PM4354

COMET-QUAD

Four Channel Combined T1/E1 Transceiver/Framer

Programmer's Guide

Proprietary and Confidential

Released

Issue 2: September, 2001

Legal Information

Copyright

Disclaimer

Trademarks

Patents

Contacting PMC-Sierra

PMC-Sierra
8555 Baxter Place Burnaby, BC
Canada V5A 4V7

Tel: +1.604.415.6000
Fax: +1.604.415.6204

Document Information: document@pmc-sierra.com
Corporate Information: info@pmc-sierra.com
Technical Support: apps@pmc-sierra.com
Web Site: http://www.pmc-sierra.com

Revision History

Issue No.	Issue Date	Details of Change
Issue 1	April 2000	Document created.
Issue 2	September 2001	Rewrite initialization section and sample configurations. Also, update document according to latest datasheet (Issue 6). Add sections to cover programming using the software driver.

Table of Contents

List of Figures

List of Tables

1 References

- PMC-1990315, PMC-Sierra, Inc., "PM4354 COMET-QUAD Four Channel Combined E1/T1 Transceiver/Framer Data Sheet", May 2001, Issue 6.
- PMC-1970624, PMC-Sierra, Inc., "PM4351 COMET Combined E1/T1 Transceiver Standard Product Data Sheet", July 1999, Issue 8.
- PMC-1990615, PMC-Sierra, Inc., "PM4351 COMET Programmer's GUIDE", December 1999, Issue 3.

2 Introduction

2.1 Scope

This document introduces the reader to the programmable features of the COMET-QUAD by providing the application interface accesses and the corresponding register accesses necessary to configure the COMET-QUAD. The document provides pseudo-code, flow charts, figures and tables necessary to program real-time operation of the COMET-QUAD chip. This document is essentially made up of two parts that mirror each other. The first part gives details on how to program the device using register access, and the second part gives details on how to program the device using the device driver.

The COMET-QUAD Programmer's Guide is a supplement to the COMET-QUAD Data Sheet (PMC-1990315). Due to the large number of configurations the COMET-QUAD can have, it is impossible to cover all configuration aspects of the chip. Please contact a PMC-Sierra Applications Engineer for information not covered in this document.

Although every effort has been taken to ensure the consistency between this document and the COMET-QUAD Data Sheet, some discrepancies may occur. In case of inconsistencies between this document and the COMET-QUAD Data Sheet the information in the data sheet should be considered accurate and take precedence over information provided in this document.

2.2 Target Audience

This document is prepared for engineers that design-in the COMET-QUAD chip and need a quick reference on how to program the COMET-QUAD. Some prior knowledge of T1, E1 and J1 framing technologies, as well as some knowledge of C programming language and computer operation is necessary to fully understand this document.

2.3 Numbering Conventions

The following numbering conventions are used throughout this document:

Binary	1010B, 011
Decimal	129, 6, 12
Hexadecimal	0x1FE2, 09FH

2.4 Offset Conventions

COMET-QUAD has four "quadrants" (ports) with identical characteristics. Each quadrant is equipped with the same set of registers shifted by the offset of 0x100, starting with offset 0x000 for the first quadrant, and ending with offset 0x300 for the fourth quadrant. In this document, the registers are referred to in two ways:

- Explicitly, by stating the register explicit hexadecimal address, and

- Implicitly, by stating the quadrant symbol 'Q' and the hexadecimal address of the corresponding register in the first quadrant.

The following is an example of explicit and implicit register notation:

- Explicit 0x211, 0C0H
- Implicit 0xQ11, QC0H

Explicit notation is used throughout this document to refer to an instance of the register in a particular quadrant, while implicit notation is used to refer to all four register instances in each of the quadrants.

2.5 Pseudo-Code Conventions

The pseudo-code in this document is shown in a C-like syntax. The code segments are not by any means compile-ready and are provided for reader's reference in order to understand a particular procedure or concept. However, the procedures were intended to be a good basis for programming the COMET-QUAD.

3 Register Description

Unless otherwise specified, COMET-QUAD registers are shown in this document using the convention **REGISTER_NAME**(register address).

All register accesses are shown in this document as:

```
READ_REG(<address>);
WRITE_REG(<address>, <value>);
```

whereby <address> is the register address from the data sheet and <value> is the data written to a register.

The COMET-QUAD registers have the following characteristics:

- All values written into unused register bits should be written with logic 0 unless otherwise stated, ensuring software compatibility with future versions of the product. Reading back unused bits can produce either logic one or a logic zero; hence, unused register bits should be masked off by software during a register read access.
- Certain register bits are reserved. To ensure that the COMET-QUAD operates as intended, reserved register bits must only be written with their default values unless otherwise stated in the datasheet.

The COMET-QUAD has 2 types of register spaces - the direct registers, which are accessed directly by the microprocessor bus interface of the COMET-QUAD, and the “indirect” registers which are accessed via assigned normal-mode buffer registers.

3.1 Direct Registers

Direct registers configure the operation of the COMET-QUAD. The registers have offsets between 0xQ00 and 0xQFF inclusive. The reader should refer to the COMET-QUAD Data Sheet [□] for the default values of these registers.

3.2 Indirect Registers

The indirect registers are for per-channel control, which is normally disabled by default, on power-up. The PCCE bit in the corresponding per-channel block must be set to enable per channel control.

The “IND” bit in the corresponding per-channel block must be set before the software can access indirect registers. On power-up or software reset, this bit is clear. See Table 1 for the location of the PCCE and IND bits for each per-channel block.

Table 1: Location of PCCE and IND Bits

Bits	Register Number		
	TPSC	RPSC	SIGX
PCCE and IND	0xQ6C	0xQ70	0xQ50

Once the "IND" bit is set, the indirect registers can be accessed via the address and data buffer registers. The following pseudo code shows how to read from and write to an indirect register. See Table 2 for explanation of pseudo code register names.

```
#define MAX_BUSY_READ   0x5
#define BIT_BUSY        0x80

/** to read an indirect register: **/

int ReadIndirectReg(int offset, unsigned char &value)
{
      int i;

      WRITE_REG(REG_CHANNEL_INDIRECT_ADDRESS, (offset|0x80));
      for (i = 0; i < MAX_BUSY_READ; i++)
      {
            value = READ_REG(REG_MICRO_ACCESS_STATUS);
            if ((value & BIT_BUSY) == 0)
            {
                  value =
                  READ_REG(REG_CHANNEL_INDIRECT_DATA_BUFFER);
                  return SUCCESS;
            }
      }
      return FAILURE;
}

/** to write to an indirect register: **/

int WriteIndirectReg(int offset, unsigned char value)
{
      unsigned char temp;
      int i;

      WRITE_REG(REG_CHANNEL_INDIRECT_DATA_BUFFER, value);
      WRITE_REG(REG_CHANNEL_INDIRECT_ADDRESS, (offset&0x7F));
      for (i = 0; i < MAX_BUSY_READ; i++)
      {
            temp = READ_REG(REG_MICRO_ACCESS_STATUS);
```

```
                if ((temp & BIT_BUSY) == 0) return SUCCESS;
           }
           return FAILURE;
      }
```

Table 2: Indirect Access Registers

Register Define	Register Number		
	TPSC	RPSC	SIGX
REG_MICRO_ACCESS_STATUS	0xQ6D	0xQ71	0xQ51
REG_CHANNEL_INDIRECT_DATA_BUFFER	0xQ6F	0xQ73	0xQ53
REG_CHANNEL_INDIRECT_ADDRESS	0xQ6E	0xQ72	0xQ52

4 Programming the COMET-QUAD Using register accesses

Typically, the software controlling the COMET-QUAD will have a few main software states:

1) Software Reset (Global)

2) Operational (Global)

a) Configuration (Per port)

b) Active (Per port)

Immediately after the COMET-QUAD completes a power-up sequence, the software begins by issuing a software reset. The software reset applies to all four ports, and resets the registers to the default values specified in the datasheet. Note that many of the register bits don't have default values – namely the read-only and unused bits.

As soon as the device is taken out of software reset, it is in an "Operational" state. This means that the device is running, and ready to be programmed to perform a useful function.

Within this "Operational" state are usually two software sub-states:

"Configuration state": Before each port can be do anything useful, the software needs to configure it (through a series of register accesses). Typically when the software is in this state, it will leave the analog transmitter the default high-impedance state to prevent garbage from being transmitted during configuration. In addition, the software will usually leave all interrupts disabled by leaving all the interrupt enable ('E' bits) as their default '0'. This is to ensure that the microprocessor doesn't receive garbage interrupts.

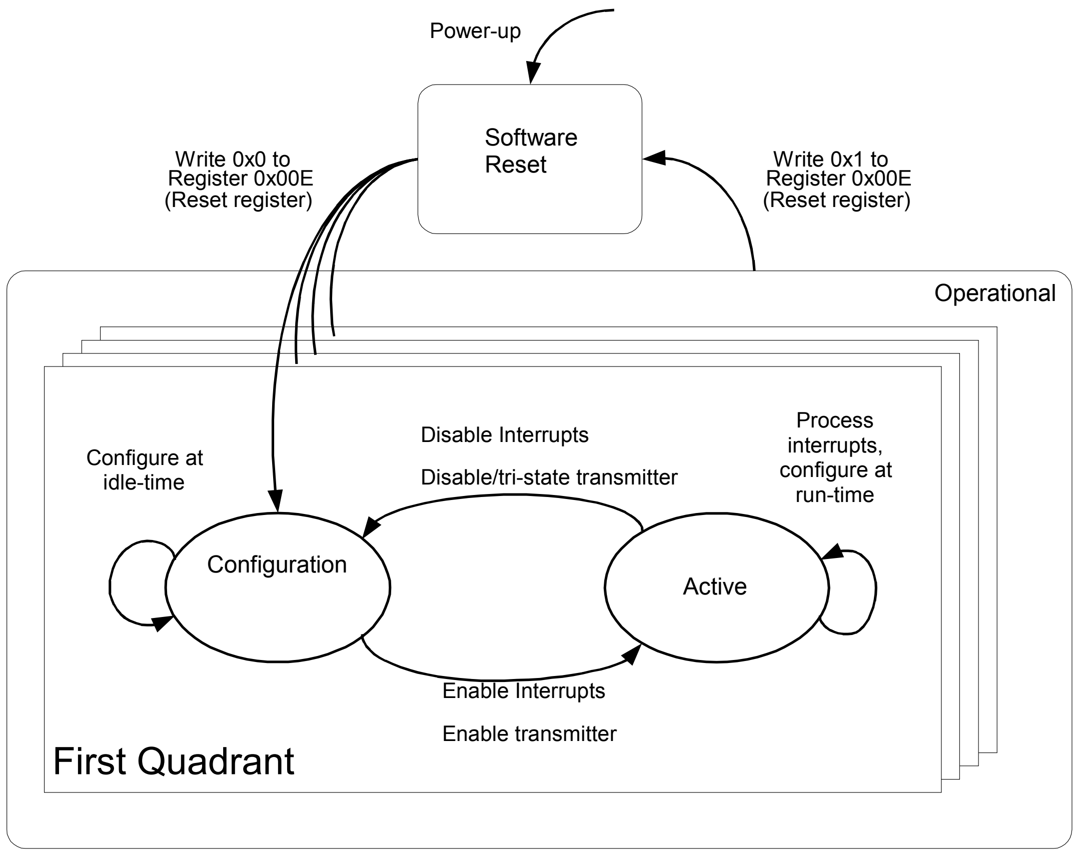
After configuration of each port, the software will typically put the port into use. The software transitions into "Active" state. It first takes the port transmitter out of its high-impedance state to enable transmission and enables the INTB output interrupt pin. Then the software typically begins processing the interrupts and monitoring status.

If during operation, the device needs to be reconfigured, the software can handle this a few ways. The simplest is to make the change "on-the-fly", staying in Active state. This can be done with most small changes. For larger changes, it is generally recommended for the software to transition into Configuration state. This is to take the device "offline", so that it won't generate any garbage or meaningless outputs during the re-configuration. For example, re-programming a different transmit pulse waveform would normally be done in Configuration state.

For significant reconfigurations, such as switching between T1 and E1 modes, the software will usually go through a Software Reset cycle, and re-configure the entire device from scratch. In particular, a software reset cycle is required any time the backplane rates are changed (ie. Changing the value of the RATE[1:0] bits).

A typical software state transition diagram is shown below in Figure 1.

Figure 1 Software State Diagram of the COMET-QUAD


Power-up
Software Reset
Write 0x0 to Register 0x00E (Reset register)
Write 0x1 to Register 0x00E (Reset register)
Operational
Configure at idle-time
Disable Interrupts
Disable/tri-state transmitter
Process interrupts, configure at run-time
Configuration
Active
Enable Interrupts
Enable transmitter
First Quadrant

4.1 Resetting the COMET-QUAD

To put the COMET-QUAD in *Reset* state, set the 'RESET' bit in register 0x00E:

```
WRITE_REG(0x00E, 0x01);
```

The device will remain in a software reset state until the 'RESET' bit is explicitly cleared:

```
WRITE_REG(0x00E, 0x00);
```

Note: *A Software Reset should be done only after a valid power-up sequence: When powering up the COMET-QUAD, all the 3.3 V power pins (analog power) should ramp-up before the 2.5 V pins (digital power). During the ramp-up, it is critical that the instantaneous voltage on the 2.5 V pins never exceeds the instantaneous voltage on the 3.3 V pins.*

The power ramp-up is considered complete when both the 2.5 V power and 3.3 V power have risen above the minimum input level thresholds specified in section 15 (DC Characteristics) of the COMET-QUAD Datasheet (PMC-1990315). These minimum thresholds are 3.135 V for the 3.3 V power and 2.3 V for the 2.5 V power.

After the ramp-up, there are three things that must be done before the device is operational:

1) The RSTB (chip reset) pin must be driven active-low for a minimum of 100 ns

2) The CSB (Chip Select) pin must be driven high concurrently with RSTB being driven low at least once in order to clear any internal test modes. (The most common way to accomplish this is to drive CSB high during the chip reset in step 1). Although there is no specified minimum time that CSB & /RSTB need to be driven for, 100 ns provides a safe margin.

3) The TRSTB (JTAG reset) must also be driven active-low for a minimum of 100 ns in order to reset the JTAG state machine so that the I/O pins are in their correct operational configuration. This JTAG reset can be performed independently of the above two steps 1) and 2).

4.2 Configuring the COMET-QUAD

To configure the COMET-QUAD for operation, the configuration routine has to implement the following procedure for all quadrants:

1. Select T1 or E1 Operation (Common for all four ports)
2. Select the Crystal Clock (Common for all four ports)
3. Select the Line Encoding
4. Select the Framing Format
5. Configure the Transmit Timing
6. Configure the Receive Timing
7. Configure the Backplane interfaces
8. Configure the Elastic Stores
9. Configure the Jitter Attenuators
10. Set the RLPS Voltage Reference value
11. Program the RLPS Equalizer RAM table
12. Enable the RLPS Equalizer

13. Program the Transmit Pulse Waveform
14. Run the XLPG Initialization Procedure
15. Run the RLPS Optimization routine (Common for all four ports)
16. Release the tri-states to begin transmission

The remainder of this section describes how to implement each of the items listed above.

4.2.1 Select T1 or E1 Operation (Common for all four ports)

Select E1 or T1 operation via the 'E1/T1B' in register 0x000 (Global Configuration). This selects E1 or T1 for all four ports.

4.2.2 Select the Crystal Clock (Common for all four ports)

Select the frequency being supplied for crystal clock (XCLK). This is done in the CSU Configuration register (0x0D6). If E1 was chosen in the previous step, then a 2.048MHz crystal must be selected here. If T1, then either a 2.048MHz or 1.544MHz crystal may be chosen.

To select the frequency for XCLK, choose the appropriate value for the MODE[2:0] bits from the table below:

Table 3: XCLK Frequencies

MODE[2:0]	XCLK frequency	T1 or E1
000	2.048 MHz	E1
001	1.544 MHz	T1
01X	Reserved	Reserved
10X	Reserved	Reserved
110	Reserved	Reserved
111	2.048 MHz	T1

Note that the setting here applies to all four ports.

4.2.3 Select the Line Encoding

If T1 mode has been selected above, the line encoding can be either AMI or B8ZS. For the receive path, the line-encoding is selected with the 'AMI' bit in register 0xQ10 (CDRC Configuration). For the transmit path, it is selected with the 'B8ZS' bit in register 0xQ54 (T1-XBAS Configuration).

If in E1 mode, the line encoding can be AMI or HDB3. For the receive path, the line-encoding is selected wiht the 'AMI' bit in register 0xQ10 (CDRC Configuration), just as in T1 mode. For the transmit path, the line-encoding is selected with the 'AMI' bit in register 0xQ80 (E1-TRAN Configuration).

4.2.4 Select the Framing Format

In T1 mode, the T1-XBAS inserts framing bits into the transmit direction (outgoing stream), while the T1-FRMR locates framing bits in the receive direction (incoming stream). Because the T1-FRMR block is not capable of detecting alarms, there is an additional block in the receive path, T1-ALMI, that is used to perform all T1 alarm detection.

The T1-XBAS, T1-FRMR, and T1-ALMI are configurable for all T1 framing formats: SF, ESF, T1DM and SLC96. The relevant registers are:

- 0xQ54 (T1-XBAS Configuration)
- 0xQ48 (T1-FRMR Configuration)
- 0xQ60 (T1-ALMI Configuration)

Each of these registers contain the bits ESF and FMS[1:0] which specify the framing format according to the following table:

Table 4: T1 Framing Modes

ESF	FMS1	FMS0	Mode
0	0	0	SF
0	0	1	T1DM
0	1	0	SLC96
0	1	1	"alternate" T1DM
1	0	0	ESF
1	0	1	Reserved
1	1	0	Reserved
1	1	1	Reserved

There is one additional register in T1 mode where the framing format needs to be specified. The SIGX block, which extracts the CAS signaling bits in the receive direction, has an 'ESF' bit in register 0xQ50. This bit must be set to '1' for ESF framing format, and set to '0' for all other T1 framing formats (SF, SLC-96, T1DM).

In E1 mode, the E1-TRAN inserts framing bytes (timeslot 0) into the outgoing stream, while the E1-FRMR locates framing bytes in the incoming stream. The E1-FRMR also detects E1 alarms.

At minimum, the E1 structure contains Basic Framing, which E1-TRAN and E1-FRMR automatically support. The Basic Framing pattern is carried in every odd framing byte (called the FAS, or "Frame Alignment Signal" bytes). In addition to Basic Framing, the E1 format optionally provides CRC Multiframing and Signaling multiframing.

To enable the optional CRC Multiframing, set GENCRC='1' in register 0xQ80 (E1-TRAN Configuration) and CRCEN='1' in register 0xQ90 (E1-FRMR Configuration). The GENCRC bit enables the E1-TRAN to insert the CRC Multiframe pattern in the transmit stream, while the CRCEN bit enables the E1-FRMR to detect the CRC Multiframe pattern in the receive stream. The CRC Multiframe pattern comprises of a MFAS (Multiframe Alignment Signal) pattern, '001011' , carried in the first bit of all even framing bytes, as well as CRC-4 computational bytes carried in the first bit of all odd framing bytes.

To enable the optional Signaling Multiframing, set SIGEN=DLEN='1' in register 0xQ80, and write CASDIS='0' in register 0xQ90. The SIGEN and DLEN bits will enable insertion of CAS (Channel Associated Signaling) – and hence Signaling Multiframing -- into timeslot 16 in the outgoing stream. The CASDIS bit controls whether Signaling Multiframe is to be detected in the receive stream.

The table below summarizes the possible settings for E1 operation:

Table 5: E1 Framing Modes

	Transmit (E1-TRAN Configuration, 0xQ80)		**Receive (E1-FRMR Configuration 0xQ90)**	
	GENCRC	**DLEN=SIGEN**	**CRCEN**	**CASDIS**
Basic Framing	0	0	0	1
Basic Framing + CRC Multiframe	1	0	1	1
Basic Framing + Signaling Multiframe	0	1	0	0
Basic Framing + CRC Multiframe *and* Signaling Multiframe	1	1	1	0

4.2.5 Configure the Transmit Timing

Configuring the timing of the transmit path involves specifying three things:

a) Source of the transmit timing. In general, there are three possible sources: (a) the backplane, either BTCLK or CMV8MCLK in H-MVIP mode (b) the Backplane datarate), (b) the external CTCLK pin (c) receive recovered clock (also called "looptiming").

b) Whether TJAT (Transmit Jitter Attenuator) is used.

c) Whether the transmit backplane (BTIF) is operating as Clock Master or Clock Slave.

The table below summarizes all the possible configurations. (It is based on the "Transmit Timing Options" table in the datasheet under the register description for 0xQ06):

Table 6: Transmit Timing configurations

	PLLREF[1:0] (Reg 0xQ06)	OCLKSEL[1:0] (Reg 0xQ06)	TXELSTBYP (Reg 0xQ06)	LINELB (Reg 0xQ06)	CMODE (0x40 BTIF Config)
<u>Transmit timing configuration:</u> Source of Transmit Timing TJAT or No TJAT Backplane Master or Slave					
Backplane With TJAT Backplane Slave	'0X'	'00'	1	0	1
Recovered clock With TJAT Backplane Master	'10'	'00'	1	0	0
Recovered clock With TJAT Backplane Slave	'10'	'00'	0	0	1
CTCLK With TJAT Backplane Master	'11'	'00'	1	0	0
CTCLK With TJAT Backplane Slave	'11'	'00'	0	0	1
Backplane No TJAT Backplane Slave	'00'	'1X'	1	0	1
CTCLK No TJAT Backplane Master	'11'	'01'	1	0	0
CTCLK No TJAT Backplane Slave	'11'	'01'	0	0	1
Lineloopback	'X0'	'00'	X	1	

4.2.6 Configure the Receive Timing

Configuring the timing of the receive path involves specifying two things:

(a) Whether the receive backplane (BRIF) is a clock master or slave. This is controlled by the 'CMODE' bit in register in register 0xQ30. In addition, the receive elastic store (RX-ELST) needs to be enabled or disabled via the 'RXELSTBYP' bit in register 0xQ02. It must be enabled anytime the backplane clock or frame pulse is a slave. In otherwords, the RX-ELST is required anytime the backplane timing differs from the line-side timing in terms of rate (clock) and/or alignment (frame pulse).

(b) Whether RJAT (Receive Jitter Attenuator) is used. This is controlled by the 'RJATBYP' bit in register 0xQ02.

4.2.7 Configure the Backplane interfaces

Configure the interface timing of the backplanes. This is controlled in the BRIF registers (0xQ30 and 0xQ31) and the BTIF Configuration register (0xQ40 and 0xQ41). In the previous two steps, the clock master or clock slave has already been specified. However, further details need to be specified, including: backplane data rate, frame pulse master or slave, frame pulse type, clock edges, and so on. Both the BRIF and BTIF contain the following bits:

RATE[1:0]: Selects the data rate of the backplane. The table below shows the setting for all available data rates. If the clock master mode was selected in the previous steps, then here you must select a backplane data rate equal the line-rate (ie. 1.544Mbit/s for T1 and 2.048Mbit/s for E1). The higher backplane rates (where the data is gapped) are supported only in clock slave mode.

Table 7: Backplane rates

RATE[1:0]	Backplane Rate
'00'	1.544 Mbit/s (T1 only)
'01'	2.048 Mbit/s (T1 or E1)
'10'	Reserved
'11'	8.192 Mbit/s H-MVIP (T1 or E1)

NX64KBIT/S[1:0]: Selects the bandwidth of data that is transferred through the backplane interfaces. A setting of '00', for "Full Frame" mode, means that the entire T1 or E1 structure will pass through the backplane.

DE: Specifies which clock edge the data (PCM and signaling) are updated or sampled on.

FE: Specifies which clock edge the frame pulse is updated or sampled on.

CMS: In clock slave mode, setting this bit allows the clock to run twice as fast as the data. The BRIF or BTIF will divide the clock by 2, ensuring the internal clock and data rate are the same.

FPMODE: Specifies whether the frame pulse is master (output) or slave (input).

In addition, you may need to specify the period of the frame pulse. The relevant bits to specify this differ between BRIF and BTIF, and also depend on whether you are in T1 or E1 mode, as well as on whether you are in frame pulse master or slave mode. The table below shows the relevant bits for each mode:

Table 8: Relevant Bits for Specifying Period of Frame Pulse

Backplane	Frame-pulse Master or Slave	Relevant Bits
BRIF	Frame-pulse Master	In E1: ROHM, BRXSMFP/BRXCMFP In T1: BRXSMFP
	Frame-pulse Slave	N/A. When the receive backplane is configured as frame pulse slave, it accepts only a basic-frame pulse, not any multi-frame pulse. Therefore, there is no need to select the period.
BTIF	Frame-pulse Master	In E1: FPTYP In T1: FPTYP, ESF_EN
	Frame-pulse Slave	In E1: FPTYP In T1: FPTYP, ESF_EN

4.2.8 Configure the Elastic Stores

Both Elastic Stores need to be configured to T1 mode or E1 mode. This is done via the 'IR' and 'OR' bits (in registers 0xQ20 for TX-ELST, and 0xQ1C for RX-ELST). For T1, set IR=OR=0. For E1, set IR=OR=1.

4.2.9 Configure the Jitter Attenuators

Both jitter-attenuators (RJAT and TJAT) need to be configured. Each have an N1/N2 divisor value that needs to be programmed for its internal PLL to function correctly. For RJAT, set N1=N2= 0x2F for T1, or N1=N2= 0xFF for E1. For TJAT, please refer to the table in the datasheet (under the register description for 0xQ1A). For your convenience, a simplified version of the table is copied below:

Table 9: Jitter Attenuator N1 and N2 values

Transmit clock reference	T1 or E1 mode	N1	N2
1.544MHz (from Backplane, Recovered, or CTCLK)	T1	0x2F	0x2F
2.048MHz (from Backplane, Recovered,	E1	0xFF	0xFF

or CTCLK)			
2.048MHz (from CTCLK)	T1	0xFF	0xC0
1.544MHz (from CTCLK)	E1	0xC0	0xFF
Nominal 1.544MHz (Backplane H-MVIP mode)	T1	0xC0	0xC0
8kHz (CTCLK)	T1	0x00	0xC0
16kHz (CTCLK)	T1	0x01	0xC0
8kHz (CTCLK)	E1	0x00	0xFF
16kHz (CTCLK)	E1	0x01	0xFF

Note: Even if the TJATs are not used, they still must be fully configured, because they are used to generate high-speed clocks used internally by the XLPG analog transmitters.

4.2.10 Set the RLPS Voltage Reference value

The RLPS (the analog receiver) requires an equalizer voltage reference value to be programmed into the 'EQ_VREF[5:0]' bits in registers 0xQDC. For T1, use the value 0x2C. For E1, use the value 0x3D.

4.2.11 Program the RLPS Equalizer RAM table

Program the Receive Equalizer RAM table into the RLPS Indirect Registers. This table must be programmed in order to recover analog pulses. This table consists of 256 rows, with each row consisting of 4 bytes. A total of 256 indirect writes are necessary to program the entire table.

A single indirect write is done as follows:

(1) Wait at least 3 line-rate cycle to ensure that previous indirect accesses are complete.

(2) Write the four data bytes into the four "data" registers: 0xQD8, 0xQD9, 0xQDA, and 0xQDB.

(3) Set the 'RWB' bit (Read Write select) in register 0xQFD to 0 to select a write operation.

(4) Write the Indirect Address into register 0xQFC (the "address" register). This initiates the indirect write, which will take one line-rate cycle to complete. The values for the Indirect Address will range from 0x00 to 0xFF, representing the 256 rows of the Equalizer RAM table.

4.2.12 Enable the RLPS Equalizer

Enable the Receive Equalizer by setting the 'EQEN' bit in registers 0xQFF to '1'. This will enable the equalizer by activating the feedback loop. The equalizer will begin to move up and down the Equalizer RAM table according to changes in the incoming signal level.

4.2.13 Program the Transmit Pulse Waveform

Program the Transmit Pulse Waveform. Each pulse waveform provided in the datasheet includes a Transmit Waveform table, which is programmed into Indirect Registers and controls the *shape* of the pulse, as well as a SCALE[4:0] value (register 0xQF0) which controls the *amplitude* of the pulse. When writing a SCALE[4:0] value into the register 0xQF0, it is usually recommended to leave the HIGHZ bit as '1' (in the same register), so that the transmitter remains in high-impedance during the remainder of the configuration. This avoids garbage from being transmitted during configuration.

Programming the Transmit Waveform table consists of 24 rows and 5 columns. Therefore, there are a total of 24x5 = 120 indirect writes are necessary to program the entire table. Each Indirect write is done as follows:

(1) Wait one line-rate cycle to ensure that previous indirect accesses are complete.

(2) Write the data byte into the "data" register (0xQF3)

(3) Write the Indirect Address into the "address" register (0xQF2). This will initiate the Indirect write. The Indirect Address is comprised of the SAMPLE[4:0] bits, which index the 24 rows of the table, and the UI[2:0] bits, which index the 5 columns of the table. Therefore, valid values for the SAMPLE index range from '00000' to '10111', and valid values for the UI index range from '000' to '100'

4.2.14 Run the XLPG Initialization Procedure

Run the XLPG Initialization procedure outlined in the Datasheet (PMC-1990315). This procedure must be run for each quadrant after device reset. Its purpose is to compensate for variations in fabrication of the XLPG analog drivers. If it isn't done, the transmit pulse waveform may vary in amplitude over time. The XLPG Initialization procedure is as follows:

(1) Write 06H to register 0xQF6 (XLPG Initialization register)

(2) Write 00H to register 0xQF4 (XLPG Configuration #1 register)

(3) Write 00H to register 0xQF5 (XLPG Configuration #2 register)

(4) Write 00H to register 0xQF6

After the sequence has been completed, the XLPG Initialization register bits must remain at logic 0 until the next time the COMET-QUAD is reset.

4.2.15 Run the RLPS Optimization routine (Common for all four ports)

Run the RLPS Optimization routine. This is a sequence of 29 writes, which is provided in the datasheet in the Operation section, under "Using the Line Receiver". For convenience, the sequence is copied in the table below. This routine needs to be run only once, after configuring all four ports.

Table 10: Write Sequence for RLPS Optimization Routine

Step	Address	Data
1	0x4D7	0x00
2	0x4F1	0x00
3	0x5F1	0x00
4	0x6F1	0x00
5	0x7F1	0x00
6	0x4F9	0x00
7	0x5F9	0x00
8	0x6F9	0x00
9	0x7F9	0x00
10	0x4F9	0x04
11	0x4FB	0x09
12	0x00B	0x20
13	Wait 1 ms	
14	0x4F9	0x00
15	0x00B	0x00
16	0x5F9	0x04
17	0x5FB	0x09
18	0x00B	0x20
19	Wait 1 ms	
20	0x5F9	0x00
21	0x00B	0x00
22	0x6F9	0x04
23	0x6FB	0x09
24	0x00B	0x20
25	Wait 1 ms	
26	0x6F9	0x00
27	0x00B	0x00
28	0x7F9	0x04
29	0x7FB	0x09
30	0x00B	0x20
31	Wait 1 ms	
32	0x7F9	0x00
33	0x00B	0x00

4.2.16 Release the tri-states to begin transmission

After the device is fully configured, one can begin transmission of T1 or E1 signals. This is done by clearing the HIGHZ bits in register 0xQF0 (XLPG Line Driver Configuration register).

4.3 Using the Per-Channel Serial Controllers

The Per-Channel Serial Controllers (TPSC and RPSC) control the per-channel functions for the transmit and receive PCM data.

The software should configure the TPSC and RPSC while in ***Idle*** state. The following steps should be followed in order:

1) Disable the PCCE bit and Enable the IND bit in the register 0xQ6C (TPSC Configuration) or register 0xQ70 (RPSC Configuration).

2) Program the required TPSC (or RPSC) configuration.

3) Enable the PCCE bit in the registers 0xQ6C (TPSC) or 0xQ70 (RPSC).

The following sections discuss step 2.

4.3.1 Using RPSC

On power up, the per-channel control is normally disabled by default. Once the RPSC is configured, the PCCE bit in register 0xQ70 must be set to activate per channel control.

The RPSC Indirect Registers (PCM Data Control byte (0x20-0x3F), Data Trunk Conditioning Code byte (0x40-0x5F), and Signaling Trunk Conditioning byte (0x61-0x7F)) are accessible on a per-channel basis. Table 11 provides a summary of some of the control bits (in the RPSC Indirect Registers 20H-3FH: PCM Data Control byte) that must be programmed to configure the channels.

Table 11: Control Bits for the PCM Data Control byte

Bit		Resulting Action
TEST	0	No test features applied.
	1	Depending on the value of the RXPATGEN bit of the PRBS Positioning/Control (0xQ0F), performs the tests: 0: Channel data is checked against the PRBS test pattern. 1: Channel data is overwritten with the PRBS test pattern.
DTRKC	0	No data trunk conditioning applied
	1	BRPCM output data is replaced with content of Data Trunk Conditioning Code Byte (0x40-0x5F) for the channel
STRKC	0	No signaling trunk conditioning applied
	1	BRSIG output data is replaced with content of Signaling Trunk Conditioning Byte (0x61-0x7F) for the channel
DMW	0	No digital milliwatt pattern replaces BRPCM output data
	1	BRPCM output data is replaced with a digital milliwatt pattern (based on the DMWALAW bit) for the channel
DMWALAW	0	Only applicable if DMW = 1. Digital milliwatt pattern used is the digital A-law pattern
	1	Only applicable if DMW = 1. Digital milliwatt pattern used is the μ-law pattern

Bit		Resulting Action
SIGNINV	0	No action
	1	Most Significant Bit of data output on BRPCM pin is inverted for the channel.

Notes:

1. RPSC indirect registers have no default values after the COMET-QUAD is reset; therefore, values must be programmed in for the desired configuration.
2. If a channels DTRKC and/or STRKC bit(s) is/are set, then values must be set in the corresponding Conditioning Code Byte. If none are set, default values will be sent.

RPSC Indirect Register Access

The RPSC indirect registers are for the per-channel control of the receive serial data stream. On power-up, the per-channel control is normally disabled by default. The PCCE bit in register 0xQ70 must be set to enable per channel control.

The "IND" bit must be set before the software can access RPSC indirect registers. On power-up or software reset, this bit is clear by default. Once the "IND" bit is set, the following pseudo code shows how to read a RPSC indirect register and how to write to a RPSC indirect register.

```
#define REG_RPSC_INDIRECT_ADDRESS                0xQ72
#define REG_RPSC_MICRO_ACCESS_STATUS        0xQ71
#define REG_RPSC_INDIRECT_DATA              0xQ73
#define MAX_BUSY_READ                       0x5
#define BIT_BUSY                            0x80

/** to read a RPSC register: **/

int ReadRpsc(int offset, unsigned char &value)
{
      int i;

      WRITE_REG(REG_RPSC_INDIRECT_ADDRESS, (offset | 0x80));
      for (i = 0; i < MAX_BUSY_READ; i++)
      {
            value = READ_REG(REG_RPSC_MICRO_ACCESS_STATUS);
            if ((value & BIT_BUSY) == 0)
            {
                  value = READ_REG(REG_RPSC_INDIRECT_DATA);
                  return SUCCESS;
            }
      }
      return FAILURE;
}

/** to write a RPSC register: **/

int WriteRpsc(int offset, unsigned char value)
```

```
{
      unsigned char temp;
      int i;

      WRITE_REG(REG_RPSC_INDIRECT_DATA, value);
      WRITE_REG(REG_RPSC_INDIRECT_ADDRESS, (offset & 0x7F));
      for (i = 0; i < MAX_BUSY_READ; i++)
      {
            temp = READ_REG(REG_RPSC_MICRO_ACCESS_STATUS);
            if ((temp & BIT_BUSY) == 0) return SUCCESS;
      }
      return FAILURE;
}
```

4.3.2 Using TPSC

The TPSC indirect registers provide per-channel serial control over the transmit data stream. The per-channel control is normally disabled by default, on power-up. The PCCE bit in register 0xQ6C must be set to activate per channel control once the RPSC is configured.

On power up, the per-channel control is normally disabled by default. Once the TPSC is configured, the PCCE bit must be set to activate per channel control. See section 0 for accessing the TPSC indirect registers.

The TPSC Indirect Registers (PCM Data Control byte (0x20-0x3F), IDLE Code byte (0x40-0x5F) and Signaling/E1 Control byte (0x60-0x7F)) are accessible on a per-channel basis.

The PCM Data Control byte provides control over how to condition the data received on the BTPCM pin, as shown in Table 12.

Table 12: Control Bits for the PCM Data Control byte

Bit		Resulting Action
INVERT	0	PCM data of this channel is not inverted.
	1	PCM data for this channel is inverted.
IDLE_CHAN	0	PCM data of this channel is not overwritten with idle code
	1	Idle code overwrites PCM data for this channel
DMW	0	PCM channel data is not replaced with digital mW pattern.
	1	Digital mW pattern replaces channel PCM data.
TEST	0	No test features applied.
	1	Depending on the value of the RXPATGEN bit of the PRBS Positioning/Control (0xQ0F), performs the tests: 0: Channel data is checked against the PRBS test pattern. 1: Channel data is overwritten with the PRBS test pattern.
LOOP	0	Channel PCM data is not looped back.
	1	PCM data for this channel is looped back to receive path.
ZCS[1:0]	00	No zero-code suppression.
	01	GTE zero-code suppression.
	10	Jammed bit 8.
	11	Bell zero-code suppression.

The IDLE Code byte stores a constant value that the programmer can transmit for any channel. The Signaling/E1 Control byte provides data manipulation in E1 mode and signaling insertion in T1 mode.

TPSC Indirect Register Access

The TPSC indirect registers are for the per-channel control of the transmit serial data stream. On power-up, the per-channel control is normally disabled by default; therefore, the PCCE bit in register 0xQ6C must be set to enable per channel control once the TPSC is configured.

The IND bit must be set before the software can access TPSC indirect registers. On power-up or software reset, this bit is clear by default. Once the IND bit is set, the following pseudo code shows how to read a TPSC indirect register and write to a TPSC indirect register.

```
#define REG_TPSC_INDIRECT_ADDRESS                0xQ6E
#define REG_TPSC_MICRO_ACCESS_STATUS        0xQ6D
#define REG_TPSC_INDIRECT_DATA              0xQ6F
#define MAX_BUSY_READ                       0x5
#define BIT_BUSY                            0x80

/** to read a TPSC register: **/

int ReadTpsc(int offset, unsigned char &value)
{
      int i;
```

```
        WRITE_REG(REG_TPSC_INDIRECT_ADDRESS, (offset | 0x80));
        for (i = 0; i < MAX_BUSY_READ; i++)
        {
                value = READ_REG(REG_TPSC_MICRO_ACCESS_STATUS);
                if ((value & BIT_BUSY) == 0)
                {
                        value = READ_REG(REG_TPSC_INDIRECT_DATA);
                        return SUCCESS;
                }
        }
        return FAILURE;
}

/** to write a TPSC register: **/

int WriteTpsc(int offset, unsigned char value)
{
        unsigned char temp;
        int i;

        WRITE_REG(REG_TPSC_INDIRECT_DATA, value);
        WRITE_REG(REG_TPSC_INDIRECT_ADDRESS, (offset & 0x7F));
        for (i = 0; i < MAX_BUSY_READ; i++)
        {
                temp = READ_REG(REG_TPSC_MICRO_ACCESS_STATUS);
                if ((temp & BIT_BUSY) == 0) return SUCCESS;
        }
        return FAILURE;
}
```

4.4 Using the Signaling Extractor

The Signaling Extraction (SIGX) block provides channel associated signaling (CAS) extraction from an E1 signaling multiframe or from SF, SLC®96, and ESF T1 Formats. The SIGX block extracts the signaling bits from the received data stream and serializes the results on the BRSIG output pin.

Control of CAS is normally disabled by default, on power-up or reset. The PCCE bit in register 0xQ50 (SIGX Configuration Register) must be set to activate per channel control. The SIGX configuration should be setup when the COMET-QUAD is in ***Configuration*** state. The following steps should be followed in order:

1) Disable the PCCE and enable the IND and COSS bit (if indirect registers need to be accessed) in register 0xQ50.

2) Program the required SIGX Configuration.

3) Disable and COSS bits and enable the PCCE bit in register 0xQ50.

Register 0xQ50 (SIGX Configuration) can represent two different registers depending on the value of its COSS bit. This is discussed below.

COSS = 0: The SIGX indirect registers are disabled on reset and have no default value when enabled. Setting the IND bit to logic 1 and the COSS bit to logic 0 in register 0xQ50 (SIGX Configuration Register) enables access to the SIGX configuration indirect registers.

The SIGX configuration indirect registers (Timeslot/Channel Signaling Data (0x20-0x3F) and Per-Timeslot Configuration (0x40-0x5F)) are accessible on a per-channel basis.

Timeslot/Channel Signaling Data registers store signaling data sent to the BRSIG pin. These registers contain the A,B,C,D bits for the ESF, SF, SLC®96 and signaling multiframe format (where bits C and D are replaced with A and B for SF and SLC®96). The Per-Timeslot Configuration registers provide data conditioning in conjunction with the RPSC Data Control bytes.

COSS = 1: The registers 0xQ50-0xQ53 (SIGX Change of Signaling State) are used to indicate a change of signaling state on a channel/timeslot. In T1 mode, COSS bits 1 to 24 represent each of the 24 channels. In E1 mode, COSS bits 1 to 30 represent each of the 30 timeslots. These registers may be polled to determine the channel that had a signaling state change. The COSS bits are cleared when the register is read.

4.4.1 SIGX Indirect Register Access

The SIGX indirect registers store the channel associated signaling from framing formats. On power-up, the per-channel control is disabled by default. The PCCE bit in register 0xQ50 (with COSS = 0) must be set to enable per channel control.

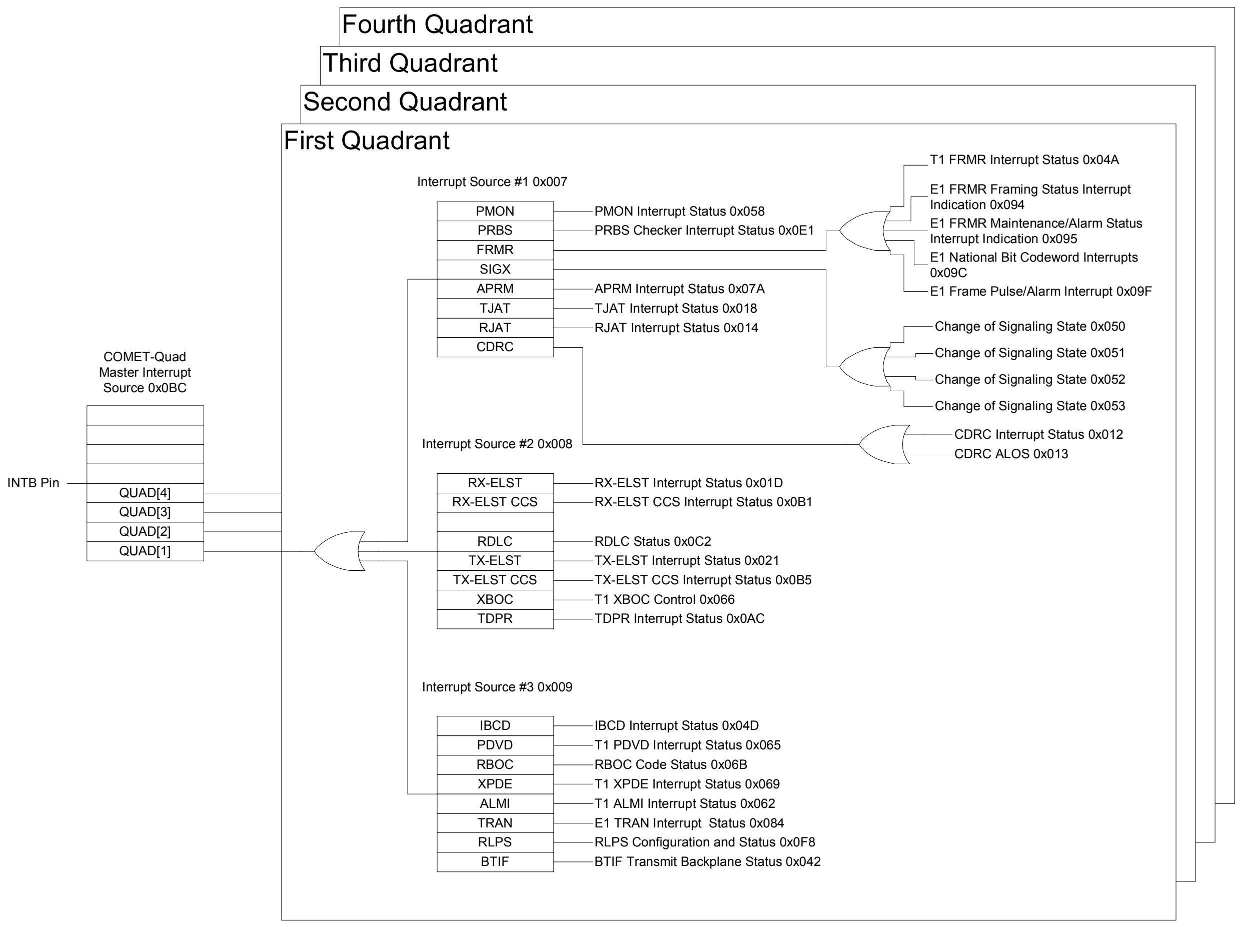
The IND bit must be set to logic 1 and the COSS bit must be set to logic 0 before the software can access SIGX indirect registers. By default, on power-up or software reset, these bits are cleared. Once the IND bit is set, the following pseudo code shows how to read a SIGX indirect register and write to a SIGX indirect register.

```
#define REG_SIGX_INDIRECT_ADDRESS           0xQ52
#define REG_SIGX_INDIRECT_STATUS       0xQ51
#define REG_SIGX_INDIRECT_DATA         0xQ53
#define MAX_BUSY_READ                  0x5
#define BIT_BUSY                       0x80

/** to read a SIGX register: **/

int ReadSigx(int offset, unsigned char &value)
{
      int i;
      WRITE_REG(REG_SIGX_INDIRECT_ADDRESS, (offset | 0x80));
      for (i = 0; i < MAX_BUSY_READ; i++)
      {
            value = READ_REG(REG_SIGX_MICRO_ACCESS_STATUS);
            if ((value&BIT_BUSY) == 0)
            {
                  value = READ_REG(REG_SIGX_INDIRECT_DATA);
                  return SUCCESS;
            }
      }
      return FAILURE;
}

/** to write a SIGX register: **/

int WriteSigx(int offset, unsigned char value)
{
      unsigned char temp;
      int i;

      WRITE_REG(REG_SIGX_INDIRECT_DATA, value);
      WRITE_REG(REG_SIGX_INDIRECT_ADDRESS, (offset & 0x7F));
      for (i = 0; i < MAX_BUSY_READ; i++)
      {
            temp = READ_REG(REG_SIGX_MICRO_ACCESS_STATUS);
            if ((temp & BIT_BUSY) == 0) return SUCCESS;
      }
      return FAILURE;
}
```

5 Interrupts using register access

COMET-QUAD has a single interrupt pin (INTB) flagging the interrupt request status for all functional blocks and quadrants. The interrupt status of each quadrant and functional block is indicated by a number of interrupt status registers, as shown in Figure 2.

Figure 2 COMET-QUAD Interrupt Status Register Hierarchy


Fourth Quadrant
Third Quadrant
Second Quadrant
First Quadrant
Interrupt Source #1 0x007
PMON
PRBS
FRMR
SIGX
APRM
TJAT
RJAT
CDRC
PMON Interrupt Status 0x058
PRBS Checker Interrupt Status 0x0E1
APRM Interrupt Status 0x07A
TJAT Interrupt Status 0x018
RJAT Interrupt Status 0x014
T1 FRMR Interrupt Status 0x04A
E1 FRMR Framing Status Interrupt Indication 0x094
E1 FRMR Maintenance/Alarm Status Interrupt Indication 0x095
E1 National Bit Codeword Interrupts 0x09C
E1 Frame Pulse/Alarm Interrupt 0x09F
Change of Signaling State 0x050
Change of Signaling State 0x051
Change of Signaling State 0x052
Change of Signaling State 0x053
CDRC Interrupt Status 0x012
CDRC ALOS 0x013
COMET-Quad Master Interrupt Source 0x0BC
INTB Pin
QUAD[4]
QUAD[3]
QUAD[2]
QUAD[1]
Interrupt Source #2 0x008
RX-ELST
RX-ELST CCS
RDLC
TX-ELST
TX-ELST CCS
XBOC
TDPR
RX-ELST Interrupt Status 0x01D
RX-ELST CCS Interrupt Status 0x0B1
RDLC Status 0x0C2
TX-ELST Interrupt Status 0x021
TX-ELST CCS Interrupt Status 0x0B5
T1 XBOC Control 0x066
TDPR Interrupt Status 0x0AC
Interrupt Source #3 0x009
IBCD
PDVD
RBOC
XPDE
ALMI
TRAN
RLPS
BTIF
IBCD Interrupt Status 0x04D
T1 PDVD Interrupt Status 0x065
RBOC Code Status 0x06B
T1 XPDE Interrupt Status 0x069
T1 ALMI Interrupt Status 0x062
E1 TRAN Interrupt Status 0x084
RLPS Configuration and Status 0x0F8
BTIF Transmit Backplane Status 0x042

5.1 Interrupt Status and Interrupt Enable Bits

The COMET-QUAD_Master_Interrupt_Register(0x0BC) indicates which COMET-QUAD quadrant has an interrupt status flagged. Each quadrant has three interrupt source registers, Interrupt_Source_#1-3(0xQ07-0xQ09), indicating which functional block has an active interrupt request.

Each functional block has at least one status register indicating what type of interrupt occurred in the block. Blocks having more than 8 types of interrupts (such as FRMR and SIGX) may have more than one interrupt status register. The status registers contain interrupt status bits (also called 'I' bits) indicating what kind of interrupt occurred within the block.

Each 'I' bit has an associated interrupt enable bit (also called the 'E' bit). If this bit is enabled, the value of the 'I' bit is propagated to the associated interrupt source register, and accordingly, to the COMET-QUAD_Master_Interrupt_Source(0x0BC) register. If the 'E' bit is disabled, the 'I' bit will still be asserted in case of an interrupt condition, but its value will not be propagated to that block's interrupt source register (and onward to the master interrupt source register). Hence, if the 'E' bit is disabled, that interrupt condition will be masked and will not generate an interrupt on the INTB pin, thus indicating that it does not need any attention from the processor. All 'E' bits are disabled upon COMET-QUAD reset, which means that the chip will not generate any interrupts after a reset. The user has to explicitly set the 'E' bits for the COMET-QUAD to generate desired interrupts.

The INTB pin on the COMET-QUAD will remain asserted until the last requested interrupt is confirmed by reading the block's status register, which means that COMET-QUAD will remain in 'interrupt' state until all the requested interrupts have been serviced.

5.2 Interrupt Reference

Table 13 below references all interrupt bits in the register they appear and the location and name of the corresponding enable/disable and status bit. The table is shown in the order displayed by Figure 2.

Table 13: Interrupt Reference

Interrupt		**Enable/Disable**		**Status**		**Comments**
Register	**"I" Bit or source group**	**Register**	**"E" Bit**	**Register**	**"V" Bit**	
Interrupt Source #1 0xQ07	PMON	------	------	------	------	This is a top-level interrupt source register so no enable/disable or status bits exist. Individual "I" bits belonging to a group are shown later in this table.
	PRBS	------	------	------	------	
	FRMR	------	------	------	------	
	SIGX	------	------	------	------	
	APRM	------	------	------	------	
	TJAT	------	------	------	------	
	RJAT	------	------	------	------	
	CDRC	------	------	------	------	
Interrupt	RX-ELST	------	------	------	------	This is a top-level

Interrupt		Enable/Disable		Status		Comments
Register	"I" Bit or source group	Register	"E" Bit	Register	"V" Bit	
Source #2 0xQ08	RX-ELST CCS	------	------	------	------	interrupt source register so no enable/disable bits exist. Individual "I" bits belonging to a group are shown later in this table.
	Unused	------	------	------	------	
	RDLC	------	------	------	------	
	TX-ELST	------	------	------	------	
	TX-ELST CCS	------	------	------	------	
	XBOC	------	------	------	------	
	TDPR	------	------	------	------	
Interrupt Source #3 0xQ09	IBCD	------	------	------	------	This is a top-level interrupt source register so no enable/disable bits exist. Individual "I" bits belonging to a group are shown later in this table.
	PDVD	------	------	------	------	
	RBOC	------	------	------	------	
	XPDE	------	------	------	------	
	ALMI	------	------	------	------	
	TRAN	------	------	------	------	
	RLPS	------	------	------	------	
	BTIF	------	------	------	------	
PMON Interrupt Enable /Status 0xQ58	XFER	PMON Interrupt Enable /Status 0xQ58	INTE	------	------	
	OVR			------	------	
PRBS Checker Interrupt Enable /Status 0xQE1	SYNCI	PRBS Checker Interrupt Enable /Status 0xQE1	SYNCE	PRBS Checker Interrupt Enable /Status 0xQE1	SYNCV	
	BEI		BEE		------	
	XFERI		XFERE		------	
T1 FRMR Interrupt Status 0xQ4A	COFAI	T1 FRMR Interrupt Enable 0xQ49	COFAE	T1 FRMR Interrupt Status 0xQ4A	------	
	FERI		FERE		------	
	BEEI		BEEE		------	
	SFEI		SFEE		------	
	MFPI		MFPE		MFP	
	INFRI		INFRE		INFR	

Interrupt		Enable/Disable		Status		Comments
Register	"I" Bit or source group	Register	"E" Bit	Register	"V" Bit	
E1 FRMR Framing Status Interrupt Indication 0xQ94	C2NCIWI	E1 FRMR Framing Status Interrupt Enable 0xQ92	C2NCIWE	E1 FRMR Framing Status 0xQ96	C2NCIWV	
	OOFI		OOFE		OOFV	
	OOSMFI		OOSMFE		OOSMFV	
	OOCMFI		OOCMFE		OOCMFV	
	COFAI		COFAE		------	
	FERI		FERE		------	
	SMFERI		SMFERE		------	
	CMFERI		CMFERE		------	
E1 FRMR Maintenance /Alarm Status Interrupt Indication 0xQ95	RAII	E1 FRMR Maintenanc e /Alarm Status Interrupt Enable 0xQ93	RAIE	E1 FRMR Maintenanc e/Alarm Status 0xQ97	RAIV	
	RMAII		RMAIE		RMAIV	
	AISDI		AISDE		AISD	
	REDI		REDE		RED	
	AISI		AISE		AIS	
	FEBEI		FEBEE		------	
	CRCEI		SMFERE		------	
E1 FRMR National Bit Codeword Interrupts 0xQ9C	Sa4I	E1 FRMR National Bit Codeword Interrupt Enable 0xQ9B	Sa4E	------	------	
	Sa5I		Sa5E		------	
	Sa6I		Sa6E		------	
	Sa7I		Sa7E		------	
	Sa8I		Sa8E		------	
E1 Frame Pulse/Alarm Interrupt 0xQ9F	OOOFI	E1 Frame Pulse /Alarm Interrupt Enable 0xQ9E	OOOFE	E1 FRMR Framing Status 0xQ96	OOOFV	
	RAICCRCI		RAICCRC E		RAICCRC V	
	CFEBEI		CFEBEE		CFEBEV	
	V52LINKI		V52LINKE		V52LINKV	
	BRFPI		BRFPE		------	
	ICSMFPI		ICSMFPE		------	
	ICMFPI		ICMFPE		------	
	ISMFPI		ISMFPE		------	

Interrupt		Enable/Disable		Status		Comments
Register	"I" Bit or source group	Register	"E" Bit	Register	"V" Bit	
SIGX Configuration /Change of Signaling State 0xQ50	COSS [25..30]	SIGX Configuration /Change of Signaling State 0xQ50	SIGE	------	------	The COSS bit in register 0x50 must be set to logic 1 for this register to be used as shown.
SIGX Microproc. Access Status/ Change of Signaling State 0xQ51	COSS [24..17]	SIGX Configuration /Change of Signaling State 0xQ50	SIGE	------	------	The COSS bit in register 0x50 must be set to logic 1 for this register to be used as shown.
SIGX Channel Indirect Address/Control/Change of Signaling State 0xQ52	COSS [16..9]	SIGX Configuration /Change of Signaling State 0xQ50	SIGE	------	------	The COSS bit in register 0x50 must be set to logic 1 for this register to be used as shown.
SIGX Channel Indirect Data Buffer/ Change of Signaling State 0xQ53	COSS [8..1]	SIGX Configuration /Change of Signaling State 0xQ50	SIGE	------	------	The COSS bit in register 0x50 must be set to logic 1 for this register to be used as shown.
T1 APRM Interrupt Status 0xQ7A	INTR	T1 APRM Configuration/Control 0xQ78	INTE	------	------	
TJAT Interrupt Status 0xQ18	OVRI	TJAT Configuration 0xQ1B	OVRE	------	------	
	UNDI		UNDE		------	

Interrupt		Enable/Disable		Status		Comments
Register	**"I" Bit or source group**	**Register**	**"E" Bit**	**Register**	**"V" Bit**	
RJAT Interrupt Status 0xQ14	OVRI	RJAT Configuration 0xQ17	OVRE	------	------	
	UNDI		UNDE		------	
CDRC Interrupt Status 0xQ12	LCVI	CDRC Interrupt Enable 0xQ11	LCVE	CDRC Interrupt Status 0xQ12	------	
	LOSI		LOSE		LOSV	
	LCSDI		LCSDE		------	
	ZNDI		ZNDE		------	
CDRC Alternate Loss of Signal 0xQ13	ALTLOSI	CDRC Alternate Loss of Signal 0xQ13	ALTLOSE	CDRC Alternate Loss of Signal 0xQ13	ALTLOS	
RX-ELST Interrupt Enable Status 0xQ1D	SLIPI	RX-ELST Interrupt Enable Status 0xQ1D	SLIPE	RX-ELST Interrupt Enable Status 0xQ1D	SLIPD	
RX-ELST CCS Interrupt Enable Status 0xQB1	SLIPI	RX-ELST CCS Interrupt Enable Status 0xQB1	SLIPE	RX-ELST CCS Interrupt Enable Status 0xQB1	SLIPD	
RDLC Status 0xQC2	OVR	RDLC Interrupt Control 0xC1	INTE	------	------	
	COLS			------	------	
	PKIN			------	------	
TX-ELST Interrupt Enable Status 0xQ21	SLIPI	TX-ELST Interrupt Enable Status 0xQ21	SLIPE	TX-ELST Interrupt Enable Status 0xQ21	SLIPD	

Interrupt		**Enable/Disable**		**Status**		**Comments**
Register	**"I" Bit or source group**	**Register**	**"E" Bit**	**Register**	**"V" Bit**	
TX-ELST CCS Interrupt Enable Status 0xQB5	SLIPI	TX-ELST CCS Interrupt Enable Status 0xQB5	SLIPE	TX-ELST CCS Interrupt Enable Status 0xQB5	SLIPD	
T1 XBOC Control 0xQ66	BOCSMPI	T1 XBOC Control 0xQ66	BOCSMPE	------	------	
TDPR Interrupt Status UDR Clear 0xQAC	PRINTI	TDPR Interrupt Enable 0xQAB	PRINTE	TDPR Interrupt Status UDR Clear 0xQAC	------	
	FULLI		FULLE		FULL	
	OVRI		OVRE		------	
	UDRI		UDRE		------	
	LFILLI		LFILLE		BLFILL	
IBCD Interrupt Enable/ Status 0xQ4D	LBAI	IBCD Interrupt Enable/ Status 0xQ4D	LBAE	IBCD Interrupt Enable/ Status 0xQ4D	LBA	
	LBDI		LBDE		LBD	
T1 PDVD Interrupt Enable/ Status 0xQ65	Z16DI	T1 PDVD Interrupt Enable/ Status 0xQ65	Z16DE	T1 PDVD Interrupt Enable/ Status 0xQ65	------	
	PDVI		PDVE		PDV	
T1 RBOC Code Status 0xQ6B	IDLEI	T1 RBOC Enable 0xQ6A	IDLI	------	------	
	BOCI		BOCE		------	
T1 XPDE Interrupt Enable/ Status 0xQ69	STUFI	T1 XPDE Interrupt Enable/ Status 0xQ69	STUFE	T1 XPDE Interrupt Enable/ Status 0xQ69	------	
	Z16DI		Z16DE		------	
	PDVI		PDVE		PDV	
T1 ALMI Interrupt Status	YELI	T1 ALMI Interrupt Enable	YELE	T1 ALMI Interrupt Status	YEL	
	REDI		REDE		RED	

Interrupt		Enable/Disable		Status		Comments
Register	"I" Bit or source group	Register	"E" Bit	Register	"V" Bit	
Status 0xQ62	AISI	Enable 0xQ61	AISE	Status 0xQ62	AIS	
E1 TRAN Interrupt Status 0xQ85	SIGMFI	E1 TRAN Interrupt Enable 0xQ84	SIGMFE	------	------	
	NFASI		NFASE		------	
	MFI		MFE		------	
	SMFI		SMFE		------	
	FRMI		FRME		------	
RLPS Configuration and Status 0xQF8	ALOSI	RLPS Configuration and Status 0xQF8	ALOSE	RLPS Configuration and Status 0xQF8	ALOSV	
BTIF Parity Configuration and Status 0xQ42	BTPCMI	BTIF Parity Configuration and Status 0xQ42	TPTYE	------	------	Both of these interrupt bits are enabled by TPTYE
	BTSIGI					

5.3 Interrupt Processing using register accesses

In order to isolate the functional block that requested an interrupt, the COMET-QUAD interrupt routine has to:

Read the COMET-QUAD_Master_Interrupt_Source(0x0BC) register to identify the COMET-QUAD quadrant that requested the interrupt,

For each quadrant that requested an interrupt, read all three interrupt source registers Interrupt_Source_#1-3(0xQ07-0xQ09) to identify the block that requested the interrupt, and

For each block that requested an interrupt, read the block's status register(s) to determine what kind of interrupt occurred, and

Service the requesting interrupt.

The following sample code demonstrates a typical interrupt service routine for COMET-QUAD:

```
/* master interrupt source register for all quadrants, */
/* quadrants are numbered 0-3                          */
#define QUAD_ISR_OFFSET 0x0bc
#define NUM_QUADRANTS 4

/* base offsets for per-quadrant interrupt source regs */
/* actual offset is quadrant * 0x100 + base offset     */
#define NUM_IS_REGISTERS 3
const unsigned long isr[NUM_IS_REGISTERS] =
      {0x007, 0x008, 0x009};

/* interrupt callback table, populated elsewhere,      */
/* these callbacks service per-block interrupts and    */
/* take the quadrant index as an argument              */
typedef void (*INTERRUPT_CALLBACK)(int quadrant);
INTERRUPT_CALLBACK intr_cbk[NUM_IS_REGISTERS][8];

void interrupt_service_routine()
{
      int i, j, q;
      unsigned char quad_is, isn;

      /* read the master interrupt source register */
      quad_is = READ_REG(QUAD_ISR_OFFSET);

      /* for each interrupting quadrant… */
      for (q = 0; q < NUM_QUADRANTS; q++)
      {
            if (quad_is & 1 << q)
            {
                  /* read all isr registers for quadrant */
                  for (i = 0; i < NUM_IS_REGISTERS; i++)
                  {
                        isn = READ_REG(0x100 * q + isr[i]);

                        if (isn)
                        {
                              for (j = 0; j < 8; j++)
                              {
                                    /* service the interrupt
                                     * flagged by the j-th bit of
                                     * i-th isr register */
                                    if (isn & (1 << j))
                                          (*(intr_cbk[i][j]))(q);
                              }
                        }
                  }
```

```
                        }
                }
        }
```

6 HDLC Programming using register access

Each COMET-QUAD quadrant has a single HDLC controller. The HDLC controllers are equipped with two 128-byte FIFOs: one for the transmit path, and one for the receive path.

6.1 Using the HDLC Transmitter (TDPR) Block

The HDLC Transmitter (TDPR) sends the HDLC frames placed in the transmit FIFO automatically at a rate provided by the transmit clock. In order to allow the TDPR to send HDLC frames at the appropriate rate, the transmit FIFO has to be filled with data at approximately the same rate. If the FIFO is filled at a high rate, the FIFO will be overrun with data. If the FIFO is filled at a low rate, the FIFO will be drained and a data underrun will occur.

The TDPR automatically starts transmitting HDLC frames when one of the following events occur:

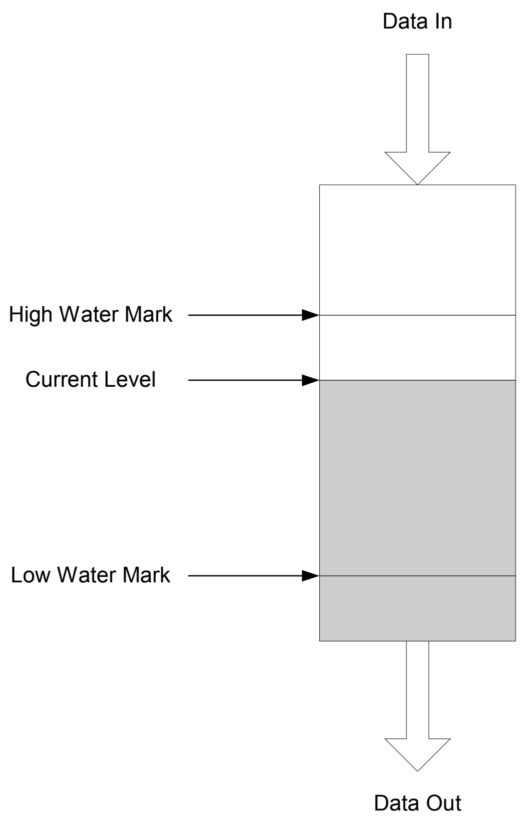
- When a complete HDLC frame is placed in the transmit FIFO, and
- When the transmit FIFO is filled to its programmable upper limit.

To ensure smooth transmission of HDLC frames, two registers are provided, as shown in Figure 3:

- Register 0xQA9 (TDPR Upper Transmit Threshold), specifying the upper fill limit at which the transmit FIFO automatically starts transmitting the data (i.e. high water mark), and
- Register 0xQAA (TDPR Lower Interrupt Threshold), specifying the lower fill limit at which an interrupt is generated to warn that the FIFO is about to be drained (i.e. low water mark).

Once transmission is started, it will not stop until all complete HDLC frames in the FIFO are transmitted AND the FIFO level is below or at the high water mark.

Figure 3 TDPR FIFO



Note: *To facilitate proper operation, TDPR registers should be accessed at the rate of the transmit clock or slower.*

6.1.1 TDPR Initialization

To initialize the TDPR, do the following:

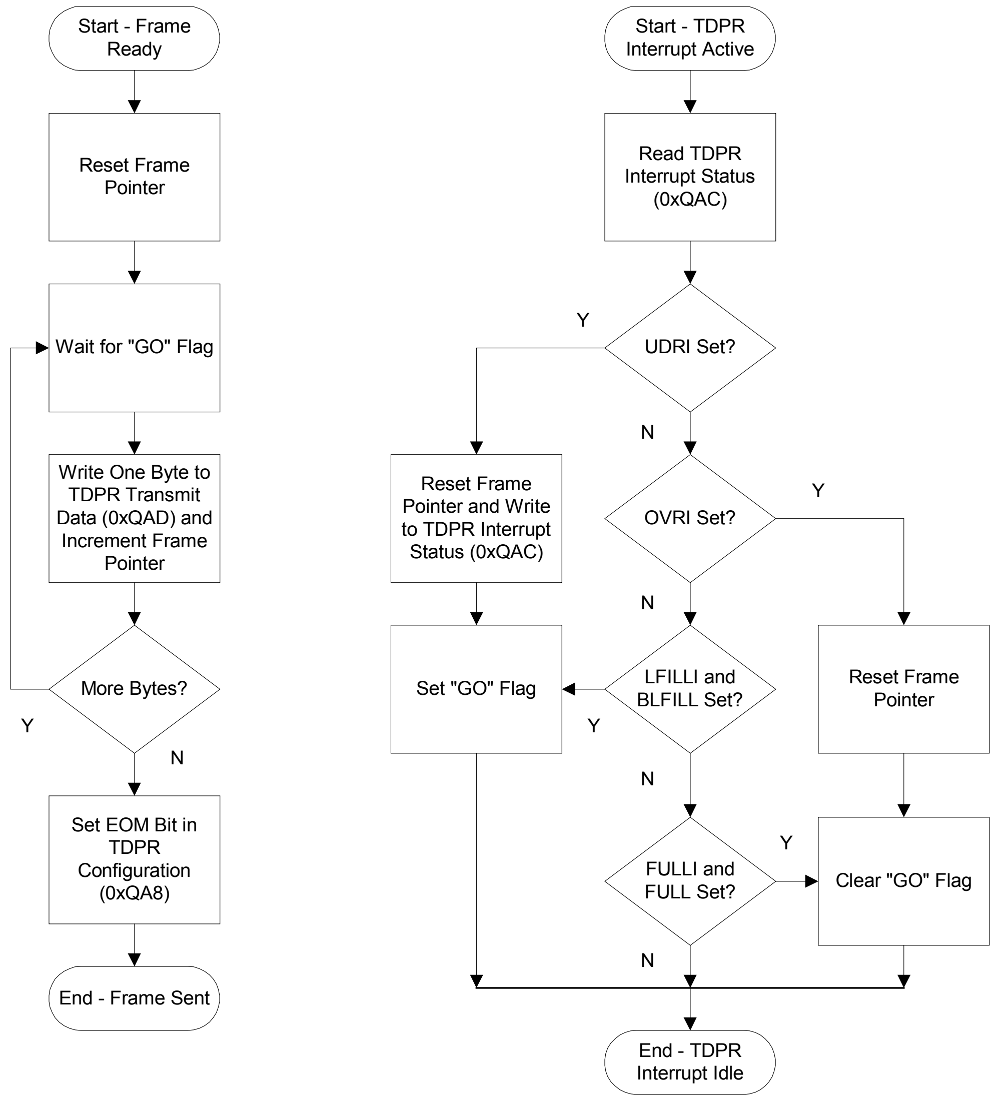
- Clear the EN bit in register 0xQA8 (TDPR Configuration) to disable the TDPR.
- Select the timeslot/channel (or select the FDL) to carry the HDLC frames by writing to register 0xQ38 (TXCI Transmit Data Link Control) and select the bits within the timeslot/channel by writing to register 0xQ39 (TXCI Transmit Data Link Bit Select).
- Set the CRC bit in register 0xQA8 register if a FCS needs to be automatically appended to the transmitted HDLC frames.
- Set the high and low water marks by writing to registers 0xQA9 and 0xQAA. After reset, these registers will default to 64 for the high water mark and 7 for the low water mark.

- Set the appropriate 'E' bits in register 0xQAB (TDPR Interrupt Enable) for the TDPR interrupts you wish to monitor.
- Set the EN bit in register 0xQA8 register to enable the TDPR.

6.1.2 Sending an HDLC Frame in Interrupt Driven Mode

The algorithm for sending an HDLC frame in interrupt driven mode and the appropriate TDPR interrupt routine are shown in Figure 4:

Figure 4 HDLC Frame Transmit Routine (left) and TDPR Interrupt Routine (right) for Interrupt Driven Mode


Start - Frame Ready
Reset Frame Pointer
Wait for "GO" Flag
Write One Byte to TDPR Transmit Data (0xQAD) and Increment Frame Pointer
More Bytes?
Y
N
Set EOM Bit in TDPR Configuration (0xQA8)
End - Frame Sent
Start - TDPR Interrupt Active
Read TDPR Interrupt Status (0xQAC)
UDRI Set?
Y
N
Reset Frame Pointer and Write to TDPR Interrupt Status (0xQAC)
OVRI Set?
Y
N
Set "GO" Flag
LFILLI and BLFILL Set?
Y
N
Reset Frame Pointer
FULLI and FULL Set?
Y
N
Clear "GO" Flag
End - TDPR Interrupt Idle

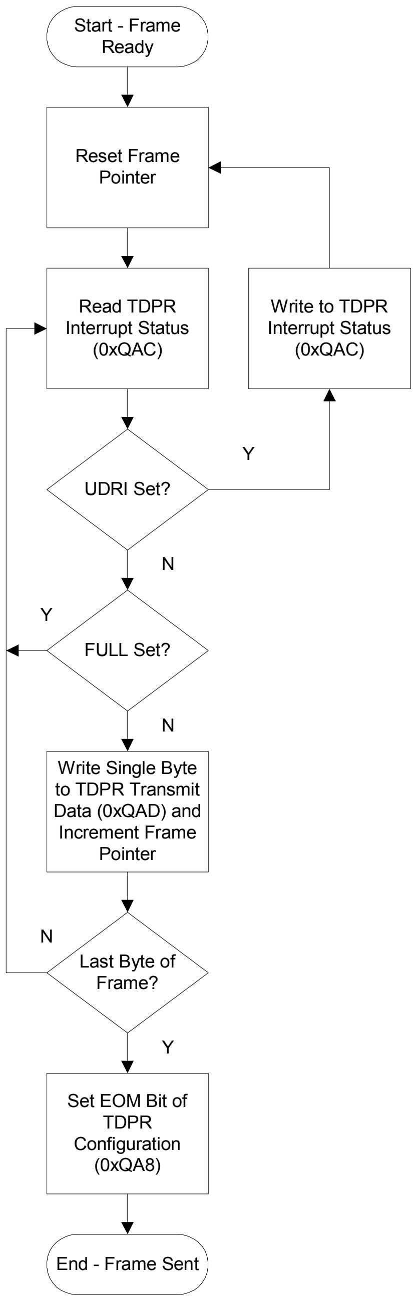
The "GO" flag in this example is used to block/unblock the task that is sending the HDLC frame, hence tuning the rate at which the FIFO is filled. This flag is set when the TDPR is initialized and is set/cleared in the interrupt routine depending on the data level in the FIFO. Under normal circumstances, the data level in the FIFO will be held between the low water mark and FIFO capacity, and LFILLI and FULLI interrupts will be generated to tune the rate at which data is written into the FIFO.

Whenever the TDPR FIFO experiences an overrun or underrun, the frame pointer of the currently sent frame is reset to point to the first byte, thus facilitating the re-transmission of the erroneous frame. In addition, when an underrun occurs, any value is written to register 0xQAC (TDPR Interrupt Status) to acknowlege the underrun and re-enable the FIFO, as required in the COMET-QUAD data sheet [].

6.1.3 Sending an HDLC Frame in Polling Mode

The routine for sending an HDLC frame in polling mode is somewhat easier to implement, but is more CPU-intensive since the HDLC transmit routine has to poll register 0xQAC in order to determine the status of the TDPR FIFO. This routine is shown in Figure 5:

Figure 5 HDLC Frame Transmit Routine for Polling Mode


Start - Frame Ready
Reset Frame Pointer
Read TDPR Interrupt Status (0xQAC)
Write to TDPR Interrupt Status (0xQAC)
UDRI Set?
Y
N
FULL Set?
Y
N
Write Single Byte to TDPR Transmit Data (0xQAD) and Increment Frame Pointer
Last Byte of Frame?
N
Y
Set EOM Bit of TDPR Configuration (0xQA8)
End - Frame Sent

The HDLC transmit routine in this example could never overflow the transmit FIFO, because it is polling the FIFO status for the FULL bit. The FULL bit denotes that (although there is room for one more byte), the FIFO should not be written to in order to avoid overruns.

In case of an underrun, the FIFO is re-enabled by writing to register 0xQAC (TDPR Interrupt Status) and the frame pointer is reset in order to facilitate re-transmission of the last HDLC frame.

6.2 Using the HDLC Receiver (RDLC) Block

The HDLC Receiver (RDLC) block receives the HDLC frames into a 128-byte deep FIFO. To avoid overruns, the FIFO has to be read at the same or higher rate than compared to the rate at which HDLC frames arrive into it.

The FIFO is accessible via the register 0xQC3 (RDLC Data) register. To control the rate at which the RDLC FIFO is read, the COMET-QUAD provides register 0xQC1 (RDLC Interrupt Control). Bits INTC[6:0] control the FIFO fill level at which the RDLC will generate an interrupt denoting that the FIFO needs to be read. In addition, RDLC could interrupt in the following cases:

- FIFO overrun (FIFO full, received data cannot be stored)
- Change of link status (an HDLC flag or HDLC abort flag encountered), and
- Packet in (a full HDLC frame was received).

In case of a FIFO overrun, the RDLC FIFO will be cleared and any data that was in it will be lost. Hence, the HDLC frame that was read at that point has to be dropped and the HDLC receive buffer reset.

In all other cases, the FIFO needs to be emptied and FIFO data has to be processed. To detect the cause of the interrupt, the user has to read the register 0xQC2 (RDLC Status). This register also contains information on the last byte read from the FIFO, and hence alignment has to be provided between the data and status register. For this purpose, whenever a change of link status is encountered, an extra dummy byte denoting the link status will be put into the FIFO. Whenever a COLS interrupt occurs, the FIFO has to be emptied to search for the flag that indicates the link status.

6.2.1 Initializing the HDLC Receiver

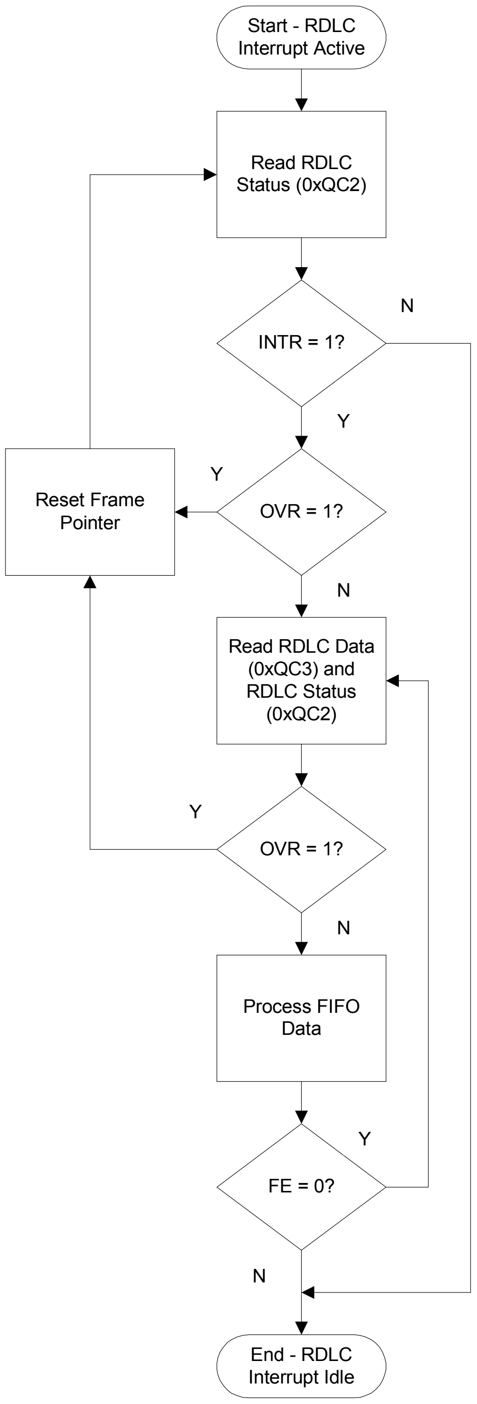
To initialize the HDLC receiver, run through the following steps:

- Clear the EN bit in register 0xQC0 (RDLC Configuration) to disable the HDLC receiver during configuration.
- Select the timeslot/channel (or select the FDL) from which to extract the HDLC frames by writing to register 0xQ38 (TXCI Transmit Data Link Control) and select the bits within the timeslot/channel by writing to register 0xQ39 (TXCI Transmit Data Link Bit Select).
- Configure the primary and secondary address match conditions by writing to registers 0xQC4 (RDLC Primary Address Match) and 0xQC5 (RDLC Secondary Address match). Set the appropriate combination of the MM and MEN bits of the register 0xQC0 (RDLC Configuration).
- Set the TR bit of the register 0xQC0 (RDLC Configuration) to reset the FIFO.
- Set the high water mark for the RDLC FIFO and enable RDLC interrupts by setting the register 0xQC1 (RDLC Interrupt Control).
- Set the EN bit in the register 0xQC0 (RDLC Configuration) to enable the HDLC receiver.

6.2.2 Processing RDLC Interrupts

To process an RDLC interrupt, follow the procedure shown in Figure 6:

Figure 6 RDLC Interrupt Routine


Start - RDLC Interrupt Active
Read RDLC Status (0xQC2)
INTR = 1?
N
Y
Reset Frame Pointer
Y
OVR = 1?
N
Read RDLC Data (0xQC3) and RDLC Status (0xQC2)
Y
OVR = 1?
N
Process FIFO Data
Y
FE = 0?
N
End - RDLC Interrupt Idle

Note that due to the fact that the HDLC frames are automatically received into the RDLC buffer, an overrun may occur at any moment, invalidating the receiving frame. This is why the OVR bit of the register 0xQC2 (RDLC Status) has to be constantly monitored in order to reset the frame buffer in case of an overrun.

In the 'Process FIFO Data' block, the user should take advantage of the PBS[2:0] bits of the register 0xQC2 (RDLC Status) to process the data read from the register 0xQC3 (RDLC Data). The interpretation of these three bits is given in the COMET-QUAD data sheet [], and the received data should be processed as follows:

- If PBS[2:0] = 000B, a non-terminating byte of the HDLC frame was received. Store the byte in the frame buffer and increment the frame buffer pointer.
- If PBS[2:0] = 001B, the received byte was a 'link activate' flag and it should be discarded. Update the link status.
- If PBS[2:0] = 010B, the received byte was a 'link abort' flag and it should be discarded. Update the link status.
- if PBS[2:0] = 1xxB, a terminating byte of the HDLC frame was received. If PBS[1:0] = 00B, the frame was received with no error. Store the byte into the frame buffer, store the frame buffer, and start a new buffer. Otherwise, use PBS[1:0] to identify the error case and discard the erroneous frame.

6.2.3 Receiving HDLC Frames in Polling Mode

To receive the HDLC frames in polling mode, implement the routine shown in Figure 6 as a timer routine. Set the timer frequency and the FIFO interrupt level (high water mark) so that the FIFO cannot be overflown between subsequent executions of the timer routine.

7 Alarms using register access

The COMET-QUAD can detect many alarm conditions and can transmit certain alarms based on the mode of operation. The following sections show the interrupt/alarm hierarchy, as well as how to insert alarms and the types of alarms the COMET-QUAD may indicate based on T1 or E1 mode.

7.1 Interrupt/Alarm Hierarchy

The receiver side of COMET-QUAD has an interrupt/alarm hierarchy shown in Figure 8 for T1 and E1 modes. The priority shown is based on order of importance for correcting alarm conditions. Interrupts/alarms shown in dotted boxes represent the same priority for the enclosed blocks. The highest priority interrupts/alarms are shown at the top and decrease in priority as the user goes down. Interrupts and alarms may propagate down so the higher level ones should be processed before processing any below. If an interrupt/alarm condition arises in the COMET-QUAD receiver side, the hierarchy should generally be followed. The diagram shows blocks for all of the interrupt source register bits. Once the highest level block is found (that is active), the diagram shows arrows to the interrupt bits that caused the interrupt/alarm to occur. The user should look in the COMET-QUAD datasheet for details about the particular interrupt.

Figure 8 generally shows the order to follow. In some cases the interrupt/alarm should be ignored. See Table 14 for transmit cases and see Table 15 for receiver cases.

Table 14: Cases to Ignore Transmitter Interrupts / Alarms

TSB Group of "I bits" to ignore	Reason
TRAN	COMET-QUAD mode is T1
APRM	COMET-QUAD mode is E1
TX-ELST	Datacom applications

Table 15: Cases to Ignore Receiver Interrupt / Alarms

TSB Group of "I bits" to ignore	Reason
RX-ELST	Datacom applications
SIGX	Datacom applications

The transmit side of COMET-QUAD does not have an interrupt/alarm hierarchy. Checking each of the blocks can identify the cause of an interrupt/alarm. Figure 7 and Figure 8 shows the interrupt bits for each interrupt/alarm case.

The next two sections show the bits that must be set to insert alarms into the transmit stream and the backplane and also show why an alarm may occur.

Figure 7 Transmitter Alarms and Interrupts

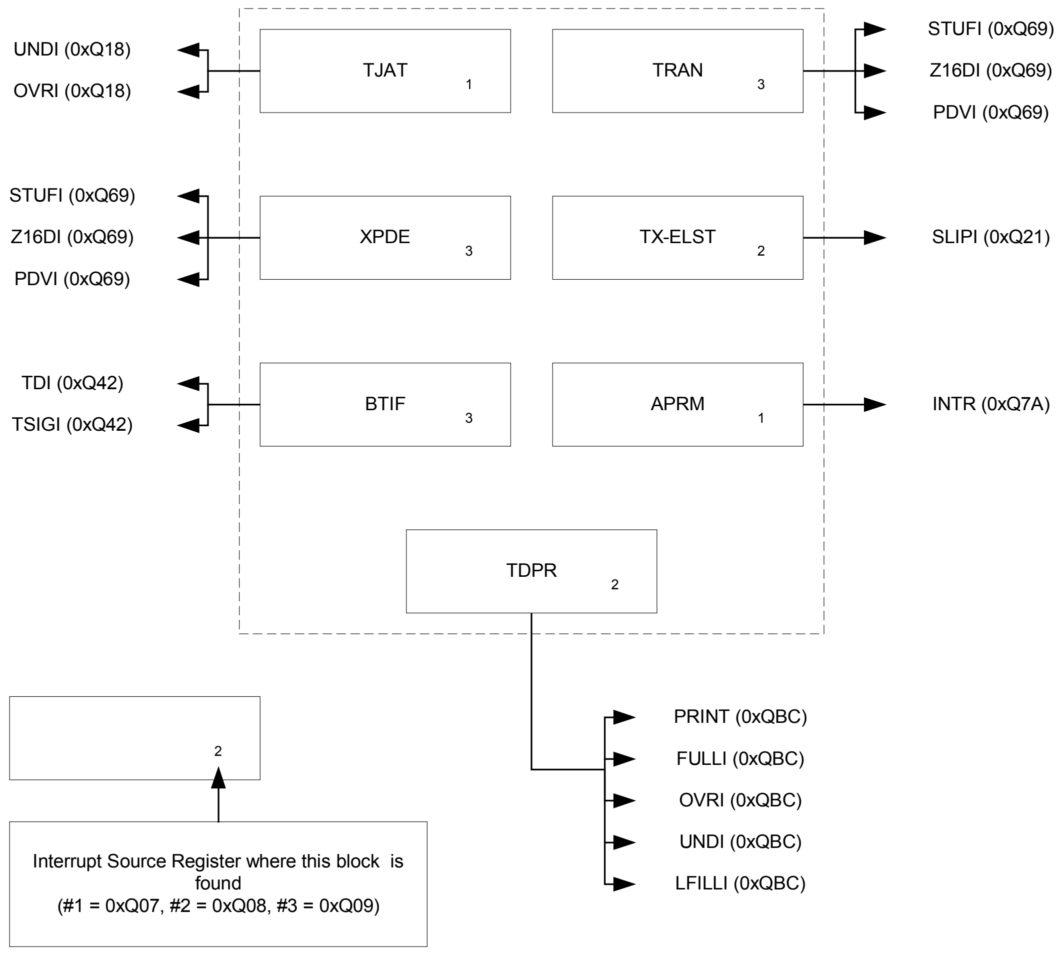
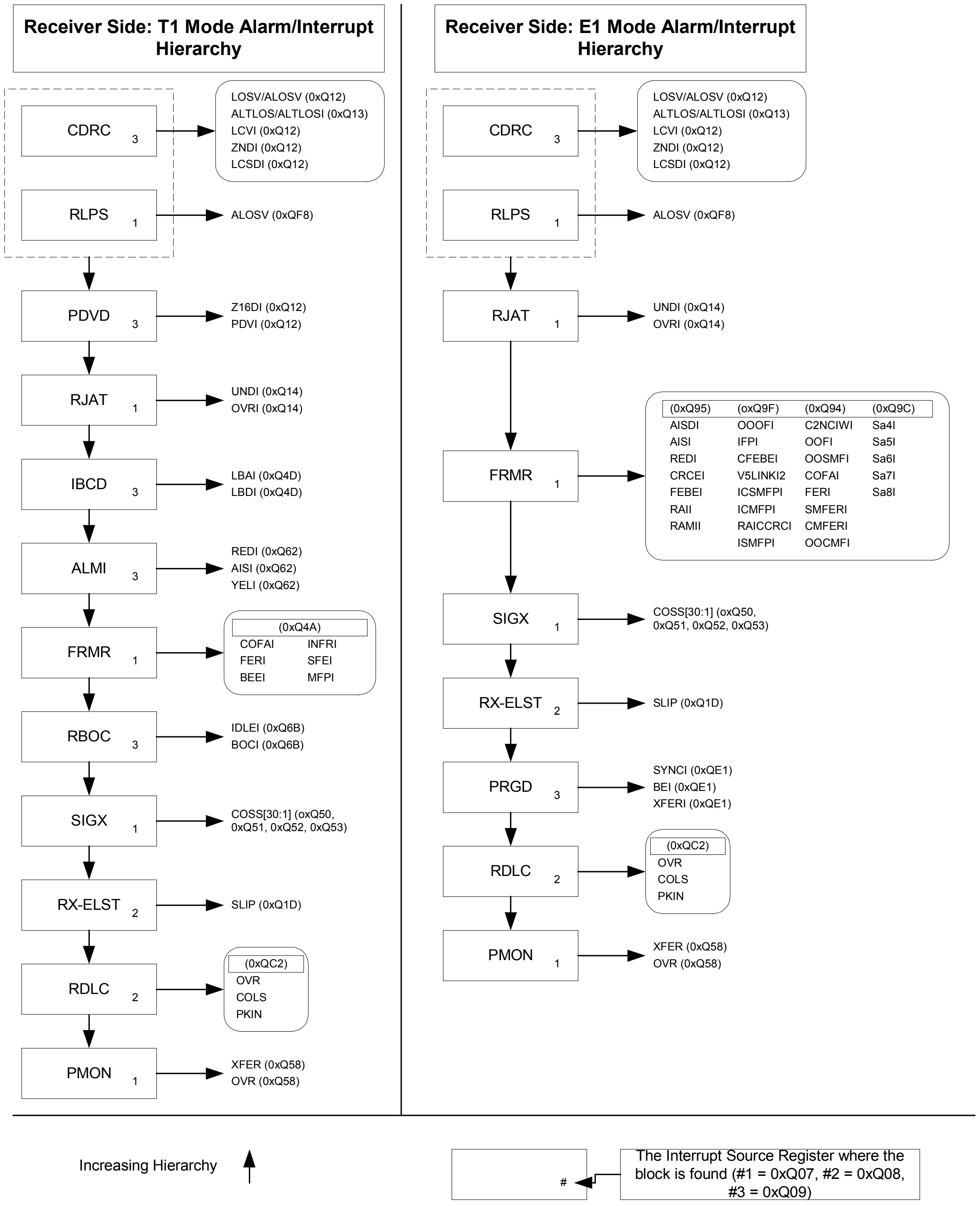


Figure 8 Receiver alarms and Interrupts for T1 and E1



7.2 T1 Mode

The sections below explain how to insert alarms into the Transmit stream and the backplane as well as why or how an alarm may occur.

7.2.1 T1 Alarm Insertion

The following tables describe how the COMET-QUAD can be configured to insert T1 alarms in the transmit data stream. Each alarm may be caused by several different conditions.

The Alarm column represents the type of alarm. The "Condition" column identifies bits that have to be set to configure the alarm. The "Register" column simply states where the bit is located.

Table 16: Transmit Alarm Insertion (T1 Mode)

Alarm	Condition	Register	Comments
YELLOW	XYEL	0xQ55	
AIS	XAIS	0xQ55	
	TAISEN=1	0xQ04	AIS signal on TXTIP & TXRING pins
	TAIS & AISE	0xQF1	TAIS bit (s/w AIS) is ORed with the external AIS input (h/w AIS) to enable AIS insertion.

Alarms are transparently sent to the Backplane Receive Interface. The AIS alarm may optionally be inserted based on the Condition below.

Table 17: Drop Alarm Insertion (T1 Mode)

Alarm	Condition	Register	Comments
AIS	AUTOAIS	0xQ03	AIS is inserted in the receive path and signaling frozen
	RAIS & (AUTOAIS (0x03) = 0)	0xQ0A	In E1/T1 mode BRPCM pin is forced to all 1's. In H-MVIP mode, MVBRD stream for the quadrant is forced to all 1's.

7.2.2 T1 Receiver Alarms

Table 18 below shows Alarms related to the Receive path of COMET-QUAD. The columns show the alarms, the register bits that can affect or indicate the alarm, the register the bits are located, possible causes for the alarm and general comments.

Table 18: Receiver Alarms (T1 Mode)

Alarm	Condition	Register	Possible Cause(s)	Comments
ALOS	DET_PER[7:0] &	0xQFA	Threshold is set too low; the link is physically down	ALOS Detection Threshold has been reached and ALOSI set.
	ALOSI	0xQF8		
LOS	LOS[1:0] &	0xQ10	An incorrect threshold is set; the link is physically down	LOS Threshold has been reached
	LOSV	0xQ12		
PDV	PDV	0xQ65	16 consecutive zeros are detected; an incorrect line code is being transmitted	Pulse Density Violation
RED	REDD	0xQ63	An incorrect framing format is being received	An OOF condition has been present for 2.55 sec (± 40 ms)
AIS	AIS	0xQ62	Alarm Detected	The carry failure alarm had a state change
	AISD	0xQ63		AIS signal was present during the last 60 ms interval
YEL	YEL	0xQ62	Alarm Detected	The carry failure alarm had a state change
	YELD	0xQ63		Yellow signal was present during the last 40 ms interval
Out of Frame	INFR=0 (T1 Mode)	0xQ4A	Clock may not be synchronized	

7.3 E1 Mode

The sections below explain how to insert alarms into the Transmit stream and the backplane as well as why or how an alarm may occur.

7.3.1 E1 Alarm Insertion

The following tables describe how the COMET-QUAD can be configured to insert E1 alarms in the transmit data stream. Each alarm may be caused by several different conditions.

The Alarm column represents the type of alarm. The "Condition" column identifies bits that have to be set to configure the alarm. The "Register" column simply states where the bit is located.

Table 19: Transmit Alarm Insertion (E1 mode)

Alarm	Condition	Register	Comments
Signaling Multiframe AIS	YBIT	0xQ81	
Timeslot 16 AIS	TS16AIS	0xQ81	
AIS	AIS	0xQ81	
	TAISEN=1	0xQ04	AIS signal on TXTIP & TXRING pins
RAIS	RAI	0xQ81	

Alarms are transparently sent to the Backplane Receive Interface. The AIS alarm may optionally inserted be inserted based on the Condition below.

Table 20: Drop Alarm Insertion (E1 mode)

Alarm	Condition	Register	Comments
AIS	AUTOAIS	0xQ03	AIS is inserted in the receive path and signaling frozen
	RAIS & (AUTOAIS (0x03) = 0)	0xQ0A	In T1/E1 mode, BRPCM pin is forced to all 1's. In H-MVIP mode, MVBRD stream for this quadrant is forced to all 1's.
	OOSMFAIS=1	0xQ00	An OOSMF indication from the E1-FRMR will send all 1's onto the BRSIG pin. In H-MVIP mode, an OOSMF indication from the E1-FRMR will send all 1's to the CASBRD pin.

7.3.2 E1 Receiver Alarms

Table 21 below shows Alarms related to the Receiver part of COMET-QUAD. The columns show the alarms, the register bits that can affect or indicate the alarm, the register location of the bits, possible causes for the alarm and general comments.

Table 21: Receiver Alarms (E1 mode)

Alarm	Condition	Register	Possible Cause(s)	Comments
ALOS	DET_PER[7:0] &	0xQFA	Threshold is set too low; the link is physically down	ALOS Detection Threshold has been reached and ALOSI set.
	ALOSI	0xQF8		
LOS	LOS[1:0] &	0xQ10	The incorrect threshold is set; the link is physically down	LOS Threshold has been reached
	LOSV	0xQ12		
PDV	PDV	0xQ65	16 consecutive zeros are detected; an incorrect line code is being transmitted	Pulse Density Violation

Alarm	**Condition**	**Regist er**	**Possible Cause(s)**	**Comments**
RED	RED	0xQ97	An incorrect framing format is being received	An OOF condition has been present for 100 ms
AIS	AISC	0xQ91	loss of frame	Depending on how the AISC bit is set, this alarm can cause the AISD bit (0x97) to be set.
	AIS	0xQ97		An out of frame all-ones condition has persisted for 100 ms
Out of Frame	OOFI (E1 Mode)	0xQ94	Clock may not be synchronized. Bits in register 0x90 may be incorrectly set. i.e. CRCEN could be disabled; CASDIS may be incorrectly set.	
loss of signaling multiframe	OOSMFV	0xQ96		Bit represents the loss of signaling multiframe
loss of CRC multiframe	OOCMFV:	0xQ96		Bit represents the loss of CRC multiframe

It should be noted that if an ALOS or LOS alarm occurs at the receive interface, the RSYNC pin represents either the ALOS or LOS alarm based on how the RSYNC_ALOSB bit in the Global_Configuration(0xQ00) register is set. When COMET-QUAD is in a loss of signal state, the RSYNC output is derived from XCLK (unless RSYNCH_MEM is set in the Receive_Options(0xQ02) register) rather than the recovered receiver clock.

8 Programming the Pattern Generator/Detector using register access

The Pattern Generator/Detector (PRDG) block is used to generate and detect pseudo-random bit patterns. The COMET-QUAD allows to select from three possible pseudo-random bit sequences. The selection of sequences is defined by the PATSEL[1:0] bits of the register 0xQE2 (PRBS Pattern Select), as shown in Table 22:

Table 22: PRBS Pattern Select

PATSEL[1:0]	Pattern
00	2^{15}-1
01	2^{20}-1
10	2^{11}-1
11	Reserved

Other PRBS options are selected in the register 0xQE0 (PRBS Generator/Checker Control).

The PRBS can be configured to accumulate bit error counts in the received pseudo-random pattern. The bit error count is accumulated as a 24-bit integer in registers 0xQE4-0xQE6 (PRBS Error Count #1-3). Bit errors are not accumulated while the pattern detector is out of synchronization. The synchronization status of the PRBS block is determined by the value of the SYNCV bit of the register 0xQE1 (PRBS Checker Interrupt Enable/Status).

To configure the PRBS block to accumulate error counts, write to register 0xQE1. Refer to the COMET-QUAD datasheet [] for the values that have to be written to this register.

9 Performance Monitoring using register access

COMET-QUAD provides counters for performance monitoring and the ability to generate automatic performance report messages in T1 mode.

9.1 Performance Monitoring Counters

The COMET-QUAD's Performance Monitoring Counters (PMON) block accumulates the following types of errors:

- CRC error events
- Frame Synchronization bit error events
- Line Code Violation events
- Out Of Frame events

Optionally, Change of Frame Alignment (COFA) events may be accumulated, with saturating counters over consecutive intervals.

When the transfer clock signal is applied, the PMON transfers the counter values into holding registers and resets the counters to begin accumulating events for the interval. The counters are reset in such a manner that error events occurring during the reset are not missed. If the holding registers are not read between successive transfer clocks, an OVERRUN register bit is asserted.

Generation of the transfer clock within the COMET-QUAD chip is performed by writing to any counter register location or by writing to register 0xQ0D (Revision/Chip ID/Quadrant PMON Update). The holding register addresses are contiguous to facilitate faster polling operations.

Table 23 and

Table 24 show all of COMET-QUAD's performance counters for each of the counter types.

Table 23 : PMON registers for T1 mode

Counter Type	Register	Description
Framing Bit Error	0xQ59	Number of framing bit error events that occurred during the last accumulation interval
Out of Frame (OOF) Errors	0xQ5A-0xQ5B	Number of OOF events that occurred during the last accumulation interval
Bit Errors	0xQ5C-0xQ5D	Number of bit error events that occurred during the previous accumulation interval
Line Code Violation (LCV)	0xQ5E-0xQ5F	Number of bipolar violations or excessive zeros that occurred during the previous accumulation interval

Table 24: PMON Registers for E1 mode

Counter Type	Register	Description
Framing Bit Error	0xQ59	Number of framing bit error events that occurred during the last accumulation interval
Far End Block Errors	0xQ5A-0xQ5B	Number of far end block error events that occurred during the previous accumulation interval
CRC Errors	0xQ5C-0xQ5D	Number of CRC error events that occurred during the previous accumulation interval
Line Code Violation (LCV)	0xQ5E-0xQ5F	Number of bipolar violations for AMI-coded signals and HDB3-coded signals that occurred during the previous accumulation interval

9.2 Automatic Performance Monitoring

The COMET-QUAD may automatically update performance reports on a per second basis if the AUTOUPDATE bit in register 0xQ78 (T1 APRM Configuration/Control) is set to logic 1.

Table 24 below shows the message structure and contents of the performance report. Table 26 and Table 27 display bit explanations.

Table 25: Performance Report Message Structure and Contents

Octet No.	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1
1	FLAG							
2	SAPI						C/R	EA
3	TEI							EA
4	CONTROL							
5	G3	LV	G4	U1	U2	G5	SL	G6
6	FE	SE	LB	G1	R	G2	Nm	NI
7	G3	LV	G4	U1	U2	G5	SL	G6
8	FE	SE	LB	G1	R	G2	Nm	NI
9	G3	LV	G4	U1	U2	G5	SL	G6
10	FE	SE	LB	G1	R	G2	Nm	NI
11	G3	LV	G4	U1	U2	G5	SL	G6
12	FE	SE	LB	G1	R	G2	Nm	NI
13	FCS							
14	FCS							
15	FLAG							

Notes:

The order of transmission of the bits is LSB (Bit 1) to MSB (Bit 8).

Table 26: Performance Report Message Structure Notes

Octet No.	Octet Contents	Interpretation
1	01111110	Opening LAPD Flag
2	00111000 00111010	From CI: SAPI=14, C/R=0, EA=0 From carrier: SAPI=14,C/R=1,EA=0
3	00000001	TEI=0,EA=1
4	00000011	Unacknowledged Frame
5,6	Variable	Data for latest second (T')
7,8	Variable	Data for Previous Second(T'-1)
9,10	Variable	Data for earlier Second(T'-2)
11,12	Variable	Data for earlier Second(T'-3)
13,14	Variable	CRC16 Frame Check Sequence
15	01111110	Closing LAPD flag

Table 27: Performance Report Message Contents

Bit Value	Interpretation
G1=1	CRC ERROR EVENT =1
G2=1	1<CRC ERROR EVENT ≤5
G3=1	5<CRC ERROR EVENT ≤10
G4=1	10<CRC ERROR EVENT ≤100
G5=1	100<CRC ERROR EVENT ≤319
G6=1	CRC ERROR EVENT ≥ 320
SE=1	Severely Errored Framing Event ≥ 1(FE shall =0)
FE=1	Frame Synchronization Bit Error Event ≥1 (SE shall=0)
LV=1	Line code violation event ≥1
SL=1	Slip Event ≥ 1
LB=1	Payload Loopback Activated
U1,U2=0	Under Study For Synchronization
R=0	Reserved (Default Value =0)
NmNI=00,01,10,11	One second Report Modulo 4 Counter

10 Diagnostics using register access

The COMET-QUAD is capable of performing four types of diagnostic tests for each of its quadrants:

- Line Loopback
- Payload Loopback
- Diagnostic Digital Loopback
- Per-Channel Loopback

Refer to the COMET-QUAD data sheet [] for a detailed description of each of these loopback modes.

The first three loopback modes are set by writing to register 0xQ0A (Master Diagnostics):

- Setting LINELB bit will switch on the Line Loopback mode.
- Setting PAYLB bit will switch on the Payload Loopback mode.
- Setting DDLB bit will switch on the Diagnostic Digital Loopback mode.

The Per-Channel Loopback mode is switch on and off by writing to TPSC indirect registers 0x20-0x3F (PCM Data Control Byte). Setting the LOOP bit in these indirect registers will switch on loopback mode for the corresponding timeslot/channel. Refer to section 3.2 for information on how to write to TPSC indirect registers.

11 Sample Configurations using register access

The following sections are provided as a quick reference for the register write sequence for commonly used configurations. The sequences do not show interrupt enable, per channel control and HDLC sequences. The reader should refer to other sections in this document for in depth discussion of these topics.

11.1 T1 Mode

Here is a sample configuration which configures one port (Port #1) for T1 ESF mode, with B8ZS line encoding. The BRIF (Receive Backplane) is configured for timing master while the BTIF (Transmit Backplane) is configured for timing slave. Steps not explicitly shown in this example are: programming the RLPS RAM Equalizer table, programming the XLPG Transmit waveform, and running the RLPS Optimization routine.

Table 28: Sample Configuration for T1 ESF mode, with B8ZS line encoding

Sample Configuration for T1 ESF mode, with B8ZS line encoding			
#	**Register**	**Value**	**Comments**
1	0x000	0x00	Select T1 operation (applies to all four ports)
2	0xQD6	0x07	Select crystal, XCLK = 2.048MHz
3	0xQ54	0x30	Transmit; Select B8ZS line encoding and ESF framing
4	0xQ10	0x00	Receive; Select B8ZS line Encoding
5	0xQ48	0x30	Receive, Select ESF framing for T1-FRMR block
6	0xQ60	0x10	Receive; Select ESF framing for T1-ALMI block
7	0xQ50	0x04	Receive, select ESF framing for SIGX block
8	0xQ06	0x01	Configure Transmit Timing; Select the first row of the "Transmit Timing" table (ie. Timing sourced from backplane BTCLK, TJAT enabled)
9	0xQ40	0x38	Transmit Backplane (BTIF): configure for clock slave, full framed, 1.544 MHz rate
10	0xQ41	0x01	Transmit Backplane (BTIF): Frame-pulse slave
11	0xQ02	0x20	Configure Receive Timing; enable RJAT and disable RX-ELST (since we will be setting the Receive Backplane to timing master)
12	0xQ30	0x00	Receive Backplane (BRIF); clock master mode, full framed, 1.544MHz rate
13	0xQ31	0x00	Receive Backplane (BRIF): Frame-pulse master
14	0xQ20	0x00	Configure TX-ELST for T1 mode
15	0xQ1C	0x00	Configure RX-ELST for T1 mode
16	0xQ19	0x2F	TJAT, set N1 value

Sample Configuration for T1 ESF mode, with B8ZS line encoding			
#	**Register**	**Value**	**Comments**
17	0xQ1A	0x2F	TJAT, set N2 value
18	0xQ1B	0x01	TJAT, set LIMIT bit
19	0xQ15	0x2F	RJAT, set N1 value
20	0xQ16	0x2F	RJAT, set N2 value
21	0xQ17	0x01	RJAT, set LIMIT bit
22	0xQDC	0x2C	Set RLPS Voltage Reference value
23			<Program RAM Equalizer table; this requires 256 Indirect Writes>
24	0xQFF	0x0B	Enable the RLPS Equalizer
25			<Program Transmit Pulse Waveform; this requires 120 Indirect Writes>
26	0xQF0	0x8C	Program the SCALE[4:0] bits for the XLPG. Leave the XLPG transmitter in tri-state.
27	0xQF6	0x06	XLPG Initialization Procedure
28	0xQF4	0x00	XLPG Initialization Procedure
29	0xQF5	0x00	XLPG Initialization Procedure
30	0xQF6	0x00	XLPG Initialization Procedure
31			<RLPS Optimization routine; this consists of 29 writes>
32	0xQF0	0x8C	Finally, release the tri-states on the XLPG transmitter by clearing the HIGHZ bit.

11.2 E1 Mode

Here is a sample configuration which configures one port (Port #1) for E1 CRC-Multiframe, with HDB3 line encoding. The transmit waveform is E1 120ohm. The BRIF (Receive Backplane) is configured for timing master while the BTIF (Transmit Backplane) is configured for timing slave. Steps not explicitly shown in this example are: programming the RLPS RAM Equalizer table, programming the XLPG Transmit waveform, and running the RLPS Optimization routine.

Table 29: Sample Configuration for E1 CRC-MF, with HDB3 line encoding

Sample Configuration for E1 CRC-MF, with HDB3 line encoding			
#	**Register**	**Value**	**Comments**
1	0x000	0x01	Select E1 operation (*applies to all four ports)
2	0xQD6	0x00	Select crystal, XCLK = 2.048MHz
3	0xQ80	0x10	Transmit; Select HDB3 line encoding and E1 CRC-multiframe
4	0xQ10	0x00	Receive; Select HDB3 line Encoding
5	0xQ90	0xC2	Receive, Select CRC-Multiframing for E1-FRMR block
6	0xQ06	0x01	Configure Transmit Timing; Select the first row of the "Transmit Timing" table (ie. Timing sourced from backplane BTCLK, TJAT enabled)
7	0xQ40	0x39	Transmit Backplane (BTIF): configure for clock slave, full framed, 2.048 MHz rate
8	0xQ41	0x01	Transmit Backplane (BTIF): Frame-pulse slave
9	0xQ02	0x20	Configure Receive Timing; enable RJAT and disable RX-ELST (since we will be setting the Receive Backplane to timing master)
10	0xQ30	0x01	Receive Backplane (BRIF); clock master mode, full framed, 2.048 MHz rate
11	0xQ31	0x00	Receive Backplane (BRIF): Frame-pulse master
12	0xQ20	0x03	Configure TX-ELST for E1 mode
13	0xQ1C	0x03	Configure RX-ELST for E1 mode
14	0xQ19	0xFF	TJAT, set N1 value
15	0xQ1A	0xFF	TJAT, set N2 value
16	0xQ1B	0x01	TJAT, set LIMIT bit
17	0xQ15	0xFF	RJAT, set N1 value
18	0xQ16	0xFF	RJAT, set N2 value
19	0xQ17	0x01	RJAT, set LIMIT bit
20	0xQDC	0x3D	Set RLPS Voltage Reference value

Sample Configuration for E1 CRC-MF, with HDB3 line encoding			
#	Register	Value	Comments
21			<Program RAM Equalizer table; this requires 256 Indirect Writes>
22	0xQFF	0x0B	Enable the RLPS Equalizer
23			<Program Transmit Pulse Waveform; this requires 120 Indirect Writes>
24	0xQF0	0x8C	Program the SCALE[4:0] bits for the XLPG. Leave the XLPG transmitter in tri-state.
25	0xQF6	0x06	XLPG Initialization Procedure
26	0xQF4	0x00	XLPG Initialization Procedure
27	0xQF5	0x00	XLPG Initialization Procedure
28	0xQF6	0x00	XLPG Initialization Procedure
29			<RLPS Optimization routine; this consists of 29 writes> (*applies to all four ports)
30	0xQF0	0x8C	Finally, release the tri-states on the XLPG transmitter by clearing the HIGHZ bit.

11.3 H-MVIP Mode

Here is a sample configuration for E1 with an H-MVIP backplane interface. Port #1 is configured for CRC-Multiframe and HDB3 line-encoding. Although only one port is configured here, the H-MVIP interface here applies to all four ports. The other three ports can be programmed identically if desired.

Table 30: Sample Configuration for E1 with an H-MVIP backplane interface

Sample Configuration for E1 CRC-MF, with HDB3 line encoding			
#	Register	Value	Comments
1	0x000	0x01	Select E1 operation (*applies to all four ports)
2	0xQD6	0x00	Select crystal, XCLK = 2.048MHz
3	0xQ80	0x10	Transmit; Select HDB3 line encoding and E1 CRC-multiframe
4	0xQ10	0x00	Receive; Select HDB3 line Encoding
5	0xQ90	0xC2	Receive, Select CRC-Multiframing for E1-FRMR block

Sample Configuration for E1 CRC-MF, with HDB3 line encoding			
#	Register	Value	Comments
6	0xQ06	0x01	Configure Transmit Timing; Select the first row of the "Transmit Timing" table (ie. Timing sourced from backplane, TJAT enabled)
7	0xQ40	0x3F	Transmit Backplane (BTIF); configure for H-MVIP mode (clock slave), full framed
8	0xQ41	0x01	Transmit Backplane (BTIF); Frame-pulse slave
	0x043	0x00	Transmit Backplane; timeslot offset =0 for port #1
9	0x143	0x01	Transmit Backplane; timeslot offset =1 for port #2
10	0x243	0x02	Transmit Backplane; timeslot offset =2 for port #3
11	0x343	0x03	Transmit Backplane; timeslot offset =3 for port #4
12	0xQ02	0x00	Configure Receive Timing; enable RJAT and RX-ELST (RX-ELST necessary because H-MVIP Receive backplane must be timing slave)
13	0xQ30	0x2F	Receive Backplane (BRIF); configure for H-MVIP mode (clock slave), full framed
14	0xQ31	0x20	Receive Backplane (BRIF); Frame-pulse slave
15	0xQ32	0x01	Receive Backplane (BRIF); Bit 0 must be set to '1' in H-MVIP mode
16	0x033	0x00	Receive Backplane; timeslot offset =0 for port #1
17	0x133	0x01	Receive Backplane; timeslot offset =1 for port #2
18	0x233	0x02	Receive Backplane; timeslot offset =2 for port #3
19	0x333	0x03	Receive Backplane; timeslot offset =3 for port #4
20	0xQ20	0x03	Configure TX-ELST for E1 mode
21	0xQ1C	0x03	Configure RX-ELST for E1 mode
22	0xQB0	0x03	Configure RX-ELST CCS for E1 mode
23	0xQB4	0x01	Configure TX-ELST CCS for E1 mode
24	0x0B8	0x01	Configure receive stream for H-MVIP mode, no signaling extraction
25	0x0B9	0x01	Configure transmit stream for H-MVIP mode, no signaling insertion
26	0xQ19	0xFF	TJAT, set N1 value
27	0xQ1A	0xFF	TJAT, set N2 value
28	0xQ1B	0x01	TJAT, set LIMIT bit
29	0xQ15	0xFF	RJAT, set N1 value
30	0xQ16	0xFF	RJAT, set N2 value
31	0xQ17	0x01	RJAT, set LIMIT bit
32	0xQDC	0x3D	Set RLPS Voltage Reference value

Sample Configuration for E1 CRC-MF, with HDB3 line encoding			
#	**Register**	**Value**	**Comments**
33			<Program RAM Equalizer table; this requires 256 Indirect Writes>
34	0xQFF	0x0B	Enable the RLPS Equalizer
35			<Program Transmit Pulse Waveform; this requires 120 Indirect Writes>
36	0xQF0	0x8C	Program the SCALE[4:0] bits for the XLPG. Leave the XLPG transmitter in tri-state.
37	0xQF6	0x06	XLPG Initialization Procedure
38	0xQF4	0x00	XLPG Initialization Procedure
39	0xQF5	0x00	XLPG Initialization Procedure
40	0xQF6	0x00	XLPG Initialization Procedure
41			<RLPS Optimization routine; this consists of 29 writes> (*applies to all four ports)
42	0xQF0	0x8C	Finally, release the tri-states on the XLPG transmitter by clearing the HIGHZ bit.

12 Device Driver Description

When using this Programmer's Guide to configure the COMET/COMET-QUAD it is assumed that the device has been reset, the device driver is in the READY state and that you have a device driver handle to the device.

To put the device in the READY state call:

```
cometqModuleOpen

cometqModuleStart

cometqAdd

cometqInit

cometqActivate
```

A handle to the device will be returned when cometqAdd is called. This handle is needed for all subsequent driver calls.

See the PMC-2001401 – COMET and COMET-Quad Device Driver Manual for a more detailed description of the device driver.

For the listing of an example application using the driver, see section 17.

The following convention will be used in the document to describe the driver functions:

```
/*** Variable Declaration ***/

varType1 parameter1;

varType2 parameter2;

:

varTypeN parameterN;

/*** Variable Initialization ***/

parameter1 = value1;

parameter2 = value2;

:

parameterN = valueN;

/*** Function Call ***/
```

```
cometqFunction (deviceHandle, parameter1,
parameter2,…,parameterN);
```

Comments will be shown using the following format:

```
/* comments */
```

13 Programming the COMET-QUAD Using the driver

The following sections describing how to program the Comet-quad, using the software driver, are written to mirror the above register sections as closely as possible.

```
/*** Global Variable Declaration ***/

sCMQ_HNDL deviceHandle;
```

13.1 Resetting the COMET-QUAD using the driver

```
/*** Function Call ***/

cometqReset (deviceHandle);
```

After a software reset, all COMET-QUAD quadrants will be in *Idle* state.

Note: A software reset should be applied to a COMET-QUAD device after a power-up and in cases when the framing mode needs to be changed from T1 to E1 and vice versa.

13.2 Configuring the COMET-QUAD using the driver

Normally the first order of business, when using the driver, is to define at least one Initialization Profile. This will take care of most of the configuration settings but for completeness the following sections are included to show how to use the driver to achieve the same results as covered in the register access sections. Refer to the section Sample configuration with the driver for an example of how to set up the initialization profiles and how to use the driver in your software application.

13.2.1 Select T1 or E1 Operation using the driver (Common for all four ports)

```
/*** Variable declaration ***/

UINT2 operMode;

/*** Variable Initialization ***/

operMode = CMQ_MODE_E1; /* or CMQ_MODE_T1 */

/*** Function Call ***/

cometqSetOperatingMode(deviceHandle, operMode);
```

13.2.2 Select the Crystal Clock using the driver (Common for all four ports)

Select the frequency being supplied for crystal clock (XCLK) and the desired transmit line-rate frequency. The transmit line-rate frequency is 1.544 MHz for T1 and 2.048 MHz for E1.

```
/*** Variable declaration ***/

UINT2 synthTxFreq;

/*** Variable Initialization ***/

synthTxFreq = CMQ_XCLK_2048_TXCLK_2048;     /* OR
                                 CMQ_XCLK_1544_TXCLK_1544,
                                 CMQ_XCLK_2048_TXCLK_1544 */

/*** Function Call ***/

cometqLineClkSvcCfg(deviceHandle, synthTxFreq);
```

Note that the setting here applies to all four ports.

13.2.3 Select the Line Decoding/Encoding using the driver

```
/*** Tx line encoder configuration ***/

UINT2 chan;

UINT2 encScheme;

/*** Variable Initialization ***/

chan = 0;  /* framer number 0,1,2,3 */

encScheme = CMQ_LINE_CODE_AMI; /* or CMQ_LINE_CODE_HDB3_E1,
                                  CMQ_LINE_CODE_B8ZS_T1 */

/*** Function Call ***/

cometqLineTxEncodeCfg(deviceHandle, chan, encScheme);


/*** Rx line decoder configuration ***/

UINT2 chan;

UINT2 encScheme;

UINT1 incXSZeros;

UINT1 E1_0162En;

/*** Variable Initialization ***/

chan = 0;  /* framer number 0,1,2,3 */
```

```
encScheme = CMQ_LINE_CODE_AMI; /* or CMQ_LINE_CODE_HDB3_E1,
                      CMQ_LINE_CODE_B8ZS_T1 */

incXSZeros = 0;       /* 1 - if you want to include excess zero
                      violations as bipolar violations */

E1_0162En = 0;        /* 1 – to enable 0.162 bipolar violation
                      definition (E1 only) */

/*** Function Call ***/

cometqLineRxEncodeCfg(deviceHandle, chan, encScheme, incXSZeros,
E1_0162En);
```

13.2.4 Select the Framing Format using the driver

```
/*** T1 Tx Mode Framer ***/

UINT2 chan;

sCMQ_CFG_T1TX_FRM frmCfg;

/*** Variable Initialization ***/

chan = 0;             /* framer number 0,1,2,3 */

frmCfg.frmMode = CMQ_FRM_MODE_T1_SF;  /* or CMQ_FRM_MODE_T1_DM,
                           CMQ_FRM_MODE_T1_SLC96,
                           CMQ_FRM_MODE_T1_DM_FDL,
                           CMQ_FRM_MODE_T1_ESF,
                           CMQ_FRM_MODE_T1_SF_JPN_ALARM,
                           CMQ_FRM_MODE_T1_DM_JPN_ALARM,
                           CMQ_FRM_MODE_T1_SLC96_JPN_ALARM,
                           CMQ_FRM_MODE_T1_DM_FDL_JPN_ALARM,
                           CMQ_FRM_MODE_T1_JT_G704,
                           CMQ_FRM_MODE_T1_UNFRAMED */

frmCfg.zSupFormat = CMQ_T1_ZSUP_NONE; /* or CMQ_T1_ZSUP_GTE,
                                CMQ_T1_ZSUP_DDS,
                                CMQ_T1_ZSUP_BELL */
```

```
frmCfg.SFSigAlignerEn = 1;          /* Enable the signaling aligner
                                    to ensure that signaling
                                    alignment between superframes on
                                    the backplane and the transmit
                                    framer is maintained. Note that
                                    when using this option, the ESF
                                    alignment option in the transmit
                                    backplane frame pulse
                                    configuration must be off.  */

/*** Function Call ***/

cometqT1TxFramerCfg(deviceHandle, chan, &frmCfg);



/*** T1 Rx Mode Framer ***/

UINT2 chan;

sCMQ_CFG_T1RX_FRM frmCfg;

/*** Variable Initialization ***/

chan = 0;  /* framer number 0,1,2,3 */

frmCfg.frmMode = CMQ_FRM_MODE_T1_SF;  /* or CMQ_FRM_MODE_T1_DM,
                                  CMQ_FRM_MODE_T1_SLC96,
                                  CMQ_FRM_MODE_T1_DM_FDL,
                                  CMQ_FRM_MODE_T1_ESF,
                                  CMQ_FRM_MODE_T1_SF_JPN_ALARM,
                                  CMQ_FRM_MODE_T1_DM_JPN_ALARM,
                                  CMQ_FRM_MODE_T1_SLC96_JPN_ALARM,
                                  CMQ_FRM_MODE_T1_DM_FDL_JPN_ALARM,
                                  CMQ_FRM_MODE_T1_JT_G704,
                                  CMQ_FRM_MODE_T1_UNFRAMED */

frmCfg.outOfFrameCriteria = CMQ_T1_OOF_2OF4;      /* or
                                  CMQ_T1_OOF_2OF5,
                                  CMQ_T1_OOF_2OF6 */

frmCfg.frmESFAlgo = CMQ_T1_ESF_FRAME_ALGO_ONE_CANDIDATE; /* or
                                  CMQ_T1_ESF_FRAME_ALGO_CRC_6 */

frmCfg.COFACntEn = 1;             /* enable counting of change of
                                   frame alignment (COFA) events
                                   rather than      out of frame
                                   alignment */

/*** Function Call ***/
```

```
cometqT1RxFramerCfg(deviceHandle, chan, &frmCfg);


/*** E1 Tx Mode Framer ***/

UINT2 chan;

sCMQ_CFG_E1TX_FRM frmCfg;

/*** Variable Initialization ***/

chan = 0;  /* framer number 0,1,2,3 */

frmCfg.frmMode = CMQ_FRM_MODE_E1;      /* or
                                       CMQ_FRM_MODE_E1_CRC_MFRM,
                                       CMQ_FRM_MODE_E1_UNFRAMED */

frmCfg.ts16Signaling = CMQ_E1_SIG_INS_NONE; /* or
                                       CMQ_E1_SIG_INS_HDLC_CCS,
                                       CMQ_E1_SIG_INS_CAS */

frmCfg.insNatIntBitEn = 1;             /* enable insertion of
                                       international and national
                                       bits  */

frmCfg.insXtraBitsEn= 1;               /* enable insertion of
                                       extra bits */

frmCfg.insFEBEEn = 0;                  /* if CRC multiframe mode
                                       selected, enable/disable
                                       insertion of far end block
                                       error (FEBE) bits */

/*** Function Call ***/

cometqE1TxFramerCfg(deviceHandle, chan, &frmCfg);


/*** E1 Rx Mode Framer ***/

UINT2 chan;

sCMQ_CFG_E1RX_FRM frmCfg;

/*** Variable Initialization ***/

chan = 0;  /* framer number 0,1,2,3 */
```

```
    frmCfg.frmMode = CMQ_FRM_MODE_E1;      /* or
                                           CMQ_FRM_MODE_E1_CRC_MFRM,
                                           CMQ_FRM_MODE_E1_UNFRAMED */

    frmCfg.CASAlignmentEn = 1;             /* Enable alignment to
                                           channel associative
                                           signaling multiframes */

    frmCfg.CRC2NCRCEn = 1;                 /* Enable checking of CRC
                                           multiframe in CRC to non-
                                           CRC internetworking mode */

    frmCfg.noReframeOnErrEn = 1;           /* Disable reframing upon
                                           any error event - framing
                                           re-resynchronization must
                                           be forced by writing to the
                                           EFR bit in the E1-FRMR
                                           Frame Alignment Options
                                           register */

    frmCfg.reframeOnXSCrcErrEn = 1;        /* Enable forced reframing
                                           when excessive CRC errors
                                           are reported */

    frmCfg.lofBit2CritEn = 1;              /* Enable loss of frame
                                           criteria: Bit 2, timeslot 0
                                           of NFAS frames is 0 for 3
                                           consecutive frames */

    frmCfg.NFASErrEn = 1;                  /* Enable errors in bit 2
                                           of timeslot 0 of NFAS
                                           frames to contribute
                                           towards the framing error
                                           count */

    frmCfg.multFASEOneFEEn = 1;            /* Enable multiple FAS
                                           errors to generate a single
                                           framing error. If      NFAS
                                           errors are counted towards
                                           framing errors, enabling
                                           this option      includes
                                           Bit 2 of timeslot 0 in NFAS
                                           frames as part of the FAS
                                           word */

frmCfg.mfrmLossAlignCrit = CMQ_E1_LOSS_MFRM_ALIGN_TS16_CRIT_NONE;
               /* or CMQ_E1_LOSS_MFRM_ALIGN_TS16_CRIT_ZERO_1_MFRM,
               CMQ_E1_LOSS_MFRM_ALIGN_TS16_CRIT_ZERO_2_MFRM */
```

```
frmCfg.AISCriteria = CMQ_E1_AIS_CRIT_3Z_IN_512BITS;    /* or
                      CMQ_E1_AIS_CRIT_2_PERIODS_3Z_IN_512BITS */

frmCfg.RAICriteria = CMQ_E1_RAI_CRIT_ALL_A_1;    /* or
                      CMQ_E1_RAI_CRIT_4_CONSEC_A_1 */

/*** Function Call ***/

cometqE1RxFramerCfg(deviceHandle, chan, &frmCfg);
```

13.2.5 Configure the Transmit Timing using the driver

Timing options can be selected by writing to register (Transmit Timing Options) 0xQ06. This is normally set by the driver when you select other options such as configuring the line transmit interface jitter attenuator (cometqLineTxJatCfg), Configuring the transmit elastic store (cometqTxElstStCfg) or configuring the transmit backplane interface.

But you can still write to the register directly to select the timing options by using cometqWrite(deviceHandle, regnum, value). Please see the COMET-QUAD Data Sheet for the description of available timing options.

13.2.6 Configure the Receive Timing using the driver

Timing options can be selected by writing to register (Receive Options) 0xQ02. This is normally set by the driver when you select other options such as configuring the line receive interface jitter attenuator (cometqLineRxJatCfg), Configuring the receive elastic store (cometqRxElstStCfg) or configuring the receive backplane interface.

But you can still write to the register directly to select the timing options by using cometqWrite(deviceHandle, regnum, value). Please see the COMET-QUAD Data Sheet for the description of available timing options.

13.2.7 Configure the Backplane interfaces using the driver

```
/*** Backplane transmit interface access configuration ***/

UINT2 chan;

sCMQ_BACKPLANE_ACCESS_CFG btifCfgData;

/*** Variable Initialization ***/

chan = 0;  /* framer number 0,1,2,3 */

btifCfgData.masterMode = 1;           /* selects slave or master
                                      configuration */
```

```
  btifCfgData.dataMode = CMQ_BACKPLANE_FULL_FRAME_MODE;  /* or
                                   CMQ_BACKPLANE_NX56K_MODE,
                                   CMQ_BACKPLANE_NX64K_MODE,
                                   CMQ_BACKPLANE_NX64K_E1_MODE */

  btifCfgData.clkTimes2 = 1;         /* select clock mode
                                   multiplication by two - if
                                   set, clock operates at twice
                                   backplane rate */

btifCfgData.dataRate = CMQ_BACKPLANE_CLK_RATE_1544;      /* or
                                   CMQ_BACKPLANE_CLK_RATE_2048,
                                   CMQ_BACKPLANE_CLK_RATE_8192 */

  /*** Function Call ***/

  cometqBTIFAccessCfg(deviceHandle, chan, &btifCfgData);


  /*** Backplane transmit interface frame configuration ***/

  UINT2 chan;

  sCMQ_CFG_BTIF_FRM frmCfg;

  /*** Variable Initialization ***/

  chan = 0;  /* framer number 0,1,2,3 */

  frmCfg.fpMaster = 1;             /* Select frame pulse signal
                                   master/slave */

  frmCfg.fpInvEn = 0;              /* enables inversion of the
                                   frame pulse signal */

  frmCfg.oddPar = 0;               /* odd/even parity selection */

  frmCfg.extParEn = 1;             /* enable extension of parity
                                   over current and previous frame
                                   */

  frmCfg.fpFrmOffset = 0;          /* offset in bytes between the
                                   framing pulse and start of next
                                   frame: valid range: 0 - 127
                                   bytes */
```

```
frmCfg.T1ESFAlign = 1;          /* for T1 mode, enables ESF
                                alignment opposed to SF
                                alignment.  This option is not
                                supported for E1 mode */

frmCfg.fpBitOffsetEn = 0;       /* enables offset between frame
                                pulse and first timeslot in bits
                                */

frmCfg.fpBitOffset = 0;         /* if bit offset is enabled,
                                this value will be applied as
                                the bit offset between the frame
                                pulse and the first timeslot */

frmCfg.tslotMapFormat = CMQ_BACKPLANE_TIMESLOT_MAP_3_OF_4;  /* or
                         CMQ_BACKPLANE_TIMESLOT_MAP_24_OF_32 */

/*** Function Call ***/

cometqBTIFFrmCfg(deviceHandle, chan, &frmCfg);



/*** Backplane receive interface access configuration ***/

UINT2 chan;

sCMQ_BACKPLANE_ACCESS_CFG brifCfgData;

/*** Variable Initialization ***/

chan = 0;  /* framer number 0,1,2,3 */

brifCfgData.masterMode = 1;     /* selects slave or master
                                configuration */

brifCfgData.dataMode = CMQ_BACKPLANE_FULL_FRAME_MODE;  /* or
                                CMQ_BACKPLANE_NX56K_MODE,
                                CMQ_BACKPLANE_NX64K_MODE,
                                CMQ_BACKPLANE_NX64K_E1_MODE */

brifCfgData.clkTimes2 = 1;      /* select clock mode
                                multiplication by two - if set,
                                clock operates at twice
                                backplane rate */

brifCfgData.dataRate = CMQ_BACKPLANE_CLK_RATE_1544;         /* or
                                CMQ_BACKPLANE_CLK_RATE_2048,
                                CMQ_BACKPLANE_CLK_RATE_8192 */
```

```
/*** Function Call ***/

cometqBRIFAccessCfg(deviceHandle, chan, &brifCfgData);


/*** Backplane receive interface frame configuration ***/

UINT2 chan;

sCMQ_CFG_BRIF_FRM frmCfg;

/*** Variable Initialization ***/

chan = 0;  /* framer number 0,1,2,3 */

frmCfg.fpMaster = 1;            /* Select frame pulse signal
                                master/slave */

frmCfg.fpmMode = CMQ_BACKPLANE_RX_FP_T1_HIGH_ON_SF_ESF;     /* or
                     CMQ_BACKPLANE_RX_FP_T1E1_HIGH_EVERY_FRAME,
                     CMQ_BACKPLANE_RX_FP_E1_HIGH_ON_CRC_MFRM,
                     CMQ_BACKPLANE_RX_FP_E1_HIGH_ON_SIG_MFRM,
                     CMQ_BACKPLANE_RX_FP_E1_COMP_MFRM,
                     CMQ_BACKPLANE_RX_FP_E1_HIGH_ON_OVERHEAD */

frmCfg.fpInvEn = 0;             /* enables inversion of the
                                frame pulse signal */

frmCfg.parInsEn = 1;            /* enable parity insertion */

frmCfg.oddPar = 0;              /* odd/even parity selection -
                                if parity insertion enabled */

frmCfg.extParEn = 1;            /* enable extension of parity
                                over current and previous frame
                                */

frmCfg.fBitFix = 1;             /* enable fixing of F bit, only
                                valid when not in parity
                                insertion mode */

frmCfg.fBitPol = 0;             /* polarity of the F bit, valid
                                only when fBitFix is 1 and not
                                in parity mode */

frmCfg.fpFrmOffset = 0;         /* offset in bytes between the
                                framing pulse and start of next
                                frame: valid range: 0 - 127
                                bytes */
```

```
frmCfg.fpBitOffsetEn = 0;         /* enables offset between frame
                                  pulse and first timeslot in bits
                                  */

frmCfg.fpBitOffset = 0;           /* if bit offset is enabled,
                                  this value will be applied as
                                  the bit     offset between the
                                  frame pulse and the first
                                  timeslot (3 bit value) */

frmCfg.altFDLEn = 0;              /* Enabling this option causes
                                  the framing bit to contain FDL
                                  data. Only supported for ESF */

frmCfg.tslotMapFormat = CMQ_BACKPLANE_TIMESLOT_MAP_3_OF_4;  /* or
                             CMQ_BACKPLANE_TIMESLOT_MAP_24_OF_32 */

/*** Function Call ***/

cometqBRIFFrmCfg(deviceHandle, chan, &frmCfg);
```

13.2.8 Configure the Elastic Stores using the driver

The following registers are set according to the operating mode when `cometqSetOperatingMode` is called.

- RX-ELST Configuration register is configured to reflect the operating mode on both COMET and COMET-Quad devices
- RX-ELST CCS Configuration register is configured to the reflect operating mode on the COMET-Quad only
- TX-ELST Configuration register is configured to reflect the operating mode on both COMET and COMET-Quad devices
- TX-ELST CCS Configuration register is configured to the reflect operating mode on the COMET-Quad only

Optionally there are driver functions to specifically configure the Elastic Stores:

```
/*** Receive elastic store configuration ***/

UINT2 chan;

sCMQ_CFG_RX_ELST elstCfg;

/*** Variable Initialization ***/
```

```
chan = 0;  /* framer number 0,1,2,3 */

elstCfg.elstEnable = 1;    /* enable/disable the elastic store */

elstCfg.idleCode = 0x55;   /* Elastic store idle code */

elstCfg.CCSidleCode = 0x55;/* Elastic store CCS idle code
                              (COMET-QUAD only) */

/*** Function Call ***/

cometqRxElstStCfg(deviceHandle, chan, &elstCfg);


/*** Transmit elastic store configuration ***/

UINT2 chan;

UINT1 elstEnable;

/*** Variable Initialization ***/

chan = 0;       /* framer number 0,1,2,3 */

elstEnable = 1; /* enable the elastic store or force bypass */

/*** Function Call ***/

cometqTxElstStCfg(deviceHandle, chan, elstEnable);
```

13.2.9 Configure the Jitter Attenuators using the driver

```
/*** Transmit jitter attenuator configuration ***/

UINT2 chan;

sCMQ_CFG_TX_JAT txJatCfg;

/*** Variable Initialization ***/

chan = 0;       /* framer number 0,1,2,3 */

txJatCfg.enable = 1;       /* enable/disable transmit jitter
                           attenuation */
```

```
txJatCfg.refDiv = 0x2F;          /* for recommended values for
                                 refDiv  refer to the TJAT
                                 Divider Control register
                                 descriptions in the COMET or
                                 COMET-Quad data sheets */

txJatCfg.outputDiv = 0x2F;       /* for recommended values for
                                 outputDiv refer to the TJAT
                                 Divider Control register
                                 descriptions in the COMET or
                                 COMET-Quad data sheets */

txJatCfg.FIFOselfCenter = 1;     /* Enables the FIFO to self-
                                 center the read pointer upon
                                 FIFO overrun or underrun */

txJatCfg.preventOvfUndf = 0;     /* Set to 1 to prevent FIFO
                                 underflows/overflows at the
                                 expense of limited jitter
                                 attenuation. 0 allows full
                                 jitter attenuation but
                                 underflows/overflows can occur
                                 on the JAT FIFO */

txJatCfg.outputClock = CMQ_TJAT_OUTPUT_CLK_INTERN_JAT; /* or
                           CMQ_TJAT_OUTPUT_CLK_CTCLK,
                           CMQ_TJAT_OUTPUT_CLK_FIFO_INPUT */

txJatCfg.pllRefClock = CMQ_TJAT_PLL_REF_CLK_FIFO_INPUT;     /* or
                                 CMQ_TJAT_PLL_REF_CLK_BACKPLANE,
                                 CMQ_TJAT_PLL_REF_CLK_RECOVERED,
                                 CMQ_TJAT_PLL_REF_CLK_CTCLK */

/*** Function Call ***/

cometqLineTxJatCfg(deviceHandle, chan, &txJatCfg);


/*** Receive jitter attenuator configuration ***/

UINT2 chan;

sCMQ_CFG_RX_JAT rxJatCfg;

/*** Variable Initialization ***/

chan = 0;        /* framer number 0,1,2,3 */
```

```
rxJatCfg.enable = 1;                /* enable/disable receive
                                    jitter attenuation */

rxJatCfg.refDiv = 0x2F;             /* Ratio between the
                                    frequency of the recovered
                                    clock and the frequency of
                                    the phase discriminator
                                    input */

rxJatCfg.outputDiv = 0x2F;          /* Ratio between the
                                    frequency of the output
                                    clock and the frequency of
                                    the phase discriminator
                                    input */

rxJatCfg.FIFOselfCenter = 1;        /* Enables the FIFO to
                                    self-center the read
                                    pointer upon FIFO overrun
                                    or underrun */

rxJatCfg.preventOvfUndf = 0;        /* Set to 1 to prevent FIFO
                                    underflows/overflows at the
                                    expense of limited jitter
                                    attenuation. 0 allows full
                                    jitter attenuation but
                                    underflows/overflows can
                                    occur on the JAT FIFO */

/*** Function Call ***/

cometqLineRxJatCfg(deviceHandle, chan, &rxJatCfg);
```

13.2.10 Set the RLPS Voltage Reference value using the driver

The Receive Line Interface Equalizer Voltage Reference is normally set automatically when a call is made to **cometqSetOperatingMode.**

The following function can be used to write to any one of the RLPS indirect registers on a specific COMET or device.

```
/*** variable declaration ***/

UINT2 frmNum;

UINT1 regNum;

UINT4 value;

/*** Variable Initialization ***/
```

```
frmNum = 0;          /* framer number 0,1,2,3 */

regNum = 0xFD;   /* indirect address */

value = 0x80;    /* value to be written */

/*** Function Call ***/

WriteRLPS(deviceHandle, frmNum, regNum, UINT4 value);
```

13.2.11 Program the RLPS Equalizer RAM table using the driver

The equalizer RAM can be configured through the following function:

```
/*** variable declaration ***/

UINT2 chan;

sCMQ_CFG_RX_ANALOG rxAnalogCfg;

/*** Variable Initialization ***/

chan = 0;        /* framer number 0,1,2,3 */

rxAnalogCfg.aLosThreshold = CMQ_RX_ALOS_9DB_THRESH;    /* or
                                      CMQ_RX_ALOS_14_5DB_THRESH,
                                      CMQ_RX_ALOS_20DB_THRESH,
                                      CMQ_RX_ALOS_22DB_THRESH,
                                      CMQ_RX_ALOS_25DB_THRESH,
                                      CMQ_RX_ALOS_30DB_THRESH,
                                      CMQ_RX_ALOS_31DB_THRESH,
                                      CMQ_RX_ALOS_35DB_THRESH */

rxAnalogCfg.aLosDetectPeriod = 0x08;  /* Duration for declaring
                                      analog loss of signal.  The
                                      actual duration used is 16
                                      x aLosDetectPeriod pulse
                                      intervals */

rxAnalogCfg.aLosClearPeriod = 0x0F;   /* Duration for clearing
                                      analog loss of signal.  The
                                      actual duration used is 16
                                      x aLosClearPeriod pulse
                                      intervals */
```

```
rxAnalogCfg.eqFreq = CMQ_RX_EQ_FREQ_T1_24_125KHZ;/* or
                                    CMQ_RX_EQ_FREQ_T1_12_063KHZ,
                                    CMQ_RX_EQ_FREQ_T1_8_0417KHZ,
                                    CMQ_RX_EQ_FREQ_T1_6_0313KHZ,
                                    CMQ_RX_EQ_FREQ_T1_4_8250KHZ,
                                    CMQ_RX_EQ_FREQ_T1_4_0208KHZ,
                                    CMQ_RX_EQ_FREQ_T1_3_4464KHZ,
                                    CMQ_RX_EQ_FREQ_T1_3_0156KHZ,
                                    CMQ_RX_EQ_FREQ_E1_32_000KHZ,
                                    CMQ_RX_EQ_FREQ_E1_16_000KHZ,
                                    CMQ_RX_EQ_FREQ_E1_10_667KHZ,
                                    CMQ_RX_EQ_FREQ_E1_8_000KHZ,
                                    CMQ_RX_EQ_FREQ_E1_6_40KHZ,
                                    CMQ_RX_EQ_FREQ_E1_5_333KHZ,
                                    CMQ_RX_EQ_FREQ_E1_4_5714KHZ,
                                    CMQ_RX_EQ_FREQ_E1_4_0KHZ */

rxAnalogCfg.eqFdBckPer = CMQ_RX_EQ_VALID_PERIOD_32;     /* or
                                    CMQ_RX_EQ_VALID_PERIOD_64,
                                    CMQ_RX_EQ_VALID_PERIOD_128,
                                    CMQ_RX_EQ_VALID_PERIOD_256 */

rxAnalogCfg.ramType = CMQ_RX_LINE_EQ_RAM_T1; /* or
                                    CMQ_RX_LINE_EQ_RAM_E1,
                                    CMQ_RX_LINE_EQ_USER_DEFINED,
                                    CMQ_RX_LINE_EQ_RETAIN_CURRENT
                                    */

rxAnalogCfg.eqCoef = array[256]; /* programmable equalizer
                                    coefficients. if ramType is
                                    user-defined,      this array
                                    should contain the new
                                    coefficients */

rxAnalogCfg.squelchEn = 1;          /* enable/disable data
                                    squelching (force to all 0's)
                                    upon analog loss of signal
                                    detection */

/*** Function Call ***/

cometqLineRxAnalogCfg(deviceHandle, chan, &rxAnalogCfg);
```

13.2.12 Enable the RLPS Equalizer using the driver

The Receive Equalizer will be enabled when a call is made to the above **cometqLineRxAnalogCfg** function See the section on Programming the RLPS Equalizer RAM table.

13.2.13 Program the Transmit Pulse Waveform using the driver

```
/*** variable declaration ***/

UINT2 chan;

sCMQ_CFG_TX_ANALOG txAnalogCfg;

/*** Variable Initialization ***/

chan = 0;        /* framer number 0,1,2,3 */

txAnalogCfg.txEn = 1;           /* If zero, transmit lines TXTIP
                                and TXRING are held in high
                                impedence state */

txAnalogCfg.wvFormType = CMQ_TX_LBO_T1_LONG_HAUL_0DB;  /* or
                      CMQ_TX_LBO_T1_LONG_HAUL_7_5DB,
                      CMQ_TX_LBO_T1_LONG_HAUL_15DB,
                      CMQ_TX_LBO_T1_LONG_HAUL_22_5DB,
                      CMQ_TX_LBO_T1_LONG_HAUL_TR62411_0DB,
                      CMQ_TX_LBO_T1_SHORT_HAUL_110FT,
                      CMQ_TX_LBO_T1_SHORT_HAUL_220FT,
                      CMQ_TX_LBO_T1_SHORT_HAUL_330FT,
                      CMQ_TX_LBO_T1_SHORT_HAUL_440FT,
                      CMQ_TX_LBO_T1_SHORT_HAUL_550FT,
                      CMQ_TX_LBO_T1_SHORT_HAUL_660FT,
                      CMQ_TX_LBO_T1_SHORT_HAUL_TR62411_110FT,
                      CMQ_TX_LBO_T1_SHORT_HAUL_TR62411_220FT,
                      CMQ_TX_LBO_T1_SHORT_HAUL_TR62411_330FT,
                      CMQ_TX_LBO_T1_SHORT_HAUL_TR62411_440FT,
                      CMQ_TX_LBO_T1_SHORT_HAUL_TR62411_550FT,
                      CMQ_TX_LBO_T1_SHORT_HAUL_TR62411_660FT,
                      CMQ_TX_LBO_E1_75OHM,
                      CMQ_TX_LBO_E1_120OHM,
                      CMQ_TX_LBO_USER_DEFINED,
                      CMQ_TX_LBO_RETAIN_CURRENT */
```

```
txAnalogCfg.wvFormScFac = 0x0B;  /* Amplitude control of DAC
                                 output - control in increments
                                 of 11.14 mA. Parameter only used
                                 if wvFormType =
                                 CMQ_TX_LBO_USER_DEFINED. A value
                                 of 0 tri-states the output line
                                 and the max value is 21 (234mA).
                                 If the transmit waveform type is
                                 not USER_DEFINED, the value
                                 corresponding to the specified
                                 line build out configuration is
                                 applied */

txAnalogCfg.wvFormData = 24x5 matrix(see driver manual);    /*
                                 User defined waveform.  There
                                 are 24 samples of five unit
                                 intervals. Each sample is a 7
                                 bit value */

txAnalogCfg.fuseDataSel = CMQ_TX_FUSE_DATA_USER_DEFINED;    /* or
                                 CMQ_TX_FUSE_DATA_LIU_FUSE */

txAnalogCfg.alogTstPosCtrl = 0x2F;

txAnalogCfg.alogTstNegCtrl = 0x2F;
                                 /* Used when fuseDataSel is
                                 CMQ_TX_FUSE_DATA_USER_DEFINED. 6
                                 bit values used to control the
                                 digital to analog converted
                                 positive or negative current
                                 control in steps of 0.78125% in
                                 either the negative or positive
                                 direction */

/*** Function Call ***/

cometqLineTxAnalogCfg(deviceHandle, chan, &txAnalogCfg);
```

13.2.14 Run the XLPG Initialization Procedure using the driver

The Initialization procedure will automatically be implemented when a call is made to the above **cometqLineTxAnalogCfg** function. See the section on Programming the Transmit Pulse Waveform

13.2.15 Run the RLPS Optimization routine using the driver

The RLPS Optimization routine is performed automatically by the **cometqLineRxAnalogCfg** function after programming the equalizer tables. See the section on Programming the RLPS Equalizer RAM table.

13.2.16 Release the tri-states to begin transmission using the driver

Transmission will be enabled when a call is made to the above **cometqLineTxAnalogCfg** function with **ptxAnalogCfg.txEn = 1**. See the section on Programming the Transmit Pulse Waveform

13.3 Using the Per-Channel Serial Controllers with the driver

13.3.1 Using RPSC with the driver

```
/*** Receive per-channel serial controller enable ***/

UINT2 chan;

UINT2 enable;

/*** Variable Initialization ***/

chan = 0;         /* framer number 0,1,2,3 */

enable = 0;       /* enable/disable Receive Serial Controller */

/*** Function Call ***/

cometqRPSCEnable(deviceHandle, chan, enable);


/*** Transmit per-channel serial controller control ***/

UINT2 chan;

UINT1 tSlot;

UINT2 rWFlag;

UINT1 ctlByte;

UINT1 trnkData;

UINT1 sigData;

/*** Variable Initialization ***/

chan = 0;         /* framer number 0,1,2,3 */

tSlot = 0x0F;     /* timeslot - E1: 0 thru 31, T1: 1 thru 24 */

rWFlag = 1;       /* Read/Write select.  Write = 1,  Read = 0 */
```

```
ctlByte = 0x00;  /* output Control byte

                 bit 7:if receive pattern generation off, data
                 routed to PRBS/PRGD checker otherwise overwritten
                 with PRBS/PRGD test pattern

                 bit 6:overwrite data with data trunk conditioning
                 code byte

                 bit 5:overwrite signaling with signaling trunk
                 conditioning code byte

                 bit 4:replace pcm data with digital miliwat pat

                 bit 3:selects A-law mW patter instead of U-law

                 bit 2:invert most significant bit of data

                 bit 1,0: unused */

trnkData = 0x00; /* output trunk conditioning data byte */

sigData = 0;     /* output signaling data byte

                 bit 7,6,5,4: unused

                 bit 3,2,1,0: A,B,C,D bits */

/*** Function Call ***/

cometqRPSCPCMCtl(deviceHandle, chan, tSlot, rWFlag, &pctlByte,
&ptrnkData, &psigData);
```

13.3.2 Using TPSC with the driver

```
/*** Transmit per-channel serial controller enable ***/

UINT2 chan;

UINT2 enable;

/*** Variable Initialization ***/

chan = 0;        /* framer number 0,1,2,3 */

enable = 0;      /* enable/disable Transmit Serial Controller */

/*** Function Call ***/

cometqTPSCEnable(deviceHandle, chan, enable);
```

```
/*** Transmit per-channel serial controller control ***/

UINT2 chan;

UINT1 tSlot;

UINT2 rWFlag;

UINT1 pctlByte;

UINT1 ptrnkData;

UINT1 psigData;

/*** Variable Initialization ***/

chan = 0;        /* framer number 0,1,2,3 */

tSlot = 0x0F;    /* timeslot - E1: 0 thru 31, T1: 1 thru 24 */

rWFlag = 1;      /* Read/Write select.  Write = 1,  Read = 0 */

ctlByte = 0x00;  /* output pcm data control byte
           bit 7,5:
                 0,0 - data unchanged
                 0,1 - only msb inverted
                 1,0 - inverted all bits
                 1,1 - invert all except msb
           bit 6:replace pcm data with trunk conditioning byte
           bit 4:(T1 only)replace pcm data with digital mW
                  pattern
           bit 3:if receive pattern generation on, data routed to
                 PRBS/PRGD checker otherwise overwritten with
                 PRBS/PRGD test pattern
           bit 2:loopback DS0
           bit 1,0:  zero code suppression format
                 0,0 - no zero suppression
                 0,1 - jammed bit 8
                 1,0 - GTE zero suppression
                 1,1 - Bell zero suppression */

trnkData = 0x00; /* output trunk conditioning data byte */
```

```
sigData = 0;     /* output signaling data byte
      bit map for E1 mode:
            bit 7,6,5:data manipulation
                  0,0,0 - data unchanged
                  0,0,1 - invert odd timeslot bits
                  0,1,0 - invert even timeslot bits
                  0,1,1 - invert all timeslot bits
                  1,0,0 - replace data with idle code
                  1,0,1 - replace data with idle code
                  1,1,0 - replace data with A-law pattern
                  1,1,1 - replace data with U-law pattern
            bit 4:when CAS enabled, signaling bits taken
                  from A,B,C,D bits
            bit 3,2,1,0:A,B,C,D bits
      bit map for T1 mode:
            bit 7:forces signaling data from A,B,C,D bits
            bit 6:enables signal insertion from ABCD bits
            bit 5,4:unused
            bit 3,2,1,0:A,B,C,D bits */

/*** Function Call ***/

cometqTPSCPCMCtl(deviceHandle, chan, tSlot, rWFlag, &ctlByte,
&trnkData, &sigData);
```

13.4 Using the Signaling Extractor with the driver

```
/*** Change of signaling state detection ***/

UINT2 chan;

UINT4 sigState;

/*** Variable Initialization ***/

chan = 0;        /* framer number 0,1,2,3 */

sigState = 0x08; /* Signal state bit map.
      E1:   Bit 0 corresponds to timeslot 1 and bit 30 corresponds
            to timeslot 31. Timeslot 0 and 16 do not have COSS
            info. Bit 15 (timeslot 16) is always set to 0.  Bit 31
            is unused and set to zero.

      T1:   Bits 0 to 23 correspond to timeslots 1 to 24
            respectively.  Bits 24 to 31 are not used and set to 0
            */

/*** Function Call ***/
```

```
cometqExtractCOSS(deviceHandle, chan, &sigState);


/*** Signaling state extraction ***/

UINT2 chan;

UINT1 timeslot;

UINT1 sigState

/*** Variable Initialization ***/

chan = 0;               /* framer number 0,1,2,3 */

timeslot = 0x0F;        /* timeslot for which to retrieve
                        signaling. For T1, this value should be
                        from 1-24.  When in E1 mode, this   value
                        should range from 1-31 as there is no
                        signaling    info for timeslot 0 */

sigState = 0;           /* output contains the signaling state
                        information for the timeslot.
                             bit 0: D bit
                             bit 1: C bit
                             bit 2: B bit
                             bit 3: A bit
                             bits 4-7: unused */

/*** Function Call ***/

cometqSigExtract(deviceHandle, chan, timeslot, &sigState);
```

14 Interrupt Processing using the driver

Callback functions are available to the application for event notification from the device driver. Applications will be notified via the callback functions for selected events of interest such as:

- Alarm conditions
- Statistics
- Diagnostics
- Line coding and conditioning
- T1/E1 Framer
- Data Link
- Performance Monitoring

Refer to the section Sample configuration with the driver for an example of how to enable the interrupts and use the callback functions.

15 HDLC Programming using the driver

```
/*** HDLC enable ***/

UINT2 idHDLC;

UINT2 enable;

/*** Variable Initialization ***/

idHDLC = 0;             /* HDLC controller: 0, 1, 2, or  3  COMET:
                        0,1 or 2 */

enable = 1;             /* enable HDLC controller if set */

/*** Function Call ***/

cometqHDLCEnable(deviceHandle, idHDLC, enable);
```

15.1 Using the HDLC Transmitter with the driver

```
/*** HDLC TX configuration ***/

UINT2 idHDLC;

sCMQ_CFG_HDLC_TX data;

sCMQ_CFG_HDLC_LINK linkLocation;

/*** Variable Initialization ***/

idHDLC = 0;             /* HDLC controller: 0, 1, 2, or  3  COMET:
                        0,1 or 2 */

linkLocation.useT1DataLink = 1;      /* use the T1 FDL data link
                                     (ESF or T1DM with FDL
                                     modes). For COMET, only
                                     valid for first HDLC
                                     controller */

linkLocation.evenFrames = 0;

linkLocation.oddFrames = 0;          /* Enable link
                                     extraction/insertion from
                                     even or odd frames.
                                     Ignored when using T1 data
                                     link */
```

```
linkLocation.timeslot = 0;              /* Timeslot to extract data
                                        link from (0 based ).
                                        Ignored if both oddFrames
                                        and evenFrames are 0 or if
                                        using T1 Data Link */

linkLocation.dataLinkBitMask = 0;       /* Bit mask selecting which
                                        of the bits in a
                                        timeslot/channel are to be
                                        used. Ignored if both
                                        oddFrames and evenFrames
                                        are 0 or if using T1 Data
                                        Link */

data.linkLocation = linkLocation;

data.flagShareEn = 1;                   /* enable sharing of
                                        start/end flags between
                                        packets */

data.crcFCSEn = 1;                      /* enable CRC frame check
                                        sequence generation */

data.pmRepEn = 1;                       /*enable performance report
                                        transmission*/

/*** Function Call ***/

cometqHDLCTxCfg(deviceHandle, idHDLC, &data);


/*** HDLC TX data ***/

UINT2 idHDLC;

UINT1 value;

/*** Variable Initialization ***/

idHDLC = 0;                    /* HDLC controller: 0, 1, 2, or  3
                               COMET: 0,1 or 2 */

value = 0x55;                  /* byte to transmit on data link */

/*** Function Call ***/

TDPRData(deviceHandle, idHDLC, value);
```

```
/*** HDLC TX control byte ***/

UINT2 idHDLC;

UINT2 hdlcAction;

/*** Variable Initialization ***/

idHDLC = 0;                           /* HDLC controller: 0, 1,
                                      2, or  3  COMET: 0,1 or 2
                                      */

hdlcAction = CMQ_TDPR_ACTION_ABORT;   /* CMQ_TDPR_ACTION_ABORT -
                                      insert HDLC abort code into
                                      data link

                                      CMQ_TDPR_ACTION_END_ABORT -
                                      end abort code insertion

                                      CMQ_TDPR_ACTION_EOM - send
                                      end of  message indicator

                                      CMQ_TDPR_ACTION_FIFOCLR -
                                      clear transmit fifo */

/*** Function Call ***/

cometqTDPRCtl(deviceHandle, idHDLC, hdlcAction);


/*** HDLC TX configure threshold ***/

UINT2 idHDLC;

UINT1 upFifoThresh;

UINT1 lowFifoThresh;

/*** Variable Initialization ***/

idHDLC = 0;                           /* HDLC controller: 0, 1,
                                      2, or  3  COMET: 0,1 or 2
                                      */

upFifoThresh = 0xF0;                  /* upper FIFO threshold for
                                      auto transmit valid values
                                      are 0 thru 127 */
```

```
lowFifoThresh = 0x0F;                    /* lower FIFO threshold for
                                         LFILL interrupt valid
                                         values are 0 thru 127. Note
                                         that the lower threshold
                                         must be less than upper
                                         threshold unless both are
                                         set to 0 */

/*** Function Call ***/

TDPRFIFOThreshCfg(deviceHandle, idHDLC, upFifoThresh,
lowFifoThresh);
```

15.2 Using the HDLC Receiver with the driver

```
/*** HDLC RX configuration ***/

UINT2 idHDLC;

sCMQ_CFG_HDLC_RX data;

sCMQ_CFG_HDLC_LINK linkLocation;

/*** Variable Initialization ***/

idHDLC = 0;                              /* HDLC controller::  0, 1,
                                         2, or  3  COMET: 0,1 or 2
                                         */

linkLocation.useT1DataLink = 1;          /* use the T1 FDL data link
                                         (ESF or T1DM with FDL
                                         modes). For COMET, only
                                         valid for first HDLC
                                         controller */

linkLocation.evenFrames = 0;

linkLocation.oddFrames = 0;              /* Enable link
                                         extraction/insertion from
                                         even or odd frames.
                                         Ignored when using T1 data
                                         link */
```

```
linkLocation.timeslot = 0;              /* Timeslot to extract data
                                        link from (0 based ).
                                        Ignored if both oddFrames
                                        and evenFrames are 0 or if
                                        using T1 Data Link */

linkLocation.dataLinkBitMask = 0;       /* Bit mask selecting which
                                        of the bits in a
                                        timeslot/channel are to be
                                        used. Ignored if both
                                        oddFrames and evenFrames
                                        are 0 or if using T1 Data
                                        Link */

data.linkLocation = linkLocation;

data.addrMatchEn = 1;                   /* force detection of
                                        packets with either an
                                        address of all 1's or
                                        matching either the primary
                                        or secondary addresses */

data.addrMaskEn = 1;                    /* ignore the two least
                                        significant bits of the
                                        primary and secondary
                                        addresses when looking for
                                        matches */

/*** Function Call ***/

cometqHDLCRxCfg(deviceHandle, idHDLC, &data);


/*** HDLC RX terminate the reception of the current data frame
***/

UINT2 idHDLC;

/*** Variable Initialization ***/

idHDLC = 0;                             /* HDLC controller: 0, 1,
                                        2, or  3  COMET: 0,1 or 2
                                        */

/*** Function Call ***/

RDLCTerm(deviceHandle, idHDLC);
```

```
/*** HDLC RX address matching ***/

UINT2 idHDLC;

UINT1 addrPri;

UINT1 addrSec;

/*** Variable Initialization ***/

idHDLC = 0;                             /* HDLC controller: 0, 1,
                                        2, or  3  COMET: 0,1 or 2
                                        */

addrPri = 0x00;                         /* primary address */

addrSec = 0x00;                         /* secondary address */

/*** Function Call ***/

RDLCAddrMatch(deviceHandle, idHDLC, addrPri, addrSec);


/*** HDLC RX configure fill level threshold ***/

UINT2 idHDLC;

UINT1 fifoThresh;

/*** Variable Initialization ***/

idHDLC = 0;                             /* HDLC controller: 0, 1,
                                        2, or  3  COMET: 0,1 or 2
                                        */

fifoThresh = 0xF0;                      /* FIFO fill level
                                        threshold (7 bit value) */

/*** Function Call ***/

RDLCFIFOThreshCfg(deviceHandle, idHDLC, fifoThresh);
```

16 Alarms using the driver

```
/*** automatic alarm response configuration ***/

UINT2 chan;

UINT1 autoYellowEn;

UINT1 autoRedEn;

UINT1 OOF_RPSCEn;

UINT1 OOF_RxELSTEn;

UINT1 autoAISEn;

/*** Variable Initialization ***/

chan = 0;                         /* framer number 0,1,2,3 */

autoYellowEn = 1;                 /* enables automatic generation
                                  of yellow or RAI alarms in the
                                  receive direction upon a red
                                  alarm */

autoRedEn = 1;                    /* enables automatic trunk
                                  conditioning onto the backplane
                                  data and signaling streams from
                                  the RPSC upon a red carrier fail
                                  alarm */

OOF_RPSCEn = 1;                   /* enables automatic trunk
                                  conditioning onto the backplane
                                  data stream for the duration of
                                  out of frame - the conditioning
                                  data is inserted from the RPSC
                                  registers */

OOF_RxELSTEn = 0;                 /* enables automatic trunk
                                  conditioning onto the backplane
                                  data stream for the duration of
                                  out of frame - the conditioning
                                  data is inserted from the Rx
                                  Elastic store idle code
                                  registers */

autoAISEn = 1;                    /* enables automatic insertion
                                  of AIS into the receive path
                                  upon loss of signal */
```

```
/*** Function Call ***/

cometqAutoAlarmCfg(deviceHandle, chan, autoYellowEn, autoRedEn,
OOF_RPSCEn, OOF_RxELSTEn, autoAISEn);



/*** alarm insertion ***/

UINT2 chan;

eCMQ_ALARM_INS alarmType;

UINT1 enable;

/*** Variable Initialization ***/

chan = 0;                         /* framer number 0,1,2,3 */

alarmType = CMQ_ALARM_INS_RX_AIS;      /* type of alarm to insert:
                           CMQ_ALARM_INS_RX_AIS - force AIS onto
                           receive backplane interface
                           CMQ_ALARM_INS_TX_YELLOW - transmit
                           yellow alarm (RAI for E1)
                           CMQ_ALARM_INS_TX_AIS - force AIS onto
                           transmit stream
                           CMQ_ALARM_INS_TX_E1_Y_BIT - E1 only.
                           Sends the timeslot 16 Y-bit alarm
                           CMQ_ALARM_INS_TX_E1_TS16_AIS - E1
                           only. Transmits AIS in timeslot 16 */

enable = 1;                       /* specifies enabling/disabling
                                  the alarm */

/*** Function Call ***/

cometqInsertAlarm(deviceHandle, chan, alarmType, enable);
```

17 Programming the Pattern Generator/Detector with the Driver

```
/*** configure the pattern generator ***/

UINT2 chan;

UINT2 genLen;

UINT2 detLen;

UINT2 unFrmGen;

UINT2 unFrmDet;

UINT2 rxPatGenLoc;

/*** Variable Initialization ***/

chan = 0;                     /* framer number 0,1,2,3 */

genLen = CMQ_PRGD_PAT_8_BIT;  /* or CMQ_PRGD_PAT_7_BIT */

detLen = CMQ_PRGD_PAT_8_BIT;  /* or CMQ_PRGD_PAT_7_BIT */

unFrmGen = 0;                 /* generate unframed
                              pattern if set */

unFrmDet = 0;                 /* detect unframed pattern
                              if set */

rxPatGenLoc = 1;              /* location of the PRBS
                              generator/detector. If set,
                              the pattern detector is
                              inserted into the transmit
                              path and the generator is
                              inserted into the receive
                              path. If clear, the
                              generator is inserted in
                              the transmit path and the
                              detector is inserted in the
                              receive path. */

/*** Function Call ***/

cometqPRGDCtlCfg(deviceHandle, chan, genLen, detLen, unFrmGen,
unFrmDet, rxPatGenLoc);
```

```
/*** configuration of the pattern type ***/

UINT2 chan;

UINT1 quasiRand;

UINT2 pat;

/*** Variable Initialization ***/

chan = 0;                               /* framer number 0,1,2,3 */

quasiRand = 0;                          /* pattern type: pseudo-
                                        random = 0
                                        quasi-random  = 1 */

pat = CMQ_PSEUDO_RANDOM_PAT_2_TO_20th_MINUS1;

/* COMET-QUAD CHOICES: CMQ_PSEUDO_RANDOM_PAT_2_TO_20th_MINUS1
                CMQ_PSEUDO_RANDOM_PAT_2_TO_15th_MINUS1
                CMQ_PSEUDO_RANDOM_PAT_2_TO_11th_MINUS1
```

```
    COMET CHOICES: CMQ_PSEUDO_RANDOM_PAT_2_TO_3RD_MINUS1
                  CMQ_PSEUDO_RANDOM_PAT_2_TO_4th_MINUS1
                  CMQ_PSEUDO_RANDOM_PAT_2_TO_5th_MINUS1
                  CMQ_PSEUDO_RANDOM_PAT_2_TO_6th_MINUS1
                  CMQ_PSEUDO_RANDOM_PAT_2_TO_7th_MINUS1
                  CMQ_PSEUDO_RANDOM_PAT_FRAC_T1_ACTIVATE
                  CMQ_PSEUDO_RANDOM_PAT_FRAC_T1_DEACTIVATE
                  CMQ_PSEUDO_RANDOM_PAT_2_TO_9th_MINUS1_O_153
                  CMQ_PSEUDO_RANDOM_PAT_2_TO_10th_MINUS1
                  CMQ_PSEUDO_RANDOM_PAT_2_TO_11th_MINUS1_O_152
                  CMQ_PSEUDO_RANDOM_PAT_2_TO_15th_MINUS1_O_151
                  CMQ_PSEUDO_RANDOM_PAT_2_TO_17th_MINUS1
                  CMQ_PSEUDO_RANDOM_PAT_2_TO_18th_MINUS1
                  CMQ_PSEUDO_RANDOM_PAT_2_TO_20th_MINUS1_O_153
                  CMQ_PSEUDO_RANDOM_PAT_2_TO_20th_MINUS1_O_151
                  CMQ_PSEUDO_RANDOM_PAT_2_TO_21th_MINUS1
                  CMQ_PSEUDO_RANDOM_PAT_2_TO_22th_MINUS1
                  CMQ_PSEUDO_RANDOM_PAT_2_TO_23th_MINUS1_O_151
                  CMQ_PSEUDO_RANDOM_PAT_2_TO_25th_MINUS1
                  CMQ_PSEUDO_RANDOM_PAT_2_TO_28th_MINUS1
                  CMQ_PSEUDO_RANDOM_PAT_2_TO_29th_MINUS1
                  CMQ_PSEUDO_RANDOM_PAT_2_TO_31ST_MINUS1
                  CMQ_PSEUDO_RANDOM_PAT_ALL_ONES
                  CMQ_PSEUDO_RANDOM_PAT_ALL_ZEROS
                  CMQ_PSEUDO_RANDOM_PAT_ALT_ONES_AND_ZEROS
                  CMQ_PSEUDO_RANDOM_PAT_DOUBLE_ALT_ONES_AND_ZEROS
                  CMQ_PSEUDO_RANDOM_PAT_3_IN_24
                  CMQ_PSEUDO_RANDOM_PAT_1_IN_16
                  CMQ_PSEUDO_RANDOM_PAT_1_IN_8
                  CMQ_PSEUDO_RANDOM_PAT_1_IN_4
                  CMQ_PSEUDO_RANDOM_PAT_INBAND_LOOPBACK_ACTIVATE
                  CMQ_PSEUDO_RANDOM_PAT_INBAND_LOOPBACK_DEACTIVATE
                  */

/*** Function Call ***/

cometqPRGDPatCfg(deviceHandle, chan, quasiRand, pat);



/*** retrieve the bit error count ***/

UINT2 chan;

UINT4 count;

/*** Variable Initialization ***/

chan = 0;        /* framer number 0,1,2,3 */
```

```
pcount = 0;           /* output total bit error count */
```

```
/*** Function Call ***/
```

```
cometqPRGDCntGet(deviceHandle, chan, &count);
```

18 Performance Monitoring with the Driver

```
/*** update performance monitor counters ***/

cometqForceStatsUpdate(deviceHandle);


/*** retrieve performance monitor statistics ***/

sCMQ_FRM_CNTS data; /* output - framer statistics structure */

/*** Function Call ***/

cometqGetStats(deviceHandle, &data);


/*** retrieve status and alarms information ***/

sCMQ_FRM_STATUS status; /* output - framer status structure */

/*** Function Call ***/

cometqGetStatus(deviceHandle, &status);


/*** retrieve clock status information ***/

UINT2 chan;

sCMQ_CLK_STATUS clkStat; /* output - clock status structure */

/*** Variable Initialization ***/

chan = 0;       /* framer number 0,1,2,3 */

/*** Function Call ***/

cometqLineClkStatGet(deviceHandle, chan, &clkStat);
```

```
/*** Enable/Disable one second update of Auto Performance Report
Monitoring (APRM) ***/

UINT2 chan;

UINT2 action;

/*** Variable Initialization ***/

chan = 0;        /* framer number 0,1,2,3 */

action = CMQ_AUTO_PMON_UPDATE_DISABLE;      /* or
                                     CMQ_AUTO_PMON_UPDATE_ENABLE,
                                     CMQ_AUTO_PMON_UPDATE_MAN */

/*** Function Call ***/

cometqPmonSet(deviceHandle, chan, action);


/*** get the current one second performance report ***/

UINT2 chan;

sCMQ_STAT_APRM pmonReport; /* output - performance report
                                          structure */

/*** Variable Initialization ***/

chan = 0;        /* framer number 0,1,2,3 */

/*** Function Call ***/

cometqPmonReportGet(deviceHandle, chan, &pmonReport);
```

19 Diagnostics with the Driver

```
/*** Register access test ***/

TestReg(deviceHandle);


/*** Framer loopback ***/

UINT2 framer;

UINT2 type;

/*** Variable Initialization ***/

framer = 0;           /* framer number 0,1,2,3 */

type = CMQ_LOOPBACK_NONE;  /* or CMQ_LOOPBACK_DIGITAL,
                           CMQ_LOOPBACK_LINE,
                           CMQ_LOOPBACK_PAYLOAD */

/*** Function Call ***/

cometqLoopFramer(deviceHandle, framer, type);


/*** DS0 loopback ***/

UINT2 framer;

UINT4 timeSlot;

UINT2 enable;

/*** Variable Initialization ***/

framer = 0;           /* framer number 0,1,2,3 */

timeSlot = 0;         /* bit mask for timeslots to loopback
                           T1:   bit 0-23  : channels 1-24
                                 bit 24-31 : unused
                           E1:   bit 0-31  : timeslots 0-31 */

enable = 1;           /* sets loop if non-zero, else clears loop
                      */
```

```
/*** Function Call ***/

cometqLoopTslots(deviceHandle, framer, timeSlot, enable);


/*** Analog transmitter bypass ***/

UINT2 chan;

UINT2 enable;

/*** Variable Initialization ***/

chan = 0;          /* framer number 0,1,2,3 */

enable = 1;        /* enable/disable analog bypass */

/*** Function Call ***/

cometqTxAnalogByp(deviceHandle, chan, enable);


/*** Analog receiver bypass ***/

UINT2 chan;

UINT2 enable;

/*** Variable Initialization ***/

chan = 0;          /* framer number 0,1,2,3 */

enable = 1;        /* enable/disable analog bypass */

/*** Function Call ***/

cometqRxAnalogByp(deviceHandle, chan, enable);
```

20 Sample Configuration with the Driver

```
/** include files **/

#include "cmq_app.h"

#include "cmq_api.h"


/* Callback functions */

void cometqCbackIntf(sCMQ_USR_CTXT usrCtxt, sCMQ_DPV *pdpv);

void cometqCbackFramer(sCMQ_USR_CTXT usrCtxt, sCMQ_DPV *pdpv);

void cometqCbackAlarmInBand(sCMQ_USR_CTXT usrCtxt, sCMQ_DPV
*pdpv);

void cometqCbackSigInsExt(sCMQ_USR_CTXT usrCtxt, sCMQ_DPV *pdpv);

void cometqCbackPMon(sCMQ_USR_CTXT usrCtxt, sCMQ_DPV *pdpv);

void cometqCbackSerialCtl(sCMQ_USR_CTXT usrCtxt, sCMQ_DPV *pdpv);


/** public data **/

extern sCMQ_MDB    *cometqMdb;


/* T1 ESF transmit and receive, 22.5 dB long haul, full T1
backplane clock master */

sCMQ_DIV T1ESFLongHaul =

   { 0,

     CMQ_ISR_MODE,

     cometqCbackIntf,

     cometqCbackFramer,

     cometqCbackAlarmInBand,

     cometqCbackSigInsExt,
     cometqCbackPMon,

     cometqCbackSerialCtl,

     1,

     {  /* analog configuration */

  CMQ_TX_LBO_T1_LONG_HAUL_22_5DB,     /* long haul 22.5 dB
                                      waveform template */

        1,                       /* enable transmitter */

        CMQ_RX_LINE_EQ_RAM_T1,   /* T1 equalizer RAM */

        CMQ_XCLK_2048_TXCLK_1544     /* XCLK scaled down to
                                      1.544 MHz */

     },

     {  /* framer configuration */

        CMQ_FRM_MODE_T1_ESF,         /* transmit T1 ESF */

        CMQ_FRM_MODE_T1_ESF          /* receive T1 ESF */

     },

     {  /* backplane configuration */

        CMQ_BACKPLANE_TX_CLOCK_MASTER_FULL_T1E1, /* tx backplane
                                      full T1 */

        CMQ_BACKPLANE_RX_CLOCK_MASTER_FULL_T1E1,      /* rx
                                      backplane full T1 */

        0, 0, 0                   /* unused when not H-MVIP */

     }

   };
```

```
/* E1 CRC-4 Multiframe transmit and receive, 75 Ohm cable,
* full E1 backplane clock master */
sCMQ_DIV E1_CRC4 =
   { 0,
     CMQ_ISR_MODE,
     cometqCbackIntf,
     cometqCbackFramer,
     cometqCbackAlarmInBand,
     cometqCbackSigInsExt,
     cometqCbackPMon,
     cometqCbackSerialCtl,
     1,
     {  /* analog configuration */
        CMQ_TX_LBO_E1_75OHM,     /* E1 75 Ohm tx waveform
                                 template      */
        1,                       /* enable transmitter  */
        CMQ_RX_LINE_EQ_RAM_E1,   /* E1 equalizer RAM  */
        CMQ_XCLK_2048_TXCLK_1544 /* XCLK scaled don to 1.544 MHz
                                 */
     },
     {  /* framer configuration */
        CMQ_FRM_MODE_E1_CRC_MFRM,     /* transmit E1 CRC-4
                                 multiframe  */
        CMQ_FRM_MODE_E1_CRC_MFRM /* receive E1 CRC-4 multiframe
                                 */
     },
     {  /* backplane configuration */
```

```
        CMQ_BACKPLANE_TX_CLOCK_MASTER_FULL_T1E1, /* tx backplane
                                        full E1   */

        CMQ_BACKPLANE_RX_CLOCK_MASTER_FULL_T1E1, /* rx backplane
                                        full E1 */

        0, 0, 0                         /* unused when not H-MVIP */

      }

    };

/*** EXAMPLE CODE***/

#define COMET_QUAD


/*****************************************************************

cometqExampleInit

DESCRIPTION:     This function is given as an example showing how
to use a COMET or COMET-Quad device when initializing from a DIV.
A typical sequence of function calls are made to show how to use
the most common APIs.

INPUTS:          usrCtxt    : user context

                 baseAddr   : base address of the device

OUTPUTS:         None

RETURN CODES:    None

*****************************************************************/
void cometqExampleInit (sCMQ_USR_CTXT usrCtxt, void *baseAddr)

{

    INT4          retCode;

    UINT2         index;


    sCMQ_MIV  miv = { 0,    /* perrModule   */

                      1,    /* maxDevs      */
```

```
                  1 }; /* maxInitProfs */


sCMQ_HNDL    deviceHandle;

UINT2         profileNum;

INT4        *perrDevice;


static sCMQ_ISR_MASK sMASK;


/* API structures */

sCMQ_CFG_TX_JAT TJATCfg;

sCMQ_CFG_RX_JAT RJATCfg;


/**** Initialize driver and add the device */

/* Open and start the driver module */

retCode = cometqModuleOpen(&miv);

retCode = cometqModuleStart();


/* Add an initialization profile */

retCode = cometqAddInitProfile(&T1ESFLongHaul, &profileNum);


/* Add one COMET or COMET-Quad device */

deviceHandle = cometqAdd(usrCtxt, baseAddr, &perrDevice);


/**** Initialize the device */

/* Initialize the device using a DIV */
```

```
    retCode = cometqInit(deviceHandle, &T1ESFLongHaul, 0);


    /* Reset the device then initialize it using the stored
profile */

    retCode = cometqReset(deviceHandle);

    retCode = cometqInit(deviceHandle, NULL, profileNum);


    /* update the configuration using a DIV */

    retCode = cometqUpdate(deviceHandle, &T1ESFLongHaul, 0);


    /* update the configuration using the stored profile */

    retCode = cometqUpdate(deviceHandle, NULL, profileNum);


/**** Perform some additional initialization not performed by the
DIV for which we may want to change default (reset) values ****/

/* configure the transmit line coding scheme from its default
(AMI) */

retCode = cometqLineTxEncodeCfg(deviceHandle, 0,
CMQ_LINE_CODE_B8ZS_T1);

#ifdef COMET_QUAD

    /* configure remaining quadrants */

retCode = cometqLineTxEncodeCfg(deviceHandle, 1,
CMQ_LINE_CODE_B8ZS_T1);

retCode = cometqLineTxEncodeCfg(deviceHandle, 2,
CMQ_LINE_CODE_B8ZS_T1);

retCode = cometqLineTxEncodeCfg(deviceHandle, 3,
CMQ_LINE_CODE_B8ZS_T1);

#endif
```

```
/* receive transmit line coding scheme default is B8Zs */


    /* configure the transmit jitter attenuator */

    TJATCfg.enable = 1;

    TJATCfg.FIFOselfCenter = 1;

    TJATCfg.preventOvfUndf = 0;

    TJATCfg.refDiv = 0x2f;

    TJATCfg.outputDiv = 0x2f;

    TJATCfg.outputClock = CMQ_TJAT_OUTPUT_CLK_INTERN_JAT;

    TJATCfg.pllRefClock = CMQ_TJAT_PLL_REF_CLK_BACKPLANE;

    retCode = cometqLineTxJatCfg(deviceHandle, 0, &TJATCfg);

    #ifdef COMET_QUAD

    /* configure remaining quadrants */

    retCode = cometqLineTxJatCfg(deviceHandle, 1, &TJATCfg);

    retCode = cometqLineTxJatCfg(deviceHandle, 2, &TJATCfg);

    retCode = cometqLineTxJatCfg(deviceHandle, 3, &TJATCfg);

    #endif


    /* disable the receive jitter attenuator */

    RJATCfg.enable = 0;

    retCode = cometqLineRxJatCfg(deviceHandle, 0, &RJATCfg);

    #ifdef COMET_QUAD

    /* configure remaining quadrants */

    retCode = cometqLineRxJatCfg(deviceHandle, 1, &RJATCfg);

    retCode = cometqLineRxJatCfg(deviceHandle, 2, &RJATCfg);
    retCode = cometqLineRxJatCfg(deviceHandle, 3, &RJATCfg);

    #endif


    /**** Configure interrupts */

    /* The device is curently in ISR mode, as initially set
inside the stored profile, configure it for polling mode instead
*/

    retCode = cometqISRConfig(deviceHandle, CMQ_POLL_MODE);


    /* Enable relevant interrupts */

    sysCometqMemSet(&sMASK, 0, sizeof(sCMQ_ISR_MASK));

    #ifdef COMET_QUAD

    /* set all four quadrants */

    for (index = 0; index < 4; index++)

    {

       sMASK.almi[index].yelEn = 1;

       sMASK.almi[index].redEn = 1;

       sMASK.almi[index].AISEn = 1;

       sMASK.cdrc[index].lcvEn = 1;

       sMASK.cdrc[index].lcvEn = 1;

       sMASK.cdrc[index].losEn = 1;

    }

    #else

    /* COMET - only one framer */

    sMASK.almi[0].yelEn = 1;

    sMASK.almi[0].redEn = 1;
```

```
        sMASK.almi[0].AISEn = 1;

        sMASK.cdrc[0].lcvEn = 1;

        sMASK.cdrc[0].lcvEn = 1;

        sMASK.cdrc[0].losEn = 1;

        #endif


        /* Set the interrupt mask */

        retCode = cometqSetMask(deviceHandle, &sMASK);


        /* Activate the device - device can now be polled */

        retCode = cometqActivate(deviceHandle);


        /* poll the interrupt status bits */

        retCode = cometqPoll(deviceHandle);


        /**** Delete the device and close the driver */

        retCode = cometqDeActivate(deviceHandle);

        retCode = cometqDelete(deviceHandle);

        retCode = cometqModuleStop();

        retCode = cometqModuleClose();

    }

    /*****************************************************************

    cometqCbackIntf

    DESCRIPTION:    This function is given as a template to use for
    the pplication callback function that would be called for
    nterface events
```

```
VALID STATES:   CMQ_ACTIVE

SIDE EFFECTS:   none

INPUTS:  usrCtxt  - user context pointer

         pdpv     - deferred Processing vector describing event
                       that occurred

OUTPUTS:        None

RETURN CODES:   None

*****************************************************************/

void cometqCbackIntf(sCMQ_USR_CTXT usrCtxt, sCMQ_DPV *pdpv)

{

sysCometqDPVBufferRtn(pdpv);

}


/*****************************************************************

cometqCbackFramer

DESCRIPTION:    This function is given as a template to use for
the application callback function that would be called for T1/E1
framing events

VALID STATES:   CMQ_ACTIVE

SIDE EFFECTS:   none

INPUTS:    usrCtxt  - user context pointer

           pdpv - deferred Processing vector describing event
                       that occurred

OUTPUTS:        None

RETURN CODES:   None

*****************************************************************/

void cometqCbackFramer(sCMQ_USR_CTXT usrCtxt, sCMQ_DPV *pdpv)
```

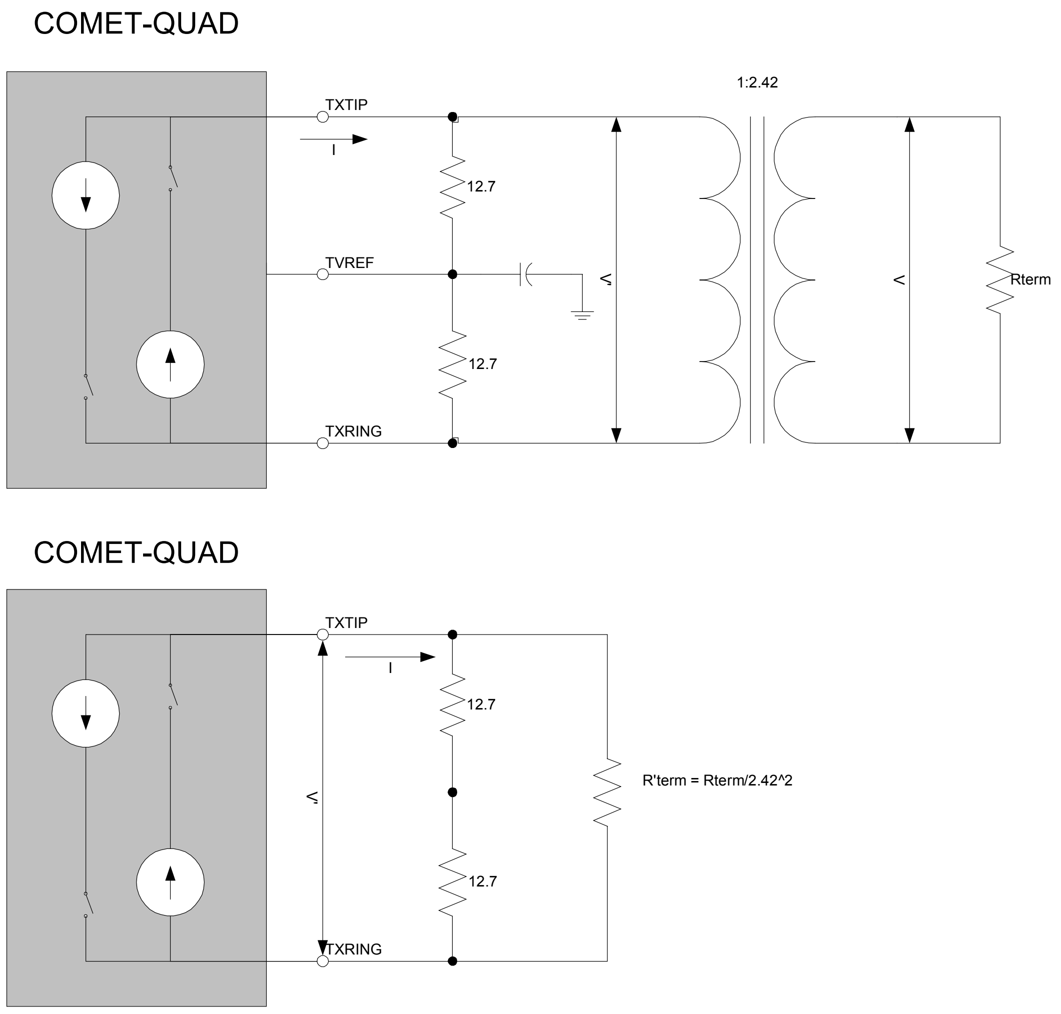
```
{

    sysCometqDPVBufferRtn(pdpv);

}


/*****************************************************************

cometqCbackAlarmInBand

DESCRIPTION:    This function is given as a template to use for
the application callback function that would be called for Alarm
and InBand notification events

VALID STATES:   CMQ_ACTIVE

SIDE EFFECTS:   none

INPUTS:         usrCtxt  - user context pointer

                pdpv     - deferred Processing vector describing
                           event that occurred

OUTPUTS:        None

RETURN CODES:   None

*****************************************************************/

void cometqCbackAlarmInBand(sCMQ_USR_CTXT usrCtxt, sCMQ_DPV
*pdpv)

{

    sysCometqDPVBufferRtn(pdpv);

}

/*****************************************************************

cometqCbackSigInsExt

DESCRIPTION:    This function is given as a template to use for
the application callback function that would be called for signal
insertion and extraction events

VALID STATES:   CMQ_ACTIVE
```

```
SIDE EFFECTS:   none

INPUTS:         usrCtxt  - user context pointer

               pdpv     - deferred Processing vector describing
                            event that occurred

OUTPUTS:        None

RETURN CODES:   None

*****************************************************************/

void cometqCbackSigInsExt(sCMQ_USR_CTXT usrCtxt, sCMQ_DPV *pdpv)

{

    sysCometqDPVBufferRtn(pdpv);

}


/*****************************************************************

cometqCbackPMon

DESCRIPTION:    This function is given as a template to use for
the application callback function that would be called for
performance monitoring events

VALID STATES:   CMQ_ACTIVE

SIDE EFFECTS:   none

INPUTS:         usrCtxt  - user context pointer

               pdpv     - deferred Processing vector describing
                            event that occurred

OUTPUTS:        None

RETURN CODES:   None

*****************************************************************/

void cometqCbackPMon(sCMQ_USR_CTXT usrCtxt, sCMQ_DPV *pdpv)

{
    sysCometqDPVBufferRtn(pdpv);

}


/******************************************************************

cometqCbackSerialCtl

DESCRIPTION:    This function is given as a template to use for
theapplication callback function that would be called for serial
controller events

VALID STATES:   CMQ_ACTIVE

SIDE EFFECTS:   none

INPUTS:         usrCtxt  - user context pointer

               pdpv     - deferred Processing vector describing
                           event that occurred

OUTPUTS:        None

RETURN CODES:   None

******************************************************************/

void cometqCbackSerialCtl(sCMQ_USR_CTXT usrCtxt, sCMQ_DPV *pdpv)

{

    sysCometqDPVBufferRtn(pdpv);

}
```

APPENDIX A: CALCULATING THE PULSE TEMPLATE VALUES

According to the COMET-QUAD data sheet the TXTIP and TXRING pins of the COMET-QUAD have to be connected to the external protection circuitry which can be converted to its Norton equivalent as shown in Figure 9:

Figure 9 The Transmit Circuit of COMET-QUAD (top) and its Norton Equivalent (bottom)



The amplitude of current at COMET-QUAD pins TXTIP and TXRING is controllable in 11.14 mA increments up to a maximum value of 21x11.14 mA = 234 mA by setting the value of SCALE[4:0] bits of the XLPG_Line_Driver_Configuration(0xQF0) register. The sample values will then be configurable in 126 equi-distant values in the range between the positive and negative current amplitude, as specified by the value of WDAT[6:0] in the pulse waveform internal registers. Refer to sections 4.2.1 and 4.2.4 for instructions on how to write to pulse waveform internal registers and set the current amplitude.

Our task here is to determine, given the resistance of the termination resistor R_{term}, the value of current going through TXTIP and TXRING that will yield the required voltage on the tips of termination resistor R_{term}. For this purpose, we will introduce the following notation:

R_{term}	is the termination resistor
R'_{term}	is the termination resistor mirrored to the primary side
R	is the resistance sensed by COMET-QUAD
V	is the desired pulse template voltage on the tips of termination resistor R_{term}
V'	is the pulse template voltage on the primary side (between TXTIP and TXRING)
I	is the line current driven by COMET-QUAD between TXTIP and TXRING
A	is the value written to SCALE[4:0].

From Figure 9, the following equations can be derived:

$V' = V / 2.42$

$V' = I * R$

$R = 25.4\ \Omega \parallel R'_{term}$

$R'_{term} = R_{term} / 2.42^2$

After a couple of simple transformations, the upper equations yield:

$I = ((2.42^2 * 25.4\Omega) + R_{term}) / (2.42 * 25.4\Omega * R_{term}) * V$

This gives us the inter-dependency between the line current, terminal resistance and the voltage on the tips of the terminal resistor. Hence, for the amplitude value of the voltage on the tips of the terminal resistor, V_a, we get:

$A = \text{round-up}(I_a/11.14\text{mA}) =$

$= \text{round-up}(((2.42^2 * 25.4\Omega) + R_{term}) / (2.42 * 25.4\Omega * R_{term}) * V_a / 11.14\text{mA})$

This represents the value that has to be written to SCALE[4:0], and since it is the line current amplitude, it corresponds to the maximum value of 0x3E that WDAT[6:0] could be set to. The samples placed in WDAT[6:0] are then scaled from the amplitude 0x3E based on the proportion:

WDAT[6:0] = round-up(0x3E * (V / V_a))

Note: *Although the maximum positive two-complement number that can be written to WDAT[6:0] is 0x3F, the value of 0x3E is taken because this corresponds to the maximum negative number that can be written to WDAT[6:0].*

Notes