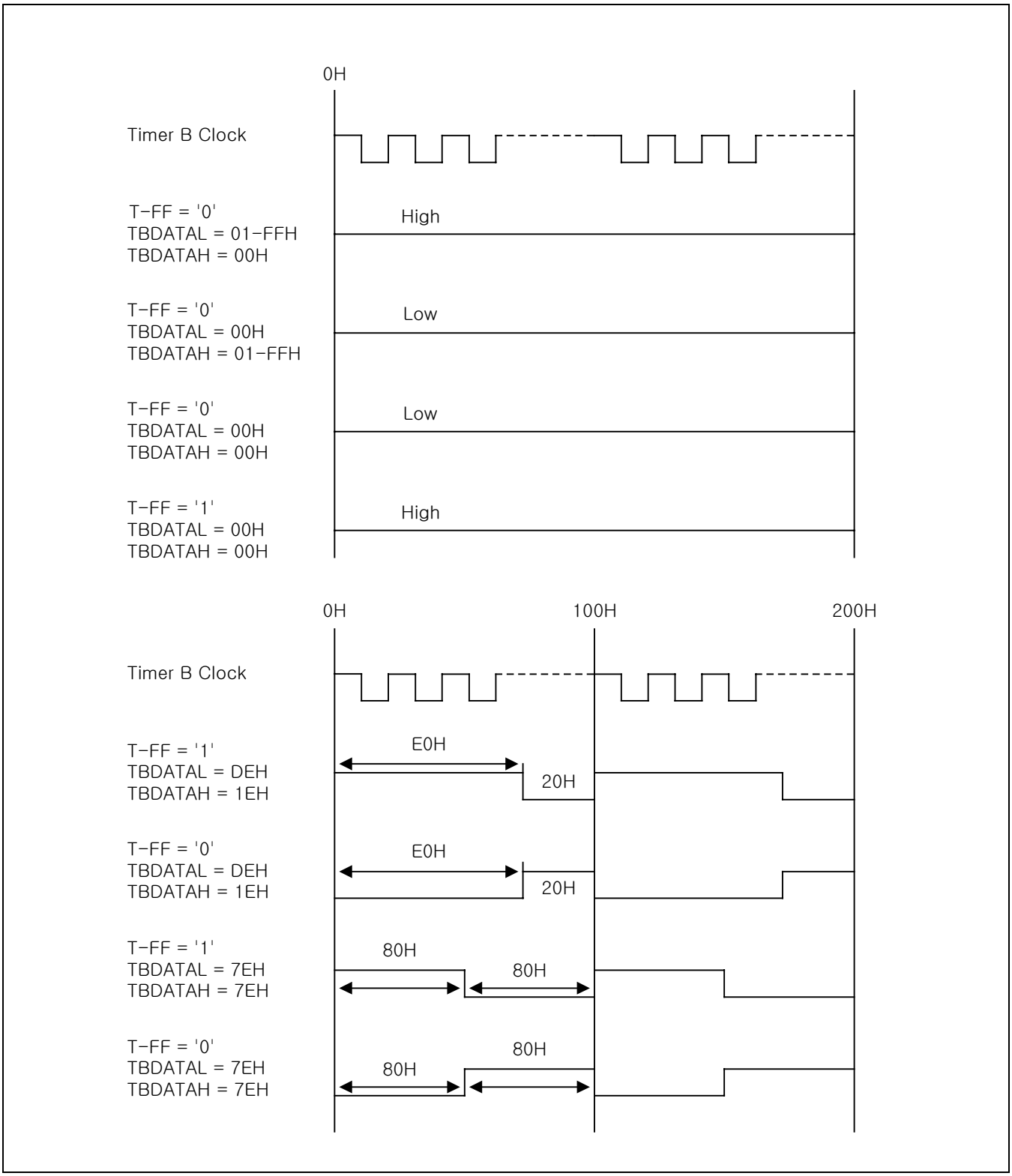
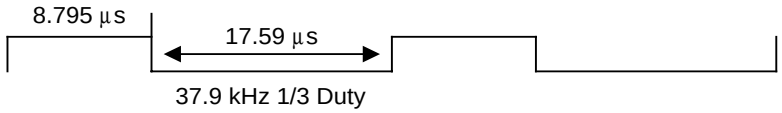


Figure 11-6. Timer B Output Flip-Flop Waveforms in Repeat Mode

☞ PROGRAMMING TIP — To generate 38 kHz, 1/3duty signal through P2.4

This example sets Timer B to the repeat mode, sets the oscillation frequency as the Timer B clock source, and TBDATAH and TBDATAL to make a 38 kHz, 1/3 Duty carrier frequency. The program parameters are:



— Timer B is used in repeat mode

— Oscillation frequency is 4 MHz (0.25 µs)

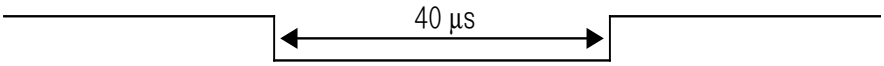
— TBDATAL = 8.795 µs/0.25 µs = 35.18, TBDATAH = 17.59 µs/0.25 µs = 70.36

— Set P2.4 to TBPWM mode.

```
        ORG     0100H               ;  Reset address
START   DI
        •
        •
        •
        LD      TBDATAH,#(70-1)     ;  Set 17.5 µs
        LD      TBDATAL,#(35-1)     ;  Set 8.75 µs
        LD      TBCON,#00100111B    ;  Clock Source ← fxx
                                    ;  Disable Timer B interrupt.
                                    ;  Select repeat mode for Timer B.
                                    ;  Start Timer B operation.
                                    ;  Set Timer B Output flip-flop (T-FF) high.
                                    ;
        LD      P2CONH,#03H         ;  Set P2.4 to TBPWM mode.
                                    ;  This command generates 38 kHz, 1/3 duty pulse signal
                                    ;  through P2.4.
        •
        •
        •
```

☞ PROGRAMMING TIP — To generate a one pulse signal through P2.4

This example sets Timer B to the one shot mode, sets the oscillation frequency as the Timer B clock source, and TBDATAH and TBDATAL to make a 40μs width pulse. The program parameters are:



— Timer B is used in one shot mode

— Oscillation frequency is 4 MHz (1 clock = 0.25 μs)

— TBDATAH = 40 μs / 0.25 μs = 160, TBDATAL = 1

— Set P2.4 to TBPWM mode

```
            ORG     0100H                ;  Reset address
START       DI
            •
            •
            •
            LD      TBDATAH,# (160-1)    ;  Set 40 μs
            LD      TBDATAL,# 1          ;  Set any value except  00H
            LD      TBCON,#00010001B     ;  Clock Source ← fOSC
                                         ;  Disable Timer B interrupt.
                                         ;  Select one shot mode for Timer B.
                                         ;  Stop Timer B operation.
                                         ;  Set Timer B output flip-flop (T-FF) high
            LD      P2CONH, #03H         ;  Set P2.4 to TBPWM mode.
            •
            •
Pulse_out:  LD      TBCON,#00010101B     ;  Start Timer B operation
                                         ;  to make the pulse at  this point.
            •                            ;  After the instruction is executed, 0.75 μs is required
            •                            ;  before the falling edge of the pulse starts.
            •
```

8-BIT TIMER C (0/1)

OVERVIEW

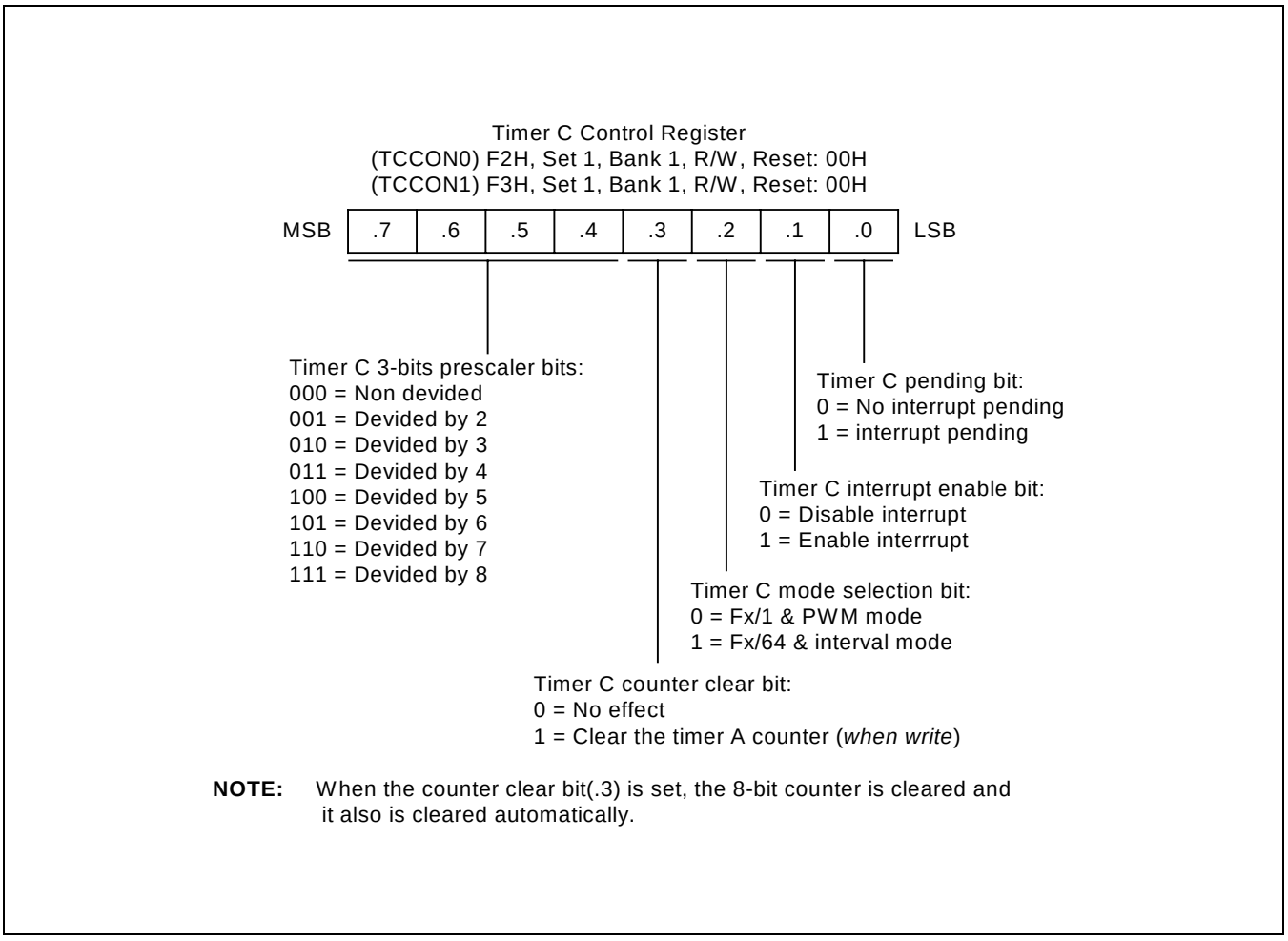
The 8-bit timer C (0/1) is an 8-bit general-purpose timer/counter. Timer C (0/1) has two operating modes, you can select one of them using the appropriate TCCON0, and TCCON1 setting:

— Interval timer mode (Toggle output at TCOUT0, TCOUT1 pin)
— PWM mode (TCOUT0, TCOUT1)

Timer C (0/1) has the following functional components:

— Clock frequency divider with multiplexer
— 8-bit counter, 8-bit comparator, and 8-bit reference data register (TCDATA0, TCDATA1)
— PWM or match output (TCOUT0, TCOUT1)
— Timer C (0) match/overflow interrupt (IRQ2, vector BCH) generation
— Timer C (1) match/overflow interrupt (IRQ2, vector BEH) generation
— Timer C (0) control register, TCCON0 (set 1, bank1, F2H, read/write)
— Timer C (1) control register, TCCON1 (set 1, bank1, F3H, read/write)

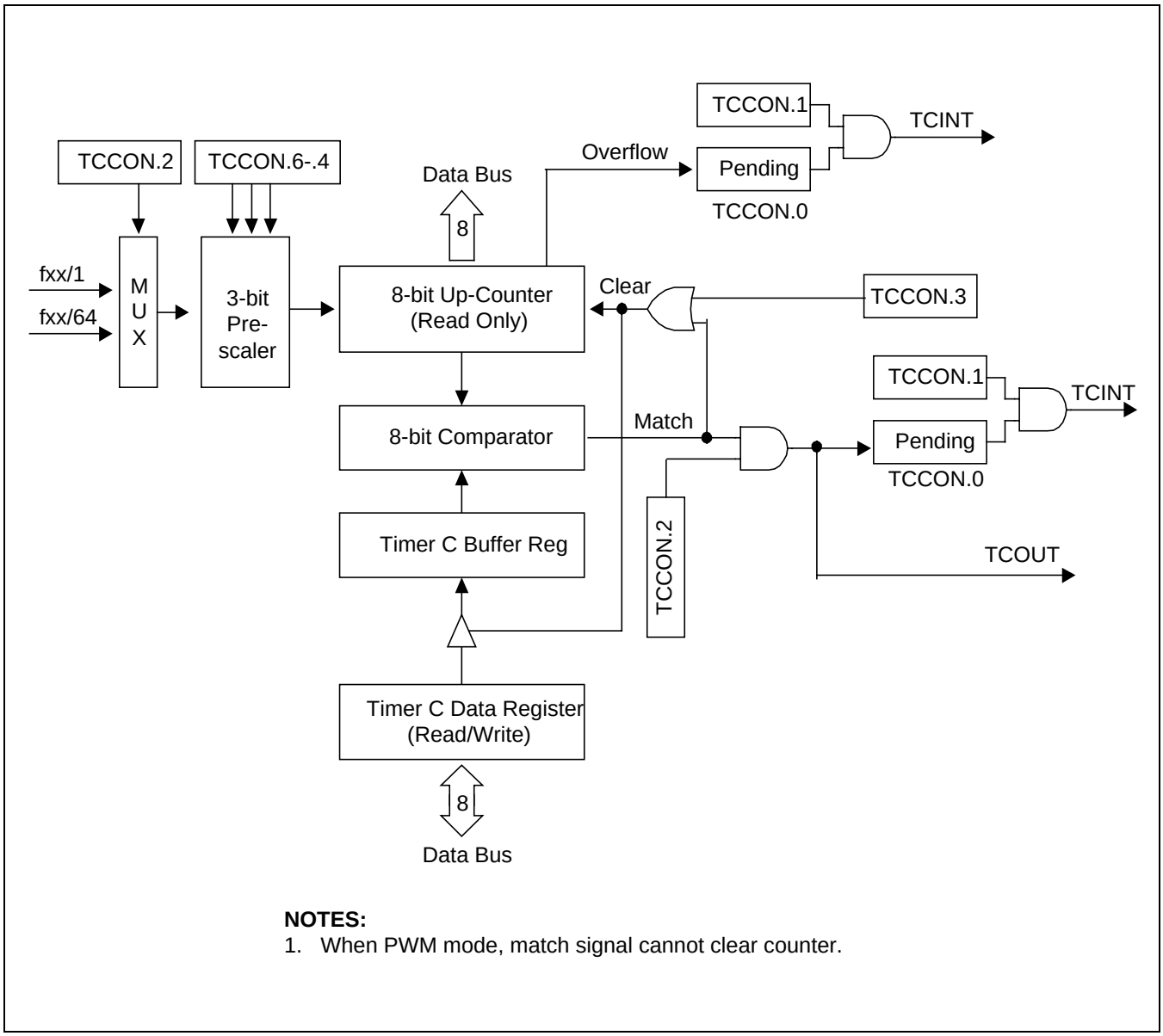
TIMER C (0/1) CONTROL REGISTER (TCCON0, TCCON1)



NOTE: When the counter clear bit(.3) is set, the 8-bit counter is cleared and it also is cleared automatically.

Figure 11-7. Timer C (0/1) Control Register (TCCON0, TCCON1)

BLOCK DIAGRAM



NOTES:
1. When PWM mode, match signal cannot clear counter.

Figure 11-8. Timer C (0/1) Functional Block Diagram

☞ PROGRAMMING TIP — Using the Timer A

```
                ORG         0000h

                VECTOR      0B8h,TAMC_INT
                VECTOR      0BAh,TAOV_INT

                ORG         0100h

INITIAL:
                LD          SYM,#00h            ;  Disable Global/Fast interrupt → SYM
                LD          IMR,#00000001b      ;  Enable IRQ0 interrupt
                LD          SPH,#00000000b      ;  Set stack area
                LD          SPL,#0FFh
                LD          BTCON,#10100011b    ;  Disable watch-dog

                LD          TADATA,#80h
                LD          TACON,#01001010b    ;  Match interrupt enable
                                                ;  3.30 ms duration (10 MHz x'tal)

                EI

MAIN:
                •
                •
                MAIN ROUTINE
                •
                •

                JR          T,MAIN

TAMC_INT:
                •
                •
                Interrupt service routine
                •
                •
                IRET

TAOV_INT:
                •
                Interrupt service routine
                •
                •
                IRET

                .END
```

☞ PROGRAMMING TIP — Using the Timer B

```
                ORG         0000h

                VECTOR      0C8h,TBUN_INT

                ORG         0100h

INITIAL:
                LD          SYM,#00h            ;  Disable Global/Fast interrupt
                LD          IMR,#00000010b      ;  Enable IRQ1 interrupt
                LD          SPH,#00000000b      ;  Set stack area
                LD          SPL,#0FFh
                LD          BTCON,#10100011b    ;  Disable Watch-dog

                LD          P2CONH,#00000011b   ;  Enable TBPWM output

                LD          TBDATAH,#80h
                LD          TBDATAL,#80h
                LD          TBCON,#11101110b    ;  Enable interrupt, repeating, fxx/8
                                                ;  Duration 206 µs (10 MHz x'tal)

                EI

MAIN:
                •
                •
                •
                MAIN ROUTINE
                •
                •
                •

                JR          T, MAIN

TBUN_INT:
                •
                •
                •
                Interrupt service routine
                •
                •
                •
                IRET

                .END
```

☞ PROGRAMMING TIP — Using the Timer C(0)

```
                ORG         0000h

                VECTOR      0BCh, TCUN_INT

                ORG         0100h

INITIAL:
                LD          SYM,#00h            ;  Disable Global/Fast interrupt
                LD          IMR,#00000100b      ;  Enable IRQ2 interrupt
                LD          SPH,#00000000b      ;  Set stack area
                LD          SPL,#11111111b
                LD          BTCON,#10100011b    ;  Disable Watch-dog, high speed

                LD          P3CONH,#00110000b   ;  Enable TCOUT0 output

                LD          TCDATA0,#80h
                LD          TCCON0,#00001110b   ;  non-divide, interval, Enable interrupt
                                                ;  Duration 0.825ms (10 MHz x'tal)

                EI

MAIN:
                •
                •
                •
                MAIN ROUTINE
                •
                •
                •

                JR          T, MAIN

TCUN_INT:
                •
                •
                •
                Interrupt service routine
                •
                •
                •
                IRET

                .END
```

12 16-BIT TIMER 1(0/1)

OVERVIEW

The S3C84BB/F84BB has two 16-bit timer/counters. The 16-bit timer 1(0/1) is a 16-bit general-purpose timer/counter. Timer 1(0/1) has three operating modes, one of which you select using the appropriate T1CON0, T1CON1 setting is:

— Interval timer mode (Toggle output at T1OUT0, T1OUT1 pin)

— Capture input mode with a rising or falling edge trigger at the T1CAP0, T1CAP1 pin

— PWM mode (T1PWM0, T1PWM1); PWM output shares their output port with T1OUT0, T1OUT1 pin

Timer 1(0/1) has the following functional components:

— Clock frequency divider (fxx divided by 1024, 256, 64, 8, or 1) with multiplexer

— External clock input pin (T1CK0, T1CK1)

— A 16-bit counter (T1CNTH0/L0, T1CNTH1/L1), 16-bit comparator, and two 16-bit reference data register (T1DATAH0/L0, T1DATAH1/L1)

— I/O pins for capture input (T1CAP0, T1CAP1), or match output (T1OUT0, T1OUT1)

— Timer 1(0) overflow interrupt (IRQ3, vector C2H) and match/capture interrupt (IRQ3, vector C0H) generation

— Timer 1(1) overflow interrupt (IRQ3, vector C6H) and match/capture interrupt (IRQ3, vector C4H) generation

— Timer 1(0) control register, T1CON0 (set 1, EAH, Bank 1, read/write)

— Timer 1(1) control register, T1CON1 (set 1, EBH, Bank 1, read/write)

FUNCTION DESCRIPTION

Timer 1 (0/1) Interrupts (IRQ3, Vectors C6H, C4H, C2H and C0H)

The timer 1(0) module can generate two interrupts, the timer 1(0) overflow interrupt (T1OVF0), and the timer 1(0) match/capture interrupt (T1INT0). T1OVF0 is interrupt level IRQ3, vector C2H. T1INT0 also belongs to interrupt level IRQ3, but is assigned the separate vector address, C0H.
A timer 1(0) overflow interrupt pending condition is automatically cleared by hardware when it has been serviced. A timer 1(0) match/capture interrupt, T1INT0 pending condition is also cleared by hardware when it has been serviced.

The timer 1(1) module can generate two interrupts, the timer 1(1) overflow interrupt (T1OVF1), and the timer 1(1) match/capture interrupt (T1INT1). T1OVF1 is interrupt level IRQ3, vector C6H. T1INT1 also belongs to interrupt level IRQ3, but is assigned the separate vector address, C4H.
A timer 1(1) overflow interrupt pending condition is automatically cleared by hardware when it has been serviced. A timer 1(1) match/capture interrupt, T1INT1 pending condition is also cleared by hardware when it has been serviced.

Interval Mode (match)

The timer 1(0) module can generate an interrupt: the timer 1(0) match interrupt (T1INT0). T1INT0 belongs to interrupt level IRQ3, and is assigned the separate vector address, C0H.
In interval timer mode, a match signal is generated and T1OUT0 is toggled when the counter value is identical to the value written to the T1 reference data register, T1DATAH0/L0. The match signal generates a timer 1(0) match interrupt (T1INT0, vector C0H) and clears the counter.

The timer 1(1) module can generate an interrupt: the timer 1(1) match interrupt (T1INT1). T1INT1 belongs to interrupt level IRQ3, and is assigned the separate vector address, C4H.
In interval timer mode, a match signal is generated and T1OUT1 is toggled when the counter value is identical to the value written to the T1 reference data register, T1DATAH1/L1. The match signal generates a timer 1(1) match interrupt (T1INT1, vector C4H) and clears the counter.

Capture Mode

In capture mode for Timer 1(0), a signal edge that is detected at the T1CAP0 pin opens a gate and loads the current counter value into the T1 data register (T1DATAH0/L0 for rising edge, or falling edge). You can select rising or falling edges to trigger this operation.
Timer 1(0) also gives you capture input source, the signal edge at the T1CAP0 pin. You select the capture input by setting the capture input selection bit in the port 3 control register, P3CONL, (set 1 bank 0, F5H).
Both kinds of timer 1(0) interrupts (T1OVF0, T1INT0) can be used in capture mode, the timer 1(0) overflow interrupt is generated whenever a counter overflow occurs, the timer 1(0) capture interrupt is generated whenever the counter value is loaded into the T1 data register (T1DATAH0/L0).
By reading the captured data value in T1DATAH0/L0, and assuming a specific value for the timer 1(0) clock frequency, you can calculate the pulse width (duration) of the signal that is being input at the T1CAP0 pin.

In capture mode for Timer 1(1), a signal edge that is detected at the T1CAP1 pin opens a gate and loads the current counter value into the T1 data register (T1DATAH1/L1 for rising edge, or falling edge). You can select rising or falling edges to trigger this operation.
Timer 1(1) also gives you capture input source, the signal edge at the T1CAP1 pin. You select the capture input by setting the capture input selection bit in the port 3 control register, P3CONL, (set 1 bank 0, F5H).
Both kinds of timer 1(1) interrupts (T1OVF1, T1INT1) can be used in capture mode, the timer 1(1) overflow interrupt is generated whenever a counter overflow occurs, the timer 1(1) capture interrupt is generated whenever the counter value is loaded into the T1 data register.
By reading the captured data value in T1DATAH1/L1, and assuming a specific value for the timer 1(1) clock frequency, you can calculate the pulse width (duration) of the signal that is being input at the T1CAP1 pin.

PWM Mode

Pulse width modulation (PWM) mode lets you program the width (duration) of the pulse that is output at the T1OUT0, T1OUT1 pin. As in interval timer mode, a match signal is generated when the counter value is identical to the value written to the timer 1(0/1) data register. In PWM mode, however, the match signal does not clear the counter but can generate a match interrupt. The counter runs continuously, overflowing at FFFFH, and then continuous increasing from 0000H. Whenever an overflow is occurred, an overflow (OVF0,1) interrupt can be generated.

Although you can use the match or the overflow interrupt in the PWM mode, these interrupts are not typically used in PWM-type applications. Instead, the pulse at the T1OUT0, T1OUT1 pin is held to low level as long as the reference data value is less than or equal to(≤) the counter value and then the pulse is held to high level for as long as the data value is greater than(>) the counter value. One pulse width is equal to t_{CLK} .

TIMER 1(0/1) CONTROL REGISTER (T1CON0, T1CON1)

You use the timer 1(0/1) control register, T1CON0, T1CON1, to

— Select the timer 1(0/1) operating mode (interval timer, capture mode, or PWM mode)

— Select the timer 1(0/1) input clock frequency

— Clear the timer 1(0/1) counter, T1CNTH0/L0, T1CNTH1/L1

— Enable the timer 1(0/1) overflow interrupt

— Enable the timer 1(0/1) match/capture interrupt

T1CON0 is located in set 1 and Bank 1 at address EAH, and is read/write addressable using Register addressing mode. T1CON1 is located in set 1 and Bank 1 at address EBH, and is read/write addressable using Register addressing mode.

A reset clears T1CON0, T1CON1 to '00H'. This sets timer 1(0/1) to normal interval timer mode, selects an input clock frequency of fxx/1024, and disables all timer 1(0/1) interrupts. To disable the counter operation, please set T1CON(0/1).7-.5 to 111B. You can clear the timer 1(0/1) counter at any time during normal operation by writing a "1" to T1CON(0/1).3. To generate the exact time interval, you should write "1" to T1CON(0/1).2 and clear appropriate pending bits of the TINTPND register.

To detect a match/capture or overflow interrupt pending condition when T1INT0, T1INT1 or T1OVF0, T1OVF1 is disabled, the application program should poll the pending bit TINTPND register, bank 0, E9H. When a "1" is detected, a timer 1(0/1) match/capture or overflow interrupt is pending.
When the sub-routine has been serviced, the pending condition must be cleared by software by writing a "0" to the interrupt pending bit. If interrupts (match/capture or overflow) are enabled, the pending bit is cleared automatically by hardware.

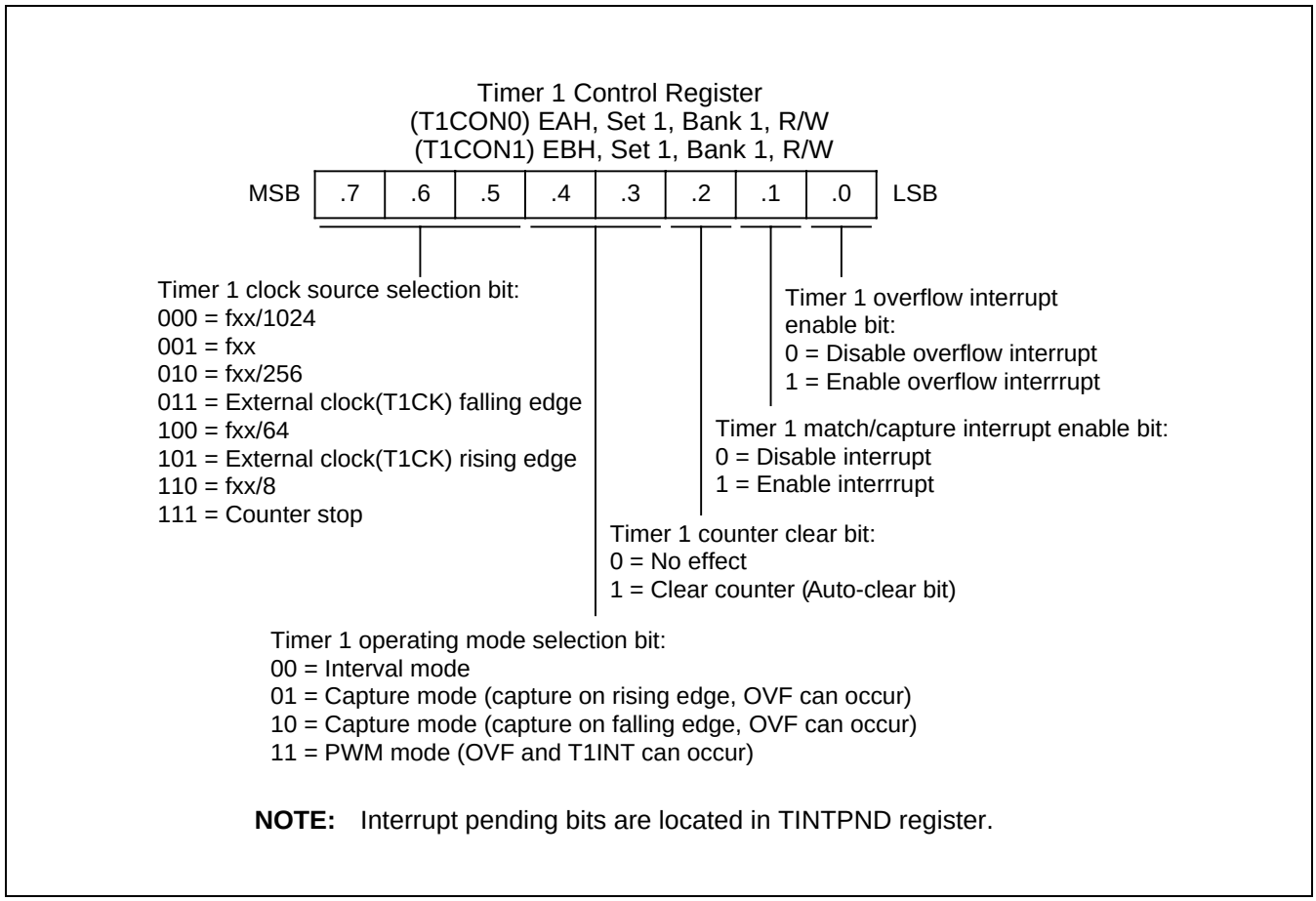


Figure 12-1. Timer 1(0/1) Control Register (T1CON0, T1CON1)

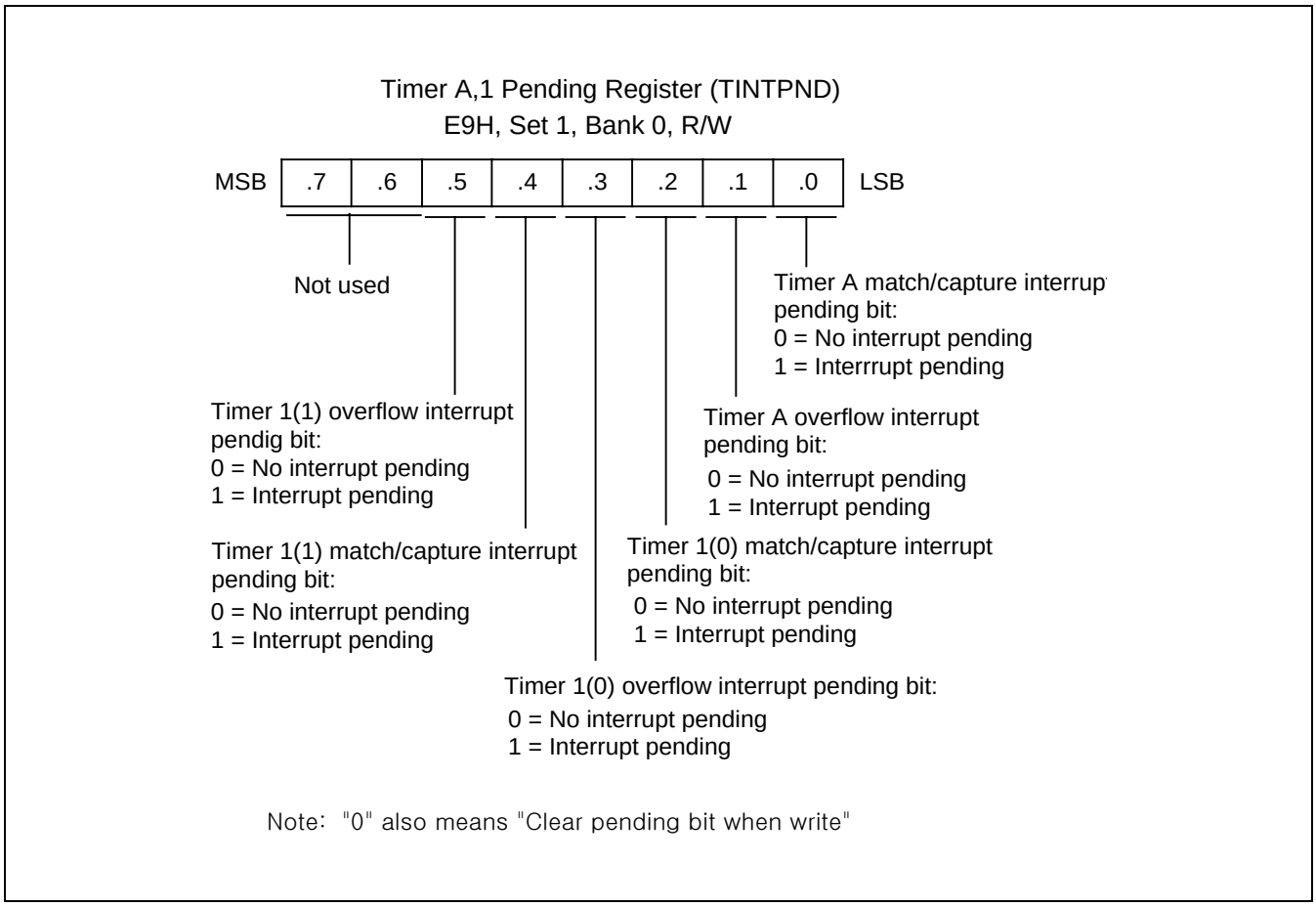
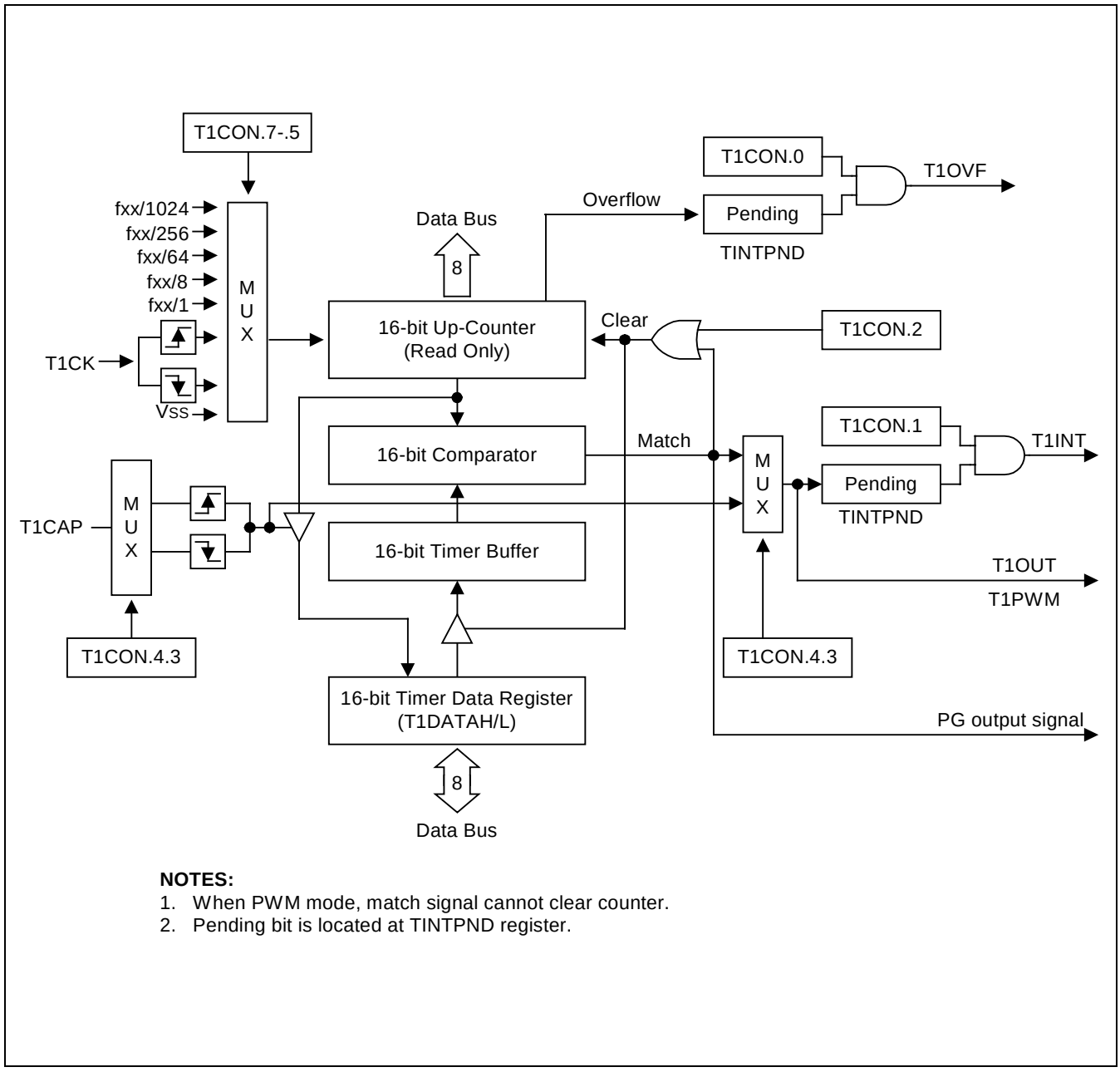


Figure 12-2. Timer A and Timer 1(0/1) Pending Register (TINTPND)

BLOCK DIAGRAM



NOTES:
1. When PWM mode, match signal cannot clear counter.
2. Pending bit is located at TINTPND register.

Figure 12-3. Timer 1(0/1) Functional Block Diagram

☞ PROGRAMMING TIP — Using the Timer 1(0)

```
                ORG         0000h

                VECTOR      0E4h,T1MC_INT

                ORG         0100h

INITIAL:
                LD          SYM,#00h            ;  Disable Global/Fast interrupt
                LD          IMR,#00001000b      ;  Enable IRQ3 interrupt
                LD          SPH,#00000000b      ;  Set stack area
                LD          SPL,#11111111b
                LD          BTCON,#10100011b    ;  Disable Watch-dog

                SB1
                LDW         T1DATAH0,#0F0h
                LD          T1CON0,#01000110b   ;  fxx/256, interval, clear counter, Enable interrupt
                                                ;  Duration 6.17ms (10 MHz x'tal)
                SB0

                EI

MAIN:
                •
                •
                •
                MAIN ROUTINE
                •
                •
                •

                JR          T,MAIN

T1MC_INT:
                •
                •
                •
                Interrupt service routine
                •
                •
                •
                IRET

                .END
```

NOTES

13 SERIAL I/O PORT

OVERVIEW

Serial I/O module, SIO can interface with various types of external devices that require serial data transfer. SIO has the following functional components:

— SIO data receive/transmit interrupt (IRQ4, vector CAH) generation
— 8-bit control register, SIOCON (set 1, bank 1, E1H, read/write)
— Clock selection logic
— 8-bit data buffer, SIODATA
— 8-bit prescaler (SIOPS), (set 1, bank 1, F4H, read/write)
— 3-bit serial clock counter
— Serial data I/O pins (P2.0–P2.1, SO, SI)
— External clock input/output pin (P2.2, SCK)

The SIO module can transmit or receive 8-bit serial data at a frequency determined by its corresponding control register settings. To ensure flexible data transmission rates, you can select an internal or external clock source.

PROGRAMMING PROCEDURE

To program the SIO modules, follow these basic steps:

1. Configure P2.1, P2.0 and P2.2 to alternative function (SI, SO, SCK) for interfacing SIO module by setting the P2CONL register to appropriately value.
2. Load an 8-bit value to the SIOCON control register to properly configure the serial I/O module. In this operation, SIOCON.2 must be set to "1" to enable the data shifter.
3. For interrupt generation, set the serial I/O interrupt enable bit, SIOCON.1 to "1".
4. To transmit data to the serial buffer, write data to SIODATA and set SIOCON.3 to 1, then the shift operation starts.
5. When the shift operation (transmit/receive) is completed, the SIO pending bit (SIOCON.0) is set to "1" and an SIO interrupt request is generated.

SIO CONTROL REGISTER (SIOCON)

The control register for the serial I/O interface module, SIOCON, is located in set 1, bank 1 at address E1H. It has the control settings for SIO module.

— Clock source selection (internal or external) for shift clock
— Interrupt enable
— Edge selection for shift operation
— Clear 3-bit counter and start shift operation
— Shift operation (transmit) enable
— Mode selection (transmit/receive or receive-only)
— Data direction selection (MSB first or LSB first)

A reset clears the SIOCON value to '00H'. This configures the corresponding module with an internal clock source, P.S clock at the SCK, selects receive-only operating mode, the data shift operation and the interrupt are disabled, and the data direction is selected to MSB-first.
So, if you want to use SIO module, you must write appropriate value to SIOCON.

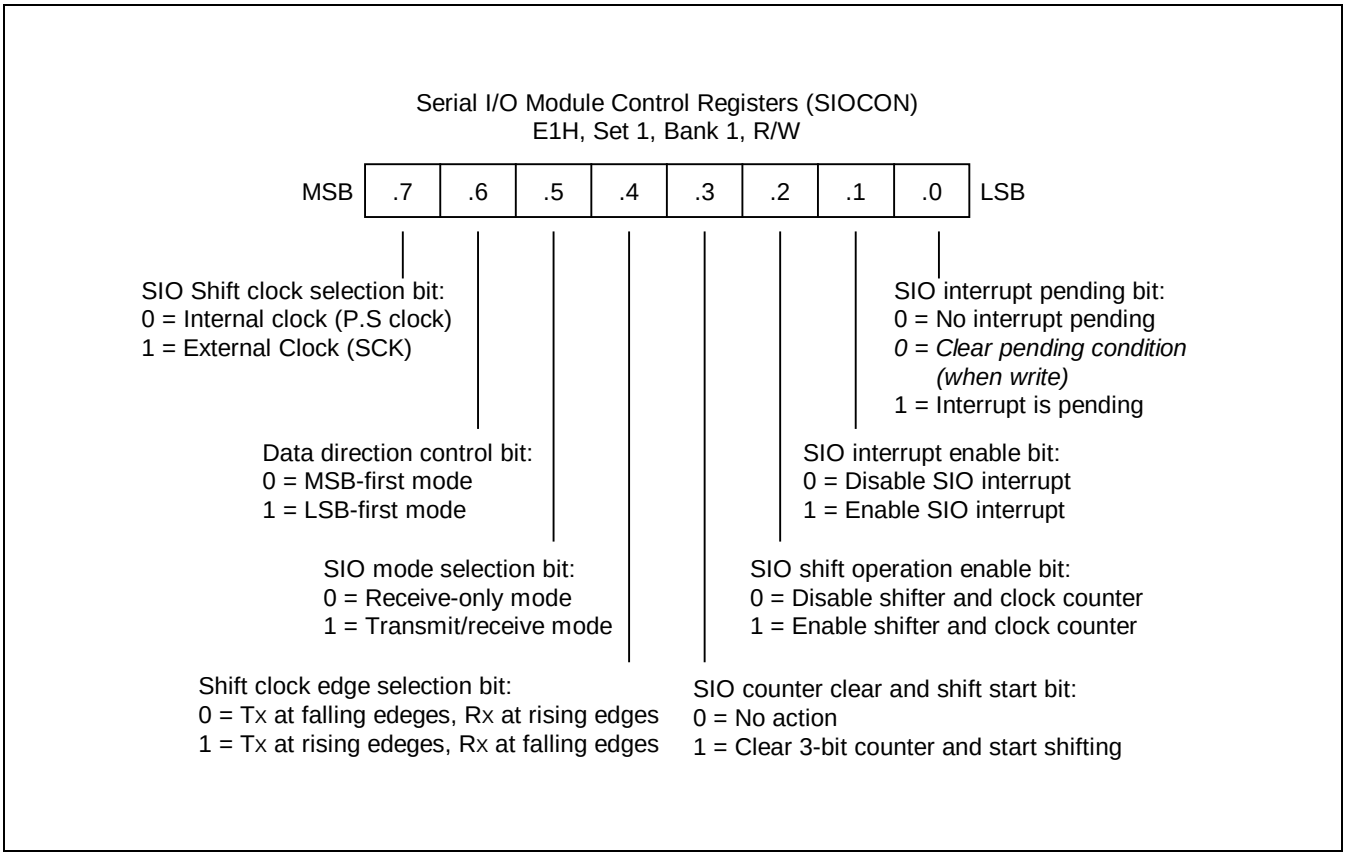


Figure 13-1. SIO Module Control Register (SIOCON)

SIO PRESCALER REGISTER (SIOPS)

The control register for the serial I/O interface module, SIOPS, is located in set 1, bank 1, at address F4H. The value stored in the SIO prescaler registers, SIOPS, lets you determine the SIO clock rate (baud rate) as follows:

Baud rate = Input clock (fxx)/[(SIOPS value + 1) x 2] or SCK input clock

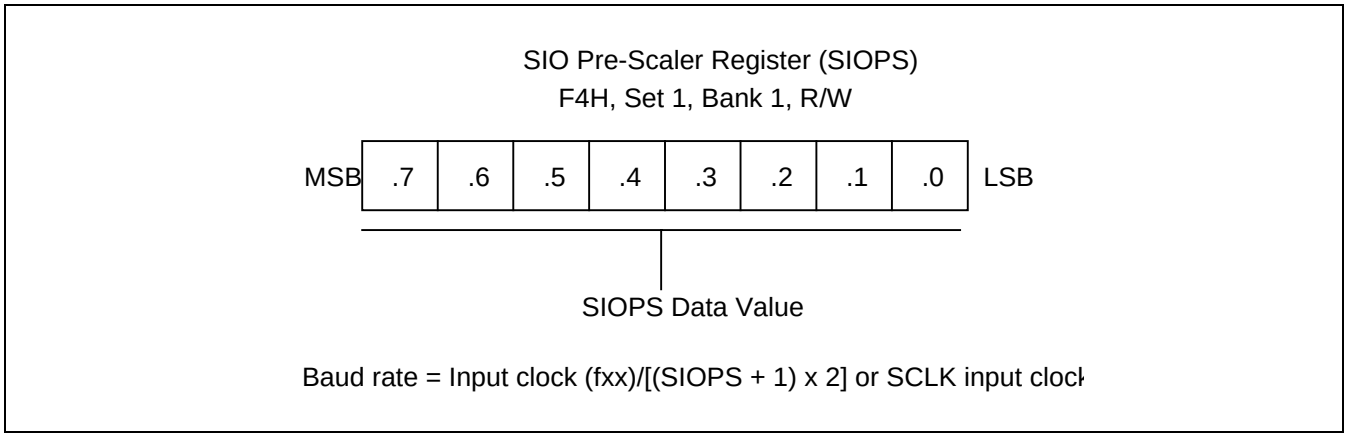


Figure 13-2. SIO Prescaler Register (SIOPS)

BLOCK DIAGRAM

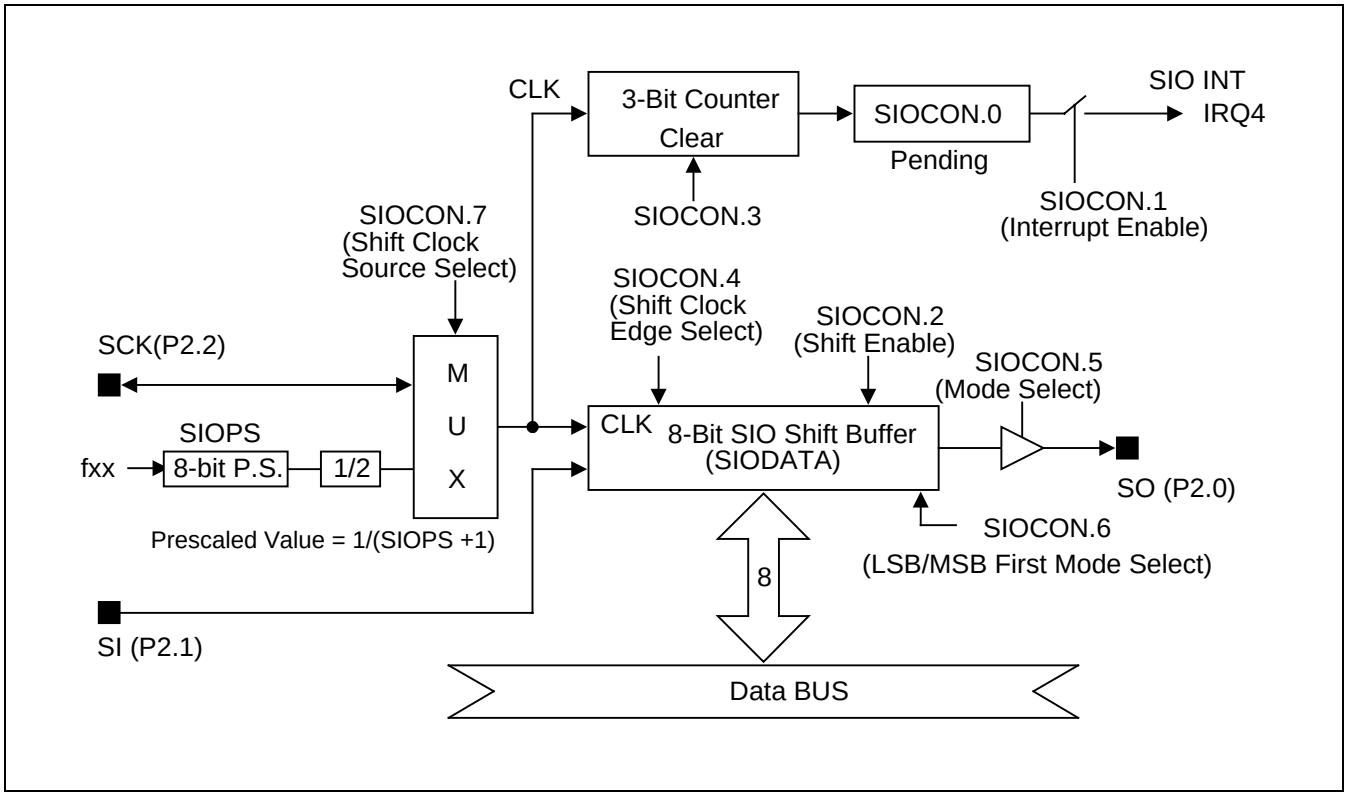


Figure 13-3. SIO Functional Block Diagram

SERIAL I/O TIMING DIAGRAMS

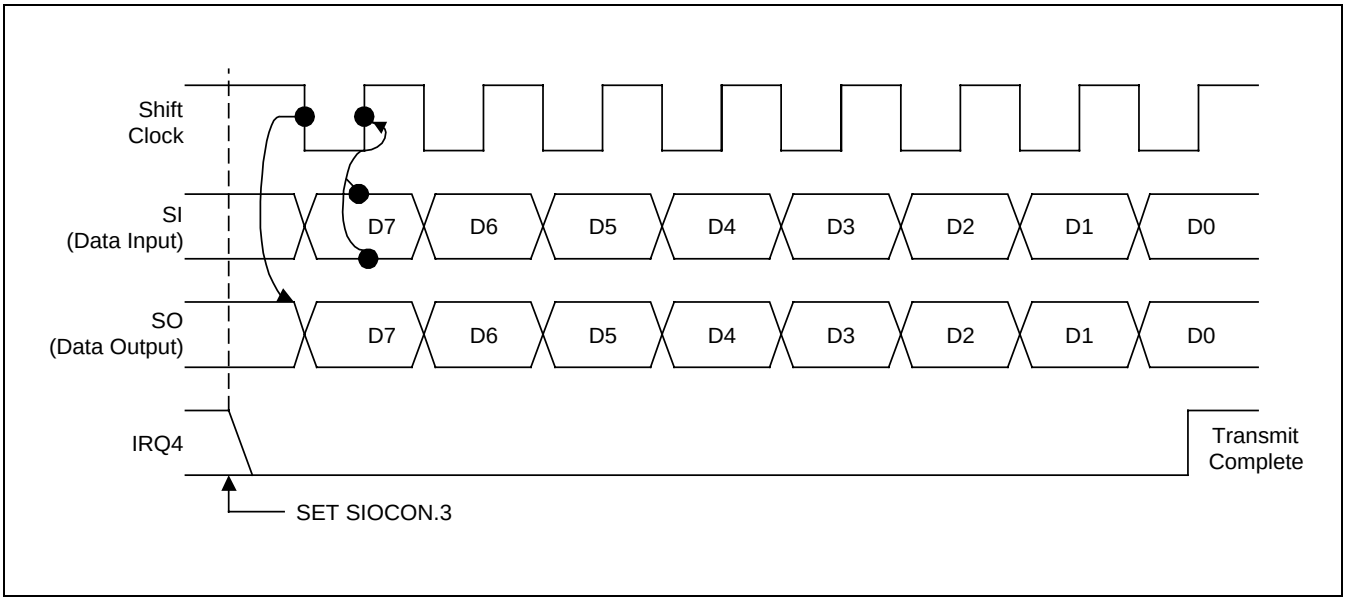


Figure 13-4. SIO Timing in Transmit/Receive Mode (Tx at falling edge, SIOCON.4=0)

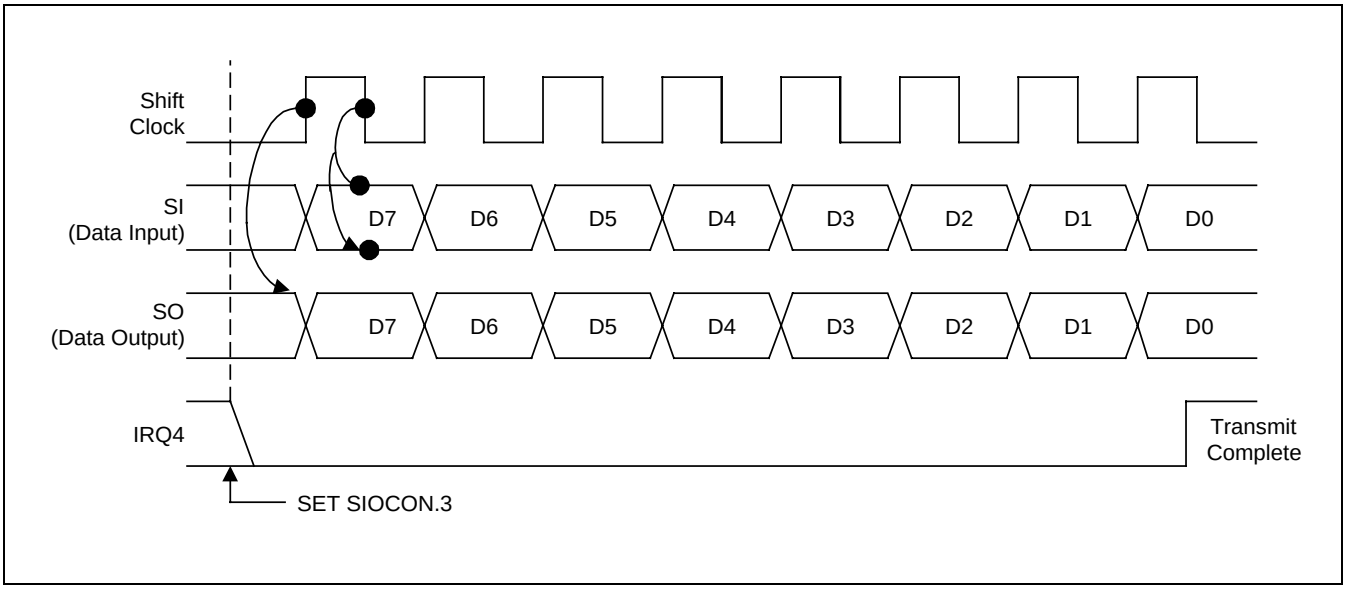


Figure 13-5. SIO Timing in Transmit/Receive Mode (Tx at rising edge, SIOCON.4=1)

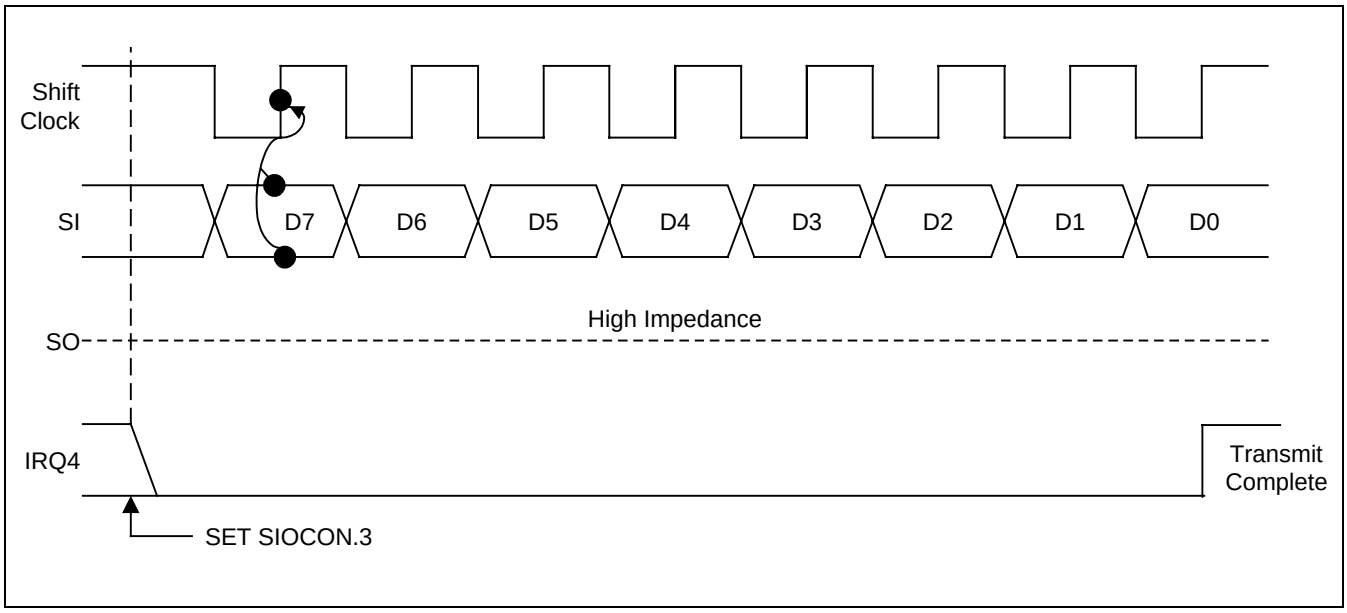


Figure 13-6. SIO Timing in Receive-Only Mode (Rising edge start)

PROGRAMMING TIP — Use Internal Clock to Transmit and Receive Serial Data

1. The method that uses interrupt is used.

```
          •
          •
          DI                              ;  Disable All interrupts
          LD        P2CONL #03H           ;  P2.2–P2.0 are selected to alternative function for
                                          ;  SI, SO, SCK, respectively

          SB1
          LD        SIODATA, TDATA        ;  Load data to SIO buffer
          LD        SIOPS, #90H           ;  Baud rate = input clock(fxx)/[(144 + 1) x 2]
          LD        SIOCON, #2EH          ;  Internal clock, MSB first, transmit/receive mode
          SB0                             ;  Select Tx falling edges to start shift operation
                                          ;  Clear 3-bit counter and start shifting
                                          ;  Enable shifter and clock counter
                                          ;  Enable SIO interrupt and clear pending
          EI
          •
          •
          •
SIOINT    PUSH      RP0                   ;
          SRP0      #RDATA                ;
          SB1
          LD        R0,SIODATA            ;  Load received data to general register
          OR        SIOCON,#08H           ;  SIO restart
          AND       SIOCON,#11111110b     ;  Clear interrupt pending bit
          POP       RP0
          IRET
```

☞ PROGRAMMING TIP — Use Internal Clock to Transfer and Receive Serial Data (Continued)

2. The method that uses software pending check is used.

```
        •
        •
        •
        DI                              ;  Disable All interrupts

        SB1
        LD      SIODATA, TDATA          ;  Load data to SIO buffer
        LD      SIOPS, #90H             ;  Baud rate = input clock(fxx)/[(144 + 1) × 2]
        LD      SIOCON, #2CH            ;  Internal clock, MSB first, transmit/receive mode
                                        ;  Select falling edges to start shift operation
                                        ;  clear 3-bit counter and start shifting
                                        ;  Disable SIO interrupt and pending clear
        EI

SIOtest: LD     R6,SIOCON               ;  To check whether transmit and receive is finished
        BTJRF   SIOtest,R6.0            ;  Check pending bit
        NOP
        AND     SIOCON,#0FEH            ;  Pending clear by software
        LD      RDATA,SIODATA           ;  Load received data to RDATA
        •
        •
        •
        SB0
        •
        •
        •
```

14 UART(0/1)

OVERVIEW

The UART block has a full-duplex serial port with programmable operating modes: There is one synchronous mode and three UART (Universal Asynchronous Receiver/Transmitter) modes:

— Serial I/O with baud rate of fxx/(16 × (BRDATA+1))

— 8-bit UART mode; variable baud rate

— 9-bit UART mode; fxx/16

— 9-bit UART mode, variable baud rate

UART receive and transmit buffers are both accessed via the data register, UDATA0, is set 1, bank 1 at address E2H, UDATA1, is set 1, bank 1 at address FAH. Writing to the UART data register loads the transmit buffer; reading the UART data register accesses a physically separate receive buffer.

When accessing a receive data buffer (shift register), reception of the next byte can begin before the previously received byte has been read from the receive register. However, if the first byte has not been read by the time the next byte has been completely received, the first data byte will be lost.

In all operating modes, transmission is started when any instruction (usually a write operation) uses the UDATA0, UDATA1 register as its destination address. In mode 0, serial data reception starts when the receive interrupt pending bit (UARTPND.1, UARTPND.3) is "0" and the receive enable bit (UARTCON0.4, UARTCON1.4) is "1". In mode 1, 2, and 3, reception starts whenever an incoming start bit ("0") is received and the receive enable bit (UARTCON0.4, UARTCON1.4) is set to "1".

PROGRAMMING PROCEDURE

To program the UART0 modules, follow these basic steps:

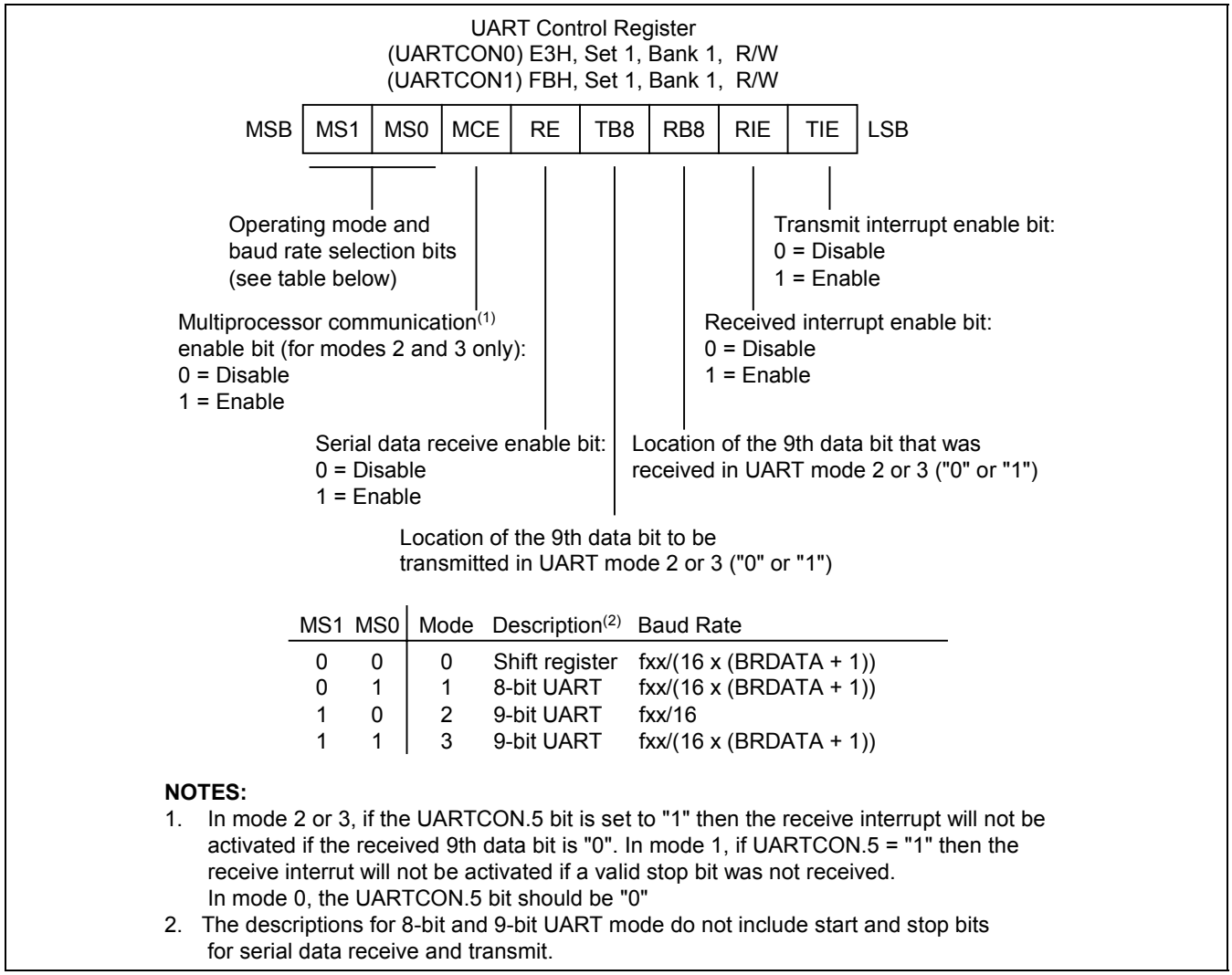
1. Configure P5.3 and P5.2 to alternative function RxD0, TxD0 for UART0 module by setting the P5CONL register to appropriatly value.
2. Load an 8-bit value to the UARTCON0 control register to properly configure the UART0 I/O module.
3. For interrupt generation, set the UART0 interrupt enable bit (UARTCON0.1 or UARTCON0.0) to "1".
4. When you transmit data to the UART0 buffer, writing data to UDATA0, the shift operation starts.
5. When the shift operation (transmit/receive) is completed, UART0 pending bit (UARTPND.1 or UARTPND.0) is set to "1" and an UART0 interrupt request is generated.

UART CONTROL REGISTER (UARTCON0, UARTCON1)

The control register for the UART is called UARTCON0 in set 1, bank 1 at address E3H, UARTCON1 in set 1, bank 1 at address FBH. It has the following control functions:

— Operating mode and baud rate selection
— Multiprocessor communication and interrupt control
— Serial receive enable/disable control
— 9th data bit location for transmit and receive operations (modes 2 and 3 only)
— UART transmit and receive interrupt control

A reset clears the UARTCON0, UARTCON1 value to "00H". So, if you want to use UART0, or UART1 module, you must write appropriate value to UARTCON0, UARTCON1.



MS1	MS0	Mode	Description(2)	Baud Rate
0	0	0	Shift register	fxx/(16 x (BRDATA + 1))
0	1	1	8-bit UART	fxx/(16 x (BRDATA + 1))
1	0	2	9-bit UART	fxx/16
1	1	3	9-bit UART	fxx/(16 x (BRDATA + 1))

NOTES:
1. In mode 2 or 3, if the UARTCON.5 bit is set to "1" then the receive interrupt will not be activated if the received 9th data bit is "0". In mode 1, if UARTCON.5 = "1" then the receive interrut will not be activated if a valid stop bit was not received. In mode 0, the UARTCON.5 bit should be "0"
2. The descriptions for 8-bit and 9-bit UART mode do not include start and stop bits for serial data receive and transmit.

Figure 14-1. UART Control Register (UARTCON0, UARTCON1)

UART INTERRUPT PENDING REGISTER (UARTPND)

The UART interrupt pending register, UARTPND is located in set 1, bank 1 at address E5H, it contains the UART0 data transmit interrupt pending bit (UARTPND.0), the receive interrupt pending bit (UARTPND.1), the UART1 data transmit interrupt pending bit (UARTPND.2), and the receive interrupt pending bit (UARTPND.3).

In mode 0, the receive interrupt pending flag UARTPND.1, UARTPND.3 is set to "1" when the 8th receive data bit has been shifted. In mode 1, 2, and 3, the UARTPND.1, UARTPND.3 bit is set to "1" at the halfway point of the stop bit's shift time. When the CPU has acknowledged the receive interrupt pending condition, the UARTPND.1, UARTPND.3 flag must then be cleared by software in the interrupt service routine.

In mode 0, the transmit interrupt pending flag UARTPND.0, UARTPND.2 is set to "1" when the 8th transmit data bit has been shifted. In mode 1, 2, or 3, the UARTPND.0, UARTPND.2 bit is set at the start of the stop bit. When the CPU has acknowledged the transmit interrupt pending condition, the UARTPND.0, UARTPND.2 flag must then be cleared by software in the interrupt service routine.

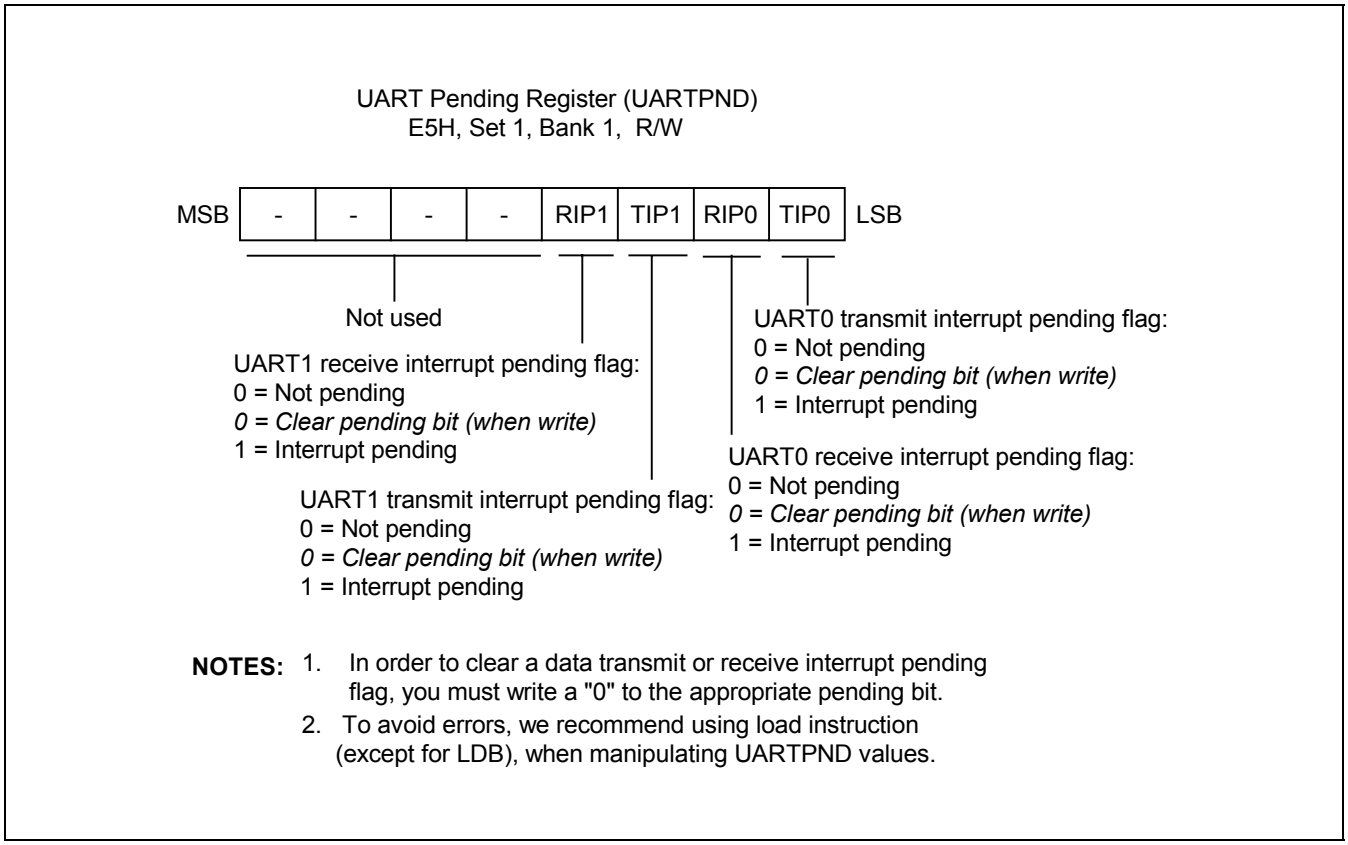


Figure 14-2. UART Interrupt Pending Register (UARTPND)

UART DATA REGISTER (UDATA0, UDATA1)

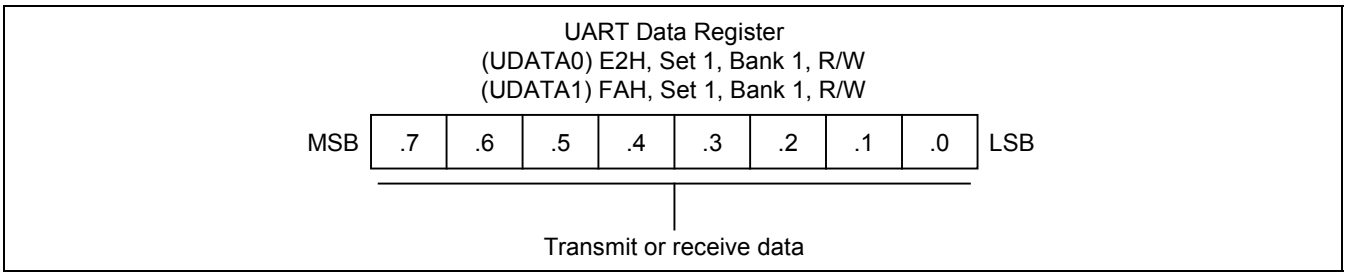


Figure 14-3. UART Data Register (UDATA0, UDATA1)

UART BAUD RATE DATA REGISTER (BRDATA0, BRDATA1)

The value stored in the UART0 baud rate register, BRDATA0, lets you determine the UART0 clock rate (baud rate). The value stored in the UART1 baud rate register, BRDATA1, lets you determine the UART1 clock rate (baud rate).

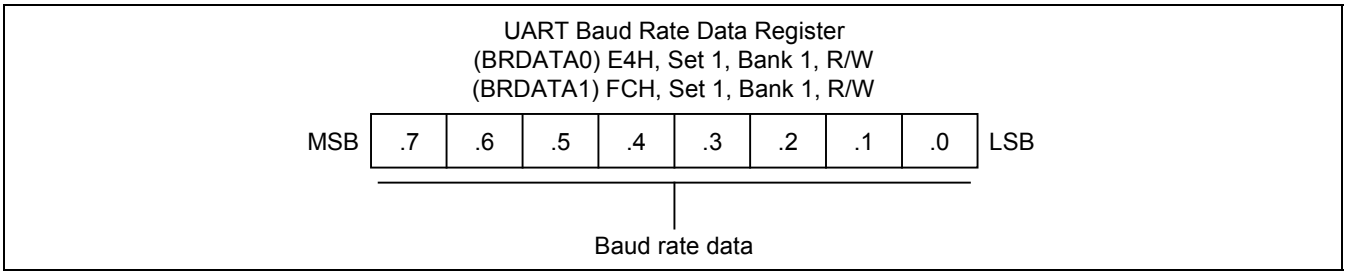


Figure 14-4. UART Baud Rate Data Register (BRDATA0, BRDATA1)

BAUD RATE CALCULATIONS (UART0)

Mode 0 Baud Rate Calculation

In mode 0, the baud rate is determined by the UART0 baud rate data register, BRDATA0 in set1, bank 1 at address E4H.

Mode 0 baud rate = fxx/(16 × (BRDATA0 + 1))

Mode 2 Baud Rate Calculation

The baud rate in mode 2 is fixed at the f_{OSC} clock frequency divided by 16:

Mode 2 baud rate = fxx/16

Modes 1 and 3 Baud Rate Calculation

In modes 1 and 3, the baud rate is determined by the UART0 baud rate data register, BRDATA0 in set 1, bank 1 at address E4H.

Mode 1 and 3 baud rate = fxx/(16 × (BRDATA0 + 1))

Table 14-1. Commonly Used Baud Rates Generated by BRDATA0, BRDATA1

Mode	Baud Rate	Oscillation Clock	BRDATA0, BRDATA1	
			Decimal	Hexdecimal
Mode 2	0.5 MHz	8 MHz	x	x
Mode 0	230,400 Hz	11.0592 MHz	02	02H
Mode 1	115,200 Hz	11.0592 MHz	05	05H
Mode 3	57,600 Hz	11.0592 MHz	11	0BH
	38,400 Hz	11.0592 MHz	17	11H
	19,200 Hz	11.0592 MHz	35	23H
	9,600 Hz	11.0592 MHz	71	47H
	4,800 Hz	11.0592 MHz	143	8FH
	62,500 Hz	10 MHz	09	09H
	9,615 Hz	10 MHz	64	40H
	38,461 Hz	8 MHz	12	0CH
	12,500 Hz	8 MHz	39	27H
	19,230 Hz	4 MHz	12	0CH
	9,615 Hz	4 MHz	25	19H

BLOCK DIAGRAM

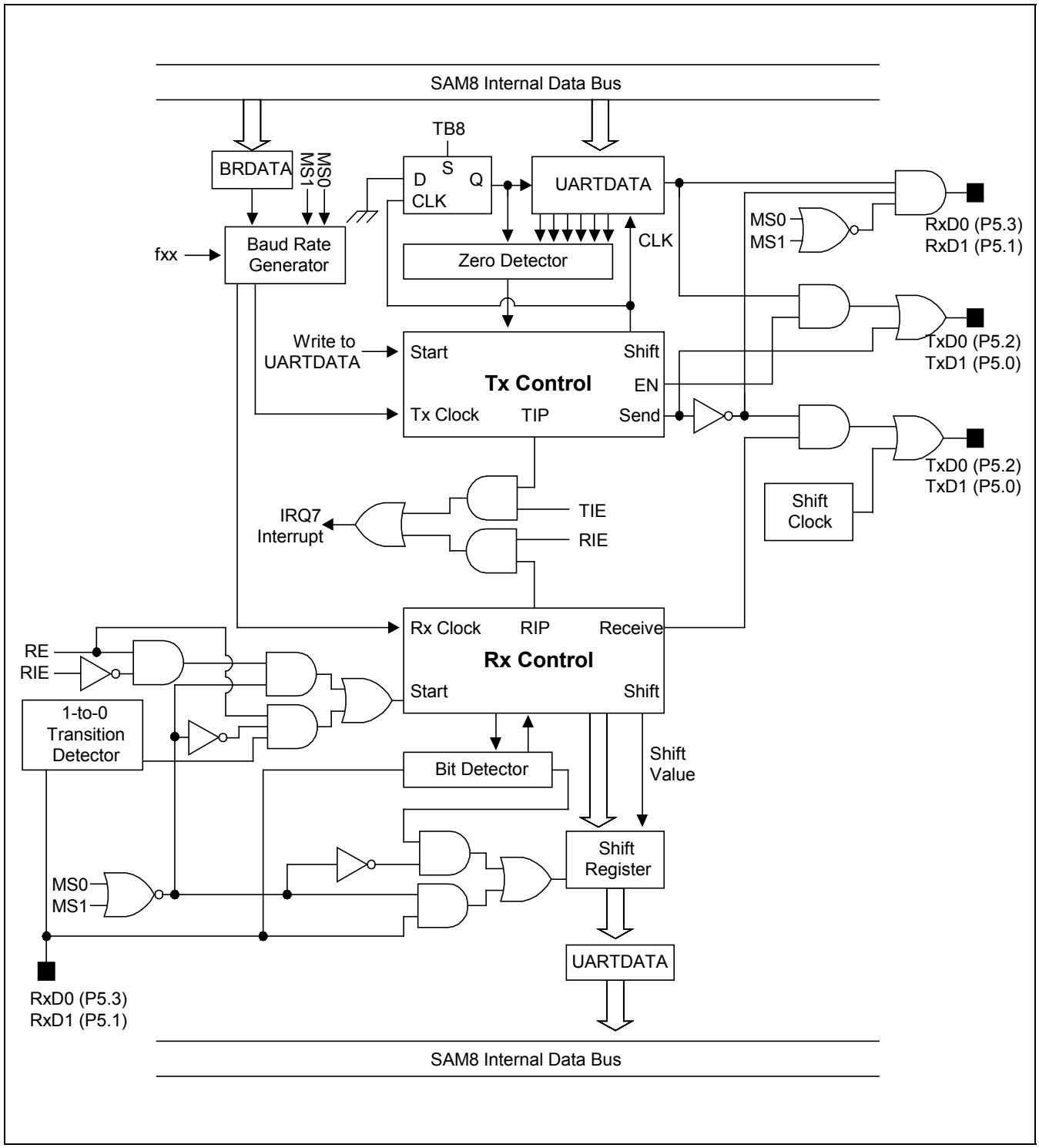


Figure 14-5. UART Functional Block Diagram

UART0 MODE 0 FUNCTION DESCRIPTION

In mode 0, UART0 is input and output through the RxD0 (P5.3) pin and TxD0 (P5.2) pin outputs the shift clock. Data is transmitted or received in 8-bit units only. The LSB of the 8-bit value is transmitted (or received) first.

Mode 0 Transmit Procedure

1. Select mode 0 by setting UARTCON0.6 and .7 to "00B".
2. Write transmission data to the shift register UDATA0 (E2H, set 1, bank 1) to start the transmission operation.

Mode 0 Receive Procedure

1. Select mode 0 by setting UATCON0.6 and .7 to "00B".
2. Clear the receive interrupt pending bit (UARTPND.1) by writing a "0" to UARTPND.1.
3. Set the UART0 receive enable bit (UARTCON0.4) to "1".
4. The shift clock will now be output to the TxD0 (P5.2) pin and will read the data at the RxD0 (P5.3) pin. A UART0 receive interrupt (IRQ7, vector F0H) occurs when UARTCON0.1 is set to "1".

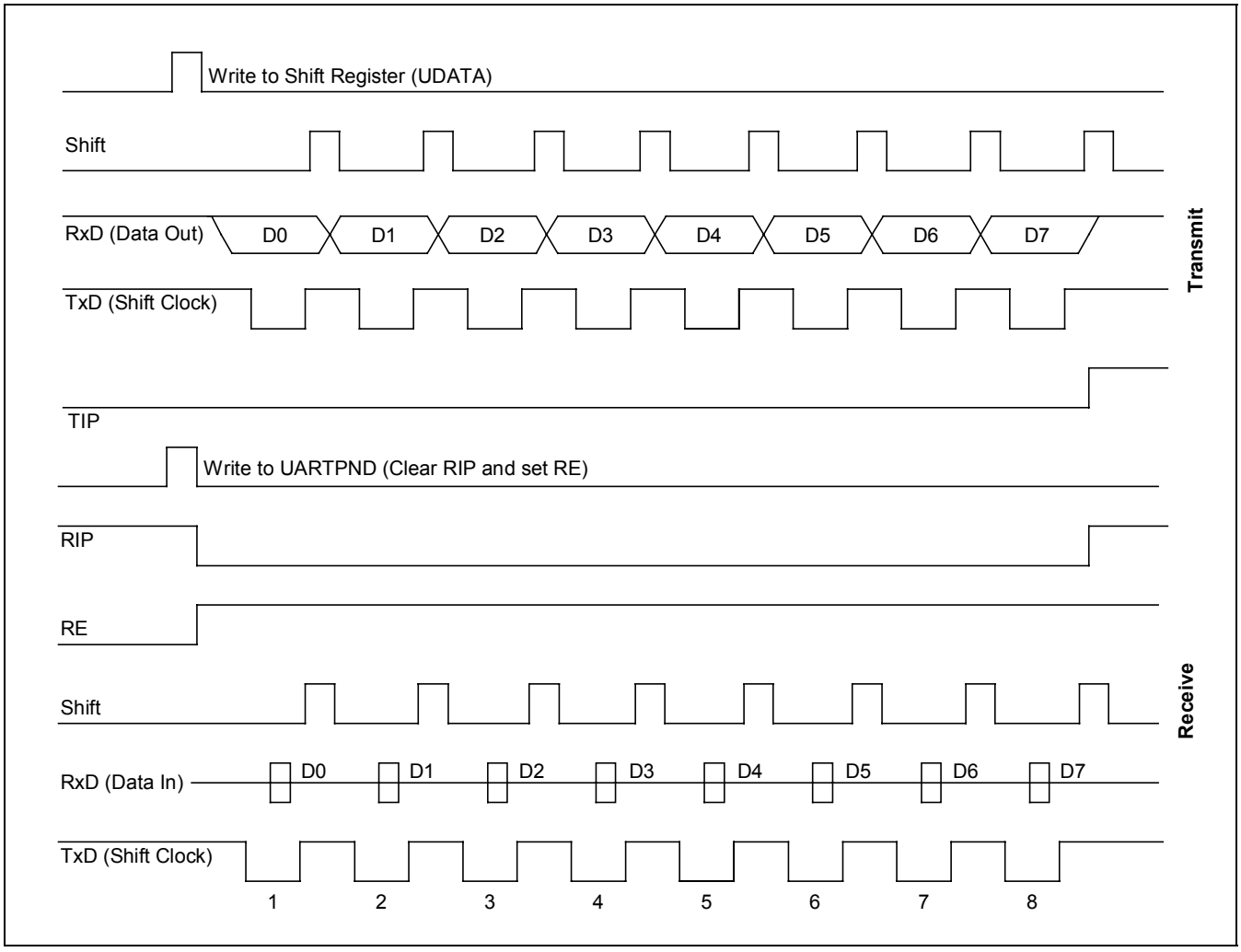


Figure 14-6. Timing Diagram for UART Mode 0 Operation

UART0 MODE 1 FUNCTION DESCRIPTION

In mode 1, 10-bits are transmitted through the TxD0 pin or received through the RxD0 pin. Each data frame has three components:

— Start bit ("0")
— 8 data bits (LSB first)
— Stop bit ("1")

When receiving, the stop bit is written to the RB8 bit in the UARTCON0 register. The baud rate for mode 1 is variable.

Mode 1 Transmit Procedure

1. Select the baud rate generated by setting BRDATA0.
2. Select mode 1 (8-bit UART0) by setting UARTCON0 bits 7 and 6 to '01B'.
3. Write transmission data to the shift register UDATA0 (E2H, set 1, bank 1). The start and stop bits are generated automatically by hardware.

Mode 1 Receive Procedure

1. Select the baud rate to be generated by setting BRDATA0.
2. Select mode 1 and set the RE (Receive Enable) bit in the UARTCON0 register to "1".
3. The start bit low ("0") condition at the RxD0 (P5.3) pin will cause the UART0 module to start the serial data receive operation.

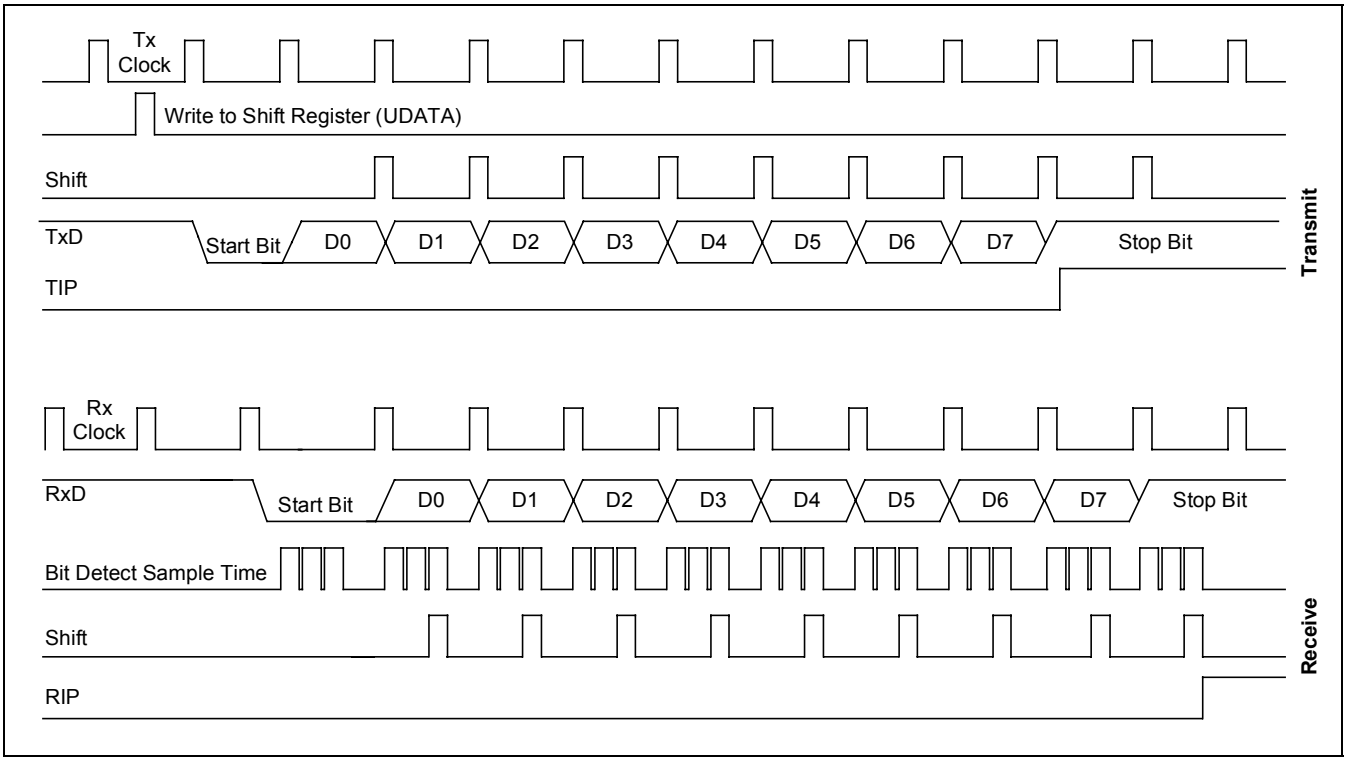


Figure 14-7. Timing Diagram for UART Mode 1 Operation

UART0 MODE 2 FUNCTION DESCRIPTION

In mode 2, 11-bits are transmitted (through the TxD0 pin) or received (through the RxD0 pin). Each data frame has four components:

— Start bit ("0")
— 8 data bits (LSB first)
— Programmable 9th data bit
— Stop bit ("1")

The 9th data bit to be transmitted can be assigned a value of "0" or "1" by writing the TB8 bit (UARTCON0.3). When receiving, the 9th data bit that is received is written to the RB8 bit (UARTCON0.2), while the stop bit is ignored. The baud rate for mode 2 is fosc/16 clock frequency.

Mode 2 Transmit Procedure

1. Select mode 2 (9-bit UART0) by setting UARTCON0 bits 6 and 7 to '10B'. Also, select the 9th data bit to be transmitted by writing TB8 to "0" or "1".
2. Write transmission data to the shift register, UDATA0 (E2H, set 1, bank 1), to start the transmit operation.

Mode 2 Receive Procedure

1. Select mode 2 and set the receive enable bit (RE) in the UARTCON0 register to "1".
2. The receive operation starts when the signal at the RxD pin goes to low level.

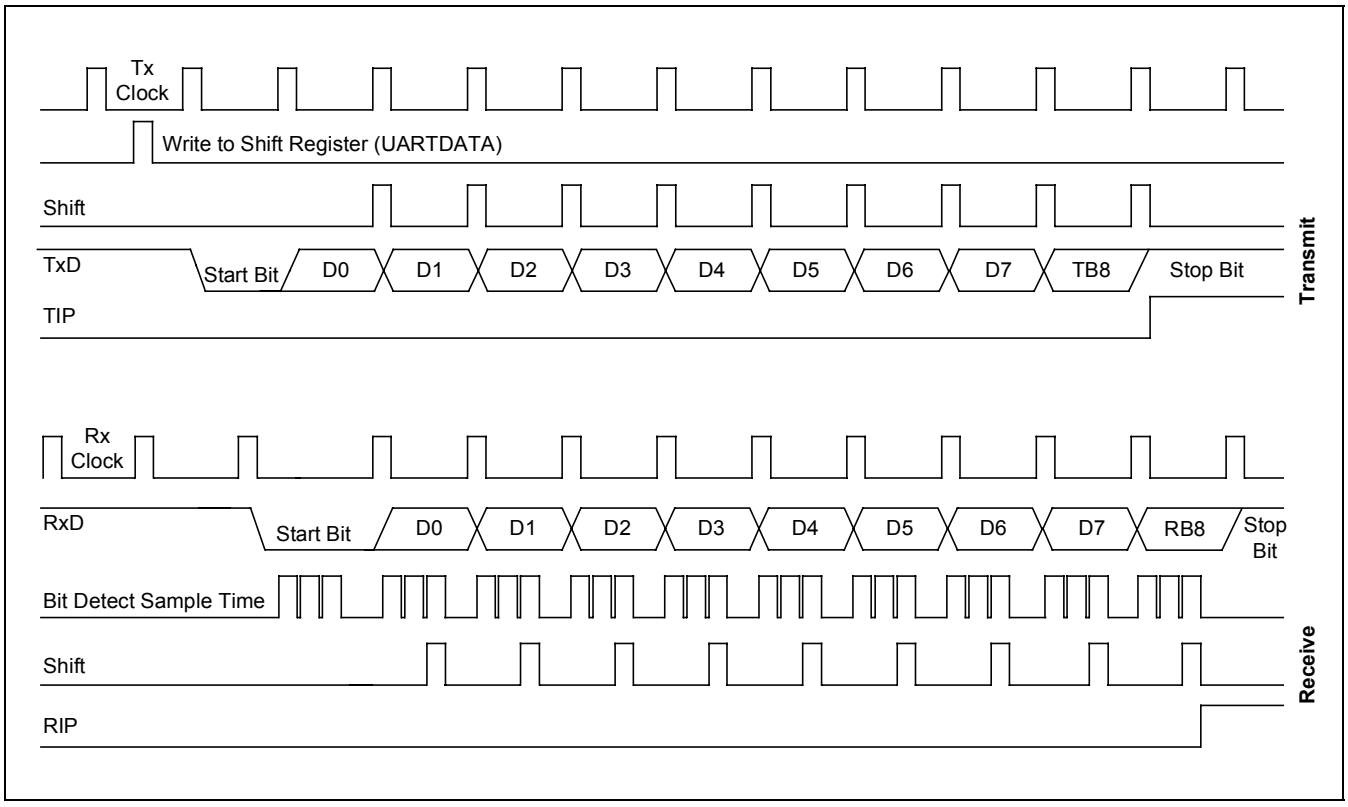


Figure 14-8. Timing Diagram for UART Mode 2 Operation

UART0 MODE 3 FUNCTION DESCRIPTION

In mode 3, 11-bits are transmitted (through the TxD0) or received (through the RxD0). Mode 3 is identical to mode 2 except for baud rate, which is variable. Each data frame has four components:

— Start bit ("0")
— 8 data bits (LSB first)
— Programmable 9th data bit
— Stop bit ("1")

Mode 3 Transmit Procedure

1. Select the baud rate generated by setting BRDATA0.
2. Select mode 3 operation (9-bit UART0) by setting UARTCON0 bits 6 and 7 to '11B'. Also, select the 9th data bit to be transmitted by writing UARTCON0.3 (TB8) to "0" or "1".
3. Write transmission data to the shift register, UDATA0 (E2H, set 1, bank 1), to start the transmit operation.

Mode 3 Receive Procedure

1. Select the baud rate to be generated by setting BRDATA0.
2. Select mode 3 and set the RE (Receive Enable) bit in the UARTCON0 register to "1".
3. The receive operation will be started when the signal at the RxD0 pin goes to low level.

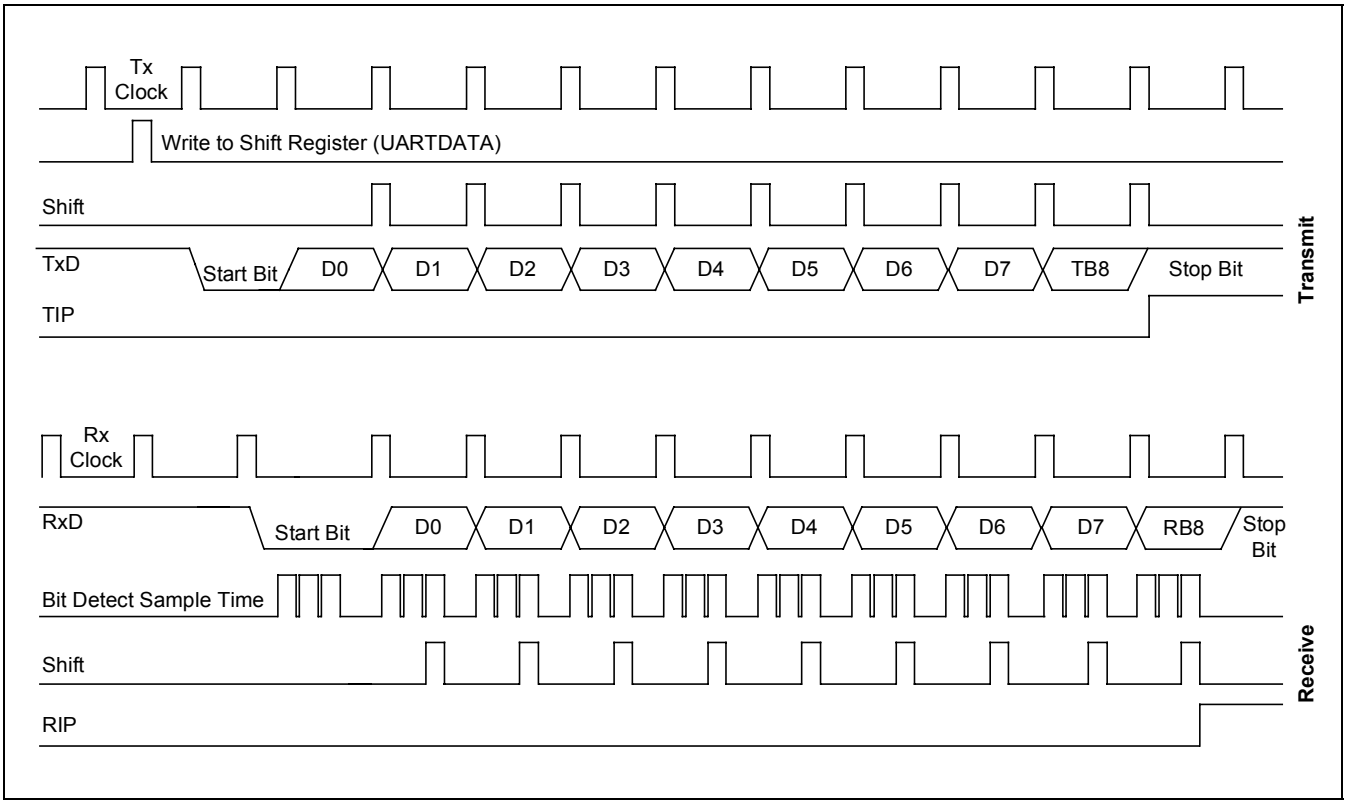


Figure 14-9. Timing Diagram for UART Mode 3 Operation

SERIAL COMMUNICATION FOR MULTIPROCESSOR CONFIGURATIONS

The S3C8-series multiprocessor communication feature lets a "master" S3C84BB/S3F84BB send a multiple-frame serial message to a "slave" device in a multi-S3C84BB/F84BB configuration. It does this without interrupting other slave devices that may be on the same serial line.

This feature can be used only in UART modes 2 or 3. In these modes 2 and 3, 9 data bits are received. The 9th bit value is written to RB8 (UARTCON0.2, or UARTCON1.2). The data receive operation is concluded with a stop bit. You can program this function so that when the stop bit is received, the serial interrupt will be generated only if RB8 = "1".

To enable this feature, you set the MCE bit in the UARTCON0/1 register. When the MCE bit is "1", serial data frames that are received with the 9th bit = "0" do not generate an interrupt. In this case, the 9th bit simply separates the address from the serial data.

Sample Protocol for Master/Slave Interaction

When the master device wants to transmit a block of data to one of several slaves on a serial line, it first sends out an address byte to identify the target slave. Note that in this case, an address byte differs from a data byte: In an address byte, the 9th bit is "1" and in a data byte, it is "0".

The address byte interrupts all slaves so that each slave can examine the received byte and see if it is being addressed. The addressed slave then clears its MCE bit and prepares to receive incoming data bytes.

The MCE bits of slaves that were not addressed remain set, and they continue operating normally while ignoring the incoming data bytes.

While the MCE bit setting has no effect in mode 0, it can be used in mode 1 to check the validity of the stop bit. For mode 1 reception, if MCE is "1", the receive interrupt will be issue unless a valid stop bit is received.

Setup Procedure for Multiprocessor Communications

Follow these steps to configure multiprocessor communications:

1. Set all S3C84BB/F84BB devices (masters and slaves) to UART mode 2 or 3.
2. Write the MCE bit of all the slave devices to "1".
3. The master device's transmission protocol is:
   - First byte: the address identifying the target slave device (9th bit = "1")
   - Next bytes: data (9th bit = "0")
4. When the target slave receives the first byte, all of the slaves are interrupted because the 9th data bit is "1". The targeted slave compares the address byte to its own address and then clears its MCE bit in order to receive incoming data. The other slaves continue operating normally.

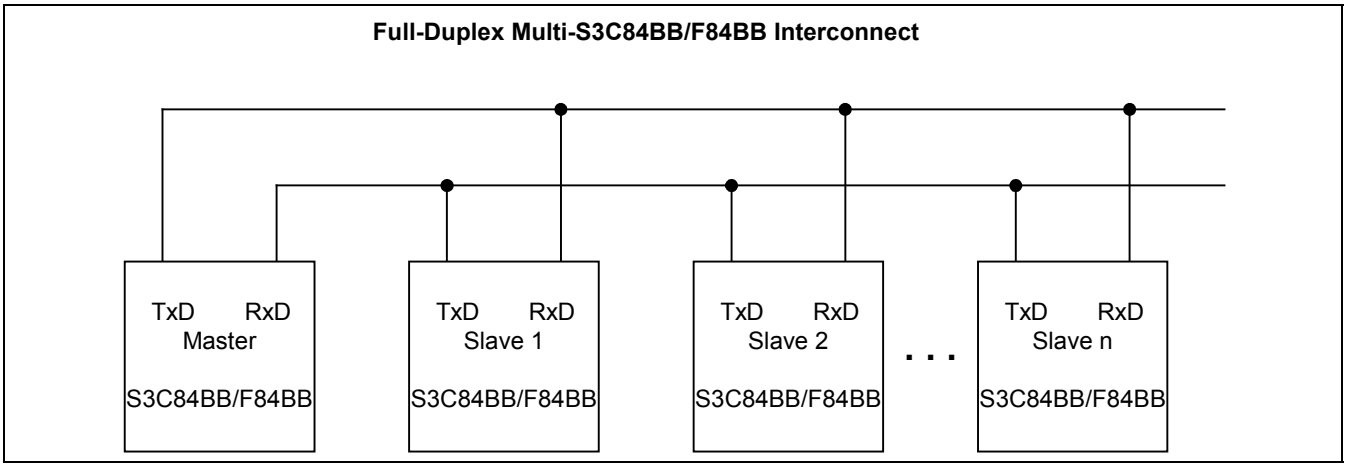


Figure 14-10. Connection Example for Multiprocessor Serial Data Communications

15 10-BIT A/D CONVERTER

OVERVIEW

The 10-bit A/D converter (ADC) module uses successive approximation logic to convert analog levels entering at one of the eight input channels to equivalent 10-bit digital values. The analog input level must lie between the AV_{REF} and AV_{SS} values. The A/D converter has the following components:

— Analog comparator with successive approximation logic

— D/A converter logic (resistor string type)

— ADC control register, ADACON (set 1, bank 1, F7H, read/write, but ADCON.3 is read only)

— Eight multiplexed analog data input pins (ADC0–ADC7)

— 10-bit A/D conversion data output register (ADDATAH, ADDATAL)

— Internal AV_{REF} and AV_{SS}

FUNCTION DESCRIPTION

To initiate an analog-to-digital conversion procedure, at first, you must configure P7.0–P7.7 to analog input before A/D conversions because the P7.0 – P7.7 pins can be used alternatively as normal data input or analog input pins. To do this, you load the appropriate value to the P7CON.0 – P7CON.7 (for ADC0 – ADC7) register.
And you write the channel selection data in the A/D converter control register ADACON to select one of the eight analog input pins (ADCn, n = 0–7) and set the conversion start or enable bit, ADACON.0.
An 10-bit conversion operation can be performed for only one analog input channel at a time.
The read-write ADACON register is located in set 1, bank 1 at address F7H.

During a normal conversion, ADC logic initially sets the successive approximation register to 200H (the approximate half-way point of an 10-bit register). This register is then updated automatically during each conversion step. The successive approximation block performs 10-bit conversions for one input channel at a time. You can dynamically select different channels by manipulating the channel selection bit value (ADACON.6–4) in the ADACON register.

To start the A/D conversion, you should set the enable bit, ADACON.0. When a conversion is completed, ADACON.3, the end-of-conversion (EOC) bit is automatically set to 1 and the result is dumped into the ADDATAH, ADDATAL registers where it can be read. The ADC module enters an idle state. Remember to read the contents of ADDATAH and ADDATAL before another conversion starts. Otherwise, the previous result will be overwritten by the next conversion result.

NOTE

Because the ADC does not use sample-and-hold circuitry, it is important that any fluctuations in the analog level at the ADC0–ADC7 input pins during a conversion procedure be kept to an absolute minimum. Any change in the input level, perhaps due to circuit noise, will invalidate the result.

A/D CONVERTER CONTROL REGISTER (ADACON)

The A/D converter control register, ADACON, is located in set1, bank 1 at address F7H. ADACON is read-write addressable using 8-bit instructions only. But EOC bit, ADACON.3 is read only. ADACON has four functions:

— Bits 6–4 select an analog input pin (ADC0–ADC7).

— Bit 3 indicates the end of conversion status of the A/D conversion.

— Bits 2–1 select a conversion speed.

— Bit 0 starts the A/D conversion.

Only one analog input channel can be selected at a time. You can dynamically select any one of the eight analog input pins, ADC0–ADC7 by manipulating the 3-bit value for ADACON.6–ADACON.4

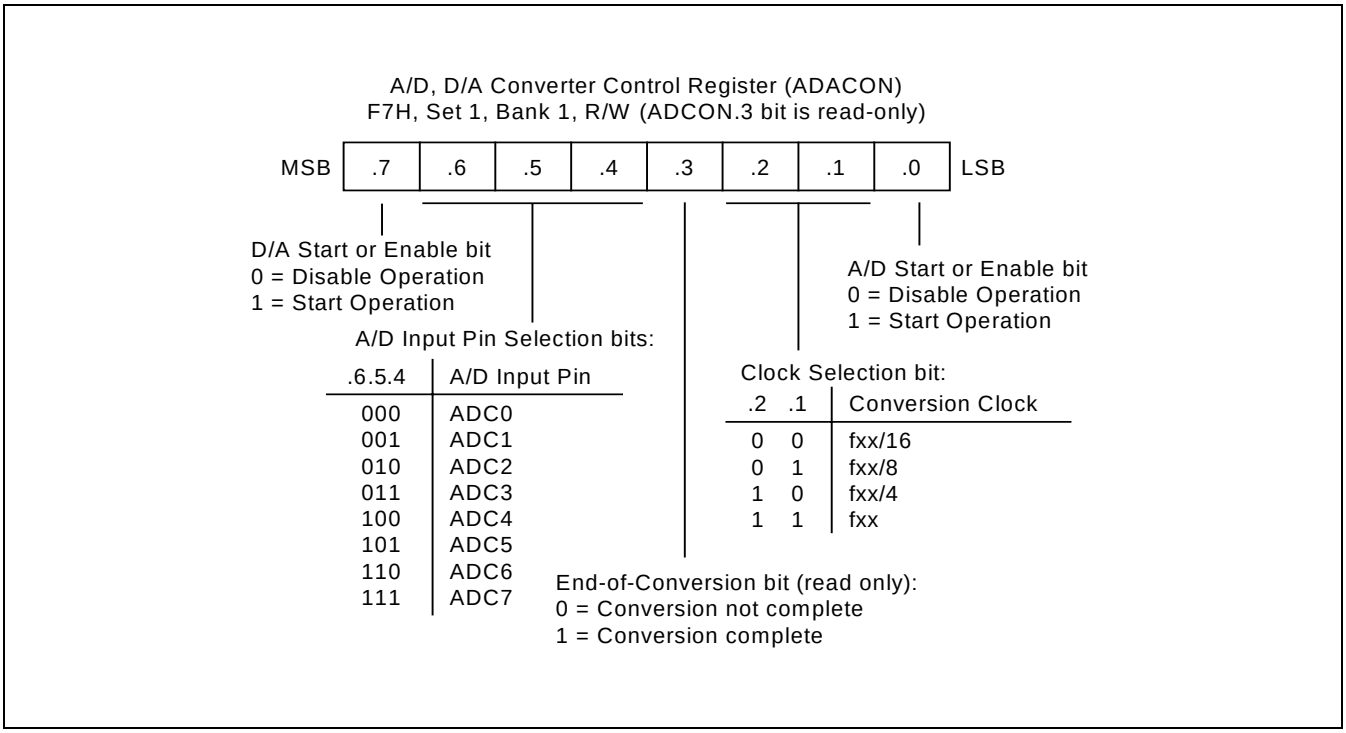


Figure 15-1. A/D Converter Control Register (ADACON)

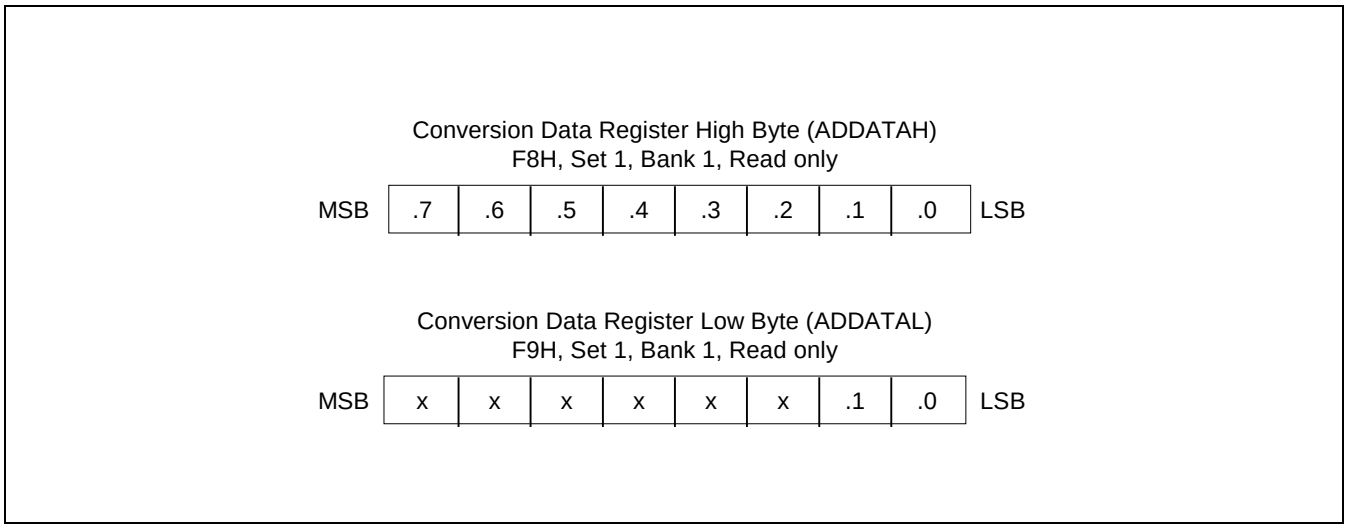


Figure 15-2. A/D Converter Data Register (ADDATAH, ADDATAL)

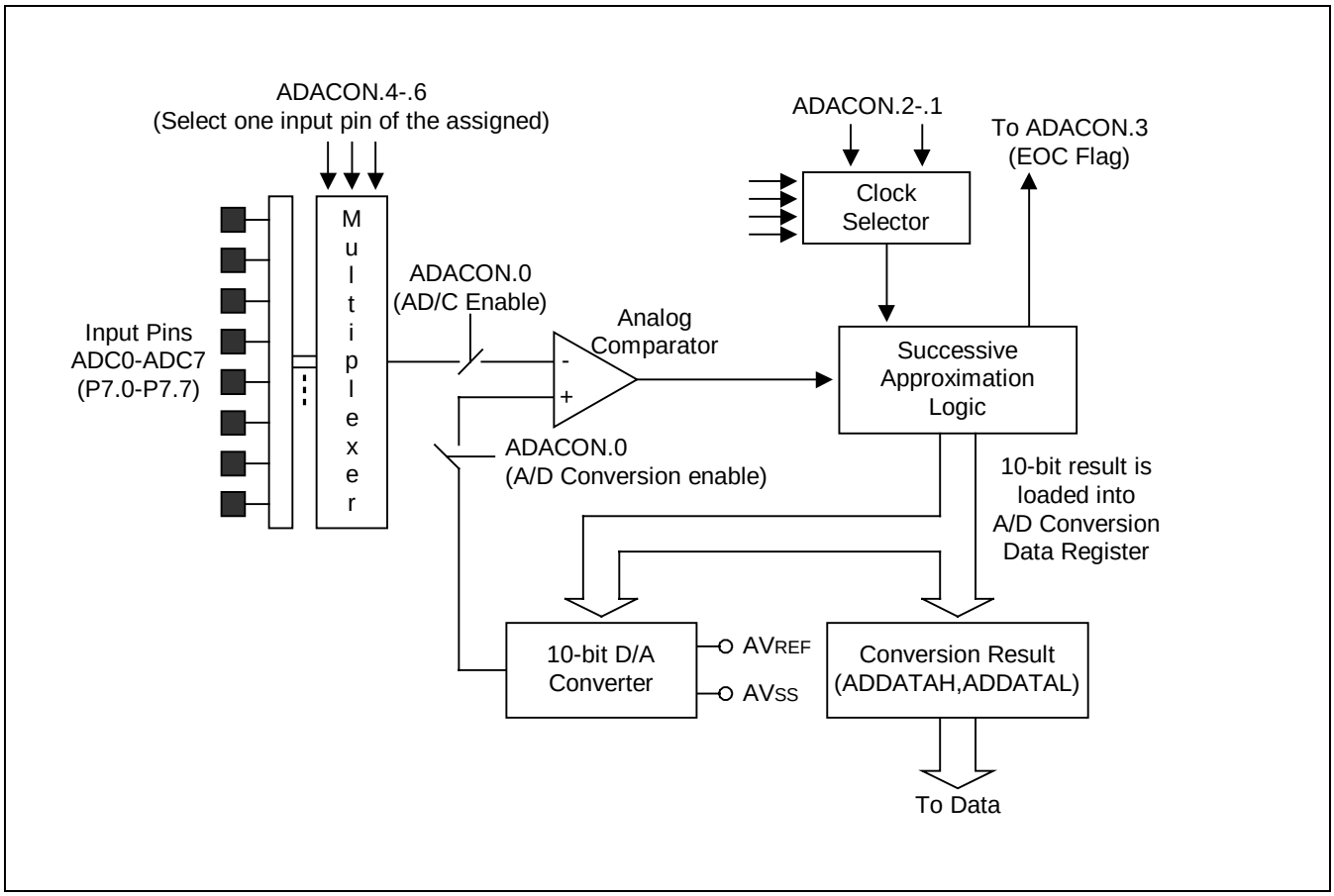


Figure 15-3. A/D Converter Circuit Diagram

INTERNAL REFERENCE VOLTAGE LEVELS

In the ADC function block, the analog input voltage level is compared to the reference voltage. The analog input level must remain within the range AV_{SS} to AV_{REF} (usually AV_{REF} = V_{DD}).

Different reference voltage levels are generated internally along the resistor tree during the analog conversion process for each conversion step. The reference voltage level for the first bit conversion is always 1/2 AV_{REF}.

CONVERSION TIMING

The A/D conversion process requires 4 steps (4 clock edges) to convert each bit and 10 clocks to step-up A/D conversion. Therefore, total of 50 clocks is required to complete a 10-bit conversion. With a 10 MHz CPU clock frequency, one clock cycle is 400 ns (4/fxx). If each bit conversion requires 4 clocks, the conversion rate is calculated as follows:

4 clocks/bit x 10-bits + step-up time (10 clock) = 50 clocks
50 clock x 400 ns = 20 μs at 10 MHz, 1 clock time = 4/fxx

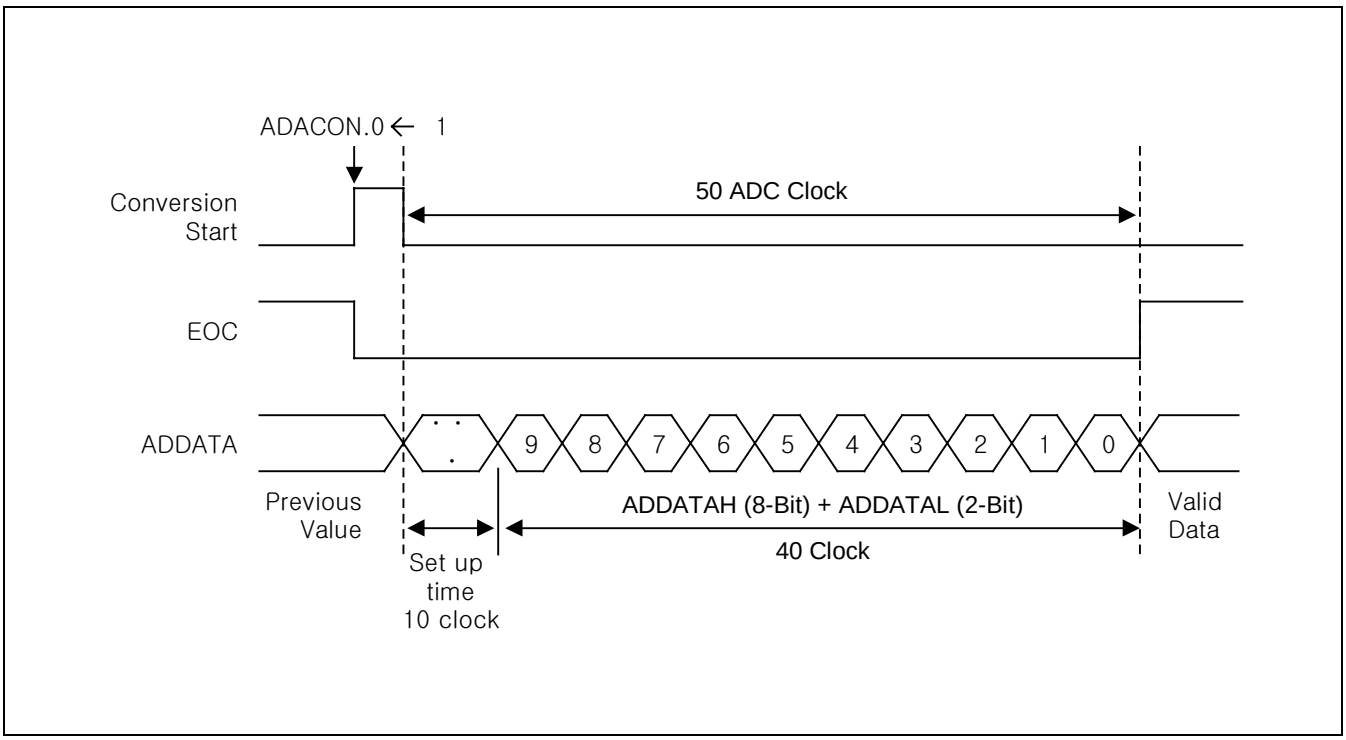
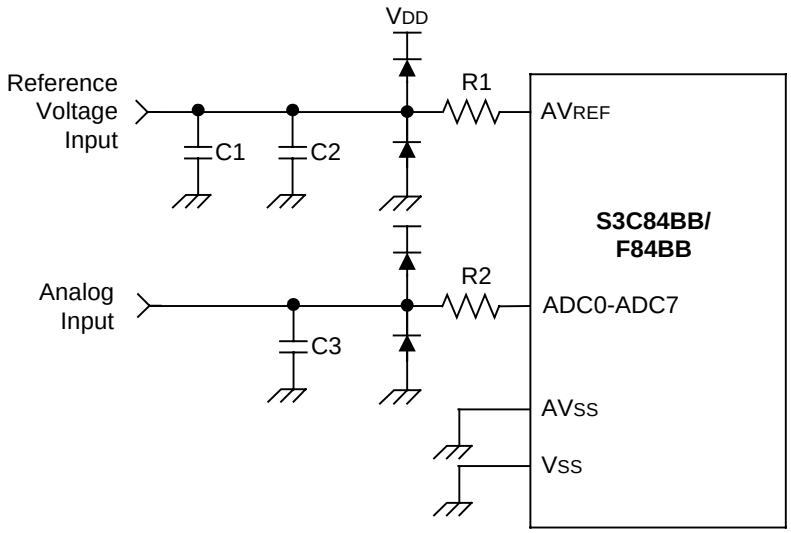


Figure 15-4. A/D Converter Timing Diagram

INTERNAL A/D CONVERSION PROCEDURE

1. Analog input must remain between the voltage range of AV_{SS} and AV_{REF}.
2. Configure P7.0–P7.7 for analog input before A/D conversions. To do this, you load the appropriate value to the P7CON (for ADC0–ADC7) register.
3. Before the conversion operation starts, you must first select one of the eight input pins (ADC0–ADC7) by writing the appropriate value to the ADACON register.
4. When conversion has been completed, (50 clocks have elapsed), the EOC, ADACON.3 flag is set to "1", so that a check can be made to verify that the conversion was successful.
5. The converted digital value is loaded to the output register, ADDATAH (8-bit) and ADDATAL (2-bit), then the ADC module enters an idle state.
6. The digital conversion result can now be read from the ADDATAH and ADDATAL register.



NOTE: The symbol "R1" signifies an offset resistor with a value of from 50 to 100 Ω.
If this resistor is omitted, the absolute accuracy will be maximum of 3LSBs.
C1=10µF, C2=100 to 1000pF, C3=100 to 1000pF, R1=50 to 100Ω, R2=10 to 1KΩ.

Figure 15-5. Recommended A/D Converter Circuit for Highest Absolute Accuracy

☞ PROGRAMMING TIP — Configuring A/D Converter

```
                •
                •
                SB0
                LD      P7CON,#11111111B    ;  P7.7–P7.0 A/D Input MODE
                •
                •

                SB1
                LD      ADACON,#00000001B   ;  Channel ADC0, Conversion start
AD0_CHK:        TM      ADACON,#00001000B   ;  A/D conversion end ? → EOC check
                JR      Z, AD0_CHK          ;  No

                LD      AD0BUFH,ADDATAH     ;  8-bit Conversion data
                LD      AD0BUFL,ADDATAL     ;  2-bit Conversion data
                SB0
                •
                •

                SB1
                LD      ADACON,#00110001B   ;  Channel AD3, fxx/16, Conversion start
AD3_CHK:        TM      ADACON,#00001000B   ;  A/D conversion end ? → EOC check
                JR      Z,AD3_CHK           ;  No

                LD      AD3BUFH,ADDATAH     ;  8-bit Conversion data
                LD      AD3BUFL,ADDATAL     ;  2-bit Conversion data
                SB0
                •
                •
```

16 8-BIT D/A CONVERTER

OVERVIEW

The S3C84BB/F84BB has 8-bit Digital-to-Analog converter with R-2R structure. This DAC(Digital-to-Analog) is used to generate analog voltage, V_{DA}, with 256 steps(2^8) The function is controlled by ADACON. To enable the converter, the ADACON.7 must be set to"1". To generate analog voltage(V_{DA}), load the appropriate value to DADATA. The level of analog voltage is determined by DADATA.

— D/A converter logic (resistor string type)

— 8-bit D/A conversion data register, DADATA (Set 1, bank1, F6H, read/write)

D/A CONVERTER CONTROL REGISTER (ADACON)

The Digital-to-Analog converter (DAC) control register, ADACON, is a 8-bit register located at F7H (set1, bank1). ADACON register controls to enable or disable the Digital-to-Analog converter (DAC).

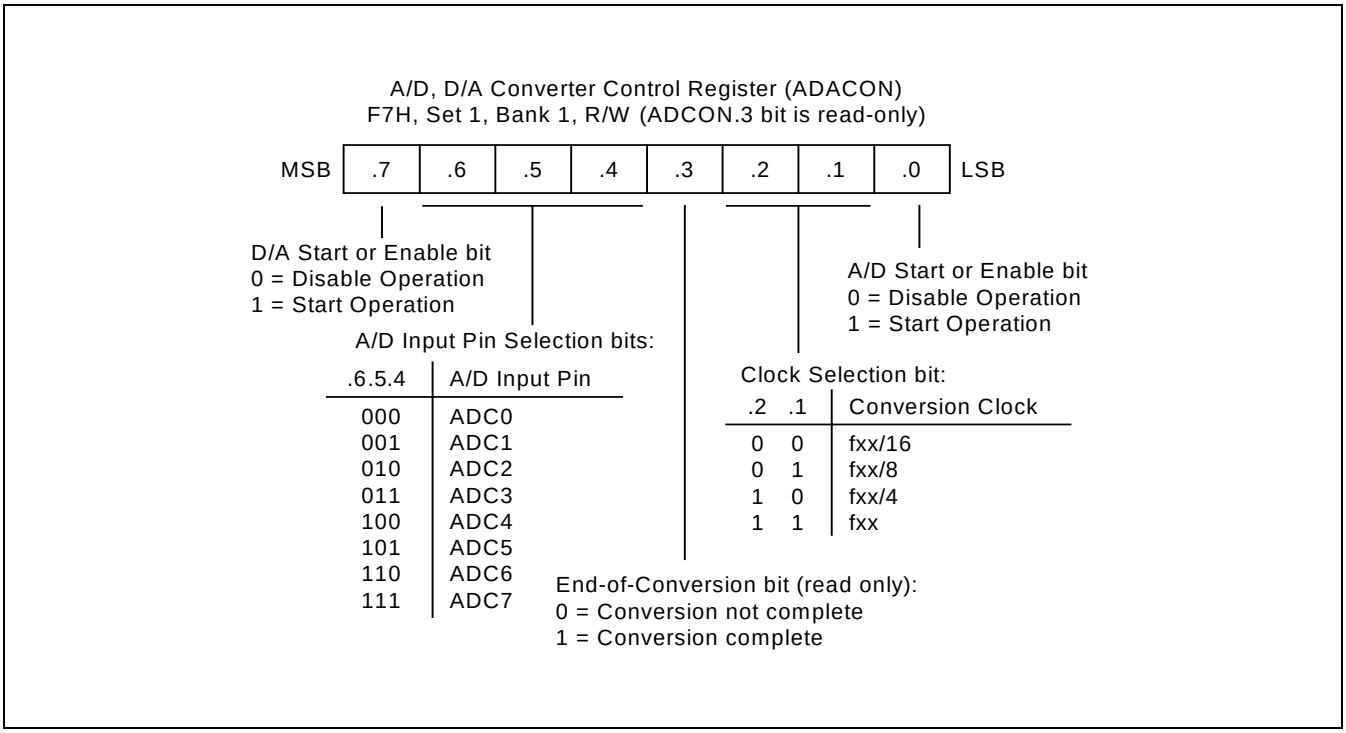


Figure 16-1. D/A Converter Control Register (ADACON)

D/A CONVERTER DATA REGISTER (DADATA)

DADATA, is a 8-bit read and write register located at F6H (set1, bank1). The DADATA specifies the digital data to generate analog voltage. ADACON values are set to logic “0” following RESET and the value disable DAC.

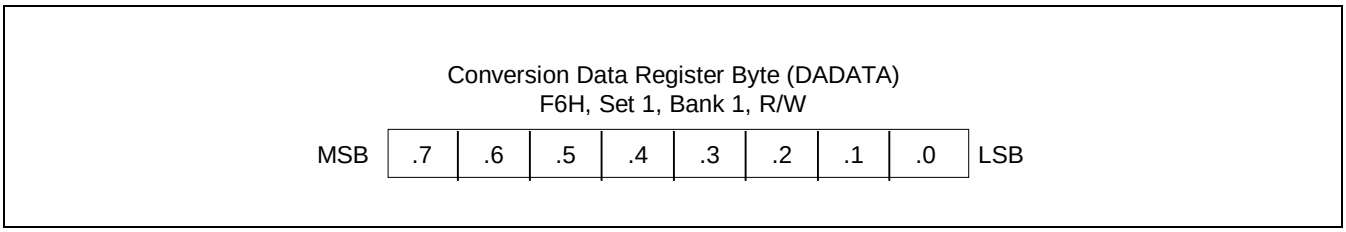


Figure 16-2. D/A Converter Data Register (DADATA)

These are the values be determined by setting just one-bit of DADATA.0-DADATA.7. The other values of DAOUT can be obtained with superimposition.

BLOCK DIAGRAM

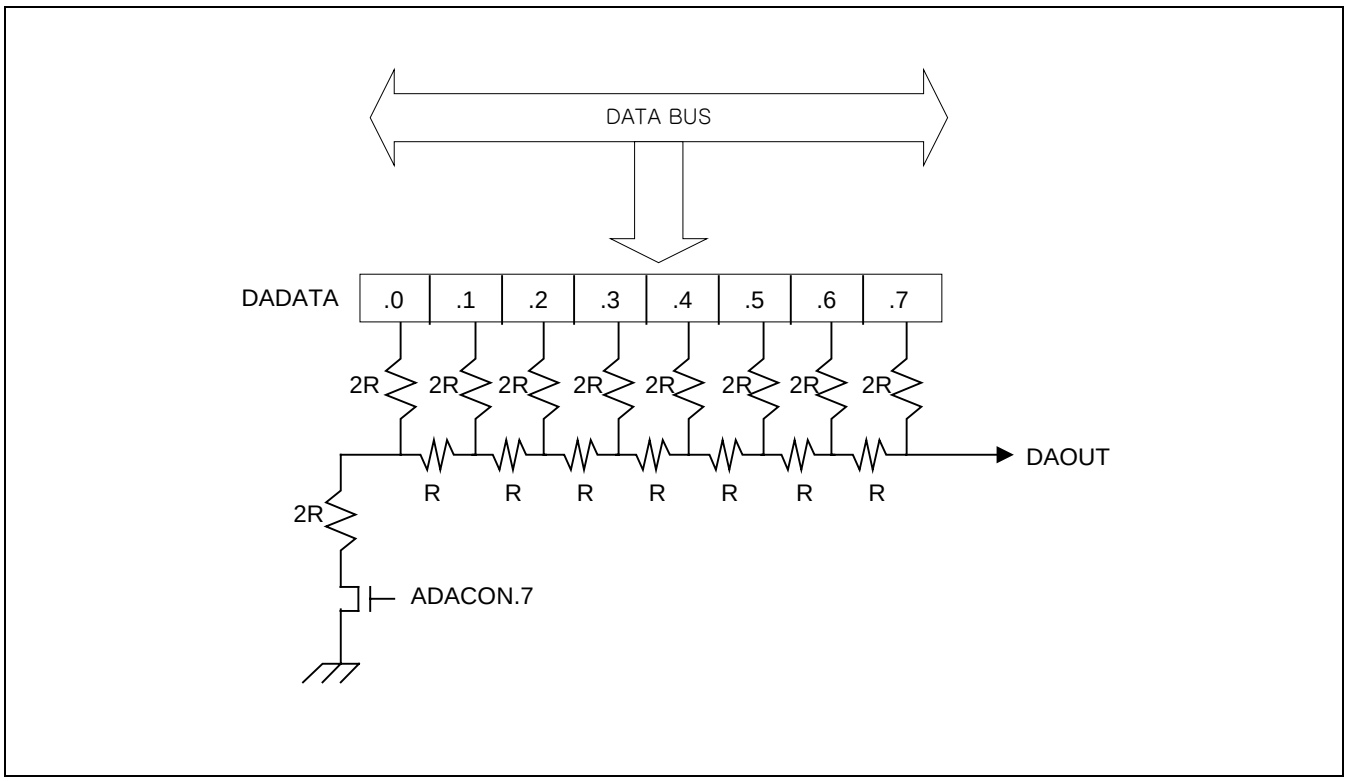


Figure 16-3. D/A Converter Circuit Diagram

Table 16-1. DADATA Setting to Generate Analog Voltage

DADATA7	DADATA6	DADATA5	DADATA4	DADATA3	DADATA2	DADATA1	DADATA0	V_{DAOUT}
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	$VDD/2^1$
0	1	0	0	0	0	0	0	$VDD/2^2$
0	0	1	0	0	0	0	0	$VDD/2^3$
0	0	0	1	0	0	0	0	$VDD/2^4$
0	0	0	0	1	0	0	0	$VDD/2^5$
0	0	0	0	0	1	0	0	$VDD/2^6$
0	0	0	0	0	0	1	0	$VDD/2^7$
0	0	0	0	0	0	0	1	$VDD/2^8$

NOTES

17 PATTERN GENERATION MODULE

OVERVIEW

PATTERN GENERATION FLOW

You can output up to 8-bit through P0.0-P0.7 by tracing the following sequence. First of all, you have to change the PGDATA into what you want to output. And then you have to set the PGCON to enable the pattern generation module and select the triggering signal. From now, bits of PGDATA are on the P0.0-P0.7 whenever the selected triggering signal occurs.

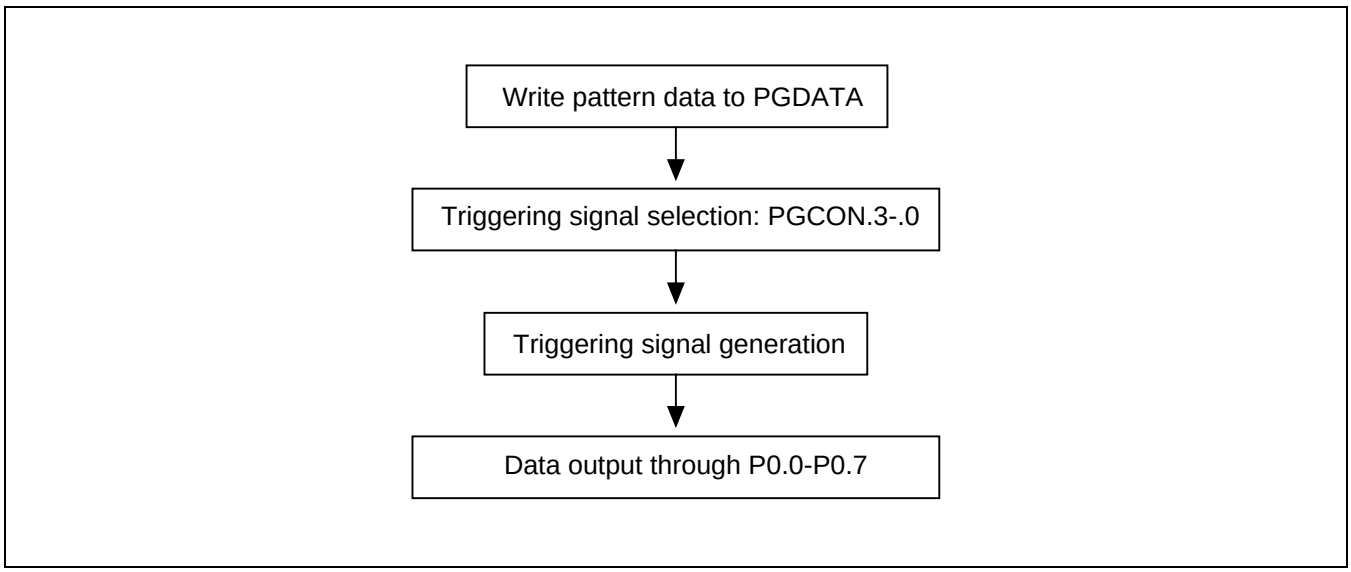


Figure 17-1. Pattern Generation Flow

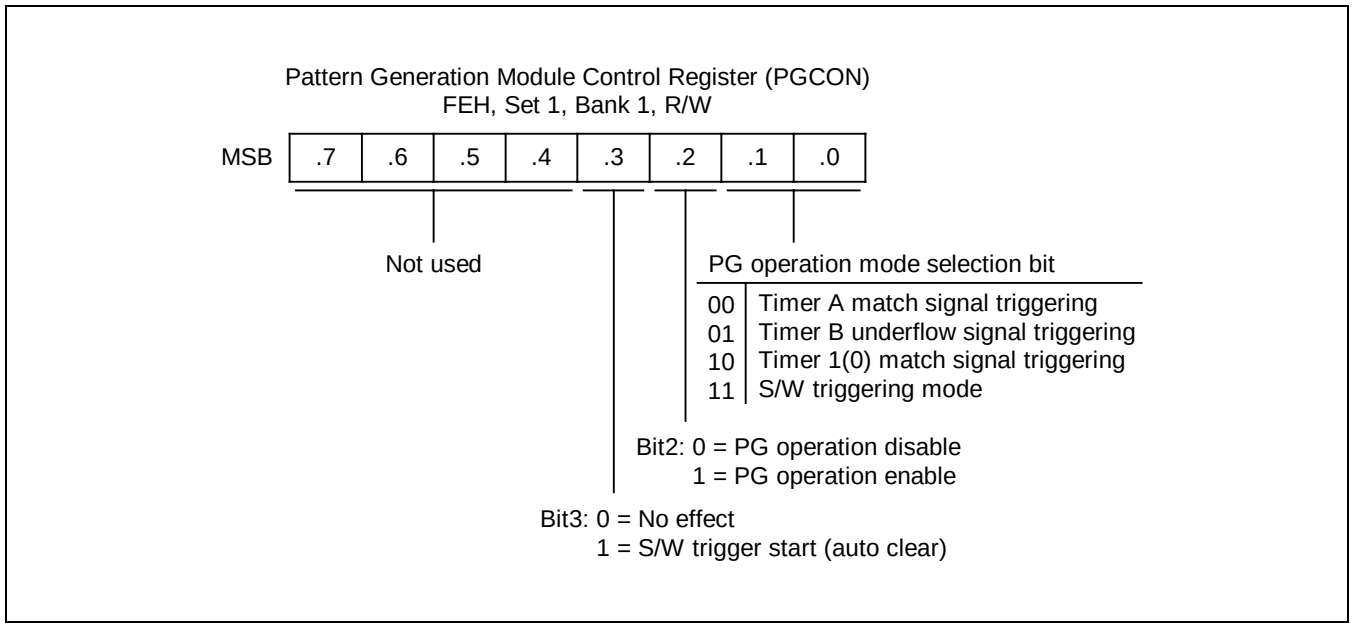


Figure 17-2. PG Control Register (PGCON)

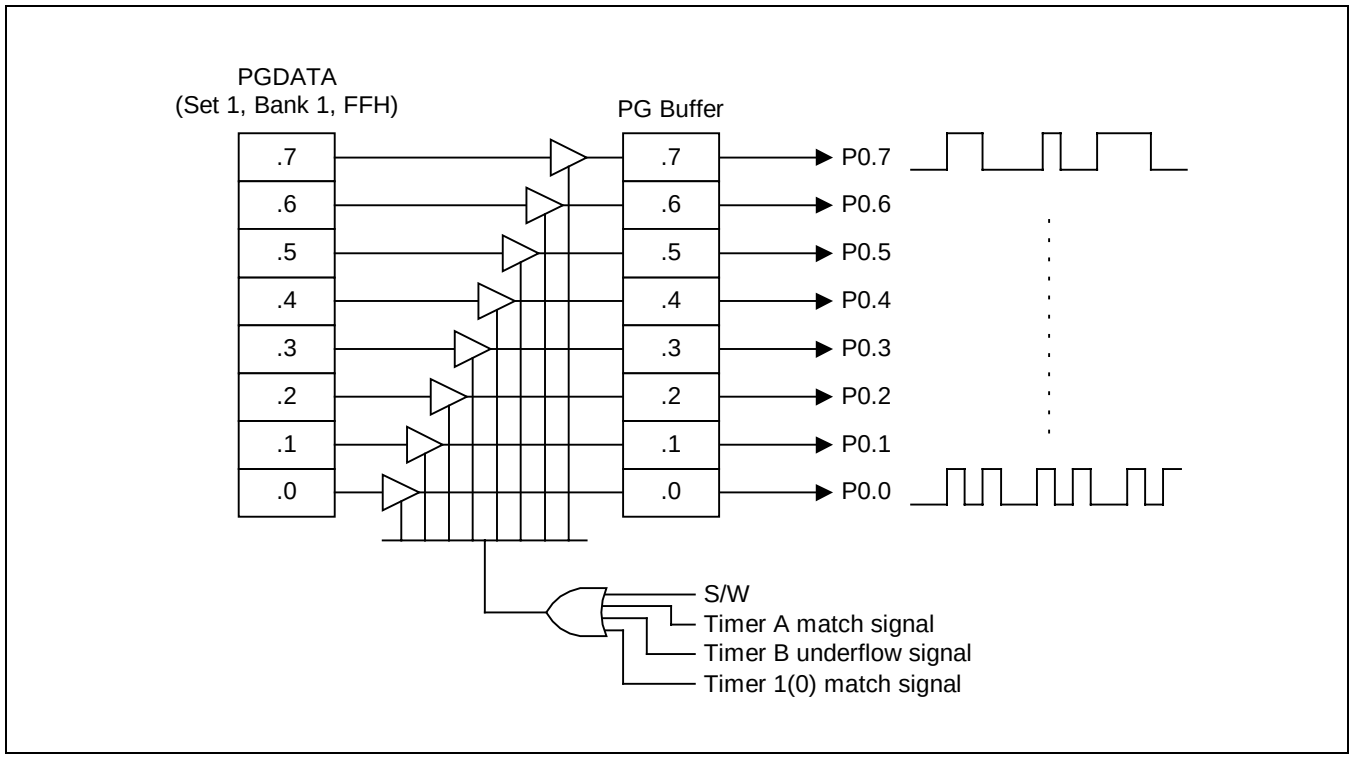


Figure 17-3. Pattern Generation Circuit Diagram

☞ Programming Tip — Using the Pattern Generation

```
        ORG     0000h

        ORG     0100h
INITIAL:
        SB0
        LD      SYM,#00h            ;  Disable Global interrupt → SYM
        LD      IMR,#01h            ;  Enable IRQ0 interrupt
        LD      SPH,#0h             ;  High byte of stack pointer → SPH
        LD      SPL,#0FFh           ;  Low byte of stack pointer → SPL
        LD      BTCON,#10100011b    ;  Disable Watch-dog
        LD      CLKCON,#00011000b   ;  Non-divided

        LD      P0CON,#11111111b    ;  Enable PG output

        EI

MAIN:
        NOP
        NOP

        SB1
        LD      PGDATA,#10101010b   ;  Setting pattern data
        OR      PGCON,#00001111b    ;  Triggering then pattern data are output
        SB0

        NOP
        NOP

        JR      T,MAIN

        .END
```

NOTES

18 EMBEDDED FLASH MEMEORY INTERFACE

OVERVIEW

The S3F84BB has an on-chip flash EEPROM instead of masked ROM. The flash EEPROM is accessed by serial data format and the type of a full flash, that is, a user can program the data in a flash memory area any time you wants. The flash EEPROM Endurance is 100 cycles for Erase/Program operation. The S3F84BB's embedded 64k-byte memory has several operating features below:

The S3F84BB has 6 pins used to read/write the flash memory, VDD/VSS, Reset, Test, SDAT and SCLK. The flash memory control block supports two kinds of program mode:

- Tool Program Mode
- User Program Mode

Tool Program Mode

The 6 pins are connected to a programming tool and programmed by Serial OTP/MTP Tools (SPW2plus single programmer, or GW-PRO2 gang programmer). The 12.5V programming power is supplied into the Vpp (Test) pin. The other modules except flash EEPROM module are at a reset state.

This mode doesn't support sector erase but chips erase and two protection modes (Hard lock protection/ Read protection).

User Program Mode

This mode supports sector erase and two protection modes.
The S3F84BB has the pumping circuit internally, therefore, 12.5V into Vpp (Test) pin is not needed. To program a flash memory in this mode several control registers will be used, refer to page 18-2. During programming/erasing flash memory, CPU will be held (30us) automatically.
Two signals, SCLK, SDAT should be made in User Program Mode by using FSCLK and FSDAT bit in FMCON register (address FDh in set1, bank1) in order to program a flash memory.
There are three kind functions – sector erase, programming, Option sector programming in User Program Mode.

Serial Interface Protocol Format

Serial interface protocol format consist of 3-byte address field and two and more byte data field.
In the 1st byte of address field, 4-bits are assigned for serial interface mode, the other 4-bits are assigned for address extension. (See Figure 18-4, and 18-5)

Data valid status of SCLK: High
Data Invalid status of SCLK: Low
START condition : SCLK = high, and SDAT = positive Edge
STOP condition : SCLK = high, and SDAT = negative Edge

Table 18-1. Command in User Program Mode

…	1st Byte				2nd Byte	3rd Byte	…	Available Program Mode
Mode	**REG/ MEMB**	**MODE (M1-M0)**	**ADDRESS (A19-A16)**	**R/WB**	**ADDRESS (A15-A8)**	**ADDRESS (A7-A0)**	**DATA (D7-D0)**	
Bit n	B23	B22-B21	B20-B17	B16	B15-B8	B7-B0		
Program	0	11b	xxxxb	0	xxh	xxh	xxh	Tool,User Program Mode
Hard Lock Protection	1	11b	0000b	0/1	---0, 1110b	--11, 1110b	----, --0-b	Tool,User Program Mode
Read Protection	1	11b	0000b	0/1	---0, 1110b	--11, 1111b	----, 0---b	Tool,User Program Mode
Sector Erase	0	10b	xxxxb	0	xxh	xxh	----, ----b	User Program Mode only

FLASH MEMORY CONTROL REGISTERS

Flash Memory Control Register

FMCON register is available only in user program mode to program some data to the flash memory.

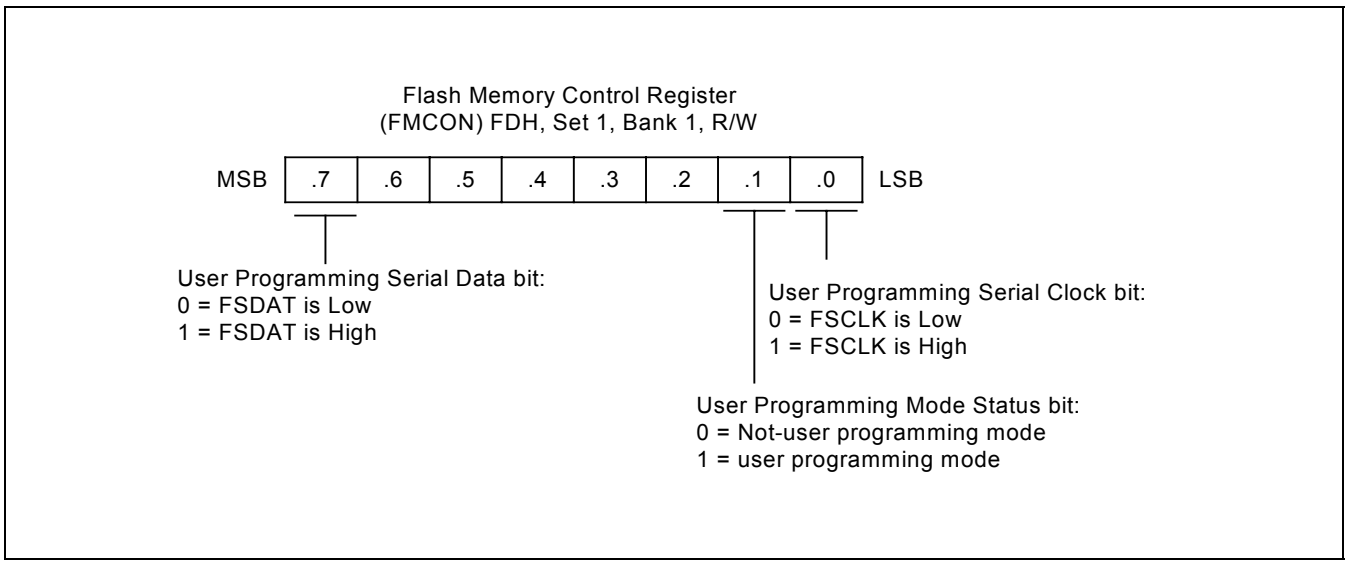


Figure 18-1. Flash Memory Control Register (FMCON)

Flash Memory User Programming Enable Register

RAM Address (00H) of page 8 is used as Flash Memory Enable Register. This location can be addressed by 1-bit or 8-bit instructions.
After reset, the user-programming mode is disabled, because the value of FMUSR is “00000000b’.
If necessary, you can use the user programming mode by setting the value of FMUSR is “10100101”.

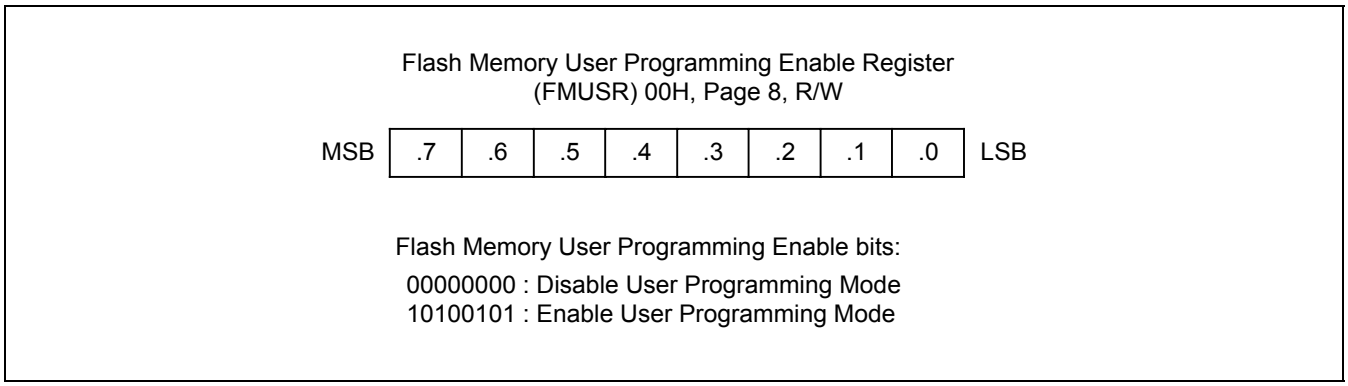


Figure 18-2. Flash Memory User Programming Enable Register (FMUSR)

The program procedure in User program Mode

1. Set Flash Memory Control Register (FMCON.1) properly to access flash memory
2. Clear FSCLK (FMCON.0) bit, FSDAT (FMCON.7) bit for the initialization of SCLK and SDAT signals
3. Enter into Sector Program Mode with instructions of “LD PP, #88H”,”LD 00H,#0A5H” orderly.
4. Make SCLK, SDAT signals for start condition with controlling FSCLK, FSDAT bits in FMCON register.
5. Make SCLK, SDAT signals for 3-byte address field with controlling FSCLK, and FSDAT bits in FMCON register.
6. Make SCLK, SDAT signals for data field with controlling FSCLK, and FSDAT bits in FMCON register.
7. Make SCLK, SDAT signals for 1-byte dummy data with controlling FSCLK, and FSDAT bits in FMCON register.
8. Make SCLK, SDAT signals for stop condition by controlling FSCLK, and FSDAT bits in FMCON register.
9. Release User Program Mode with instruction of “LD PP, #88H”, and ”LD 00H, #00H” orderly.

SECTOR ERASE

User can erase a flash memory partially by using sector erase function only in User Program Mode.
The only unit of flash memory to be erased and written in User Program Mode is called sector.
S3F84BB has 120 sectors to be erased written in flash memory. Sectors have all 512-byte sizes as program memory areas. Sector Erase is not supported in Tool Program Modes (MDS mode).
Minimum 2ms to maximum 100ms delay time for erase is required after setting sector address.

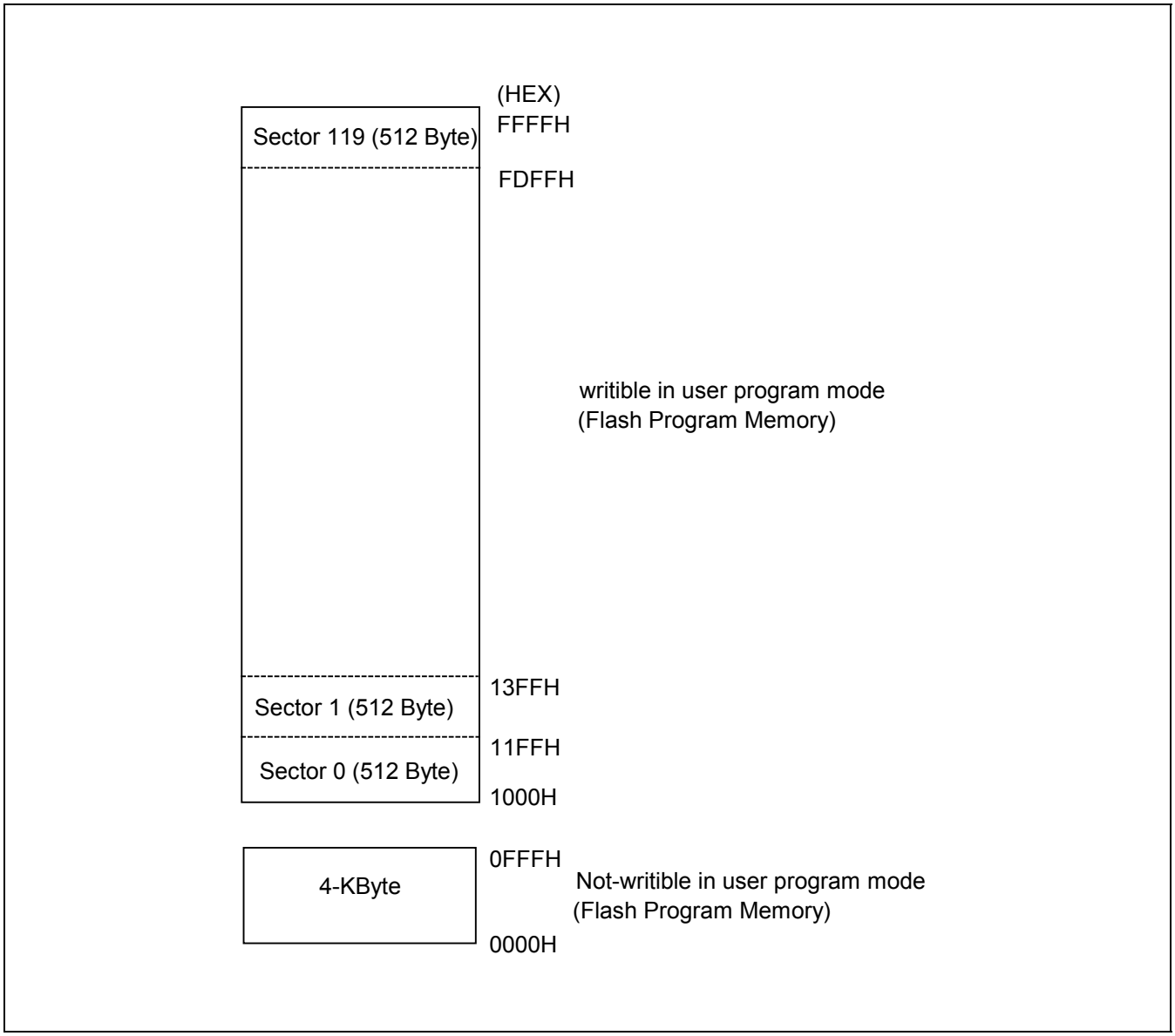


Figure 18-3. Sectors in User Program Mode

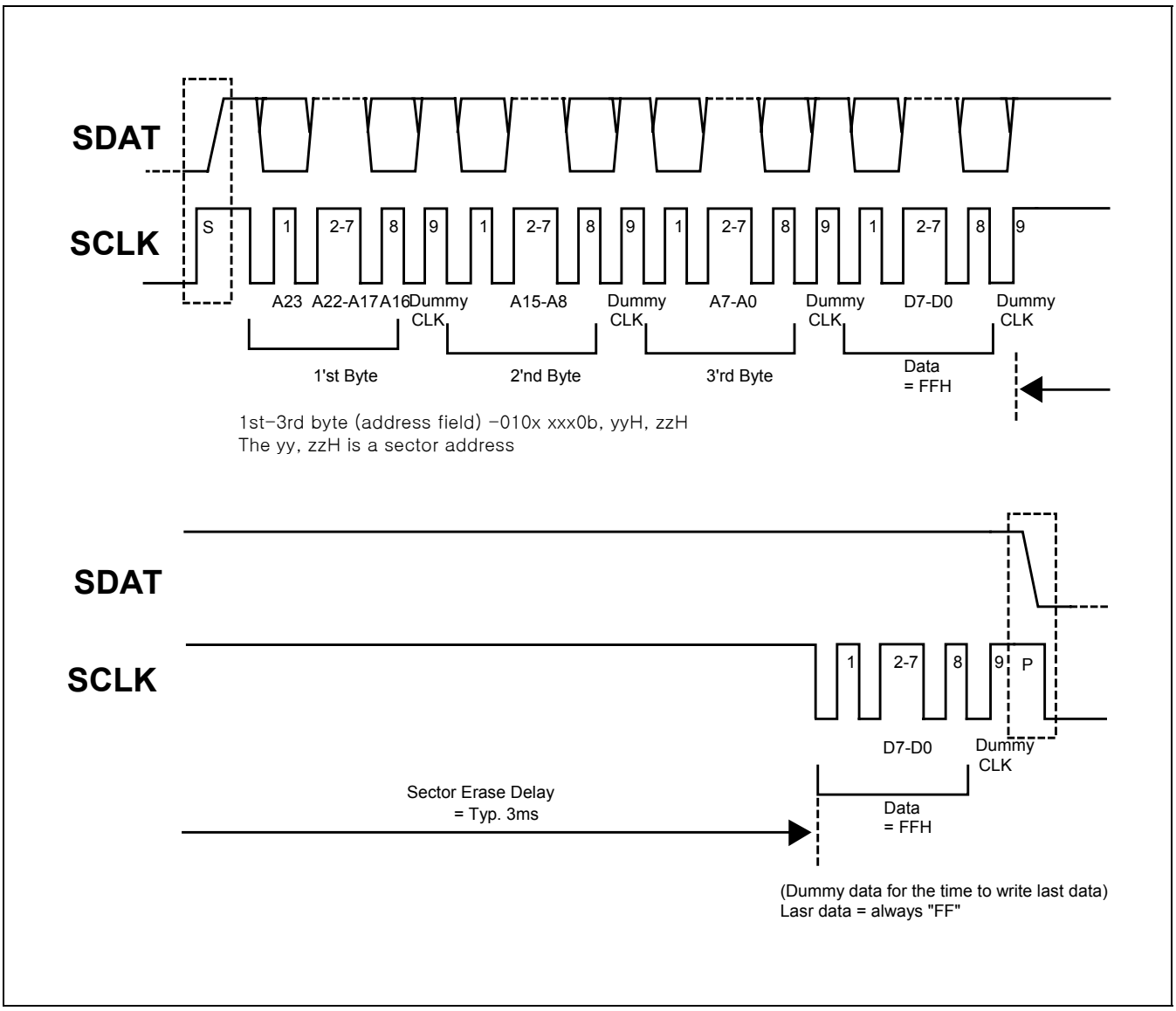


Figure 18-4. Sectors Erase Wave Form

☞PROGRAMMING TIP — Sector Erase

```
                SB1
                CLR     FMCON                   ; clear register
                SB0

                LD      CLKCON, #00H            ; cpu clock is 16-divide

LOOPE:          NOP
                LD      PP, #88H                ;
                LD      00H,#0A5H               ; User Program mode enable
                LD      PP,#00H                 ;

                SB1
ENT:            OR      FMCON,#00000010B        ; flag enable
                TM      FMCON,#00000010B        ; flag check
                JR      Z, ENT
                SB0

SPGM:           SB1                             ; Start
                OR      FMCON,#00000001B        ; SCLK=1
                OR      FMCON,#10000000B        ; SDAT=1
                SB0

                LD      R8,#01000000B           ; 1'st Byte (Sector Erase mode)
                CALL    PGM
                LD      R8,#00010000B           ; 2nd Byte, Address=1000H (Sector 0)
                CALL    PGM
                LD      R8,#00000000B           ; 3rd Byte
                CALL    PGM

                LD      R15,#0EBH               ; delay for typical 3ms when 10MHz oscillator used
DELAY:          DJNZ    R15,DELAY               ; ((1/(10MHz/16))x8cycle) x 235 = 3.008 [ms]

                LD      R8,#0FFH                ; dummy data
                CALL    PGM

                SB1
                AND     FMCON,#01111111B        ; SDAT=0
                AND     FMCON,#11111110B        ; SCLK=0, Stop
                SB0

                LD      PP,#88H
                LD      00H,#00H                ; User Program Mode Disable
                LD      PP,#00H
```

☞PROGRAMMING TIP — Sector Erase (Continued)

```
            SB1
REL:        AND     FMCON,#11111101B    ; flag disable
            TM      FMCON,#00000010B    ; flag check
            JR      NZ, REL
            SB0

            JP      END_SYM             ; END

PGM:        SB1
            AND     FMCON,#11111110B    ; SCLK=0

            CALL    WAIT

            LD      R9, #08H            ; Rotate time

PGMB:       RL      R8                  ; msb -> lsb
            LDB     FMCON.7,R8          ; FMCON.7 ← R8.0
            OR      FMCON,#00000001B    ; SCLK=1
            AND     FMCON,#11111110B    ; SCLK=0
            DJNZ    R9, PGMB

            OR      FMCON,#10000000B    ; SDAT=1
            OR      FMCON,#00000001B    ; SCLK=1
            SB0
            RET

WAIT:       NOP
            NOP
            LD      R15, #0FFH          ; 00H  <- FFH
LOOP0:      DJNZ    R15, LOOP0
            RET

END_SYM:    NOP


            .END
```

PROGRAMMING

A flash memory is programmed in one byte unit after sector erase.
The write operation of programming starts at a falling edge of dummy clock when a start address and data have been transmitted, and finishes at a falling edge of last SCLK for next data transmission.
The next data to write is transmitted during the previous data is writing. So, S3F84BB has 8-bit buffer register to write data to flash cell and shift register to receive the next data to be written.
The address of next data increments automatically at a dummy clock after previous data has been transmitted.
Dummy data (FFh) is required after transmission of the last data because the time to write the last data to flash cell is needed.
Programming finished when stop condition occurs after the Dummy clock has been transmitted.

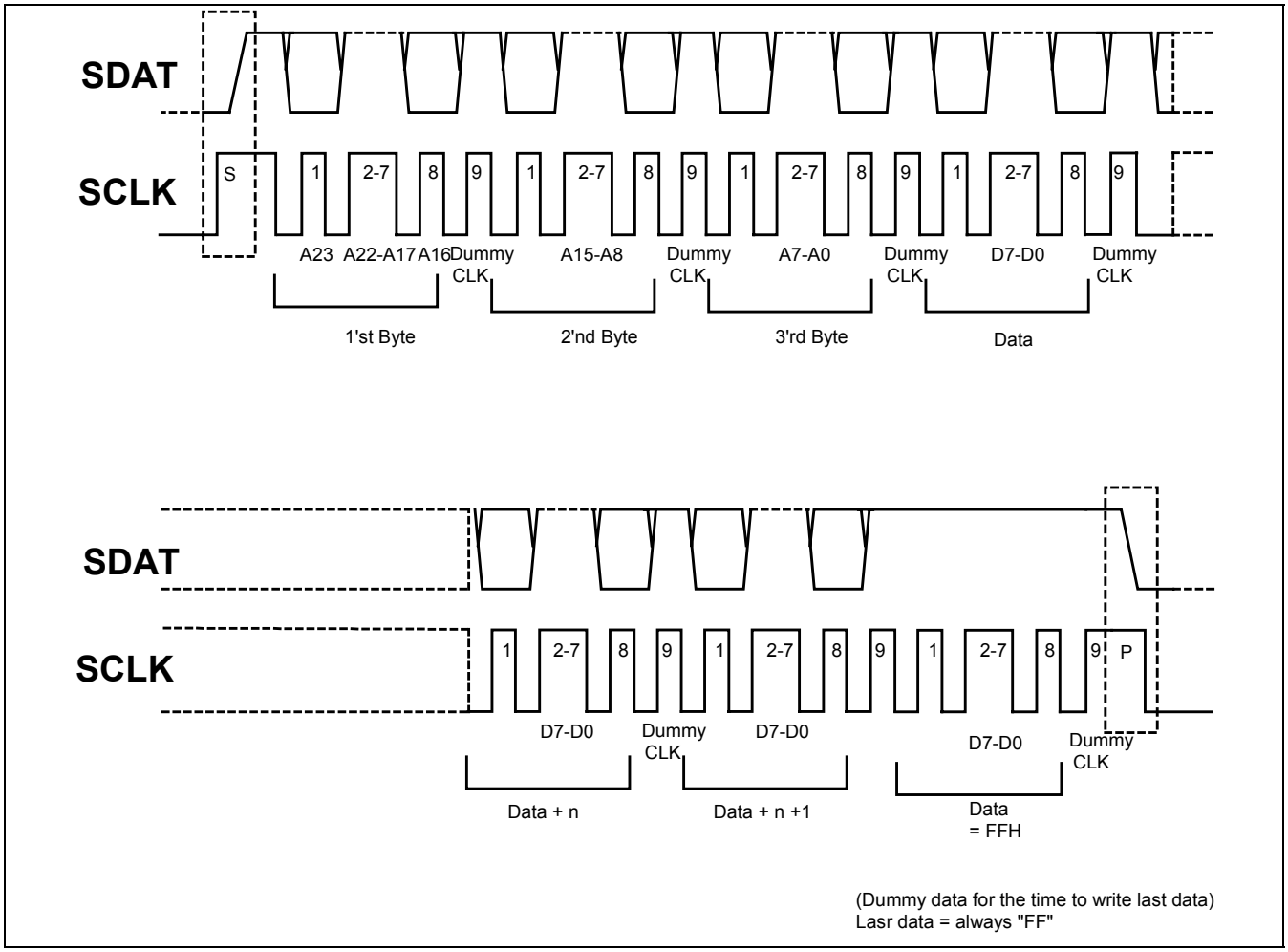


Figure 18-5. Program Wave Form

☞PROGRAMMING TIP — Programming

```
                SB1
                CLR     FMCON                   ; clear register
                SB0

                LD      CLKCON, #00H            ; cpu clock is 16-divide

LOOPE:          NOP
                LD      PP, #88H                ;
                LD      00H,#0A5H               ; User Program mode enable
                LD      PP,#00H                 ;

                SB1
ENT:            OR      FMCON,#00000010B        ; flag enable
                TM      FMCON,#00000010B        ; flag check
                JR      Z, ENT
                SB0

SPGM:           SB1                             ; Start
                OR      FMCON,#00000001B        ; SCLK=1
                OR      FMCON,#10000000B        ; SDAT=1
                SB0

                LD      R8,#01100010B           ; 1'st Byte (Programming mode)
                CALL    PGM
                LD      R8,#00010000B           ; 2nd Byte, Address=1000H (Sector 0)
                CALL    PGM
                LD      R8,#00000000B           ; 3rd Byte
                CALL    PGM

                LD      R3, #00H                ; Write Address = 1000h ~ 10FFh
ADR:            LD      R8, #66H                ; Write Data = 66h
                CALL    PGM
                INC     R3
                JR  NZ, ADR

                LD      R8,#0FFH                ; dummy data
                CALL    PGM

                SB1
                AND     FMCON,#01111111B        ; SDAT=0
                AND     FMCON,#11111110B        ; SCLK=0, Stop
                SB0

                LD      PP,#88H
                LD      00H,#00H                ; User Program Mode Disable
                LD      PP,#00H
```

☞PROGRAMMING TIP — Programming (Continued)

```
            SB1
REL:        AND     FMCON,#11111101B    ; flag disable
            TM      FMCON,#00000010B    ; flag check
            JR      NZ, REL
            SB0

            JP      END_SYM             ; END

PGM:        SB1
            AND     FMCON,#11111110B    ; SCLK=0

            CALL    WAIT

            LD      R9, #08H            ; Rotate time

PGMB:       RL      R8                  ; msb -> lsb
            LDB     FMCON.7,R8          ; FMCON.7  ← R8.0
            OR      FMCON,#00000001B    ; SCLK=1
            AND     FMCON,#11111110B    ; SCLK=0
            DJNZ    R9, PGMB

            OR      FMCON,#10000000B    ; SDAT=1
            OR      FMCON,#00000001B    ; SCLK=1
            SB0
            RET

WAIT:       NOP
            NOP
            LD      R15, #0FFH          ; 00H  <- FFH
LOOP0:      DJNZ    R15, LOOP0
            RET

END_SYM:    NOP


            .END
```

DATA PROTECTION

Option Sector Programming (Protection option in User Programming Mode)

User Program Mode can support Hard lock protection and Read protection when they have not been selected in Tool program mode yet. The data programmed by a user flash memory need to be protected at the fields of application.
The flash memory control block in the S3F84BB protects the data with two protection modes:

- Hardware protection (Hard Lock Protection)
- Read protection

These protection modes can be enabled by the option selection at a tool program mode or setting the smart option at a user program mode.

Hardware Protection (Hard Lock Protection)

If this function is enable user or any other thing cannot write and erase the data in a flash memory area. Hard Lock function can be set up in the tool program mode as well as a user program mode. Besides this protection could be released (cleared) by the chip erase execution at a tool program mode.

Read Protection

There are many users who do not want their code data to be read by any others. Read protection solves this matter by preventing the flash data from being read serially at a tool program mode and is no effective at a user program mode. When this function is enable reading or verifying the flash data at a tool program mode results in zero read out. Read protection can be released (cleared) by the chip erase execution at a tool program mode.

NOTES;

1. To enable Hard lock Protection, set the data of address 0E3Eh to “00h” in User Program Mode.
2. To enable Read Protection, set the data of address 0E3Fh to “00h” in User Program Mode.

☞PROGRAMMING TIP — Option Sector Programming (Hard Lock Protection in User Program Mode)

```
                SB1
                CLR     FMCON                   ; clear register
                SB0

                LD      CLKCON, #00H            ; cpu clock is 16-divide

LOOPE:          NOP
                LD      PP, #88H                ;
                LD      00H,#0A5H               ; User Program mode enable
                LD      PP,#00H                 ;

                SB1
ENT:            OR      FMCON,#00000010B        ; flag enable
                TM      FMCON,#00000010B        ; flag check
                JR      Z, ENT
                SB0

SPGM:           SB1                             ; Start
                OR      FMCON,#00000001B        ; SCLK=1
                OR      FMCON,#10000000B        ; SDAT=1
                SB0

                LD      R8,#11100000B           ; 1'st Byte (Hard Lock Protection mode)
                CALL    PGM
                LD      R8,#00001110B           ; 2nd Byte=0E3EH
                CALL    PGM
                LD      R8,#00111110B           ; 3rd Byte
                CALL    PGM

                LD      R8,#00H                 ; Data = 00h(Hard Lock Data)
                CALL    PGM

                LD      R8,#0FFH                ; dummy data
                CALL    PGM

                SB1
                AND     FMCON,#01111111B        ; SDAT=0
                AND     FMCON,#11111110B        ; SCLK=0, Stop
                SB0

                LD      PP,#88H
                LD      00H,#00H                ; User Program Mode Disable
                LD      PP,#00H
```

☞PROGRAMMING TIP — Option Sector Programming (Hard Lock Protection - Continued)

```
                SB1
REL:            AND     FMCON,#11111101B        ; flag disable
                TM      FMCON,#00000010B        ; flag check
                JR      NZ, REL
                SB0

                JP      END_SYM                 ; END

PGM:            SB1
                AND     FMCON,#11111110B        ; SCLK=0

                CALL    WAIT

                LD      R9, #08H                ; Rotate time

PGMB:           RL      R8                      ; msb -> lsb
                LDB     FMCON.7,R8              ; FMCON.7 ← R8.0
                OR      FMCON,#00000001B        ; SCLK=1
                AND     FMCON,#11111110B        ; SCLK=0
                DJNZ    R9, PGMB

                OR      FMCON,#10000000B        ; SDAT=1
                OR      FMCON,#00000001B        ; SCLK=1
                SB0
                RET

WAIT:           NOP
                NOP
                LD      R15, #0FFH              ; 00H  <- FFH
LOOP0:          DJNZ    R15, LOOP0
                RET

END_SYM:        NOP


                .END
```

Note: It is possible to adopt protection option (read or hard lock protection) in User Program Mode only when it hasn't been adopted by the programmer tools (in Tool Mode before).

19 ELECTRICAL DATA

OVERVIEW

In this chapter, S3C84BB/F84BB electrical characteristics are presented in tables and graphs. The information is arranged in the following order:

- Absolute maximum ratings
- Input/output capacitance
- D.C. electrical characteristics
- A.C. electrical characteristics
- Oscillation characteristics
- Oscillation stabilization time
- Data retention supply voltage in stop mode
- A/D converter electrical characteristics

Table 19-1. Absolute Maximum Ratings

(T_A= 25 °C)

Parameter	Symbol	Conditions	Rating	Unit
Supply voltage	V_{DD}		– 0.3 to +6.5	V
Input voltage	V_I		– 0.3 to V_{DD} + 0.3	
Output voltage	V_O		– 0.3 to V_{DD} + 0.3	
Output current high	I_{OH}	One I/O pin active	– 18	mA
		All I/O pins active	– 60	
Output current low	I_{OL}	One I/O pin active	+30	
		Total pin current for port	+100	
Operating temperature	T_A		– 40 to + 85	°C
Storage temperature	T_{STG}		– 65 to + 150	

Table 19-2. D.C. Electrical Characteristics

(T_A = -25 °C to + 85 °C, V_{DD} = 2.7 V to 5.5 V)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Operating voltage	V_{DD}	f_{CPU} = 10 MHz	2.7	–	5.5	V
Input high voltage	V_{IH1}	All input pins except V_{IH2}	0.8 V_{DD}	–	V_{DD}	
	V_{IH2}	X_{IN}	V_{DD}-0.5		V_{DD}	
Input low voltage	V_{IL1}	All input pins except V_{IL2}	–	–	0.2 V_{DD}	
	V_{IL2}	X_{IN}	–		0.4	

Table 19-2. D.C. Electrical Characteristics (Continued)

(T_A = -25 °C to + 85 °C, V_{DD} = 2.7 V to 5.5 V)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Output high voltage	V_{OH1}	V_{DD} = 5 V; I_{OH} = -1 mA All output pins except Port 0,2,6	V_{DD} – 1.0	–	–	V
	V_{OH2}	V_{DD} = 5 V; I_{OH} = -4 mA Port 0,2	V_{DD} – 2.0			
Output low voltage	V_{OL1}	V_{DD} = 5 V; I_{OL} = 2 mA All output pins except Port 0,2,6	–	0.12	2.0	
	V_{OL2}	V_{DD} = 5 V; I_{OL} = 15 mA Port 0,2,6		0.6	2.0	
Input high leakage current	I_{LIH1}	V_{IN} = V_{DD} All input pins except I_{LIH2}	–	–	3	μA
	I_{LIH2}	V_{IN} = V_{DD} X_{IN}			20	
Input low leakage current	I_{LIL1}	V_{IN} = 0 V All input pins except I_{LIL2}	–	–	-3	
	I_{LIL2}	V_{IN} = 0 V X_{IN}			-20	
Output high leakage current	I_{LOH}	V_{OUT} = V_{DD} All I/O pins and Output pins	–	–	5	
Output low leakage current	I_{LOL}	V_{OUT} = 0 V All I/O pins and Output pins	–	–	-5	
Pull-up resistor	R_{L1}	V_{IN} = 0 V; V_{DD} = 5 V ±10 % Port 0–8, T_A = 25°C	30	46	80	kΩ
	R_{L2}	V_{IN} = 0 V; V_{DD} = 5 V ±10% RESET only, T_A=25 °C	120	240	320	

Table 19-2. D.C. Electrical Characteristics (Concluded)

(T_A = -25 °C to + 85 °C, V_{DD} = 2.7 V to 5.5 V)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Supply current (1)	I_{DD1}	V_{DD} = 5 V ± 10 % 10 MHz crystal oscillator	–	8.5	20	mA
	I_{DD2}	Idle mode: V_{DD} = 5 V ± 10 % 10 MHz crystal oscillator		2.5	5	
	I_{DD3}	Stop mode: V_{DD} = 5 V ± 10 % T_A = 25 °C		1	3	μA

NOTES:

1. Supply current does not include current drawn through internal pull-up resistors or external output current loads.

Table 19-3. A.C. Electrical Characteristics

(T_A = -25 °C to +85 °C, V_{DD} = 2.7 V to 5.5 V)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Interrupt input high, low width (P4.0–P4.7) (P8.5, P8.6)	t_{INTH}, t_{INTL}	V_{DD} = 5 V	180	–	–	ns
RESET input low width	t_{RSL}	V_{DD} = 5 V	1.0	–	–	µs

NOTE: User must keep more large value then min value.

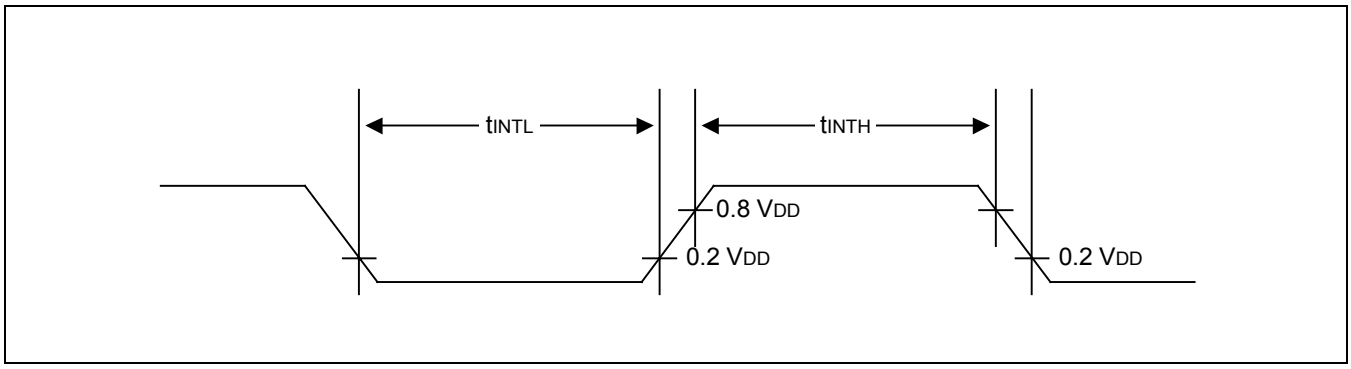


Figure 19-1. Input Timing for External Interrupts (Ports 4, Port 8.5, Port 8.6)

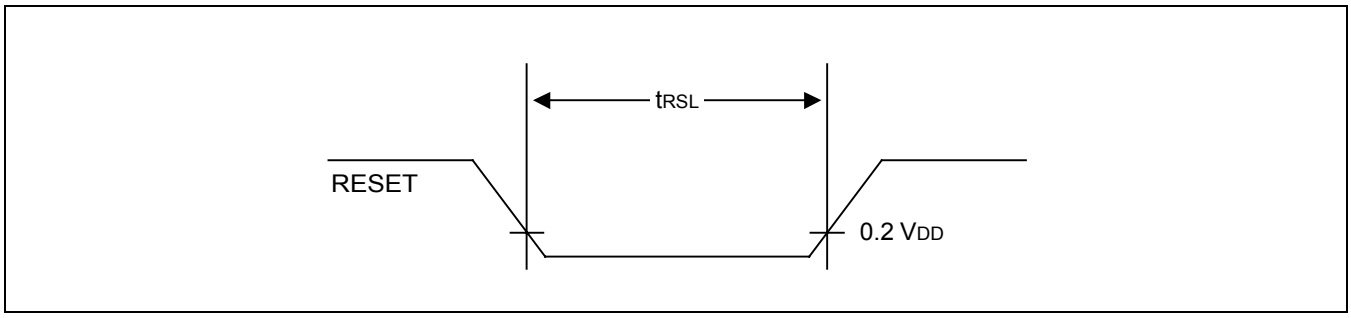


Figure 19-2. Input Timing for RESET

Table 19-4. Input/Output Capacitance

(T_A = -25 °C to +85 °C, V_{DD} = 0 V)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Input capacitance	C_{IN}	f = 1 MHz; unmeasured pins are tied to V_{SS}	–	–	10	pF
Output capacitance	C_{OUT}					
I/O capacitance	C_{IO}					

Table 19-5. Data Retention Supply Voltage in Stop Mode

(T_A = -25 °C to + 85 °C)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Data retention supply voltage	V_{DDDR}	Stop mode	2	–	5.5	V
Data retention supply current	I_{DDDR}	V_{DDDR} = 2.0 V, Stop mode	–	–	3	µA

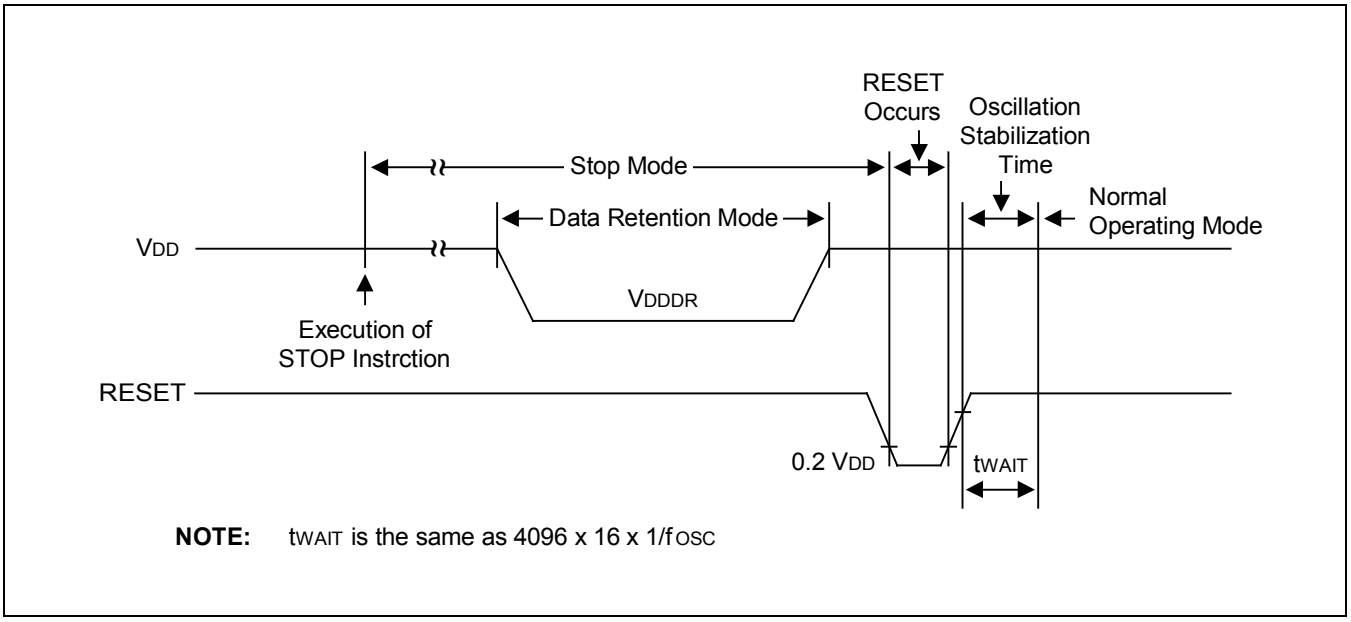


Figure 19-3. Stop Mode Release Timing Initiated by RESET

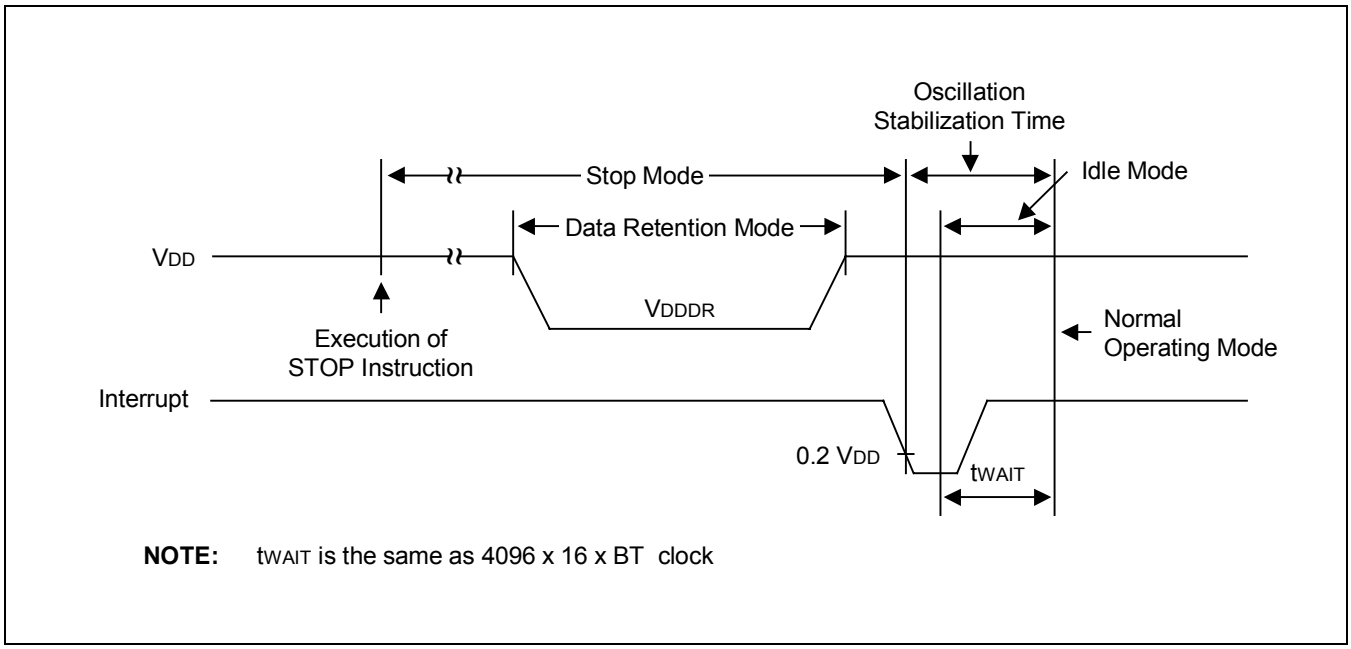


Figure 19-4. Stop Mode Release Timing Initiated by Interrupts

Table 19-6. A/D Converter Electrical Characteristics

(T_A = - 25 °C to +85 °C, V_{DD} = 2.7 V to 5.5 V, V_{SS} = 0 V)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Resolution			–	10	–	bit
Total accuracy		V_{DD} = 5.12 V	–	–	±3	LSB
Integral Linearity Error	ILE	AV_{REF} = 5.12V	–	–	±2	
Differential Linearity Error	DLE	AV_{SS} = 0 V fxx = 10 MHz	–	–	±1	
Offset Error of Top	EOT		–	±1	±3	
Offset Error of Bottom	EOB		–	±0.5	±2	
Conversion time (1)	T_{CON}	10-bit resolution 50 x 4/fxx, fxx = 10MHz	20	–	–	μs
Analog input voltage	V_{IAN}	–	AV_{SS}	–	AV_{REF}	V
Analog input impedance	R_{AN}	–	2	1000	–	MΩ
Analog reference voltage	AV_{REF}	–	2.5	–	V_{DD}	V
Analog ground	AV_{SS}	–	V_{SS}	–	V_{SS} +0.3	
Analog input current	I_{ADIN}	AV_{REF} = V_{DD} = 5V	–	–	10	μA
Analog block current (2)	I_{ADC}	AV_{REF} = V_{DD} = 5V	–	1	3	mA
		AV_{REF} = V_{DD} = 3V		0.5	1.5	
		AV_{REF} = V_{DD} = 5V When Power Down mode		100	500	nA

NOTES:
1. 'Conversion time' is the time required from the moment a conversion operation starts until it ends.
2. I_{ADC} is an operating current during A/D conversion.

Table 19-7. D/A Converter Electrical Characteristics

(T_A = - 25 °C to +85 °C, V_{DD} = 2.7 V to 5.5 V, V_{SS} = 0 V)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Resolution	–	–	–	–	8	bits
Absolute Accuracy	–		– 3	–	3	LSB
Differential Linearity Error	DLE		– 1	–	1	LSB
Setup Time	t_{SU}		–	–	5	μs
Output Resistance	R_O		4.5	5	5.5	kΩ

Table 19-8. Flash Memory D.C. Electrical Characteristics

(T_A = - 25 °C to +85 °C, V_{DD} = 2.7 V to 5.5 V, V_{SS} = 0 V)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Logic power supply	V_{DD}		2.7	5.0	5.5	V
Flash memory operating current (F_{DD})	F_{DD1}	V_{DD} = 2.7 V to 5.5 V during reading	–	40	80	mA
	F_{DD2}	V_{DD} = 2.7 V to 5.5 V during programming	–	40	80	mA
	F_{DD3}	V_{DD} = 2.7 V to 5.5 V during erasing	–	40	80	mA

Table 19-9. Flash Memory A.C. Electrical Characteristics

(T_A = - 25 °C to +85 °C, V_{DD} = 2.7 V to 5.5 V, V_{SS} = 0 V)

Parameter	Symbol	Conditions	Min	Typ	Max	Unit
Programming time(1)	Ftp	V_{DD} = 2.7 V to 5.5 V	20	30	300	uS
Chip Erasing time(2)	Ftp1		–	–	10	mS
Sector Erasing time(3)	Ftp2		–	2		mS
Data access time	Ft_{RS}		–	50	–	mS
Number of writing/erasing	Fnwe	–	–	100		Times

NOTES:

1. The Programming time is the time during which one byte(8-bit) is programmed.
2. The chip erasing time is the time during which all 64K-byte block is erased.
3. The sector erasing time is the time during which all 60K-byte block is erased.

Table 19-10. Main Oscillator Frequency (f_{OSC1})

(T_A = -25 °C to +85 °C, V_{DD} = 2.7 V to 5.5 V)

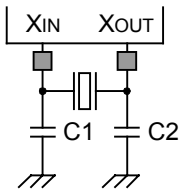
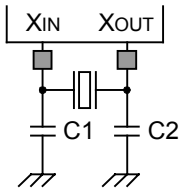
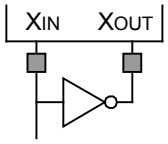
Oscillator	Clock Circuit	Test Condition	Min	Typ	Max	Unit
Crystal		Crystal oscillation frequency	1	–	10	MHz
Ceramic		Ceramic oscillation frequency	1	–	10	
External clock		X_{IN} input frequency	1	–	10	

Table 19-11. Main Oscillator Clock Stabilization Time (t_{ST1})

(T_A = -25 °C to +85 °C, V_{DD} = 2.7 V to 5.5 V)

Oscillator	Test Condition	Min	Typ	Max	Unit
Crystal	V_{DD} = 2.7 V to 5.5 V	–	–	10	ms
Ceramic	Stabilization occurs when V_{DD} is equal to the minimum oscillator voltage range.	–	–	4	
External clock	X_{IN} input high and low level width (t_{XH}, t_{XL})	50	–	–	ns

NOTE: Oscillation stabilization time (t_{ST1}) is the time required for the CPU clock to return to its normal oscillation frequency after a power-on occurs, or when Stop mode is ended by a RESET signal.

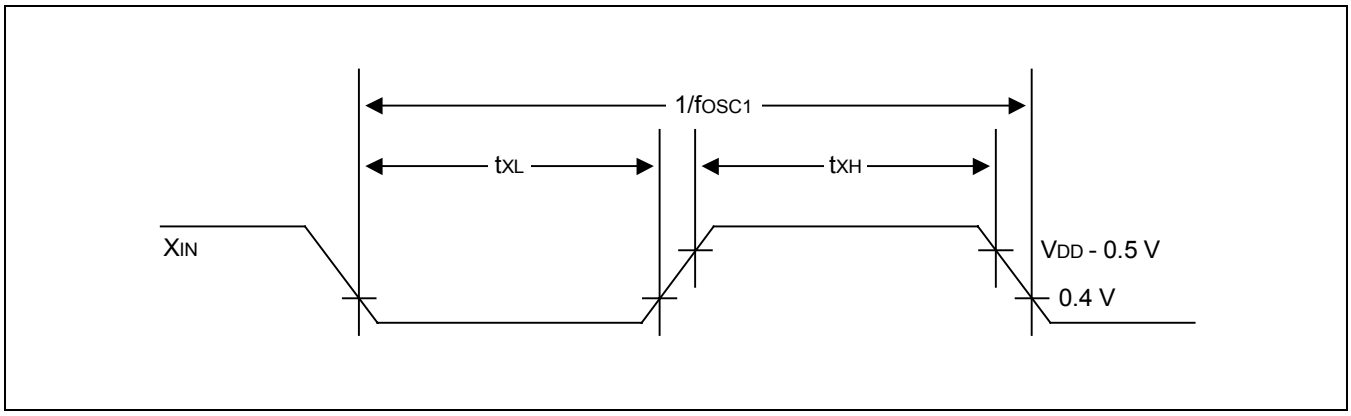


Figure 19-5. Clock Timing Measurement at X_{IN}

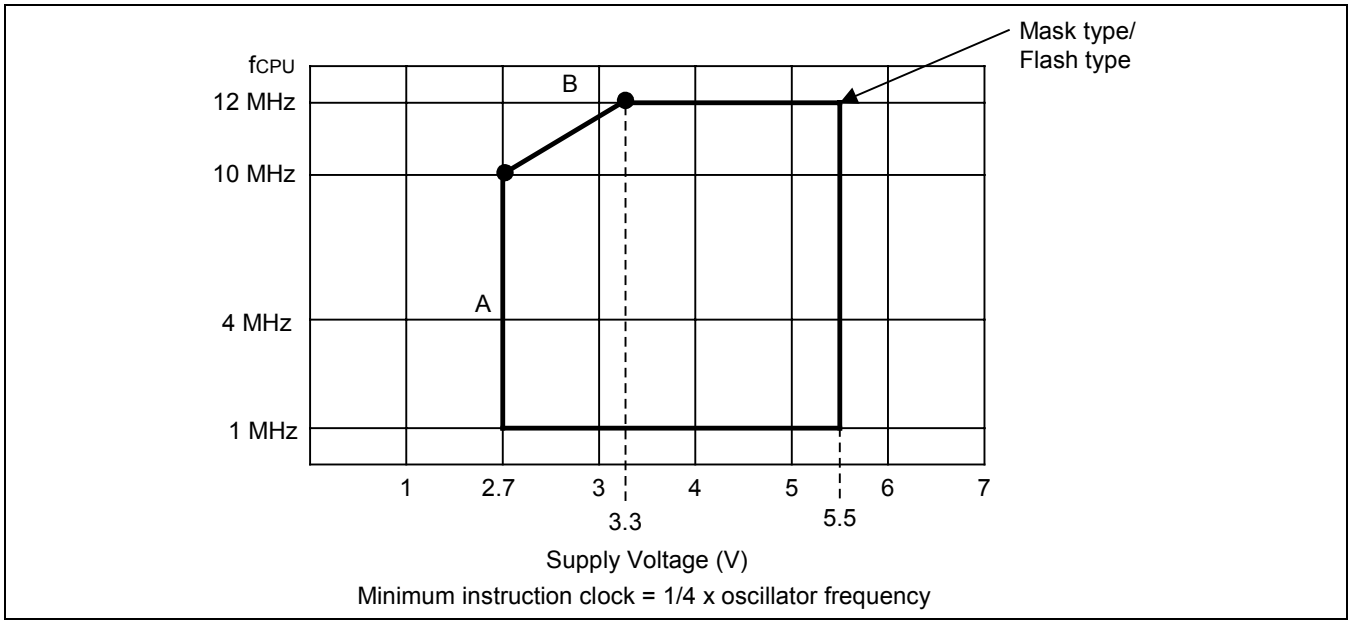
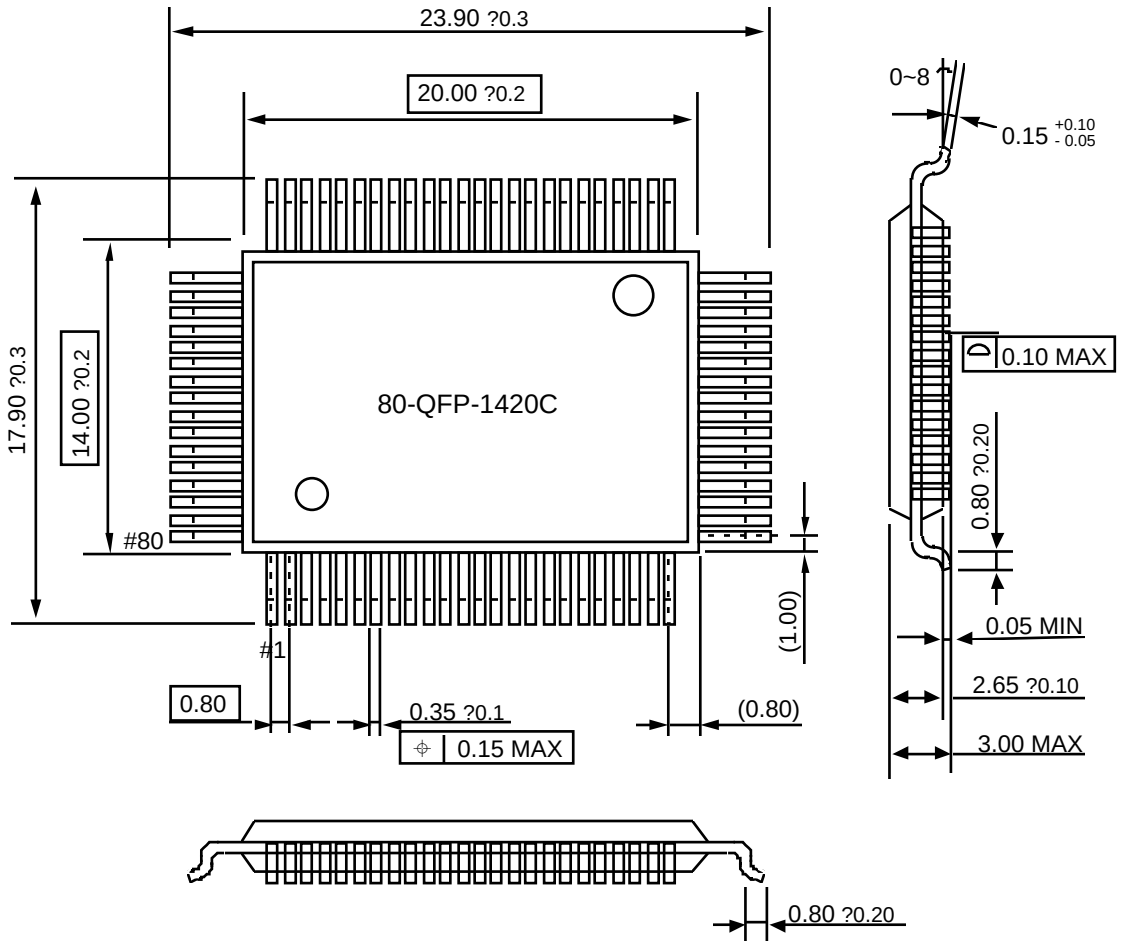


Figure 19-6. Operating Voltage Range

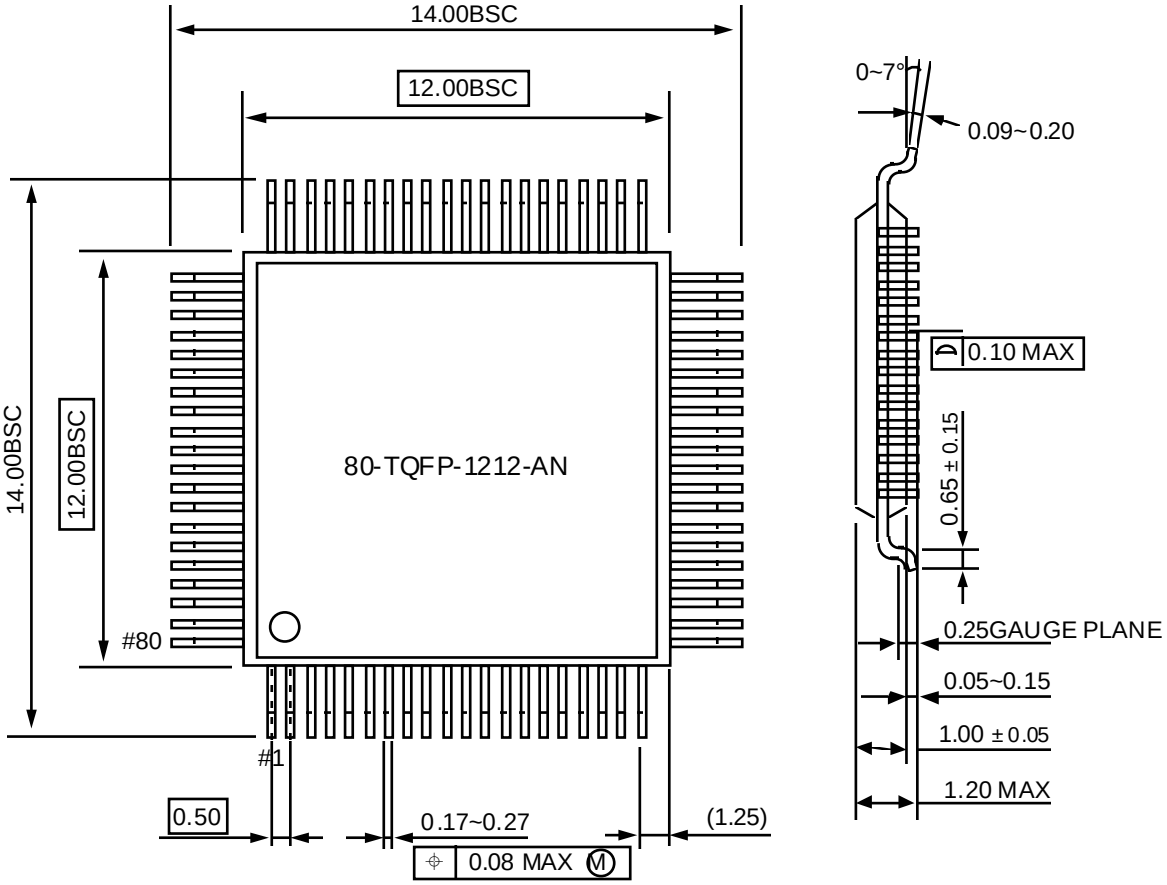
NOTES

20 MECHANICAL DATA



NOTE: Dimensions are in millimeters.

Figure 20–1. S3C84BB/F84BB 80-QFP Standard Package Dimension (in Millimeters)



NOTE: Dimensions are in millimeters.

Figure 20–2. S3C84BB/F84BB 80-TQFP Standard Package Dimension (in Millimeters)

21 DEVELOPMENT TOOLS

OVERVIEW

Samsung provides a powerful and easy-to-use development support system in turnkey form. The development support system is configured with a host system, debugging tools, and support software. For the host system, any standard computer that operates with Win95/98/2000 as its operating system can be used. One type of debugging tool including hardware and software is provided: the sophisticated and powerful in-circuit emulator, SMDS2+ or SK-1000, for S3C7, S3C9, S3C8 families of microcontrollers. The SMDS2+ and SK-1000 is a new and improved version of SMDS2. Samsung also offers support software that includes debugger, assembler, and a program for setting options.

SHINE

Samsung Host Interface for In-Circuit Emulator, SHINE (Smart Studio in case of SK-1000), is a multi-window based debugger for SMDS2+. SHINE provides pull-down and pop-up menus, mouse support, function/hot keys, and context-sensitive hyper-linked help. It has an advanced, multiple-windowed user interface that emphasizes ease of use. Each window can be sized, moved, scrolled, highlighted, added, or removed completely.

SASM ASSEMBLER

The SASM88 is a relocatable assembler for Samsung's S3C8-series microcontrollers. The SASM88 takes a source file containing assembly language statements and translates into a corresponding source code, object code and comments. The SASM88 supports macros and conditional assembly. It runs on the MS-DOS operating system. It produces the relocatable object code only, so the user should link object file. Object files can be linked with other object files and loaded into memory.

SAMA ASSEMBLER

The Samsung Arrangeable Microcontroller (SAM) Assembler, SAMA, is a universal assembler, and generates object code in standard hexadecimal format. Assembled program code includes the object code that is used for ROM data and required SMDS program control data. To assemble programs, SAMA requires a source file and an auxiliary definition (DEF) file with device specific information.

HEX2ROM

HEX2ROM file generates ROM code from HEX file which has been produced by assembler. ROM code must be needed to fabricate a microcontroller which has a mask ROM. When generating the ROM code (.OBJ file) by HEX2ROM, the value "FF" is filled into the unused ROM area up to the maximum ROM size of the target device automatically.

TARGET BOARDS

Target boards are available for all S3C8-series microcontrollers. All required target system cables and adapters are included with the device-specific target board.

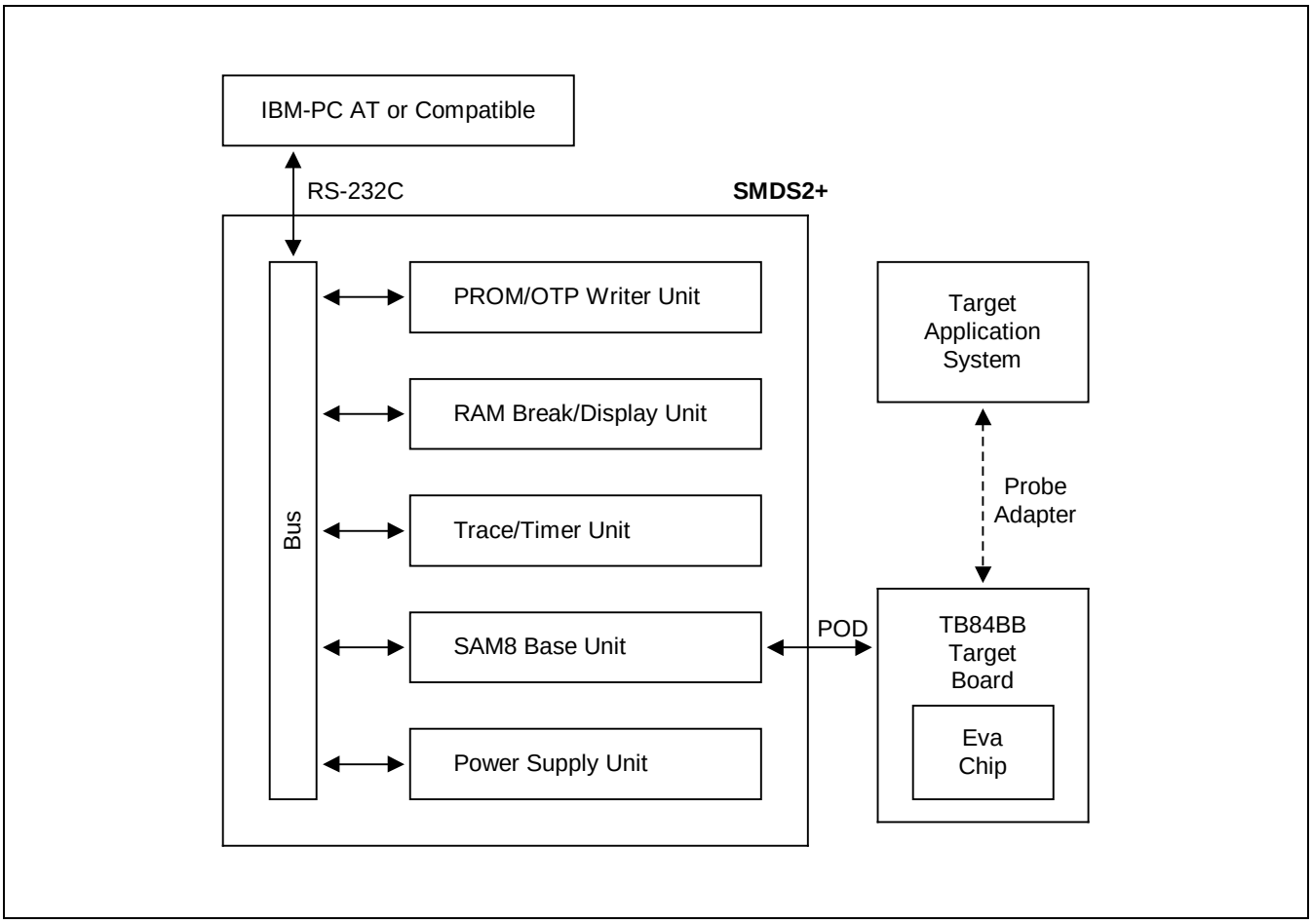


Figure 21-1. SMDS+ Product Configuration (SK-1000 is single cabinet type)

TB84BB TARGET BOARD

The TB84BB target board is used for the S3C84BB/F84BB microcontroller. It is supported by the SMDS2, SMDS2+, SK-820, or SK-1000 development system.

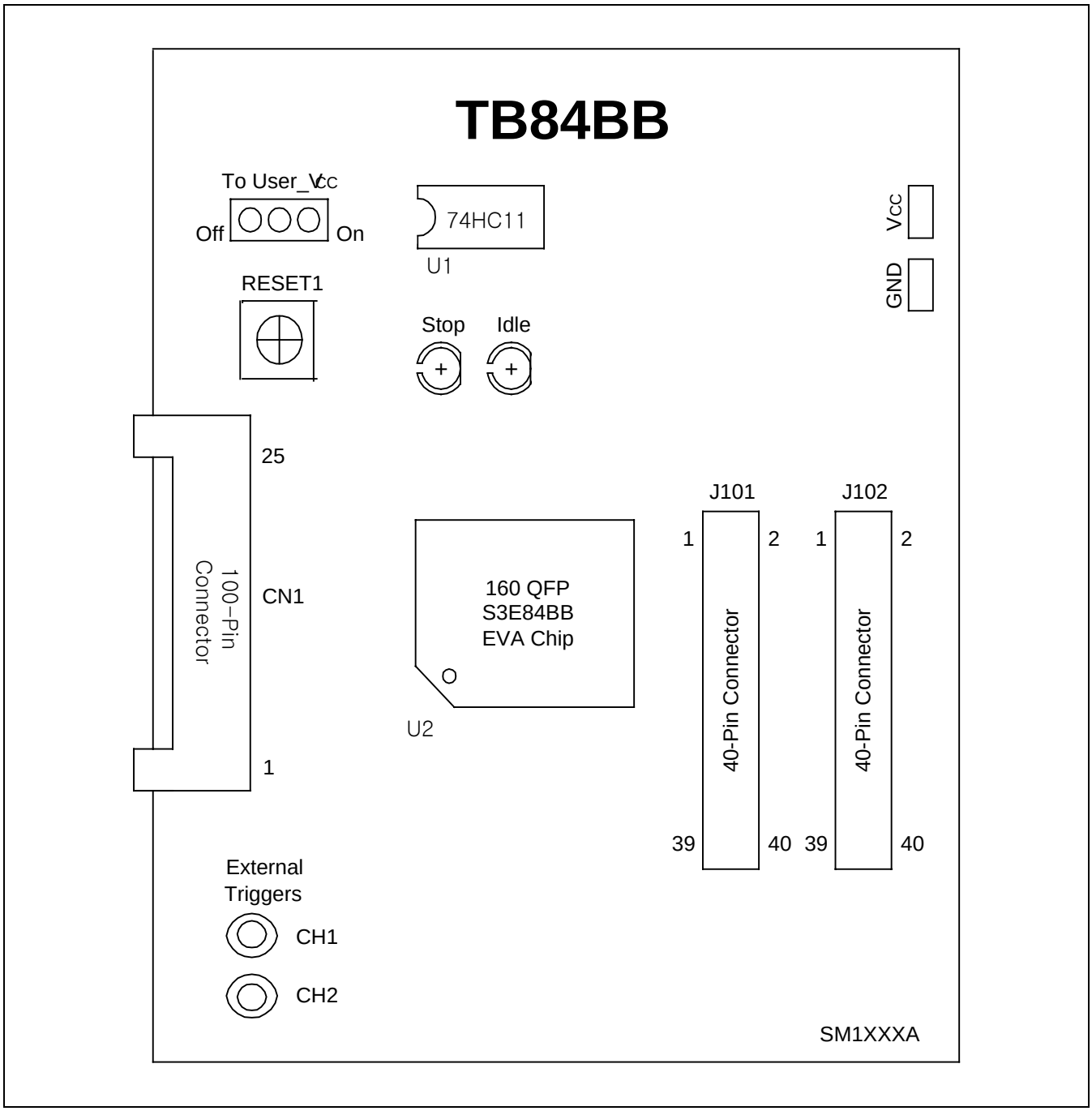


Figure 21–2. TB84BB Target Board Configuration

Table 21-1. Power Selection Settings for TB84BB

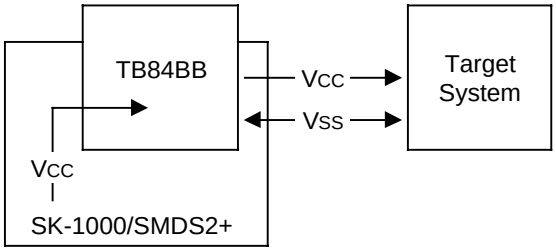
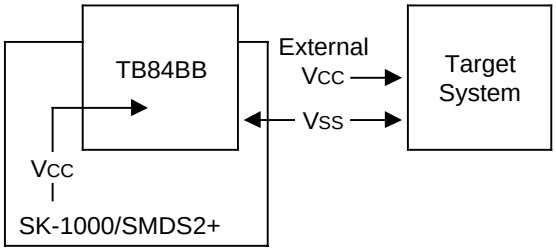
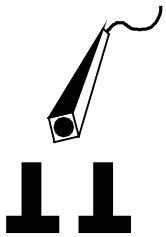
"To User_V_{CC}" Settings	Operating Mode	Comments
To User_Vcc Off  On		The ICE (SK-1000/SMDS2+) supplies V_{CC} to the target board (evaluation chip) and the target system.
To User_Vcc Off  On		The ICE (SK-1000/SMDS2+) supplies V_{CC} only to the target board (evaluation chip). The target system must have its own power supply.

Table 21-2. Using Single Header Pins as the Input Path for External Trigger Sources

Target Board Part	Comments
External Triggers Ch1 Ch2	 Connector from External Trigger Sources of the Application System You can connect an external trigger source to one of the two external trigger channels (CH1 or CH2) only for the SMDS2+ breakpoint and trace functions.

IDLE LED

The Green LED is ON when the evaluation chip (S3E84BB) is in idle mode.

STOP LED

The Red LED is ON when the evaluation chip (S3E84BB) is in stop mode.

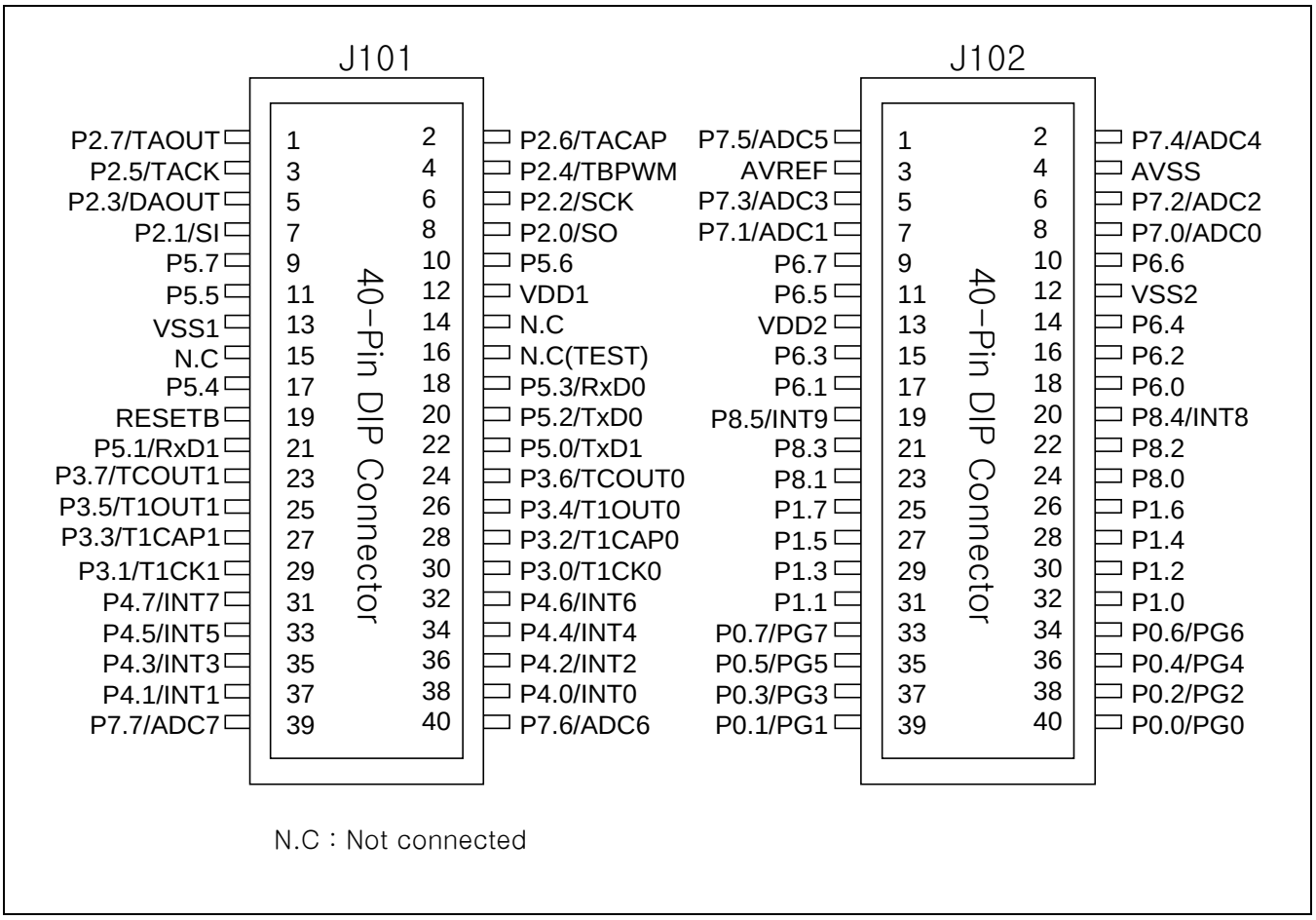


Figure 21–3. 40-Pin Connectors for TB84BB (S3C84BB, 80-QFP Package)

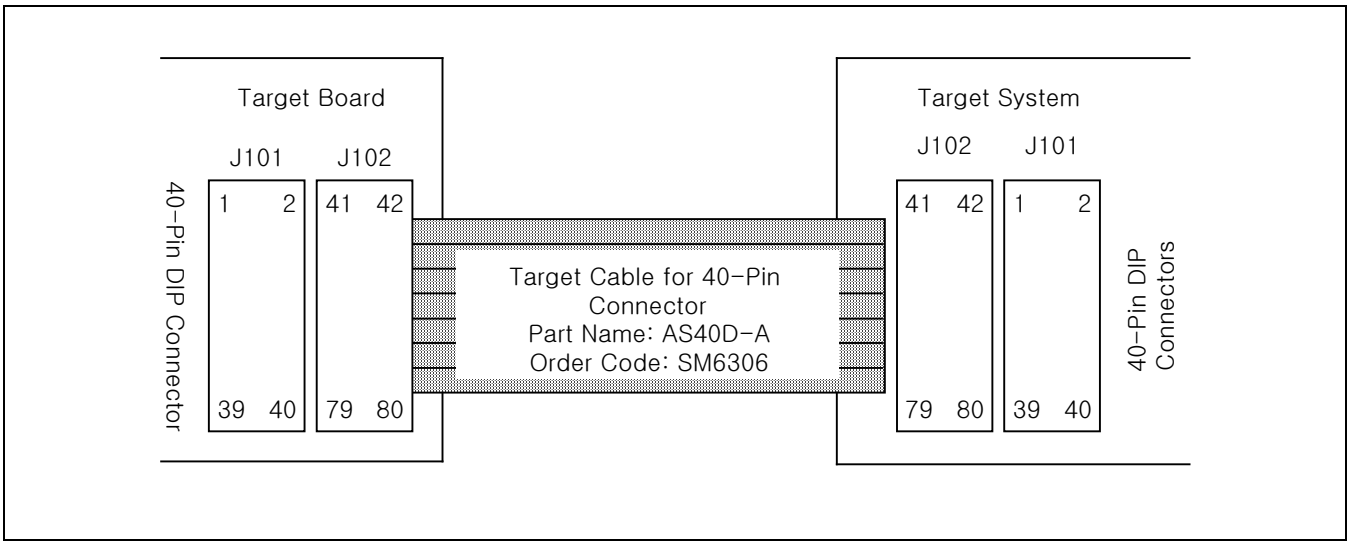


Figure 21–4. TB84BB Cable for 80-QFP Adapter

SAMSUNG ELECTRONICS

S3C SERIES MASK ROM ORDER FORM

Product description:

Device Number: S3C84BB_____- _____(write down the ROM code number)

Product Order Form: ☐ Package ☐ Pellet ☐ Wafer Package Type: _____

Package Marking (Check One):

☐ Standard

SEC @ YWW
Device Name

☐ Custom A
(Max 10 chars)

@ YWW
Device Name

☐ Custom B
(Max 10 chars each line)

@ YWW

@ : Assembly site code, Y : Last number of assembly year, WW : Week of assembly

Delivery Dates and Quantities:

Deliverable	Required Delivery Date	Quantity	Comments
ROM code	–	Not applicable	See ROM Selection Form
Customer sample			
Risk order			See Risk Order Sheet

Please answer the following questions:

☞ **For what kind of product will you be using this order?**

☐ New product ☐ Upgrade of an existing product

☐ Replacement of an existing product ☐ Other

If you are replacing an existing product, please indicate the former product name

( )

☞ **What are the main reasons you decided to use a Samsung microcontroller in your product?**

Please check all that apply.

☐ Price ☐ Product quality ☐ Features and functions

☐ Development system ☐ Technical support ☐ Delivery on time

☐ Used same micom before ☐ Quality of documentation ☐ Samsung reputation

Mask Charge (US$ / Won): _____

Customer Information:

Company Name: _____ Telephone number _____

Signatures: _____ _____

(Person placing the order) (Technical Manager)

(For duplicate copies of this form, and for additional ordering information, please contact your local Samsung sales representative. Samsung sales offices are listed on the back cover of this book.)

S3C SERIES
REQUEST FOR PRODUCTION AT CUSTOMER RISK

Customer Information:

Company Name: _____

Department: _____

Telephone Number: _____ Fax: _____

Date: _____

Risk Order Information:

Device Number: S3C_____- _____ (write down the ROM code number)

Package: Number of Pins: _____ Package Type: _____

Intended Application: _____

Product Model Number: _____

Customer Risk Order Agreement:

We hereby request SEC to produce the above named product in the quantity stated below. We believe our risk order product to be in full compliance with all SEC production specifications and, to this extent, agree to assume responsibility for any and all production risks involved.

Order Quantity and Delivery Schedule:

Risk Order Quantity: _____ PCS

Delivery Schedule:

Delivery Date (s)	Quantity	Comments

Signatures: _____ _____

(Person Placing the Risk Order) (SEC Sales Representative)

(For duplicate copies of this form, and for additional ordering information, please contact your local Samsung sales representative. Samsung sales offices are listed on the back cover of this book.)

S3C84BB MASK OPTION SELECTION FORM

Device Number: S3C___-_____(write down the ROM code number)

Attachment (Check one): ☐ Diskette ☐ PROM

Customer Checksum: _____

Company Name: _____

Signature (Engineer): _____

Please answer the following questions:

☞ **Application** (Product Model ID: _____)

☐ Audio	☐ Video	☐ Telecom
☐ CD Databank	☐ Caller ID	☐ CD Game
☐ Industrials	☐ Home Appliance	☐ Office Automation
☐ Remocon	☐ Other	

Please describe in detail its application

(For duplicate copies of this form, and for additional ordering information, please contact your local Samsung sales representative. Samsung sales offices are listed on the back cover of this book.)

S3F84BB SERIES FLASH MCU
FACTORY WRITING ORDER FORM (1/2)

Product Description:

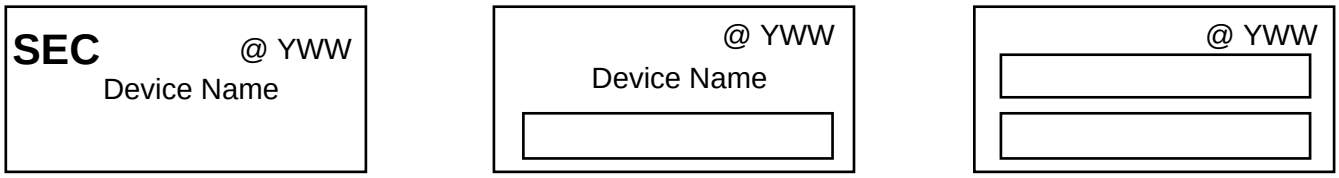
Device Number: S3F84BB__-_____(write down the ROM code number)

Product Order Form: ☐ Package ☐ Pellet ☐ Wafer

If the product order form is package: Package Type: _____

Package Marking (Check One):

☐ Standard ☐ Custom A (Max 10 chars) ☐ Custom B (Max 10 chars each line)



@ : Assembly site code, Y : Last number of assembly year, WW : Week of assembly

Delivery Dates and Quantity:

ROM Code Release Date	Required Delivery Date of Device	Quantity

Please answer the following questions:

☞ **What is the purpose of this order?**

☐ New product development ☐ Upgrade of an existing product

☐ Replacement of an existing microcontroller ☐ Other

If you are replacing an existing microcontroller, please indicate the former microcontroller name

( )

☞ **What are the main reasons you decided to use a Samsung microcontroller in your product? Please check all that apply.**

☐ Price ☐ Product quality ☐ Features and functions

☐ Development system ☐ Technical support ☐ Delivery on time

☐ Used same MCU before ☐ Quality of documentation ☐ Samsung reputation

Customer Information:

Company Name: _____ Telephone number _____

Signatures: _____ _____

(Person placing the order) (Technical Manager)

(For duplicate copies of this form, and for additional ordering information, please contact your local Samsung sales representative. Samsung sales offices are listed on the back cover of this book.)

S3F84BB FLASH MCU
FACTORY WRITING ORDER FORM (2/2)

Device Number: S3F84BB__-_____ (write down the ROM code number)

Customer Checksums: _____

Company Name: _____

Signature (Engineer): _____

Read Protection [1]: ☐ **Yes** ☐ **No**

Please answer the following questions:

☞ **Are you going to continue ordering this device?**

☐ Yes ☐ No

If so, how much will you be ordering? _____pcs

☞ **Application** (Product Model ID: _____)

☐ Audio	☐ Video	☐ Telecom
☐ LCD Databank	☐ Caller ID	☐ LCD Game
☐ Industrials	☐ Home Appliance	☐ Office Automation
☐ Remocon	☐ Other	

Please describe in detail its application

_____

NOTES

1. Once you choose a read protection, you cannot read again the programming code from the ROM.
2. FLASH MCU Writing will be executed in our manufacturing site.
3. The writing program is completely verified by a customer. Samsung does not take on any responsibility for errors occurred from the writing program.

(For duplicate copies of this form, and for additional ordering information, please contact your local Samsung sales representative. Samsung sales offices are listed on the back cover of this book.)