PM4354

COMET-QUAD

COMET-QUAD EVALUATOR BOARD SOFTWARE

PRELIMINARY

ISSUE 2: AUGUST 2001

REVISION HISTORY

Issue No.	Issue Date	Details of Change
1	May 2000	Document Created.
2	August 2001	Updated document to reflect software changes related to release of new comet-quad software driver 1.0

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

The COMET-QUAD Evaluator Software is part of the COMET-QUAD Evaluator Kit and allows for the evaluation and demonstration of PMC-Sierra's PM4354 COMET-QUAD device. The kit also provides an environment for the development and integration of software using the software driver.

The evaluation software consists of two parts. The Graphic Display provides a way for the user to access the COMET-QUAD device registers and the Tcl console provides an interface to exercise COMET-QUAD driver APIs.

1.1 Purpose

The COMET-QUAD Evaluator software is designed to assist software developers in designing or integrating the COMET-QUAD device driver into their system. It helps to reduce development time when using PMC-Sierra's COMET-QUAD device and driver. The purpose of this document is to provide a detailed description of the COMET-QUAD Evaluator Software design.

1.2 Scope

This document describes proper use of the software and demonstrates the functionality of the COMET-QUAD device on the evaluation board. The document gives a thorough description of each window display and use of each Tcl command. The document is prepared for COMET-QUAD Evaluation Kit Software users. The evaluation software is designed and built on top of the COMET-QUAD Device Driver. The user should refer to the documents listed in the reference section for a more in-depth understanding of COMET-QUAD device operation and driver design.

2 INSTALLATION

2.1 System Requirements

This program requires a Windows 95, 98, NT or 2000 PC with a Pentium and a PCI bus. For best performance, a clock speed of 200MHz and RAM size of 32Mbytes or greater is recommended. A lower performance PC can be used but the window displays may appear slow. The graphical user interface is best viewed with a monitor of size 17" or larger. The display resolution is best viewed with 1024X768.

2.2 Installation Steps

1. Insert the installation CD and run the executable file "setup.exe".

2. For best results, adjust the display monitor resolution via the control panel or the task bar. This program is best viewed with a resolution of 1024x768 and 256 or more colors.

3. Run the program from the start menu under PROGRAMS | PMC_SIERRA | COMETQUAD GUI or by double clicking on the file “cometgui.exe” within the Windows Explorer.

4. The program will notify the user if it does not detect the PCI Evaluator Kit board on the PCI bus. The user should follow the hardware installation notes to install the hardware.

2.3 Un-Installation Steps

1. Open the control panel and select the add/remove software icon.

2. Select "COMETQUAD" from the list and press the remove button.

3 SOFTWARE FILES LIST

3.1 OS Dependent Files

Table 1 - Windows 95/98 Files

File Name	Description
comettcl.exe	This is the Tcl console executable file for Win95 that has all the COMET-QUAD driver API commands built in.
cometquad.exe	This is the GUI file for Win95 that executes the main window.
comet.vxd	This is the Windows 95/98 driver. This driver is used by the GUI and Tcl Console to provide access to the COMET-QUAD Evaluation Board.

Table 2 - Windows NT/2000 Files

File Name	Description
comettcl.exe	This is the Tcl console executable file for WinNT that has all the COMET-QUAD driver API commands built in.
cometquad.exe	This is the GUI file for WinNT that executes the main window.
comet.sys	This is the Windows NT/2000 driver. This driver is used by the GUI and Tcl Console to provide access to the COMET-QUAD Evaluation Board.

3.2 OS Independent Files

Table 3 - Miscellaneous Files

File Name	Description
cometquad.hlp	This is the Windows help file that is built based on the COMET-QUAD Data sheet.
cometquad.EMF	The COMET-QUAD block diagram.

File Name	Description
cometquad_t1_esf_sh.tcl	This script sets up the COMET-QUAD for Short Haul (0-110ft) T1 ESF external payload loop back (for all channels) using the driver API's.
cometquad_e1_crcmf.tcl	This script sets up the COMET-QUAD for E1 CRC-Multi-Frame external payload loop back (for all channels) using the driver API's.
cometquad_reg_t1_esf_sh.tcl	This script sets up the COMET-QUAD for Short Haul (0-110ft) T1 ESF external payload loop back (for all channels) using register writes only.
cometquad_reg_e1_crcmf.tcl	This script sets up the COMET-QUAD for E1 CRC-Multi-Frame external payload loop back (for all channels) using register writes only.

4 GRAPHIC USER INTERFACE

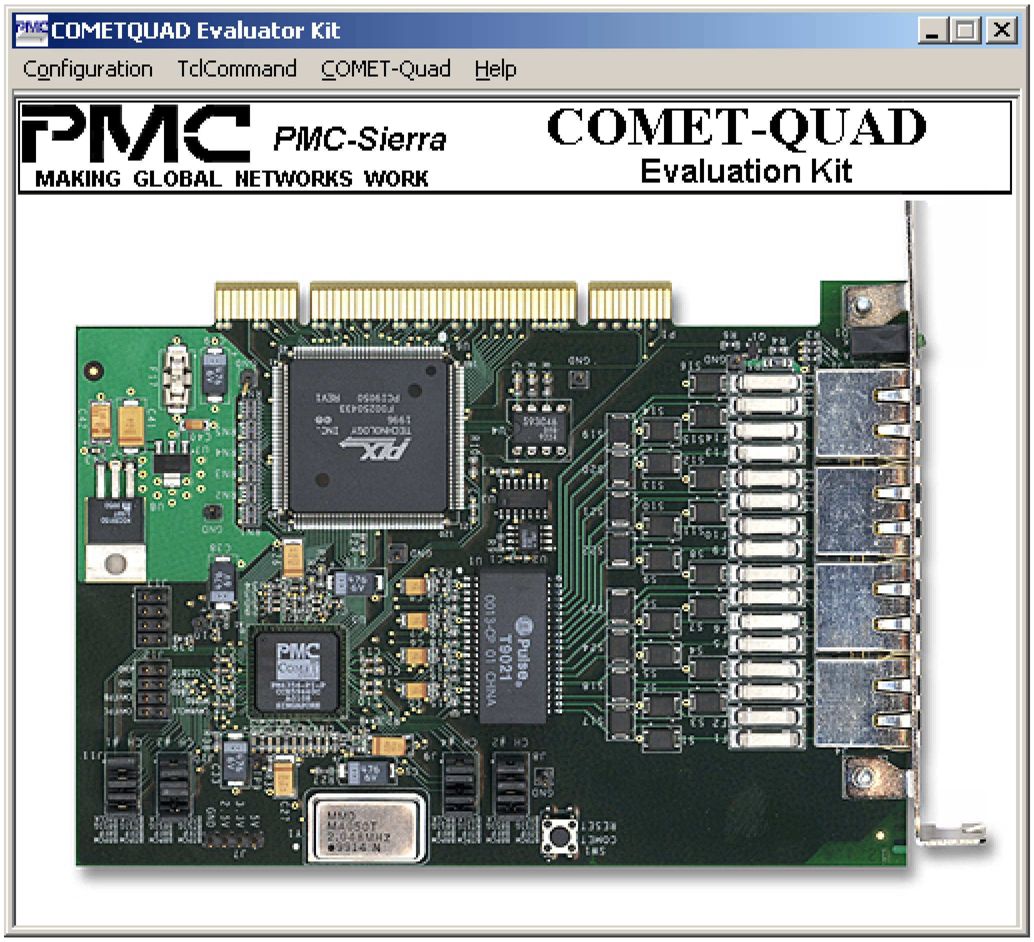
The graphic windows provide users easy access to the COMET-QUAD device. Additionally, the program provides interactive access to the COMET-QUAD datasheet via various window displays. This information is in the form of Windows Help. Help is obtained by a mouse left-click on a window control when the help cursor appears, or by accessing help from the main menu.

The cursor changes from an arrow icon to a hand icon to indicate that the mouse can be left-clicked at the current position. The hand icon indicates that an action will occur when the mouse is left-clicked. The action will depend on where the mouse cursor is positioned.

Some windows allow the user to select from a pop-up menu when the mouse is right-clicked. The operation of each window is discussed in the following sections.

4.1 Main Window

Figure 1 - Main Window Display



The main window display (Figure 1) is shown when the program starts. The display contains a main menu and a bitmap photo of the PCI add-in card.

A popup menu can be launched with a right-click anywhere in the main window display. This menu contains the same items as the "COMET-QUAD" main menu item.

The main menu allows the user to display detailed information concerning one of three areas:

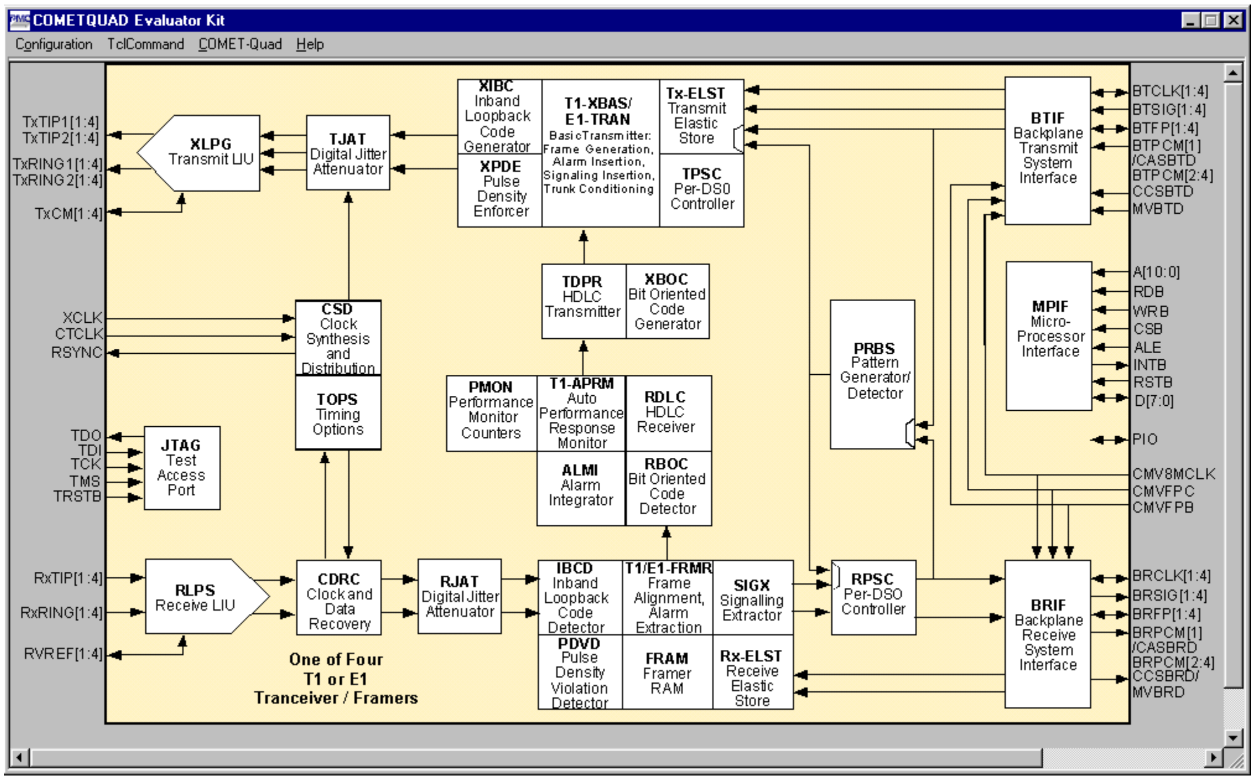
1. "Configuration" menu item provides selections to display board status and to exit the program

2. “TCL Command” menu item provides selections to launch the Tcl console, run Tcl scripts, and view Tcl log files.

3. "COMET-QUAD" menu item provides selections to launch detailed displays associated with the COMET-QUAD device: channel selection window, registers read/write window, alarm status window, interrupt status window, and HDLC statistic displays.

4. "Help" menu item provides selections to launch documentation concerning the COMET-QUAD, this program or the board.

Figure 2 - COMET-QUAD Block Diagram

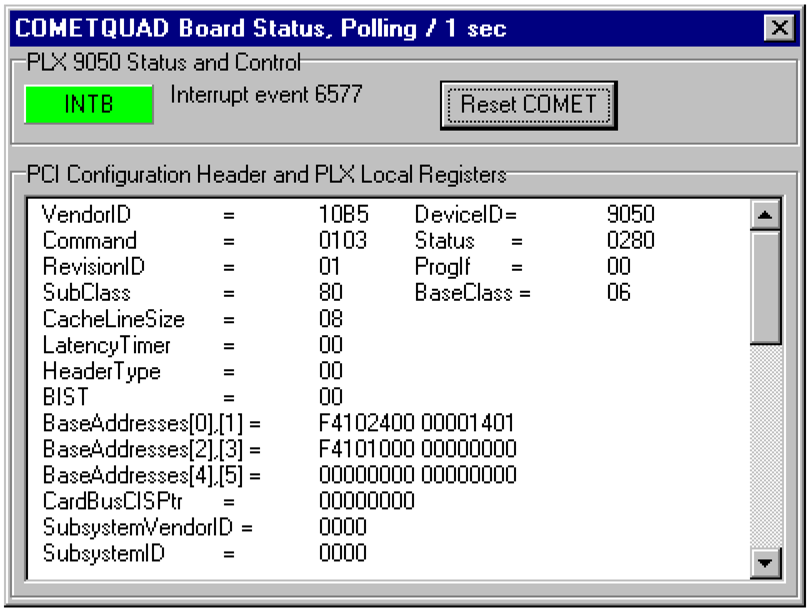

COMETQUAD Evaluator Kit
Configuration
TclCommand
COMET-Quad
Help
TxTIP1[1:4]
TxTIP2[1:4]
TxRING1[1:4]
TxRING2[1:4]
TxCM[1:4]
XLPG Transmit LIU
TJAT Digital Jitter Attenuator
XIBC Inband Loopback Code Generator
XPDE Pulse Density Enforcer
T1-XBAS/ E1-TRAN Basic Transmitter: Frame Generation, Alarm Insertion, Signaling Insertion, Trunk Conditioning
Tx-ELST Transmit Elastic Store
TPSC Per-DS0 Controller
BTIF Backplane Transmit System Interface
BTCLK[1:4]
BTSIG[1:4]
BTFP[1:4]
BTPCM[1] /CASBTD
BTPCM[2:4]
CCSBTD
MVBTD
XCLK
CTCLK
RSYNC
CSD Clock Synthesis and Distribution
TOPS Timing Options
TDPR HDLC Transmitter
XBOC Bit Oriented Code Generator
PRBS Pattern Generator/ Detector
MPIF Micro-Processor Interface
A[10:0]
RDB
WRB
CSB
ALE
INTB
RSTB
D[7:0]
PIO
CMV8MCLK
CMVFPC
CMVFPB
TDO
TDI
TCK
TMS
TRSTB
JTAG Test Access Port
PMON Performance Monitor Counters
T1-APRM Auto Performance Response Monitor
RDLC HDLC Receiver
ALMI Alarm Integrator
RBOC Bit Oriented Code Detector
RxTIP[1:4]
RxRING[1:4]
RVREF[1:4]
RLPS Receive LIU
CDRC Clock and Data Recovery
RJAT Digital Jitter Attenuator
IBCD Inband Loopback Code Detector
PDVD Pulse Density Violation Detector
T1/E1-FRMR Frame Alignment, Alarm Extraction
FRAM Framer RAM
SIGX Signalling Extractor
Rx-ELST Receive Elastic Store
RPSC Per-DS0 Controller
BRIF Backplane Receive System Interface
BRCLK[1:4]
BRSIG[1:4]
BRFP[1:4]
BRPCM[1] /CASBRD
BRPCM[2:4]
CCSBRD/ MVBRD
One of Four T1 or E1 Tranceiver / Framers

If the mouse is placed above the COMET-QUAD in the bitmap, the cursor will appear as a hand. A left-click on the COMET-QUAD will replace the bitmap with a COMET-QUAD block diagram (Figure 2). The same change can be performed by selecting "COMET-QUAD | Show Block Diagram" in the main menu, or on the right-click popup menu item.

When the block diagram is shown in the main window the user can move the mouse across the block diagram and left-click on a block, or pin name. When the hand cursor is left-clicked a popup menu allows the user to access a register associated with the block, or to display help information describing the block. The user can directly access help information describing a pin when the help cursor is left-clicked above the pin name.

4.2 COMET-QUAD Board Status

Figure 3 - Board Control and Status


COMETQUAD Board Status, Polling / 1 sec
PLX 9050 Status and Control
INTB
Reset COMET
PCI Configuration Header and PLX Local Registers

Interrupt event 6577

```
VendorID          =    10B5     DeviceID=      9050
Command           =    0103     Status    =    0280
RevisionID        =    01       ProgIf    =    00
SubClass          =    80       BaseClass =    06
CacheLineSize     =    08
LatencyTimer      =    00
HeaderType        =    00
BIST              =    00
BaseAddresses[0],[1] = F4102400 00001401
BaseAddresses[2],[3] = F4101000 00000000
BaseAddresses[4],[5] = 00000000 00000000
CardBusCISPtr     =    00000000
SubsystemVendorID =    0000
SubsystemID       =    0000
```

This display is launched from the main menu item "Configuration | Board Control ". It provides access to the PLX 9050 chip that bridges the PCI bus to the COMET-QUAD device.

4.2.1 PCI Configuration Header and PLX Local Registers Display

The PCI Configuration Header and PLX local register values are shown in the display window. The information is only retrieved once when the program starts.

The following provides some brief and useful information about these values.

- VendorID and DeviceId are used by the GUI to recognize the PCI card.
- BaseAddress[0] is the physical address of the PLX registers.

- BaseAddress[2] is the physical address of the COMET-QUAD registers
- InterruptLine is the IRQ number assigned by the OS.
- InterruptPin is the pin that is routed to the COMET-QUAD INTB pin.

Please refer to PLX9050 datasheet and PCI specification for more detailed information on these register values.

4.2.2 INTB button

When the program is started, a periodic one-second timer is activated. On timeout, the PLX9050 interrupt status register (INTCSR) is read to determine the state of the COMET-QUAD INTB pin. The local interrupt status bit (bit2) of the PLX9050 interrupt status register is read and displayed. When it is set, INTB pin is active and the INTB panel is displayed red in color. When the COMET-QUAD interrupt is cleared, INTB pin is inactive and the INTB panel is displayed green in color.

When this button is clicked, the COMET-QUAD INTB pin is routed to the PCI interrupt pin by asserting the PCI Interrupt Enable bit of the register INTCSR. The Interrupt Service Routine of the program is then associated with the COMET-QUAD INTB signal.

In this GUI design, three windows are added with interrupt capabilities along with original timer polling routines. Upon receiving an interrupt event, the ISR manually calls the timer timeout event of the following windows: Status window, Interrupt windows, and HDLC window. Enabling the PCI interrupt enhances the performance of the GUI to reflect almost real-time status of the COMET-QUAD registers.

The Title bar displays the mode of operation (Polling or Interrupt driven). A counter near the INTB button displays the number of interrupt serving attempts.

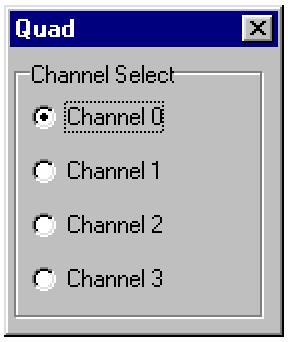
NOTE: if the INTB can't be cleared, the counter accumulates really fast and endless running of the interrupt routine can decrease computer performance dramatically.

4.2.3 Reset button

The "Reset" button instructs the PLX 9050 chip to toggle the COMET-QUAD RSTB pin. This is a hardware reset of the COMET-QUAD. Pressing the reset button on the board also causes register values to be reset to their default values.

4.3 Channel Select

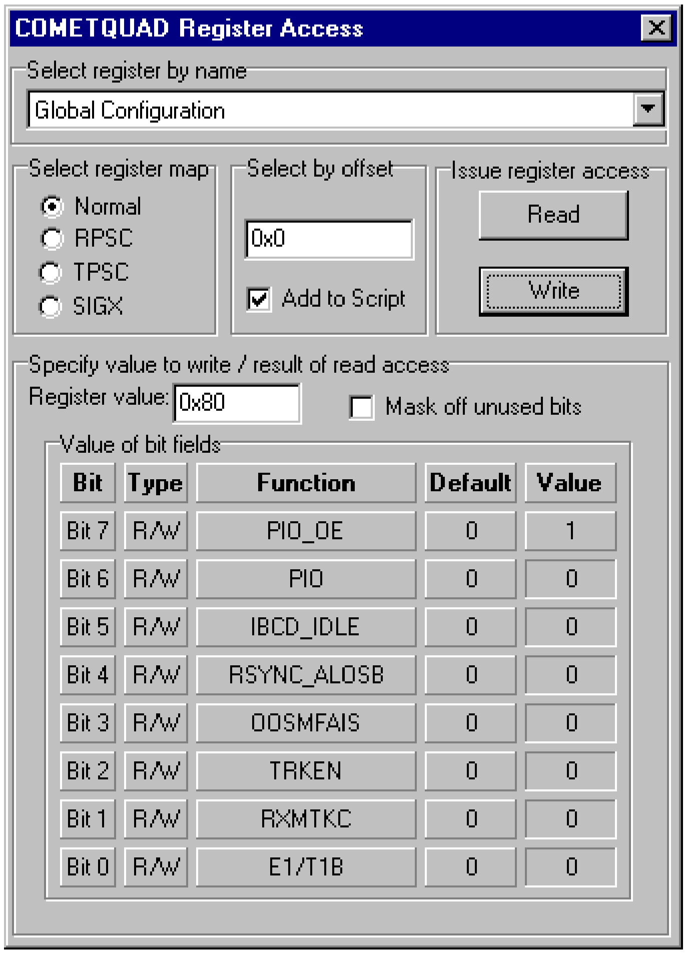
Figure 4 - Channel Select



This window is launched from the main menu under COMET-Quad or by right-clicking the "COMETQUAD Channel Select" menu item. This window provides a means for windows to display different COMET-QUAD channels. The change in channels affects three windows: Status window, Interrupt windows, and Register Access window. The title bar in these windows indicates the current channel of operation.

4.4 COMET-QUAD Register Access

Figure 5 - Register Access



Global Configuration

Select register map: [✓ Normal] RPSC TPSC SIGX

Select by offset: 0x0

[✓] Add to Script

Register value: 0x80

[ ] Mask off unused bits

Bit	Type	Function	Default	Value
Bit 7	R/W	PIO_OE	0	1
Bit 6	R/W	PIO	0	0
Bit 5	R/W	IBCD_IDLE	0	0
Bit 4	R/W	RSYNC_ALOSB	0	0
Bit 3	R/W	OOSMFAIS	0	0
Bit 2	R/W	TRKEN	0	0
Bit 1	R/W	RXMTKC	0	0
Bit 0	R/W	E1/T1B	0	0

The COMET-QUAD registers can be directly accessed via the register access window. This window is launched from the main menu under COMET-Quad or by right-clicking the "COMET-QUAD | Register Access" menu item.

This display has a "Read" and a "Write" button that provides read or write access to the COMET-QUAD device. A register is selected by left-clicking one of the register spaces of the COMET-QUAD - either normal, RPSC, TPSC or SIGX. Then the register is selected by name in the pull down list, or by directly typing in the register offset box.

The register value can be viewed after the "Read" button is left-clicked, or alternatively the user can specify a register value to write.

Within the control titled "Value of bit fields", clicking on one of the bit panels in the Value column toggles that value bit. Clicking on the Value panel clears all value bits. Clicking on the Default panel sets the value to the register default. Alternatively the user can specify the value directly in the control titled "Register value". Once the value is specified, the user can issue a write access to the COMET-QUAD by left-clicking the "Write" button.

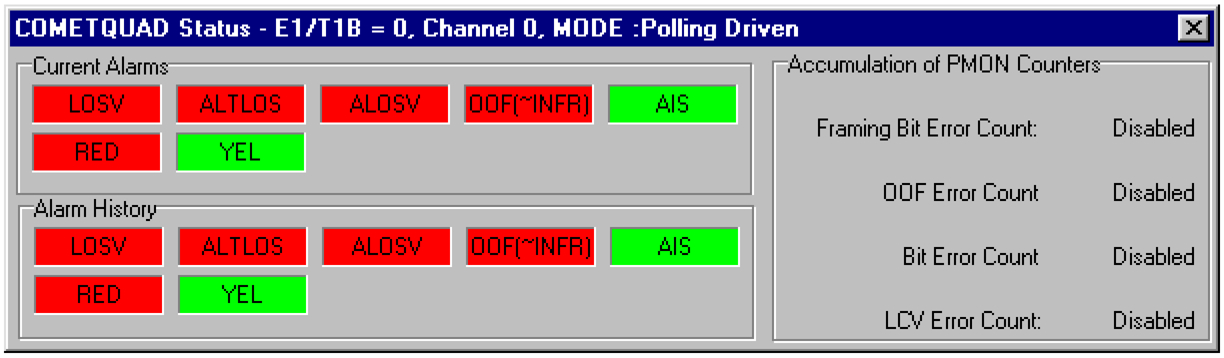
The name of each bit field for the register offset is shown in this window. If the user moves the mouse over any of the bit names, the cursor changes to a help icon. Left-clicking at this time launches context sensitive help for the register display.

4.4.1 Add to Script Check Box

This check box provides users a way to record direct or indirect read/write accesses from the GUI. When the check box is clicked, the script, “tmpScript.tcl” is opened. Read and Write accesses from clicking window buttons are automatically appended to the script. NOTE that the command “INITIALIZE” must be added to the fist line of the script so that COMET-QUAD driver api can be run. In addition, “UPDATELOGFILE” must be added to the end of the script to update the output log file.

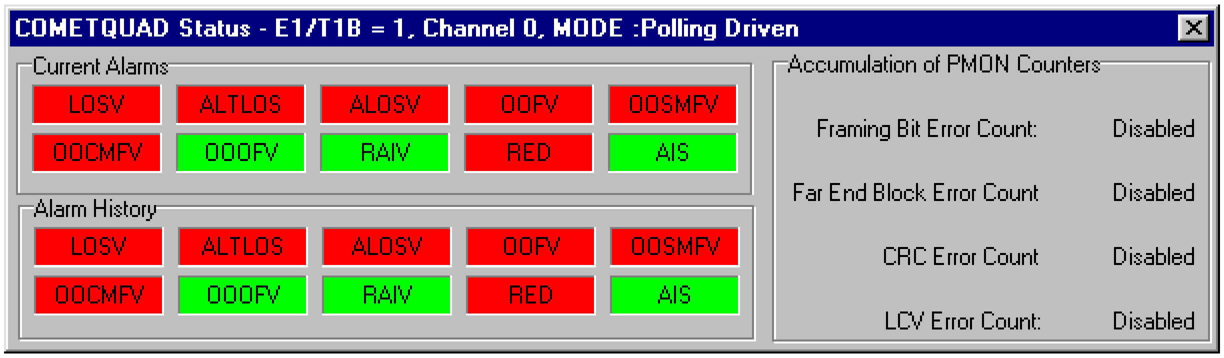
4.5 COMET-QUAD Status

Figure 6 - T1 Status



Framing Bit Error Count: Disabled
OOF Error Count Disabled
Bit Error Count Disabled
LCV Error Count: Disabled

Figure 7 - E1 Status


COMETQUAD Status - E1/T1B = 1, Channel 0, MODE :Polling Driven

Current Alarms

🟥 LOSV	🟥 ALTLOS	🟥 ALOSV	🟥 OOFV	🟥 OOSMFV
🟥 OOCMFV	🟩 OOOFV	🟩 RAIV	🟥 RED	🟩 AIS

Alarm History

🟥 LOSV	🟥 ALTLOS	🟥 ALOSV	🟥 OOFV	🟥 OOSMFV
🟥 OOCMFV	🟩 OOOFV	🟩 RAIV	🟥 RED	🟩 AIS

Accumulation of PMON Counters

Framing Bit Error Count:	Disabled
Far End Block Error Count	Disabled
CRC Error Count	Disabled
LCV Error Count:	Disabled

The COMET-QUAD status window is launched from the main menu item "COMET-QUAD | Status".

This window appears differently for T1 configurations than for E1 configurations. In both cases, it shows the status of some alarms and the accumulated count of the performance monitor error counters. The status window determines whether the configuration is an E1 or T1 application by reading the E1/T1Bit from the register value database.

The alarm status is shown as a red color when the associated status bit value is set. It is shown as a green color when the associated status bit value is clear. Status bit values are taken from the register value database instead of the COMET-QUAD registers. Alarm panels that correspond with database values that have not been updated (invalid values) are shown as grey.

Each alarm status has a current and a history panel. A right-click of the mouse enables the user to select a menu item from a popup menu. Selecting "Clear History" turns all history panels to grey. The next timeout updates the register value database (during a register access) and that causes the alarm and history panels to change to either red or green.

The “Enable/Disable Register Access” menu item of the popup menu can be selected to enable, or disable, register access. Register reads are performed periodically (1 seconds) to update the register value database for alarms and history when the window is visible. Choosing “Disable Register Access” stops the periodic register read accesses. When register access is disabled the alarm text heading includes the text “Disabled Register Access”, and the PMON counters show the text “disabled”. PMON counters are not accumulated when register access is disabled but they are accumulated when the window is not visible.

The following tables show the registers and bits that are read for status display

Table 4 - T1 Status Window Description

Status	**"LOSV"**	**"ALTLOS"**	**"ALOSV"**	**"OOF (~INFR)"**	**"AIS"** **"RED"** **"YEL"**
Register	0xQ12	0xQ13	0xQF8	0xQ4A	0xQ62
Bit(s)	bit0	bit0	bit6	bit0	bit0 bit1 bit2

Table 5 - E1 Status Window Description

Status	**"LOSV"**	**"ALTLOS"**	**"ALOSV"**	**"OOFV"** **"OOSMFV"** **"OOCMFV"** **"OOOFV"**	**"RAIV"** **"RED"** **"AIS"**
Register	0xQ12	0xQ13	0xQF8	0xQ96	0xQ97
Bit(s)	bit0	bit0	bit6	bit6 bit5 bit4 bit3	bit7 bit3 bit2

The performance monitor counters shown in this display are an accumulation of the PMON counters within the COMET-QUAD device. The polling of counters can be enabled/disabled by a right-click to launch the popup menu and a left-click on the "specify counter polling interval" menu item. A non-zero polling value causes the PMON counter registers to be read periodically. A zero value disables polling of the PMON counters. At each poll interval register 0xQ59 is written to transfer the internal counts to the visible registers. Then each register is read and added to the displayed value. PMON counter accumulations are also disabled if the user chooses to disable register access. When the accumulations are disabled the counter value is not displayed. Instead the text "disabled" is displayed.

The accumulated counts shown in the window can be cleared by selecting "Clear Counters" from the popup menu.

For T1 applications the following PMON counters are shown in this window:

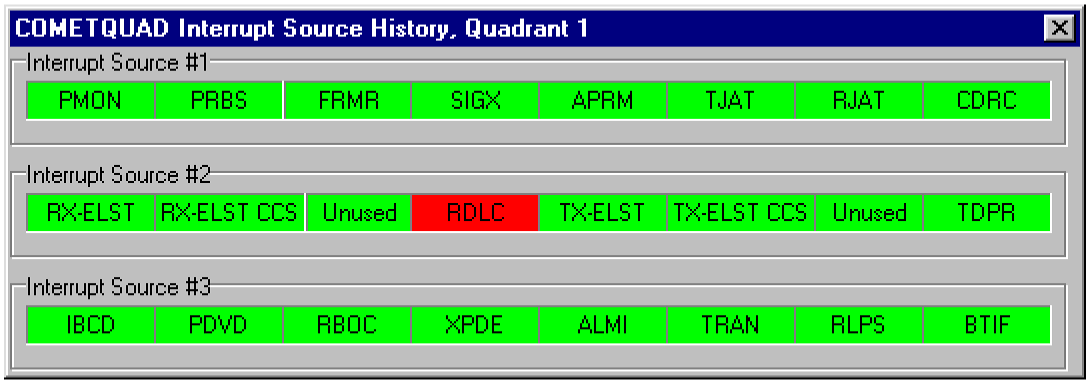
- Framing Bit Error Count
- OOF Error Count if the CCOFA bit of the receive options register is clear. Otherwise it's the COFA Error Count.
- Bit Error Count
- LCV Error Count

For E1 applications the following PMON counters are shown in this window:

- Framing Bit Error Count
- Far End Block Error Count
- CRC Error Count
- LCV Error Count

4.6 COMET-QUAD Interrupt Sources

Figure 8 - Interrupt Source History


COMETQUAD Interrupt Source History, Quadrant 1

Interrupt Source #1

PMON	PRBS	FRMR	SIGX	APRM	TJAT	RJAT	CDRC

Interrupt Source #2

RX-ELST	RX-ELST CCS	Unused	RDLC	TX-ELST	TX-ELST CCS	Unused	TDPR

Interrupt Source #3

IBCD	PDVD	RBOC	XPDE	ALMI	TRAN	RLPS	BTIF

The COMET-QUAD Interrupt Sources window is launched by selecting the main menu item "COMET-QUAD | Interrupts".

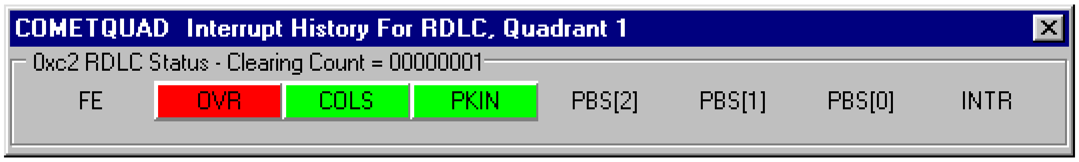
The window shows the status of all interrupt sources. Panels representing bits of the three registers that are the source of all interrupts can be colored red, green or grey. A grey panel indicates that the interrupt source bit has not been read and updated in the register value database. A red color indicates an active interrupt the last time this source bit was read. A green color indicates the interrupt source was not active the last time this source bit was read.

The cursor appears as a help icon when it is moved around the boundary line of a group box on this display. A left-click with the help icon launches register specific help.

The cursor appears as a hand icon when it is moved across bit panels in this window. If the user left-clicks with the hand icon then an Interrupt Status History window for the selected source bit panel is displayed.

4.7 COMET-QUAD Interrupt Status History

Figure 9 - Interrupt Status History Display


COMETQUAD Interrupt History For RDLC, Quadrant 1

0xc2 RDLC Status - Clearing Count = 00000001

The COMET-QUAD Interrupt Status History window is launched with a left-click on a source bit of the COMET-QUAD Interrupt Sources window. Each source bit represents a COMET-QUAD block.

The window shows the status history of each interrupt status bit associated with the selected COMET-QUAD block. Status bits are green if all previous accesses to the register had an inactive (clear) bit. Status bits are red if one or more previous accesses to the register had an active (set) bit. The most current status for the bits within a register is obtained by a right-click on the status bit to launch a pop-up menu. If the menu item "Read register and clear status" is selected then the COMET-QUAD register is read, status bits are cleared, and the register value database is updated with the new register status/history.

Interrupt status history is cleared from the display by a right-click on the display to launch a popup menu, and choosing the "Clear Register History" menu item. Status bits are grey if the register has not yet been read from the COMET-QUAD.

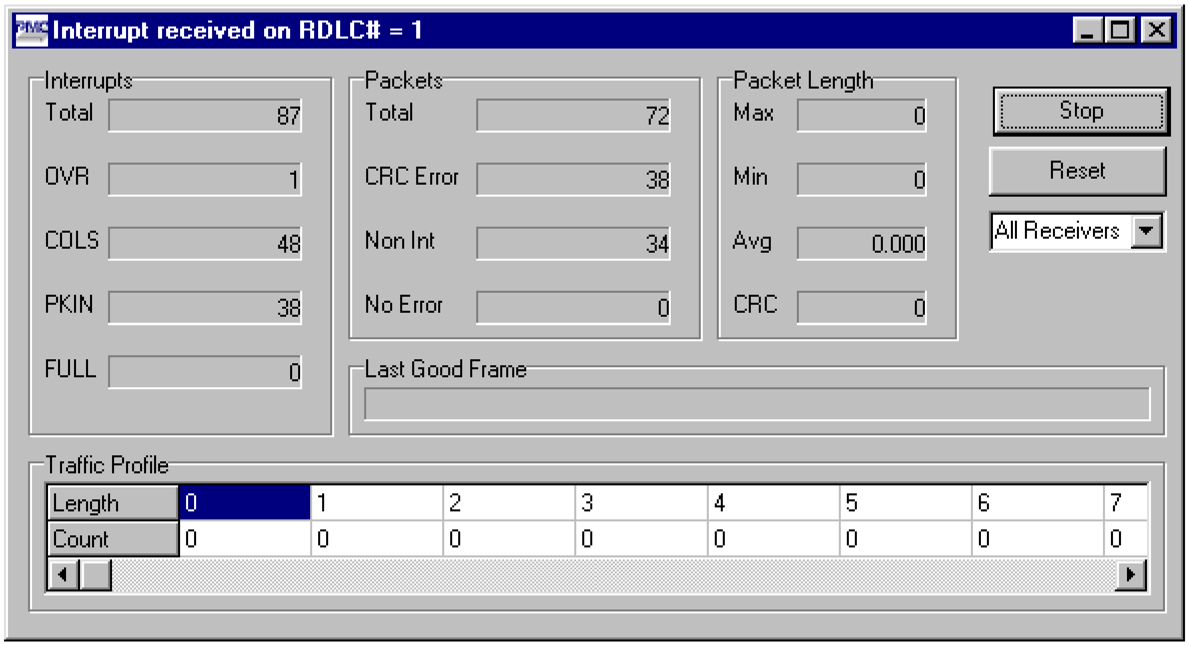
A clearing count is provided beside the name of each register. The clearing count increments by one when the register is read, and one or more of the interrupt status bits are active.

A depressed panel indicates the status bit is enabled - that it may activate the INTB pin if the status bit is active (red). A panel which appears to stick out of the window, indicates the status bit is disabled - that it could not activate the INTB pin when the status bit is active (red). A hand cursor appears over a status bit and allows the user to enable/disable the individual interrupts associated with the status bits.

When the cursor is moved across the window it may appear as a help icon and can be left-clicked to launch help on the register underneath the cursor.

4.8 HDLC Window

Figure 10 - HDLC Display



Interrupts

Total	87
OVR	1
COLS	48
PKIN	38
FULL	0

Packets

Total	72
CRC Error	38
Non Int	34
No Error	0

Packet Length

Max	0
Min	0
Avg	0.000
CRC	0

Last Good Frame

Traffic Profile

Length	0	1	2	3	4	5	6	7
Count	0	0	0	0	0	0	0	0

The COMET-QUAD HDLC window is launched by selecting the main menu item "COMET-QUAD | HDLC ".

This GUI demonstrates the RDLC receiver Block of the COMET-QUAD. Each quadrant of COMET-QUAD has an HDLC controller. The user can select any of the four controllers to monitor from the Pull-down list.

When the start button is clicked, a thread that runs continuously in a loop starts to poll the INTR bit of the RDLC Status registers (0xQC2). If INTR bit (bit0) is asserted, the program reads the status registers and RDLC Data FIFO. The data collected is then organized and stored in the database. The GUI display uses a 0.1s timer to poll the database and display the statistics.

When the Stop button is clicked, the thread is blocked and stopped to collect statistics. When the Reset button is clicked, the database is reset to 0 for all statistics.

5 TCL COMMANDS

The Tcl commands are derived based on the COMET-QUAD device driver APIs. This section describes the Tcl commands and input parameters that can be used to exercise to the driver APIs. It is advised that the user refers to the COMET-QUAD Device Driver Design Specification and COMET-QUAD Device Driver User's Guide when using the Tcl script commands.

The Tcl interface to the driver API is built to familiarize the user with the driver architecture. Therefore, each Tcl command is almost built for one-to-one mapping to the actual API calls. There are only a few exceptions.

1. The input deviceHandle is omitted in the command
2. The output pointers of the input command are omitted.
3. Some Tcl commands (APIs) can only be used in certain device states. Please refer to COMET-QUAD Device Driver User's Guide for each Tcl command(API)'s valid states.
4. The device state can be manipulated using Device Management commands: "cometqInit", "cometqActivate", "cometqReset", "cometqDeActivate"
5. The output of the API is echoed on the Tcl console. It is also written to the file "outfile.txt"

- <> brackets around a field imply that the textual representation of a number or character strings must be entered in the field.
- [ ] brackets around a field, or fields, imply that the field(s) is optional.

Note that in Tcl script language, { } bracket is used to group a list of inputs into one field. Therefore, when an input field, such as a pointer to a block, requires multiple entries, { } is used to group the block.

5.1 Device Management

The following APIs are executed

INITIALIZATION

*Note: INITIALIZATION sets the device to T1 mode by default.

To change the operating mode to E1, user needs to reset (cometqReset) and re-initialize (cometqInit) and then call “cometqSetOperatingMode”, followed by reactivation (cometqActivate).

cometqModuleOpen

cometqModuleStart

cometqAdd

cometqInit

cometqActivate

CLOSE

cometqDeActivate

cometqDelete

cometqModuleStop

cometqModuleClose

UPDATELOGFILE

This command closes the filestream and writes the the up-to-date Tcl outputs to the “outfile.txt”

5.1.1 cometqInit

Description	`Initialize comet-Quad device to default setting.`
	`( with input field pdiv = NULL, profileNum = 0 )`
	`Change device state from “Present” to “Inactive”.`

Map to API	`INT4 cometqInit(sCMQ_HNDL deviceHandle, sCMQ_DIV *pdiv, UINT2 profileNum);`
Input field(s)	`none`
Output	`none if success`
Example	`cometqInit`

5.1.2 cometqReset

Description	Applies a software reset to the COMET or COMET-Quad device. Also resets all the DDB contents. Change device state from "Inactive" or "Active" to "Present".
Map to API	`INT4 cometqReset(sCMQ_HNDL deviceHandle);`
Input field(s)	none
Output	none if success
Example	cometqReset

5.1.3 cometqActivate

Description	The cometqActivate API activates the device by enabling the interrupt if the device is in ISR mode. Change device state from "Inactive" to "Active"
Map to API	`INT4 cometqActivate(sCMQ_HNDL deviceHandle);`
Input field(s)	none
Output	none if success
Example	cometqActivate

5.1.4 cometqDeActivate

Description	The cometqDeActivate API Deactivates the device by disabling interrupts if the device is in ISR mode and retaining interrupt mask within the DDB. Change device state from "Active" to "Inactive"
Map to API	`INT4 cometqDeActivate(sCMQ_HNDL deviceHandle);`
Input field(s)	none
Output	none if success
Example	cometqDeActivate

5.2 Device Read and Write

5.2.1 cometqRead <regNum>

Description	Reading from the Device Registers.
Map to API	`UINT1 cometqRead(sCMQ_HNDL deviceHandle, UINT2 regNum)`

Input field(s)	`<regNum> any number from 0x0 to 0x3ff`
Output	`print the Register value if success`
Example	`cometqRead 0x100`

5.2.2 cometqWrite <regNum> <value>

Description	`Writing to Device Registers.`
Map to API	`UINT1 cometqWrite(sCMQ_HNDL deviceHandle, UINT2 regNum, UINT1 value)`

Input field(s)	`<regNum> any number from 0x0 to 0x3ff`
	`<value> any number from 0x0 to 0xff`

Output	`none if success`
Example	`cometqWrite 0x100 0x1`

5.2.3 cometqReadBlock <startRegNum> <size>

Description	`Reading from a block of Device Registers`
Map to API	`UINT1 cometqReadBlock(sCMQ_HNDL deviceHandle, UINT2 startRegNum, UINT2 size, UINT1 *pblock)`

Input field(s)	`<startRegNum> any number from 0x0 to 0x3ff`
	`<size> any number from 0x0 to 0x3ff`

Output	`print the Block of Register values if success`
Example	`cometqReadBlock 0 5`

5.2.4 cometqWriteBlock <startRegNum> <size> <block> <mask>

Description	`Writing to a Block of Device Registers`
Map to API	`UINT1 cometqWriteBlock(sCMQ_HNDL deviceHandle, UINT2 startRegNum, UINT2 size, UINT1 *pblock, UINT1 *pmask)`

Input field(s)	`<startRegNum> any number from 0x0 to 0x3ff`
	`<size> any number from 0x0 to 0x3ff`
	`<block> a set of numbers from 0x0 to 0xff`
	`<mask> a set of numbers from 0x0 to 0xff`

Output	`none if success`
Example	`cometqWriteBlock 0 5 {1 2 3 4 5} {0xff 0 0xff 0xff 0}`

5.2.5 cometqReadFr <frmNum> <regNum>

Description	`Reading from Framer Device Registers:`
Map to API	`UINT1 cometqReadFr(sCMQ_HNDL deviceHandle, UINT1 frmNum, UINT2 regNum)`

Input field(s)	`<frmNum> any number from 0 to 3`
	`<regNum> any number from 0x0 to 0xff`

Output	`print the Register value if success`
Example	`cometqReadFr 0 0x1`

5.2.6 cometqWriteFr <frmNum> <regNum> <value>

Description	`Writing to Framer Device Registers`
Map to API	`UINT1 cometqWriteFr(sCMQ_HNDL deviceHandle, UINT1 frmNum, UINT2 regNum, UINT1 value)`

Input field(s)	`<frmNum> any number from 0 to 3`
	`<regNum> any number from 0x0 to 0xff`
	`<value> any number from 0x0 to 0xff`

Output	`none if success`
Example	`cometqWriteFr 0 0x1 0xff`

5.2.7 cometqReadFrInd <frmNum> <section> <regNum>

Description	`Reading from Device Indirect Registers`
Map to API	`UINT1 cometqReadFrInd(sCMQ_HNDL deviceHandle, UINT1 frmNum, eCMQ_SECTION section, UINT2 regNum)`

Input field(s)	`<frmNum> any number from 0 to 3`
	`<section> CMQ_SIGX_SECT or 0,`
	`          CMQ_TPSC_SECT or 1,`
	`          CMQ_RPSC_SECT or 2,`
	`          CMQ_XLPG_SECT or 3`
	`<regNum> any number in the valid address range`

Output	`print the Register value if success`
Example	`cometqReadFrInd 0 CMQ_SIGX_SECT 0x20`

5.2.8 cometqWriteFrInd <frmNum> <section> <regNum> <value>

Description	`Writing to Device Indirect Registers`
Map to API	`UINT1 cometqWriteFrInd(sCMQ_HNDL deviceHandle, UINT1 frmNum eCMQ SECTION section, UINT2 regNum, UINT1`

	value)

Input field(s)	`<frmNum> any number from 0 to 3`
	`<section> CMQ_SIGX_SECT or 0,`
	`CMQ_TPSC_SECT or 1,`
	`CMQ_RPSC_SECT or 2,`
	`CMQ_XLPG_SECT or 3`
	`<regNum> any number in the valid address range`
	`<value> any number from 0x0 to 0xff`

Output	none if success
Example	cometqWriteFrInd 0 CMQ_SIGX_SECT 0x20 0x1

5.2.9 cometqReadRLPS <frmNum> <regNum>

Description	Reading from Device RLPS Indirect Registers
Map to API	UINT1 cometqReadRLPS(sCMQ_HNDL deviceHandle, UINT2 frmNum,UINT1 regNum)

Input field(s)	`<frmNum> any number from 0 to 3`
	`<regNum> any number from 0x0 to 0xff`

Output	print the Register value if success
Example	cometqReadRLPS 0 0x1

5.2.10 cometqWriteRLPS <frmNum> <regNum> <value>

Description	Writing to Device RLPS Indirect Registers
Map to API	UINT4 cometqWriteRLPS(sCMQ_HNDL deviceHandle, UINT2 frmNum, UINT1 regNum, UINT4 value)

Input field(s)	`<frmNum> any number from 0 to 3`
	`<regNum> any number from 0x0 to 0xff`
	`<value> any number from 0x0 to 0xff`

Output	none if success
Example	cometqWriteRLPS 0 0x1 0xff

5.3 Line Side Interface

5.3.1 cometqLineTxEncodeCfg <Chan> <encScheme>

Description	Configure transmit line encoding

Map to API	`INT4 cometqLineTxEncodeCfg(sCMQ_HNDL deviceHandle, UINT2 chan, eCMQ_LINE_CODE encScheme);`

Input field(s)	`<Chan> any number from 0 to 3`
	`<encScheme> CMQ_LINE_CODE_AMI or 0,`
	`CMQ_LINE_CODE_HDB3_E1 or 1,`
	`CMQ_LINE_CODE_B8ZS_T1 or 2`

Output	`none if success`
Example	`cometqLineTxEncodeCfg 1 1`

5.3.2 cometqLineRxEncodeCfg <Chan> <encScheme> <incXSZeros> <E1_O162En>

Description	`Configure receive line encoding`
Map to API	`INT4 cometqLineRxEncodeCfg(sCMQ_HNDL deviceHandle, UINT2 chan, eCMQ_LINE_CODE encScheme, UINT1 incXSZeros, UINT1 E1_O162En);`

Input field(s)	`<Chan> any number from 0 to 3`
	`<encScheme> CMQ_LINE_CODE_AMI or 0,`
	`CMQ_LINE_CODE_HDB3_E1 or 1,`
	`CMQ_LINE_CODE_B8ZS_T1 or 2`
	`< incXSZeros> 0 or 1`
	`< E1_O162En > 0 or 1`

Output	`none if success`
Example	`cometqLineRxEncodeCfg 0 1 0 1`

5.3.3 cometqLineTxAnalogCfg <Chan> < sCMQ_CFG_TX_ANALOG >

Description	`Configure analog transmit line`
Map to API	`INT4 cometqLineTxAnalogCfg(sCMQ_HNDL deviceHandle, UINT2 chan, sCMQ_CFG_TX_ANALOG* ptxAnalogCfg);`

Input field(s)	`<Chan> any number from 0 to 3`
	`< sCMQ_CFG_TX_ANALOG > =`
	`{ <txEn> <wvFormType> <wvFormScFac> { <wvFormData> } <fuseDataSel> <alogTstPosCtrl> <alogTstNegCtrl> }`
	`<txEn>: 0 or 1`
	`<wvFormType>: CMQ_TX_LBO_T1_LONG_HAUL_0DB or 0,`
	`CMQ_TX_LBO_T1_LONG_HAUL_7_5DB or 1,`
	`CMQ_TX_LBO_T1_LONG_HAUL_15DB or 2,`
	`CMQ_TX_LBO_T1_LONG_HAUL_22_5DB or 3,`
	`CMQ_TX_LBO_T1_LONG_HAUL_TR62411_0DB or 4,`
	`CMQ_TX_LBO_T1_SHORT_HAUL_110FT or 5,`
	`CMQ_TX_LBO_T1_SHORT_HAUL_220FT or 6,`

	CMQ_TX_LBO_T1_SHORT_HAUL_330FT or 7, CMQ_TX_LBO_T1_SHORT_HAUL_440FT or 8, CMQ_TX_LBO_T1_SHORT_HAUL_550FT or 9, CMQ_TX_LBO_T1_SHORT_HAUL_660FT or 10, CMQ_TX_LBO_T1_SHORT_HAUL_TR62411_110FT or 11, CMQ_TX_LBO_T1_SHORT_HAUL_TR62411_220FT or 12, CMQ_TX_LBO_T1_SHORT_HAUL_TR62411_330FT or 13, CMQ_TX_LBO_T1_SHORT_HAUL_TR62411_440FT or 14, CMQ_TX_LBO_T1_SHORT_HAUL_TR62411_550FT or 15, CMQ_TX_LBO_T1_SHORT_HAUL_TR62411_660FT or 16, CMQ_TX_LBO_E1_75OHM or 17, CMQ_TX_LBO_E1_120OHM or 18, CMQ_TX_LBO_USER_DEFINED or 19, CMQ_TX_LBO_RETAIN_CURRENT or 20 <wvFormScFac>: Scale factor <wvFormData>: a [24][5] array Note: This array entry is only needed when wvFormType = CMQ_TX_LBO_USER_DEFINED <fuseDataSel>: CMQ_TX_FUSE_DATA_TST_CTL or 0 CMQ_TX_FUSE_DATA_ENABLE or 1 CMQ_TX_FUSE_DATA_MAX or 2
Output	none if success
Example 1	cometqLineTxAnalogCfg 0 \\ { 1 CMQ_TX_LBO_USER_DEFINED 0xC \\ {{0 0x1f 0x16 0x6 0x1} \\ {0 0x20 0x15 0x5 0x2} \\ {0 0x20 0x15 0x5 0x3} \\ {0 0x20 0x15 0x5 0x4} \\ {0 0x20 0x15 0x5 0x5} \\ {0 0x20 0x15 0x5 0x6} \\ {0 0x20 0x15 0x5 0x7} \\ {0 0x20 0x15 0x5 0x8} \\ {0 0x20 0x15 0x5 0x9} \\ {0 0x20 0x15 0x5 0xa} \\ {0 0x20 0x15 0x5 0xb} \\ {0 0x20 0x15 0x5 0xc} \\ {0 0x1f 0x16 0x6 0xd} \\ {0 0x20 0x15 0x5 0xe} \\ {0 0x20 0x15 0x5 0xf} \\ {0 0x20 0x15 0x5 0x10} \\ {0 0x20 0x15 0x5 0x11} \\ {0 0x20 0x15 0x5 0x12} \\ {0 0x20 0x15 0x5 0x13} \\ {0 0x20 0x15 0x5 0x14} \\ {0 0x20 0x15 0x5 0x15} \\ {0 0x20 0x15 0x5 0x16} \\

	{0 0x20 0x15 0x5 0x17} \ {0 0x20 0x15 0x5 0x18}} \ 1 1 1 }
Example 2	cometqLineTxAnalogCfg 0 { 1 0 0xC 1 1 1 }

5.3.4 cometqLineRxAnalogCfg <Chan> < sCMQ_CFG_RX_ANALOG >

Description	Configure analog receive line
Map to API	INT4 cometqLineRxAnalogCfg(sCMQ_HNDL deviceHandle, UINT2 chan, sCMQ_CFG_RX_ANALOG *prxAnalogCfg);
Input field(s)	<Chan> any number from 0 to 3 < sCMQ_CFG_RX_ANALOG > = { < aLosThreshold > <aLosDetectPeriod> <aLosClearPeriod> < eqFreq> < eqFdBckPer> <ramType> { <eqCoef> } <squelchEn> } < aLosThreshold >: CMQ_RX_ALOS_9DB_THRESH or 0, CMQ_RX_ALOS_14_5DB_THRESH or 1, CMQ_RX_ALOS_20DB_THRESH or 2, CMQ_RX_ALOS_22DB_THRESH or 3, CMQ_RX_ALOS_25DB_THRESH or 4, CMQ_RX_ALOS_30DB_THRESH or 5, CMQ_RX_ALOS_31DB_THRESH or 6, CMQ_RX_ALOS_35DB_THRESH or 7 <eqFreq>: CMQ_RX_EQ_FREQ_T1_24_125KHZ or 0, CMQ_RX_EQ_FREQ_T1_12_063KHZ or 1, CMQ_RX_EQ_FREQ_T1_8_0417KHZ or 2, CMQ_RX_EQ_FREQ_T1_6_0313KHZ or 3, CMQ_RX_EQ_FREQ_T1_4_8250KHZ or 4, CMQ_RX_EQ_FREQ_T1_4_0208KHZ or 5, CMQ_RX_EQ_FREQ_T1_3_4464KHZ or 6, CMQ_RX_EQ_FREQ_T1_3_0156KHZ or 7, CMQ_RX_EQ_FREQ_E1_32_000KHZ or 8, CMQ_RX_EQ_FREQ_E1_16_000KHZ or 9, CMQ_RX_EQ_FREQ_E1_10_667KHZ or 10, CMQ_RX_EQ_FREQ_E1_8_000KHZ or 11, CMQ_RX_EQ_FREQ_E1_6_40KHZ or 12, CMQ_RX_EQ_FREQ_E1_5_333KHZ or 13, CMQ_RX_EQ_FREQ_E1_4_5714KHZ or 14, CMQ_RX_EQ_FREQ_E1_4_0KHZ or 15 <eqFdBckPer>: CMQ_RX_EQ_VALID_PERIOD_32 or 0, CMQ_RX_EQ_VALID_PERIOD_64 or 1,

	CMQ_RX_EQ_VALID_PERIOD_128 or 2, CMQ_RX_EQ_VALID_PERIOD_256 or 3 <ramType>: CMQ_RX_LINE_EQ_RAM_T1 or 0, CMQ_RX_LINE_EQ_RAM_E1 or 1, CMQ_RX_LINE_EQ_USER_DEFINED or 2, CMQ_RX_LINE_EQ_RETAIN_CURRENT or 3 <eqCoef>: a [256] array Note: This array entry is only needed when ramType = CMQ_RX_LINE_EQ_USER_DEFINED. <squelchEn>: 0 or 1
Output	none if success

Example 1	cometqLineRxAnalogCfg 0 \\
	{ 1 3 3 0 0 \\
	 CMQ_RX_LINE_EQ_USER_DEFINED \\
	 { 0x0 0x0 0x0 0x0 0x0 0x0 0x0 0x0 0x0 0x0\\
	   0x0 0x0 0x0 0x0 0x0 0x0 0x0 0x0 0x0 0x0\\
	   . . .
	   . . .
	   . . . (256 entries) }\\
	   1 }

Example 2	cometqLineRxAnalogCfg 0 { 1 3 3 0 0 0 1 }

5.3.5 cometqLineTxJatCfg <Chan> < sCMQ_CFG_TX_JAT >

Description	Configure transmit line jitter attenuation
Map to API	INT4 cometqLineTxJatCfg(sCMQ_HNDL deviceHandle, UINT2 chan, sCMQ_CFG_TX_JAT *ptxJatCfg);
Input field(s)	<Chan> any number from 0 to 3 < sCMQ_CFG_TX_JAT > = { < enable > < refDiv > < outputDiv > <FIFOselfCenter> < preventOvfUndf > < outputClock > <pllRefClock > } < enable >: 0 or 1 < refDiv >: a ratio (refer to data sheet) < outputDiv >: a ratio (refer to data sheet) < FIFOselfCenter >: 0 or 1 < preventOvfUndf >: 0 or 1 < outputClock >: CMQ_TJAT_OUTPUT_CLK_INTERN_JAT or 0, CMQ_TJAT_OUTPUT_CLK_CTCLK or 1, CMQ_TJAT_OUTPUT_CLK_FIFO_INPUT or 2 < pllRefClock >: CMQ_TJAT_PLL_REF_CLK_FIFO_INPUT or 0,

	CMQ_TJAT_PLL_REF_CLK_BACKPLANE or 1, CMQ_TJAT_PLL_REF_CLK_RECOVERED or 2, CMQ_TJAT_PLL_REF_CLK_CTCLK or 3
Output	none if success
Example	cometqLineTxJatCfg 0 { 1 1 1 1 1 0 0 }

5.3.6 cometqLineRxJatCfg <Chan> < sCMQ_CFG_RX_JAT >

Description	Configure receive line jitter attenuation
Map to API	INT4 cometqLineRxJatCfg(sCMQ_HNDL deviceHandle, UINT2 chan, sCMQ_CFG_RX_JAT *prxJatCfg);
Input field(s)	<Chan> any number from 0 to 3 < sCMQ_CFG_RX_JAT > = { < enable > < refDiv > < outputDiv > < FIFOselfCenter > < preventOvfUndf > } < enable >: 0 or 1 < refDiv >: a ratio (refer to data sheet) < outputDiv >: a ratio (refer to data sheet) < FIFOselfCenter >: 0 or 1 < preventOvfUndf >: 0 or 1
Output	none if success
Example	cometqLineRxJatCfg 0 { 1 1 1 0 0 }

5.3.7 cometqLineClkSvcCfg < eCMQ_CSU_SVC_CLK >

Description	Configure clock service unit
Map to API	INT4 cometqLineClkSvcCfg(sCMQ_HNDL deviceHandle, eCMQ_CSU_SVC_CLK synthTxFreq);
Input field(s)	< eCMQ_CSU_SVC_CLK > = CMQ_XCLK_2048_TXCLK_2048 or 0, CMQ_XCLK_1544_TXCLK_1544 or 1, CMQ_XCLK_2048_TXCLK_1544 or 2
Output	none if success
Example 1	cometqLineClkSvcCfg {CMQ_XCLK_2048_TXCLK_2048}
Example 2	cometqLineClkSvcCfg 1

5.3.8 cometqLineRxClkCfg <Chan> < sCMQ_CFG_RX_CLK >

Description	Configure receive line clock

Map to API	`INT4 cometqLineRxClkCfg(sCMQ_HNDL deviceHandle, UINT2 chan, sCMQ_CFG_RX_CLK *prxClkCfg);`

Input field(s)	`<Chan> any number from 0 to 3`
	`< sCMQ_CFG_RX_CLK > =`
	`{ < recoverClkSel > < LOSThresh >}`
	`<recoverClkSel>: CMQ_RECOVER_CLK_LOW_FREQ_JAT or 0,`
	`                 CMQ_RECOVER_CLK_HIGH_FREQ_JAT or 1`
	`<LOSThresh>: CMQ_LOS_THRESH_PCM_10_HDB3 or 0,`
	`             CMQ_LOS_THRESH_PCM_15_B8ZS or 1,`
	`             CMQ_LOS_THRESH_PCM_15_AMI or 2,`
	`             CMQ_LOS_THRESH_PCM_31 or 3,`
	`             CMQ_LOS_THRESH_PCM_63 or 4,`
	`             CMQ_LOS_THRESH_PCM_175 or 5`

Output	`none if success`
Example	`cometqLineRxClkCfg 0 { 1 3 }`

5.4 System Side Interface

5.4.1 cometqBTIFAccessCfg <Chan> < sCMQ_BACKPLANE_ACCESS_CFG >

Description	`Configure backplane transmit interface`
Map to API	`INT4 cometqBTIFAccessCfg(sCMQ_HNDL deviceHandle, UINT2 chan, sCMQ_BACKPLANE_ACCESS_CFG* pBTIFCfgData);`

Input field(s)	`<Chan> any number from 0 to 3`
	`< sCMQ_BACKPLANE_ACCESS_CFG > =`
	`{ < masterMode > < dataMode > < clkTimes2 > < dataRate > }`
	`< masterMode >: 0 or 1`
	`< dataMode >: CMQ_BACKPLANE_FULL_FRAME_MODE or 0,`
	`              CMQ_BACKPLANE_NX56K_MODE or 1,`
	`              CMQ_BACKPLANE_NX64K_MODE or 2,`
	`              CMQ_BACKPLANE_NX64K_E1_MODE or 3`
	`< clkTimes2 >: 0 or 1`
	`< dataRate >: CMQ_BACKPLANE_CLK_RATE_1544 or 0,`
	`              CMQ_BACKPLANE_CLK_RATE_2048 or 1,`
	`              CMQ_BACKPLANE_CLK_RATE_8192 or 2`

Output	`none if success`
Example	`cometqBTIFAccessCfg 1 { 1 0 1 0 }`

5.4.2 cometqBTIFFrmCfg <Chan> < sCMQ_CFG_BTIF_FRM >

Description	`Configure backplane transmit interface`
Map to API	`INT4 cometqBTIFFrmCfg(sCMQ_HNDL deviceHandle, UINT2 chan, sCMQ_CFG_BTIF_FRM *pfrmCfg);`

Input field(s)	`<Chan> any number from 0 to 3`
	`< sCMQ_CFG_BTIF_FRM > =`
	`{< fpMaster > < fpInvEn > < oddPar > < extParEn > < fpFrmOffset > < T1ESFAlign > < fpBitOffsetEn > < fpBitOffset > < tslotMapFormat > }`
	`< fpMaster >: 0 or 1`
	`< fpInvEn >: 0 or 1`
	`< oddPar >: 0 or 1`
	`< extParEn > 0 or 1`
	`< fpFrmOffset > Offset in bytes`
	`< T1ESFAlign > 0 or 1 (T1 Mode Only)`
	`< fpBitOffsetEn > 0 or 1`
	`< fpBitOffset > Offset in bits`
	`< tslotMapFormat >`
	`CMQ_BACKPLANE_TIMESLOT_MAP_3_OF_4 or 0,`
	`CMQ_BACKPLANE_TIMESLOT_MAP_24_OF_32 or 1`

Output	`none if success`
Example	`cometqBTIFFrmCfg 1 {0 1 0 1 1 1 1 1 0}`

5.4.3 cometqBRIFAccessCfg <Chan >< sCMQ_BACKPLANE_ACCESS_CFG >

Description	`Configure backplane transmit interface`
Map to API	`INT4 cometqBRIFAccessCfg(sCMQ_HNDL deviceHandle, UINT2 chan, sCMQ_BACKPLANE_ACCESS_CFG* pBRIFCfgData);`

Input field(s)	`<Chan> any number from 0 to 3`
	`< sCMQ_BACKPLANE_ACCESS_CFG > =`
	`{ < masterMode > < dataMode > < clkTimes2 > < dataRate > }`
	`< masterMode >: 0 or 1`
	`< dataMode >: CMQ_BACKPLANE_FULL_FRAME_MODE or 0,`
	`CMQ_BACKPLANE_NX56K_MODE or 1,`
	`CMQ_BACKPLANE_NX64K_MODE or 2,`
	`CMQ_BACKPLANE_NX64K_E1_MODE or 3`
	`< clkTimes2 >: 0 or 1`
	`< dataRate >: CMQ_BACKPLANE_CLK_RATE_1544 or 0,`
	`CMQ_BACKPLANE_CLK_RATE_2048 or 1,`
	`CMQ_BACKPLANE_CLK_RATE_8192 or 2`

Output	`none if success`
Example	`cometqBRIFAccessCfg 1 { 0 1 1 0 }`

5.4.4 cometqBRIFFrmCfg <Chan> < sCMQ_CFG_BRIF_FRM >

Description	`Configure backplane receive interface`
Map to API	`INT4 cometqBRIFFrmCfg(sCMQ_HNDL deviceHandle, UINT2 chan, sCMQ_CFG_BRIF_FRM *pfrmCfg);`

Input field(s)	`<Chan> any number from 0 to 3`
	`< sCMQ_CFG_BRIF_FRM > =`
	`{< fpMaster > < fpmMode > < fpInvEn > < parInsEn > < oddPar > < extParEn > < fBitFix > < fBitPol > < fpFrmOffset > < fpBitOffsetEn > < fpBitOffset > < altFDLEn > < tslotMapFormat >}`
	`< fpMaster >: 0 or 1`
	`< fpmMode >:`
	`CMQ_BACKPLANE_RX_FP_T1_HIGH_ON_SF_ESF or 0,`
	`CMQ_BACKPLANE_RX_FP_T1E1_HIGH_EVERY_FRAME or 1,`
	`CMQ_BACKPLANE_RX_FP_E1_HIGH_ON_CRC_MFRM or 2,`
	`CMQ_BACKPLANE_RX_FP_E1_HIGH_ON_SIG_MFRM or 3,`
	`CMQ_BACKPLANE_RX_FP_E1_COMP_MFRM or 4,`
	`CMQ_BACKPLANE_RX_FP_E1_HIGH_ON_OVERHEAD or 5`
	`< fpInvEn >: 0 or 1`
	`< parInsEn >: 0 or 1`
	`< oddPar >: 0 or 1`
	`< extParEn > : 0 or 1`
	`< fBitFix >: 0 or 1`
	`< fBitPol >: 0 or 1`
	`< fpFrmOffset > Offset in bytes`
	`< fpBitOffsetEn >: 0 or 1`
	`< fpBitOffset > Offset in bits`
	`< altFDLEn >: 0 or 1`
	`< tslotMapFormat >`
	`CMQ_BACKPLANE_TIMESLOT_MAP_3_OF_4 or 0,`
	`CMQ_BACKPLANE_TIMESLOT_MAP_24_OF_32 or 1`

Output	`none if success`
Example	`cometqBRIFFrmCfg 1 {0 1 0 1 0 1 0 1 0 1 0 1 0}`

5.4.5 cometqHMVIPCfg < rxHMVIPMode > < txHMVIPMode > <chan 0 data> <chan 1 data> <chan 2 data> <chan 3 data>

Description	`Configure the H-MVIP receive and transmit interface`
Map to API	`INT4 cometqHMVIPCfg (sCMQ_HNDL deviceHandle, sCMQ_CFG_HMVIP *pHMVIPCfg)`

Input field(s)	`< rxHMVIPMode >:     CMQ_BACKPLANE_HMVIP_MODE or 0,`
	`                     CMQ_BACKPLANE_HMVIP_CCS or 1`
	`                     CMQ_BACKPLANE_HMVIP_DISABLE or 2`
	`< txHMVIPMode >:     CMQ_BACKPLANE_HMVIP_MODE or 0,`
	`                     CMQ_BACKPLANE_HMVIP_CCS or 1,`
	`                     CMQ_BACKPLANE_HMVIP_DISABLE or 2`
	`<chan 0, 1, 2, 3 data>:  ( insert at ) { timeslot 15 timeslot 16    timeslot 31 }`

Output	none if success
Example	`cometqHMVIPCfg 1  1  {1 0 0} {0 0 0} {1 1 1} {1 0 1}`

5.4.6 cometqTxElstStCfg <Chan >< elstEnable >

Description	Configure the elastic store in receive data path
Map to API	`INT4 cometqTxElstStCfg(sCMQ_HNDL deviceHandle, UINT2 chan, UINT1 elstEnable);`
Input field(s)	<Chan> any number from 0 to 3 < elstEnable > 0 or 1
Output	none if success
Example	`cometqTxElstStCfg 0 1`

5.4.7 cometqRxElstStCfg <Chan >< sCMQ_CFG_RX_ELST >

Description	Configure the elastic store in receive data path

Map to API	`INT4 cometqRxElstStCfg(sCMQ_HNDL deviceHandle, UINT2 chan,`
	`                        sCMQ_CFG_RX_ELST* pElstCfg);`

Input field(s)	<Chan> any number from 0 to 3 < sCMQ_CFG_RX_ELST > = { < elstEnable > < idleCode > < CCSidleCode > } < elstEnable >: 0 or 1 < idleCode >: idle code < CCSidleCode >: ccs idle code
Output	none if success
Example	`cometqRxElstStCfg 0 { 1 1 1 }`

5.5 T1 /E1 Framers

5.5.1 cometqSetOperatingMode <operMode>

Description	`This function specifies whether the device will operate in T1 or E1 mode.`
Map to API	`INT4 cometqSetOperatingMode(sCMQ_HNDL deviceHandle, eCMQ_OPER_MODE operMode);`

Input field(s)	`<operMode>: CMQ_MODE_E1 or 0,`
	`CMQ_MODE_T1 or 1`

Output	`none if success`
Example	`cometqSetOperatingMode CMQ_MODE_E1`

5.5.2 cometqT1TxFramerCfg <Chan> < sCMQ_CFG_T1TX_FRM >

Description	`T1 transmit framer configuration`
Map to API	`INT4 cometqT1TxFramerCfg (sCMQ_HNDL deviceHandle, UINT2 chan, sCMQ_CFG_T1TX_FRM *pfrmCfg)`

Input field(s)	`<Chan> any number from 0 to 3`
	`< sCMQ_CFG_T1TX_FRM > =`
	`{< frmMode > < zSupFormat > < SFSigAlignerEn >}`
	`< frmMode >:  CMQ_FRM_MODE_E1 or 0,`
	`CMQ_FRM_MODE_T1_SF or 3,`
	`CMQ_FRM_MODE_T1_DM or 4,`
	`CMQ_FRM_MODE_T1_SLC96 or 5,`
	`CMQ_FRM_MODE_T1_DM_FDL or 6,`
	`CMQ_FRM_MODE_T1_ESF or 7,`
	`CMQ_FRM_MODE_T1_SF_JPN_ALARM or 8,`
	`CMQ_FRM_MODE_T1_DM_JPN_ALARM or 9,`
	`CMQ_FRM_MODE_T1_SLC96_JPN_ALARM or 10,`
	`CMQ_FRM_MODE_T1_DM_FDL_JPN_ALARM or 11,`
	`CMQ_FRM_MODE_T1_JT_G704 or 12,`
	`CMQ_FRM_MODE_T1_UNFRAMED or 13`
	`< zSupFormat >: CMQ_T1_ZSUP_NONE or 0,`
	`CMQ_T1_ZSUP_GTE or 1,`
	`CMQ_T1_ZSUP_DDS or 2,`
	`CMQ_T1_ZSUP_BELL or 3`
	`< SFSigAlignerEn >: 0 or 1`

Output	`none if success`
Example	`cometqT1TxFramerCfg  2 { 1 2 0 }`

5.5.3 cometqT1RxFramerCfg <Chan> < sCMQ_CFG_T1RX_FRM >

Description	`T1 receive framer configuration`
Map to API	`INT4 cometqT1RxFramerCfg (sCMQ_HNDL deviceHandle, UINT2 chan, sCMQ_CFG_T1RX_FRM *pfrmCfg)`

Input field(s)	`<Chan> any number from 0 to 3`
	`< sCMQ_CFG_T1RX_FRM > =`
	`{< frmMode > < outOfFrameCriteria> < frmESFAlgo> < COFACntEn >}`
	`< frmMode >:`
	`    CMQ_FRM_MODE_T1_SF or 3,`
	`    CMQ_FRM_MODE_T1_DM or 4,`
	`    CMQ_FRM_MODE_T1_SLC96 or 5,`
	`    CMQ_FRM_MODE_T1_DM_FDL or 6,`
	`    CMQ_FRM_MODE_T1_ESF or 7,`
	`    CMQ_FRM_MODE_T1_SF_JPN_ALARM or 8,`
	`    CMQ_FRM_MODE_T1_DM_JPN_ALARM or 9,`
	`    CMQ_FRM_MODE_T1_SLC96_JPN_ALARM or 10,`
	`    CMQ_FRM_MODE_T1_DM_FDL_JPN_ALARM or 11,`
	`    CMQ_FRM_MODE_T1_JT_G704 or 12,`
	`    CMQ_FRM_MODE_T1_UNFRAMED or 13`
	`< outOfFrameCriteria >: CMQ_T1_OOF_2OF4 or 0,`
	`                        CMQ_T1_OOF_2OF5 or 1,`
	`                        CMQ_T1_OOF_2OF6 or 2`
	`< frmESFAlgo >:`
	`CMQ_T1_ESF_FRAME_ALGO_ONE_CANDIDATE or 0,`
	`CMQ_T1_ESF_FRAME_ALGO_CRC_6 or 1`
	`< COFACntEn >: 0 or 1;`

Output	`none if success`
Example	`cometqT1RxFramerCfg   1 { 3 1 1 1 }`

5.5.4 cometqE1TxFramerCfg <Chan> < sCMQ_CFG_E1TX_FRM >

Description	`E1 transmit framer configuration`
Map to API	`INT4 cometqE1TxFramerCfg (sCMQ_HNDL deviceHandle, UINT2 chan, sCMQ_CFG_E1TX_FRM *pfrmCfg)`

Input field(s)	`<Chan> any number from 0 to 3`
	`< sCMQ_CFG_E1TX_FRM > =`
	`{< frmMode > < ts16Signaling > < insNatIntBitEn > < insXtraBitsEn > < insFEBEEn >}`
	`< frmMode >:`
	`    CMQ_FRM_MODE_E1 or 0,`
	`    CMQ_FRM_MODE_E1_CRC_MFRM or 1,`

	`CMQ_FRM_MODE_E1_UNFRAMED or 2,`
	`< ts16Signaling >: CMQ_E1_SIG_INS_NONE or 0,`
	`CMQ_E1_SIG_INS_HDLC_CCS or 1,`
	`CMQ_E1_SIG_INS_CAS or 2`
	`< insNatIntBitEn >: 0 or 1`
	`< insXtraBitsEn >: 0 or 1`
	`< insFEBEEn >: 0 or 1`

Output	`none if success`
Example	`cometqE1TxFramerCfg  1 { 0 0 1 1 1 }`

5.5.5 cometqE1RxFramerCfg <Chan> < sCMQ_CFG_T1RX_FRM >

Description	`E1 receive framer configuration`
Map to API	`INT4 cometqE1RxFramerCfg (sCMQ_HNDL deviceHandle, UINT2 chan, sCMQ_CFG_E1RX_FRM *pfrmCfg)`

Input field(s)	`<Chan> any number from 0 to 3`
	`< sCMQ_CFG_E1RX_FRM > =`
	`{< frmMode > < CASAlignmentEn > < CRC2NCRCEn > < noReframeOnErrEn > < reframeOnXSCrcErrEn > < lofBit2CritEn > < NFASErrEn > < multFASEOneFEEn > < mfrmLossAlignCrit > < AISCriteria > < RAICriteria >}`
	`< frmMode >:`
	`CMQ_FRM_MODE_E1 or 0,`
	`CMQ_FRM_MODE_E1_CRC_MFRM or 1,`
	`CMQ_FRM_MODE_E1_UNFRAMED or 2,`
	`< CASAlignmentEn >: 0 or 1`
	`< CRC2NCRCEn >: 0 or 1`
	`< noReframeOnErrEn >: 0 or 1`
	`< reframeOnXSCrcErrEn >: 0 or 1`
	`< lofBit2CritEn >:0 or 1`
	`< NFASErrEn >:0 or 1`
	`< multFASEOneFEEn >:0 or 1`
	`< mfrmLossAlignCrit >:`
	`CMQ_E1_LOSS_MFRM_ALIGN_TS16_CRIT_NONE or 0,`
	`CMQ_E1_LOSS_MFRM_ALIGN_TS16_CRIT_ZERO_1_MFRM or 1,`
	`CMQ_E1_LOSS_MFRM_ALIGN_TS16_CRIT_ZERO_2_MFRM or 2`
	`< AISCriteria >:`
	`CMQ_E1_AIS_CRIT_3Z_IN_512BITS or 0,`
	`CMQ_E1_AIS_CRIT_2_PERIODS_3Z_IN_512BITS or 1`
	`< RAICriteria >: CMQ_E1_RAI_CRIT_ALL_A_1 or 0,`
	`CMQ_E1_RAI_CRIT_4_CONSEC_A_1 or 1`

Output	`none if success`

Example	`cometqE1RxFramerCfg   1 {1 0 1 0 1 0 1 0 1 0 1}`

5.5.6 cometqE1TxSetExtraBits <Chan > < extraBits >

Description	Allows user to set extra bit values that will be inserted into bits 5, 7 and 8 of timeslot 16 if enabled in the E1 transmit framer configuration.
Map to API	`INT4 cometqE1TxSetExtraBits(sCMQ_HNDL deviceHandle, UINT2 chan,  UINT1 extraBits);`
Input field(s)	<Chan> any number from 0 to 3 < extraBits > bitmap (bit 0: X1 – timeslot 16, bit 5) (bit 1: X3 – timeslot 16, bit 7) (bit 2: X4 – timeslot 16, bit 8) (bit 3-7: unused)
Output	none if success
Example	`cometqE1TxSetExtraBits 1 0x03`

5.5.7 cometqE1TxSetIntBits <Chan > < intBits >

Description	Set value of the international bits to insert into the E1 stream. Must be enabled first in the configuration.
Map to API	`INT4 cometqE1TxSetIntBits(sCMQ_HNDL deviceHandle, UINT2 chan, UINT1 intBits);`
Input field(s)	<Chan> any number from 0 to 3 < intBits > bitmap containing international bits bit 0: Si[0]: inserted into NFAS frames bit 1: Si[1]: inserted into FAS frames bit 2-7: unused
Output	none if success
Example	`cometqE1TxSetIntBits 1 0xFF`

5.5.8 cometqE1TxSetNatBits <Chan > < codeSelect > <natBits > <natBitsEn >

Description	Set National bits. Must be enabled first in the framer configuration.
Map to API	`INT4 cometqE1TxSetNatBits(sCMQ_HNDL deviceHandle, UINT2 chan, eCMQ_E1_NAT_BIT codeSelect, UINT1 natBits, UINT1 natBitsEn)`
Input field(s)	<Chan> any number from 0 to 3 <codeSelect>: CMQ_E1_NAT_BIT_SA4 or 0, CMQ_E1_NAT_BIT_SA5 or 1, CMQ_E1_NAT_BIT_SA6 or 2, CMQ_E1_NAT_BIT_SA7 or 3,

	`CMQ_E1_NAT_BIT_SA8 or 4`
	`<natBits >: bitmap containing national bit values (over one sub-multiframe)`
	`bit 0 : first Sa`$_i$` position in SMF`
	`bit 1 : second Sa`$_i$` position in SMF`
	`bit 2 : third Sa`$_i$` position in SMF`
	`bit 3 : fourth Sa`$_i$` position in SMF`
	`bits 4-7 : unused`
	`<natBitsEn >: enables/disables each bit position in natBits:`
	`bit 0 : enable first Sa`$_i$` bit`
	`bit 1 : enable second Sa`$_i$` bit`
	`bit 2 : enable third Sa`$_i$` bit`
	`bit 3 : enable fourth Sa`$_i$` bit`
	`bits 4-7 : unused`

Output	`none if success`
Example	`cometqE1TxSetNatBits 3 2 0xff 0xff`

5.5.9 cometqE1RxGetExtraBits <Chan >

Description	`Get extra bits and the y bit from ts16 frame 0 of the last received signaling multiframe`
Map to API	`INT4 cometqE1RxGetExtraBits(sCMQ_HNDL deviceHandle, UINT2 chan, UINT1* pExtraBits);`
Input field(s)	`<Chan> any number from 0 to 3`

Output	`pExtraBits : bitmap containing extra bits & y bit`
	`bit 0 : X1 (from timeslot 16, bit 5)`
	`bit 1 : Y bit (from timeslot 16, bit 6)`
	`bit 2 : X3 (from timeslot 16, bit 7)`
	`bit 3 : X4 (from timeslot 16, bit 8)`
	`bits 4-7 : unused`

Example	`cometqE1RxGetExtraBits 2`

5.5.10 cometqE1RxGetIntBits <Chan >

Description	`This function returns the international bits from the incoming E1 stream for the last frame. S`$_i$`[0] is updated every NFAS frame while S`$_i$`[1] is updated every FAS frame.`
Map to API	`INT4 cometqE1RxGetIntBits(sCMQ_HNDL deviceHandle, UINT2 chan, UINT1* pIntBits);`
Input field(s)	`<Chan> any number from 0 to 3`

Output	`pIntBits : bitmap containing international bits`
	`bit 0 : S`$_i$`[0]: inserted into NFAS frames`

	bit 1 : S_i[1]: inserted into FAS frames bit 2-7: unused
Example	`cometqE1RxGetIntBits 2`

5.5.11 cometqE1RxGetNatBitsNFAS <Chan >

Description	This function returns the value of the national bits from the incoming E1 stream from the last NFAS frame. This API allows the user to process the national bits on a frame by frame basis, without waiting for the complete submultiframe.
Map to API	`INT4 cometqE1RxGetNatBitsNFAS(sCMQ_HNDL deviceHandle, UINT2 chan, UINT1* pNatBits);`
Input field(s)	<Chan> any number from 0 to 3
Output	pNatBits : bitmap containing national bit values bit 0 : Sa 4 bit 1 : Sa 5 bit 2 : Sa 6 bit 3 : Sa 7 bit 4 : Sa 8 bits 5-7 : unused
Example	`cometqE1RxGetNatBitsNFAS 2`

5.5.12 cometqE1RxGetNatBitsSMFRM <Chan > <codeSelect>

Description	This function returns a complete national bit codeword for a specified national bit in the incoming E1 stream for the last submultiframe.
Map to API	`INT4 cometqE1RxGetNatBitsSMFRM(sCMQ_HNDL deviceHandle, UINT2 chan, eCMQ_E1_NAT_BIT codeSelect, UINT1* pNatBits);`
Input field(s)	<Chan> any number from 0 to 3 <codeSelect>: CMQ_E1_NAT_BIT_SA4 or 0, CMQ_E1_NAT_BIT_SA5 or 1, CMQ_E1_NAT_BIT_SA6 or 2, CMQ_E1_NAT_BIT_SA7 or 3, CMQ_E1_NAT_BIT_SA8 or 4
Output	pNatBits : bitmap containing national bit values (over one sub-multiframe) bit 0 : first Sa_i position in SMF bit 1 : second Sa_i position in SMF bit 2 : third Sa_i position in SMF

	bit 3 : fourth Sa_i position in SMF bits 4-7 : unused
Example	cometqE1RxGetNatBitsSMFRM 1 1

5.6 Signal Insertion / Extraction

5.6.1 cometqSigExtractCOSS <Chan>

Description	This function provides a bitmap corresponding to change of signaling state (COSS) information for a T1 or E1 stream. After determining on which E1 timeslot or T1 channel the signaling state has changed through the use of this function, the user should call cometqSigExtract for the new signaling state.
Map to API	INT4 cometqExtractCOSS(sCMQ_HNDL deviceHandle, UINT2 chan, UINT4 *psigState)
Input field(s)	<Chan> any number from 0 to 3
Output	pSigState: 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.
Example	cometqSigExtractCOSS 0

5.6.2 cometqSigExtract <Chan> <timeslot>

Description	Signaling state
Map to API	INT4 cometqSigExtract(sCMQ_HNDL deviceHandle, UINT2 chan, UINT1 timeslot, UINT1 *pSigState)
Input field(s)	<Chan> any number from 0 to 3 <timeslot> timeslot for which to retrieve signaling
Output	pSigState : 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`

Example	`cometqSigExtract 1 12`

5.6.3 cometqSigTslotTrnkDataCfg <Chan> <readWrite> <timeslot> [<SigConfig>]

Description	This API allows the user to enable change of signal state debouncing by only allowing a COSS event to be generated when the new signaling data is received twice consecutively. Also, DS0 manipulation of the PCM data stream can be performed here. DS0 PCM data manipulation performed by the signal extraction block occurs before PCM data manipulation performed by the RPSC block (via cometqRPSCPCMCtrl).
Map to API	`INT4 cometqSigTslotTrnkDataCfg(sCMQ_HNDL deviceHandle, UINT2 chan, UINT2 readWrite, UINT1 timeslot, UINT1 *pSigConfig);`

Input field(s)	`<Chan> any number from 0 to 3`
	`<readWrite> 0 for read, 1 for write`
	`<timeslot> timeslot to operate on`
	`<SigConfig> config data`

Output	`pSigConfig : pointer to config data`
	
	`bit map for T1 mode:`
	
	`bit 0: enable COSS debouncing`
	`bit 1: logic level when bit fixing enabled`
	`bit 2: enable bit fixing`
	`bit 3: invert all PCM data bits`
	`bit 4-7: unused`
	
	`bit map for E1 mode:`
	
	`bit 0: enable COSS debouncing`
	`bit 1: unused`
	`bit 2,3: 0,0 – no inversion`
	`0,1 – invert odd PCM bits`
	`1,0 – invert even PCM bits`
	`1,1 – invert all bits`
	`bit 4-7: unused`

Example 1 (Read)	`cometqSigTslotTrnkDataCfg 1 0 13`

Example 2 (Write)	`cometqSigTslotTrnkDataCfg 1 1 12 0xff`

5.7 Alarm Control and InBand Communications

5.7.1 cometqAutoAlarmCfg <Chan> < autoYellowEn > < autoRedEn > < OOF_RPSCEn > <OOF_RxELSTEn > < autoAISEn >

Description	Enable or disable the possible automatic alarm responses
Map to API	`INT4 cometqAutoAlarmCfg(sCMQ_HNDL deviceHandle, UINT2 chan,  UINT1 autoYellowEn, UINT1 autoRedEn, UINT1 OOF_RPSCEn, UINT1 OOF_RxELSTEn, UINT1 autoAISEn);`

Input field(s)	`<Chan> any number from 0 to 3`
	`< autoYellowEn >: 0 or 1`
	`< autoRedEn >: 0 or 1`
	`< OOF_RPSCEn >: 0 or 1`
	`< OOF_RxELSTEn >: 0 or 1`
	`< autoAISEn >: 0 or 1`

Output	none if success
Example	`cometqAutoAlarmCfg 0 1 0 1 0 1`

5.7.2 cometqInsertAlarm <Chan> <alarmType> <enable>

Description	Insert or disable insertion of alarm

Map to API	`INT4 cometqInsertAlarm(sCMQ_HNDL deviceHandle, UINT2 chan,`
	`eCMQ_ALARM_INS alarmType, UINT1 enable);`

Input field(s)	`<Chan> any number from 0 to 3`
	`<alarmType>:  CMQ_ALARM_INS_RX_AIS or 0,`
	`CMQ_ALARM_INS_TX_YELLOW or 1,`
	`CMQ_ALARM_INS_TX_AIS or 2,`
	`CMQ_ALARM_INS_TX_E1_Y_BIT or 3,`
	`CMQ_ALARM_INS_TX_E1_CHAN16 or 4`
	`<enable>: 0 or 1`

Output	none if success
Example	`cometqInsertAlarm 1 0 1`

5.7.3 cometqHDLCEnable <idHDLC> <enable>

Description	`Enable HDLC inband communications`
Map to API	`INT4 cometqHDLCEnable(sCMQ_HNDL deviceHandle, UINT2 idHDLC, UINT2 enable)`

Input field(s)	`<idHDLC> HDLC controller 0, 1, 2, 3,`
	`<enable> 0 or 1`

Output	`none if success`
Example	`cometqHDLCEnable 1 1`

5.7.4 cometqHDLCRxCfg <idHDLC>< sCMQ_CFG_HDLC_RX >

Description	`HDLC configuration`
Map to API	`INT4 cometqHDLCRxCfg(sCMQ_HNDL deviceHandle, UINT2 idHDLC, sCMQ_CFG_HDLC_RX* pData);`

Input field(s)	`<idHDLC> : HDLC controller 0, 1, 2, 3`
	`< sCMQ_CFG_HDLC_RX >= { { linkLocation } < addrMatchEn >< addrMaskEn > }`
	`{ linkLocation } = { < useT1DataLink >< evenFrames >< oddFrames >< timeslot >< dataLinkBitMask > }`
	`< useT1DataLink >:0 or 1`
	`< evenFrames >:0 or 1 (ignored when using T1 Data Link)`
	`< oddFrames >:0 or 1 (ignored when using T1 Data Link)`
	`< timeslot >: timeslot`
	`< dataLinkBitMask >: bit mask`
	`< addrMatchEn >: 0 or 1`
	`< addrMaskEn >: 0 or1`

Output	`none if success`
Example	`cometqHDLCRxCfg 0 {{ 1 0 1 0 1} 0 1}`

5.7.5 cometqHDLCTxCfg <idHDLC>< sCMQ_CFG_HDLC_TX >

Description	`HDLC configuration`
Map to API	`INT4 cometqHDLCTxCfg(sCMQ_HNDL deviceHandle, UINT2 idHDLC, sCMQ_CFG_HDLC_TX* pData);`

Input field(s)	`<idHDLC> : HDLC controller 0, 1, 2, 3`
	`< sCMQ_CFG_HDLC_TX > ={{ linkLocation } < flagShareEn ><crcFCSEn ><pmRepEn> }`
	`{ linkLocation } = { < useT1DataLink >< evenFrames >< oddFrames >< timeslot >< dataLinkBitMask > }`
	`< useT1DataLink >:0 or 1`

	`< evenFrames >:0 or 1 (ignored when using T1 Data Link)`
	`< oddFrames >:0 or 1 (ignored when using T1 Data Link)`
	`< timeslot >: timeslot`
	`< dataLinkBitMask >: bit mask`
	`< flagShareEn >:0 or 1`
	`< crcFCSEn >:0 or 1`
	`< pmRepEn >:0 or 1`

Output	`none if success`
Example	`cometqHDLCTxCfg 1 {{1 0 1 0 1} 0 0 0 }`

5.7.6 cometqTDPRData < idHDLC > <value>

Description	`This function transmits a data byte on the specified HDLC link.`
Map to API	`NT4 cometqTDPRData(sCMQ_HNDL deviceHandle, UINT2 idHDLC, UINT1 value);`

Input field(s)	`< idHDLC > : HDLC controller 0, 1, 2, 3`
	`<value> : any number from 0 to 0xff`

Output	`none if success`
Example	`cometqTDPRData 1 0xf`

5.7.7 cometqTDPRCtl < idHDLC > <hdlcAction>

Description	`With this function the user can transmit a control byte on an HDLC link or force a FIFO clear.`
Map to API	`INT4 cometqTDPRCtl(sCMQ_HNDL deviceHandle, UINT2 idHDLC, eCMQ_TDPR_ACTION hdlcAction)`

Input field(s)	`< idHDLC > : HDLC controller 0, 1, 2, 3`
	`< hdlcAction >: CMQ_TDPR_ACTION_ABORT or 0,`
	`CMQ_TDPR_ACTION_END_ABORT or 1,`
	`CMQ_TDPR_ACTION_EOM or 2,`
	`CMQ_TDPR_ACTION_FIFOCLR or 3`

Output	`none if success`
Example	`cometqTDPRCtl 1 CMQ_TDPR_ACTION_ABORT`

5.7.8 cometqTDPRFIFOThreshCfg < idHDLC > <upFifoThresh> <lowFifoThresh>

Description	`Configures the upper and lower limits of the HDLC transmitter FIFO for one of the transmit HDLC Controllers.`
Map to API	`INT4 cometqTDPRFIFOThreshCfg (sCMQ_HNDL deviceHandle, UINT2 idHDLC, UINT1 upFifoThresh, UINT1 lowFifoThresh)`

Input field(s)	`< idHDLC > : HDLC controller 0, 1, 2, 3`
	`<upFifoThresh> FIFO Watermark for auto transmit`
	`valid values are 0 thru 127`
	`<lowFifoThresh> FIFO Watermark for internal interrupt`
	`valid values are 0 thru 127`
	
	`* low threshold must be less than up threshold unless both are set to 0`

Output	`none if success`
Example	`cometqTDPRFIFOThreshCfg 1 3 2`

5.7.9 cometqRDLCTerm < idHDLC >

Description	Forces the RDLC to immediately terminate the reception of the current data frame. This function causes the current data frame to terminate and the FIFO buffer to empty.
Map to API	`INT4 cometqRDLCTerm (sCMQ_HNDL deviceHandle, UINT2 idHDLC)`
Input field(s)	`< idHDLC > : HDLC controller 0,1, 2, 3`
Output	`none if success`
Example	`cometqRDLCTerm 1`

5.7.10 cometqRDLCAddrMatch < idHDLC > < addrPri > < addrSec >

Description	This function configures the primary and secondary addresses used with address matching on an HDLC receiver. When address masking is enabled, the lower two bits of each of the primary and secondary addresses are masked during comparison.
Map to API	`INT4 cometqRDLCAddrMatch(sCMQ_HNDL deviceHandle, UINT2 idHDLC, UINT1 addrPri, UINT1 addrSec)`

Input field(s)	`< idHDLC > : HDLC controller 0, 1, 2, 3`
	`< addrPri > primary address`
	`< addrSec > secondary address`

Output	`none if success`
Example	`cometqRDLCAddrMatch 1 2 3`

5.7.11 cometqRDLCFIFOThreshCfg < idHDLC > < fifoThresh>

Description	Configures the fill level threshold for an RDLC FIFO. The fill level threshold is the number of bytes that

	must be present in the FIFO before an interrupt is generated.
Map to API	INT4 cometqRDLCFIFOThreshCfg(sCMQ_HNDL deviceHandle, UINT2 idHDLC, UINT1 fifoThresh)
Input field(s)	< idHDLC > : HDLC controller 0, 1, 2, 3 <fifoThresh> FIFO fill level threshold (7 bit value)
Output	none if success
Example	cometqRDLCFIFOThreshCfg 1 2

5.7.12 cometqIBCDActLpBkCfg < chan > < patLen > <pat>

Description	Configures the detection of inband activate loopback code length and pattern.
Map to API	INT4 cometqIBCDActLpBkCfg(sCMQ_HNDL deviceHandle, UINT2 chan, UINT2 patLen, UINT2 pat)
Input field(s)	< chan > : channel from 0 to 3 <patLen > 5,6,7,8 <pat>: loop code pattern
Output	none if success
Example	cometqIBCDActLpBkCfg 1 5 1

5.7.13 cometqIBCDDeActLpBkCfg < chan > < patLen > <pat>

Description	Configures the detection of inband de-activate loopback code length and pattern.
Map to API	INT4 cometqIBCDDeActLpBkCfg(sCMQ_HNDL deviceHandle, UINT2 chan, UINT1 patLen, UINT2 pat)
Input field(s)	< chan > : channel from 0 to 3 <patLen > 5,6,7,8 <pat>: loop code pattern
Output	none if success
Example	cometqIBCDDeActLpBkCfg 1 6 1

5.7.14 cometqIBCDTxCfg < chan > < patLen > <pat> <enable>

Description	Enables/disables transmission of the inband transmit loopback code and allows configuration of the code length and pattern when activating. When disabling transmission of the loopback code, the parameters pat and patLen are not used.
Map to API	INT4 cometqIBCDTxCfg (sCMQ_HNDL deviceHandle, UINT2 chan, UINT2 patLen, UINT1 pat, UINT2 enable)

Input field(s)	`< chan > : channel from 0 to 3`
	`<patLen > 5,6,7,8`
	`<pat>: loop code pattern`
	`<enable> 0 or 1`

Output	`none if success`
Example	`cometqIBCDTxCfg 1 7 1 0`

5.7.15 cometqBOCTxCfg < chan > < boc > <repCount>

Description	Configures the bit oriented code for transmission. On COMET-Quad devices, the associated repeat count can also be configured. The value of parameter boc is transmitted with the least significant bit leading. To terminate BOC transmission, boc should be set to all 1's. Valid only in T1 mode.
Map to API	`INT4 cometqBOCTxCfg(sCMQ_HNDL deviceHandle, UINT2 chan, UINT1 boc, UINT1 repCount)`

Input field(s)	`< chan > : channel from 0 to 3`
	`< boc > 6 bit BOC code pattern. Setting this value to all 1's disables BOC transmission`
	`<repCount> number of consecutive BOC codes to transmit. Valid numbers from 0 to 15`

Output	`none if success`
Example	`cometqBOCTxCfg 1 2 3`

5.7.16 cometqBOCRxCfg < chan > <detectCriteria>

Description	Configures bit oriented code detection criteria by allowing the user to select the threshold that determines whether or not a valid BOC code has been detected. Valid BOC detection criteria are 8 out of 10 matching values (or alternately 4 out of 5 matching values). Valid only in T1 mode.
Map to API	`INT4 cometqBOCRxCfg(sCMQ_HNDL deviceHandle, UINT2 chan, eCMQ_RBOC_CRITERIA detectCriteria)`

Input field(s)	`< chan > : channel from 0 to 3`
	`<detectCriteria>: CMQ_RX_BOC_VALID_4OF5 or 0,`
	`CMQ_RX_BOC_VALID_8OF10 or 1`

Output	`none if success`
Example	`cometqBOCRxCfg 1 CMQ_RX_BOC_VALID_4OF5`

5.7.17 cometqBOCRxGet < chan >

Description	`This function retrieves the current bit oriented code from the holding registers.  Valid only in T1 mode.`
Map to API	`INT4 cometqBOCRxGet(sCMQ_HNDL deviceHandle, UINT2 chan, UINT1 *prxBOC)`
Input field(s)	`< chan > : channel from 0 to 3`
Output	`rxBOC = Last BOC received (6 bit value)`
Example	`cometqBOCRxGet 2`

5.8 Serial Controller

5.8.1 cometqTPSCEnable < chan > <enable>

Description	`Transmit per-channel serial controller`
Map to API	`INT4 cometqTPSCEnable(sCMQ_HNDL deviceHandle, UINT2 chan, UINT2 enable)`

Input field(s)	`< chan > : channel from 0 to 3`
	`<enable> 0 or 1`

Output	`none if success`
Example	`cometqTPSCEnable 2 1`

5.8.2 cometqTPSCPCMCtl < chan > <tSlot> <rWFlag> [<ctlByte> <trnkData> <sigData>]

Description	`Transmit per-channel serial controller`
Map to API	`INT4 cometqTPSCPCMCtl(sCMQ_HNDL deviceHandle, UINT2 chan, UINT1 tSlot ,UINT2 rWFlag, , UINT1 *pctlByte, UINT1 *ptrnkData, UINT1 *psigData)`

Input field(s)	`< chan > : channel from 0 to 3`
	`<tSlot> E1 from 0 to 31, T1 from 0 to 23`
	`<rWFlag> 0 for Read 1 for Write`
	`<ctlByte> when read control byte(when rWFlag=0)`
	`bit 7,4:`
	`pcm data unchanged         = 0,0`
	`invert pcm data             = 1,0`
	`invert pcm data MSB        = 0,1`
	`invert pcm data except MSB = 1,1`
	`bit 5:`

	`replace pcm data with idle pat = 1`
	`bit 4:`
	`replace pcm data with miliwatt pat = 1`
	`bit 3:`
	`replace pcm data with DPGR = 1`
	`bit 2:`
	`loopback Tx timeslot data with Rx data = 1`
	`bit 1,0 :`
	`zero code suppression format`
	`no zero suppression    = 0`
	`jammed bit 8              = 1`
	`GTE zero suppression = 2`
	`Bell zero suppression  = 3`
	`<trnkData>: trunk conditioning data byte(when rWFlag=0)`
	`<sigData>: signaling data byte (when rWFlag=0)`
	`bit map for E1 mode:`
	`bit 7,6,5:`
	`sub timeslot bits with trunk data = 0`
	`invert odd timeslot bits  = 1`
	`invert even timeslot bits = 2`
	`invert all timeslot bits  = 3`
	`sub timeslot bits with A-Law pattern  = 4`
	`sub timeslot bits with U-Law pattern  = 5`
	`sub timeslot bits with trunk data = 6`
	`sub timeslot bits with trunk data  = 7`
	`bit 4:`
	`This bit is only valid when CAS is selected in the T1-Tran configuration register.`
	`Signal trunken sourced from BTSIG via the format specified E1-Tran configuration reg = 0`
	`signal trunken sourced from  bit 3 thru 0  = 1`
	`bit 3,2,1,0:              signal trunken bits`
	
	`bit map for T1 mode:`
	`bit 7:`
	`signal trunken sourced from BTSIG = 0`
	`signal trunken sourced from  bit 3 thru 0  = 1`
	`bit 6: disable signal insertion = 0`
	`enable signal insertion = 1`
	`bit 5,4:`
	`unused`
	`bit 3,2,1,0:`
	`signal trunken bits`

Output	`when Read (Wflag =1)`
	`print if success`

	ctlByte : control byte trnkData : trunk conditioning data sigData: signaling data byte
Example	cometqTPSCPCMCtl 1 2 0

5.8.3 cometqRPSCEnable < chan > <enable>

Description	Receive per-channel serial controller
Map to API	INT4 cometqRPSCEnable(sCMQ_HNDL deviceHandle, UINT2 chan, UINT2 enable)
Input field(s)	< chan > : channel from 0 to 3 <enable> 0 or 1
Output	none if success
Example	cometqRPSCEnable 2 1

5.8.4 cometqRPSCPCMCtl < chan > <tSlot> <rWFlag> [<ctlByte> <trnkData> <sigData>]

Description	Transmit per-channel serial controller
Map to API	INT4 cometqTPSCPCMCtl(sCMQ_HNDL deviceHandle, UINT2 chan, , UINT1 tSlot,UINT2 rWFlag, UINT1 *pctlByte, UINT1 *ptrnkData, UINT1 *psigData)
Input field(s)	< chan > : channel from 0 to 3 <tSlot> E1 from 0 to 31, T1 from 0 to 23 <rWFlag> 0 for Read 1 for Write <ctlByte> :control byte(when rWFlag=1) bit 7: leave pcm data unchanged = 0 replace pcm data with DPGR = 1 bit 6: leave pcm data unchanged = 0 replace pcm data with trunken condition code byte = 1 bit 5: leave signaling data unchanged = 0 replace pcm data with signal trunken condition code byte = 1 bit 4: leave pcm data unchanged = 0 replace pcm data with miliwat pat = 1 bit 3: leave pcm data unchanged = 0

	replace pcm data with U LAW pat = 1 bit 2: leave pcm data unchanged = 0 invert most significant bit of pcm data = 1 bit 1,0: unused trnkData : trunk conditioning data byte sigData : signaling data byte bit 7,6,5,4: unused bit 3,2,1,0: A,B,C,D bits
Output	when Read (Wflag =1) print if success ctlByte : control byte trnkData : tronk conditioning data sigData: signaling data byte bit 7,6,5,4 unused, bit 3,2,1,0 signal trunken bits
Example	cometqRPSCPCMCtl 1 2 1 0xff 0x11 0x22 cometqRPSCPCMCtl 1 2 0

5.8.5 cometqTxTrnkCfg < chan > <enable>

Description	Transmit Trunk Conditioning
Map to API	INT4 cometqTxTrnkCfg (sCMQ_HNDL deviceHandle, UINT2 chan, UINT2 enable)
Input field(s)	< chan > : channel from 0 to 3 <enable> 0 or 1
Output	none if success
Example	cometqTxTrnkCfg 3 1

5.8.6 cometqRxTrnkCfg < chan > <enable>

Description	Receive Trunk Conditioning
Map to API	INT4 cometqRxTrnkCfg (sCMQ_HNDL deviceHandle, UINT2 chan, UINT2 enable)
Input field(s)	< chan > : channel from 0 to 3 <enable> 0 or 1
Output	none if success
Example	cometqRxTrnkCfg 3 0

5.8.7 cometqPRGDCtlCfg< chan > < genLen ><detLen> <unFrmGen> <unFrmDet> <rxPatGenLoc>

Description	Receive Trunk Conditioning
Map to API	INT4 cometqPRGDCtlCfg(sCMQ_HNDL deviceHandle, UINT2 chan, eCMQ_PRGD_PAT_LEN genLen, eCMQ_PRGD_PAT_LEN detLen, UINT2 unFrmGen, UINT2 unFrmDet, UINT2 rxPatGenLoc);
Input field(s)	< chan > : channel from 0 to 3 < genLen > : select 7 or 8 bit pattern insertion: CMQ_PRGD_PAT_8_BIT, CMQ_PRGD_PAT_7_BIT < detLen > :select 7 or 8 bit pattern detection: CMQ_PRGD_PAT_8_BIT, CMQ_PRGD_PAT_7_BIT <unFrmGen> 0 or 1 <unFrmDet> 0 or 1 <rxPatGenLoc>: location of the PRBS generator / detector. If set the pattern detector is inserted into the transmit path and the detector 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.
Output	none if success
Example	cometqPRGDCtlCfg 1 0 1 0 1 0

5.8.8 cometqPRGDPatCfg < chan > < quasiRand > <pat>

Description	This API provides configuration of the pattern type that the pseudo-random generator/detector uses. The user can specify the pattern type and select between a pseudo-random or a quasi-random sequence.
Map to API	INT4 cometqPRGDPatCfg(sCMQ_HNDL deviceHandle, UINT2 chan, UINT1 quasiRand, eCMQ_PSEUDO_RANDOM_PATTERN pat);
Input field(s)	< chan > : channel from 0 to 3 < quasiRand >: 0 or 1 <pat>: * = Comet-Quad Device CMQ_PSEUDO_RANDOM_PAT_2_TO_3RD_MINUS1 or 0, CMQ_PSEUDO_RANDOM_PAT_2_TO_4th_MINUS1 or 1, CMQ_PSEUDO_RANDOM_PAT_2_TO_5th_MINUS1 or 2, CMQ_PSEUDO_RANDOM_PAT_2_TO_6th_MINUS1 or 3, CMQ_PSEUDO_RANDOM_PAT_2_TO_7th_MINUS1 or 4, CMQ_PSEUDO_RANDOM_PAT_FRAC_T1_ACTIVATE or 5, CMQ_PSEUDO_RANDOM_PAT_FRAC_T1_DEACTIVATE or 6, CMQ_PSEUDO_RANDOM_PAT_2_TO_9th_MINUS1_O_153 or 7,

	CMQ_PSEUDO_RANDOM_PAT_2_TO_10th_MINUS1 or 8, * CMQ_PSEUDO_RANDOM_PAT_2_TO_11th_MINUS1 or 9, CMQ_PSEUDO_RANDOM_PAT_2_TO_11th_MINUS1_O_152 or 10, * CMQ_PSEUDO_RANDOM_PAT_2_TO_15th_MINUS1 or 11, CMQ_PSEUDO_RANDOM_PAT_2_TO_15th_MINUS1_O_151 or 12, CMQ_PSEUDO_RANDOM_PAT_2_TO_17th_MINUS1 or 13, CMQ_PSEUDO_RANDOM_PAT_2_TO_18th_MINUS1 or 14, * CMQ_PSEUDO_RANDOM_PAT_2_TO_20th_MINUS1 or 15, CMQ_PSEUDO_RANDOM_PAT_2_TO_20th_MINUS1_O_153 or 16, CMQ_PSEUDO_RANDOM_PAT_2_TO_20th_MINUS1_O_151 or 17, CMQ_PSEUDO_RANDOM_PAT_2_TO_21th_MINUS1 or 18, CMQ_PSEUDO_RANDOM_PAT_2_TO_22th_MINUS1 or 19, CMQ_PSEUDO_RANDOM_PAT_2_TO_23th_MINUS1_O_151 or 20, CMQ_PSEUDO_RANDOM_PAT_2_TO_25th_MINUS1 or 21, CMQ_PSEUDO_RANDOM_PAT_2_TO_28th_MINUS1 or 22, CMQ_PSEUDO_RANDOM_PAT_2_TO_29th_MINUS1 or 23, CMQ_PSEUDO_RANDOM_PAT_2_TO_31ST_MINUS1 or 24, CMQ_PSEUDO_RANDOM_PAT_ALL_ONES or 25, CMQ_PSEUDO_RANDOM_PAT_ALL_ZEROS or 26, CMQ_PSEUDO_RANDOM_PAT_ALT_ONES_AND_ZEROS or 27, CMQ_PSEUDO_RANDOM_PAT_DOUBLE_ALT_ONES_AND_ZEROS or 28, CMQ_PSEUDO_RANDOM_PAT_3_IN_24 or 29, CMQ_PSEUDO_RANDOM_PAT_1_IN_16 or 30, CMQ_PSEUDO_RANDOM_PAT_1_IN_8 or 31, CMQ_PSEUDO_RANDOM_PAT_1_IN_4 or 32, CMQ_PSEUDO_RANDOM_PAT_INBAND_LOOPBACK_ACTIVATE or 33, CMQ_PSEUDO_RANDOM_PAT_INBAND_LOOPBACK_DEACTIVATE or 34
Output	none if success
Example	cometqPRGDPatCfg 1 2 3

5.8.9 cometqPRGDErrInsCfg <errRate >

Description	configure the data generator and detector error insertion rate. For COMET only
Map to API	INT4 cometqPRGDErrInsCfg(sCMQ_HNDL deviceHandle, eCMQ_ERROR_RATE errRate);
Input field(s)	<errRate>: CMQ_ERROR_RATE_OFF or 0, CMQ_ERROR_RATE_SINGLE or 1, CMQ_ERROR_RATE_10_TO_MINUS1 or 2, CMQ_ERROR_RATE_10_TO_MINUS2 or 3, CMQ_ERROR_RATE_10_TO_MINUS3 or 4, CMQ_ERROR_RATE_10_TO_MINUS4 or 5, CMQ_ERROR_RATE_10_TO_MINUS5 or 6, CMQ_ERROR_RATE_10_TO_MINUS6 or 7, CMQ_ERROR_RATE_10_TO_MINUS7 or 8

Output	`none if success`
Example	`cometqPRGDErrInsCfg 8`

5.9 Interrupt Service Functions

NOTE:

- The Tcl console only supports Polling mode. Polling can be done by calling "cometqPoll".
- ISR and ISR Mask related APIs functions are omitted.
- For the purpose of demonstrating interrupt events, only CDRC interrupts are enabled.
- Callback functions used in this Tcl console are those provided in the Beta Driver cmq_app.c file.

5.9.1 cometqPoll

Description	`Commands the driver to poll the interrupt registers in the device.`
Map to API	`INT4 cometqPoll(sCMQ_HNDL deviceHandle);`
Input field(s)	`none`
Output	`Invoke callbacks`
Example	`cometqPoll`

5.10 Status and Statistics Functions

5.10.1 cometqForceStatsUpdate

Description	`This function forces the performance monitor counters obtained by calling cometqGetStats to be updated from the internal holding registers.  This function must be called before cometqGetStats is invoked.`
Map to API	`INT4 cometqForceStatsUpdate(sCMQ_HNDL deviceHandle);`
Input field(s)	`none`
Output	`none if success`
Example	`cometqForceStatsUpdate`

5.10.2 cometqGetStats

Description	This function retrieves framing statistics from the hardware T1/E1 performance monitoring registers. When calling this function, it is assumed that the user forced the registers to update by calling cometqForceStatsUpdate.
Map to API	INT4 cometqGetStats(sCMQ_HNDL deviceHandle, sCMQ_FRM_CNTS *pData);
Input field(s)	none
Output	framing stats
Example	cometqGetStats

5.10.3 cometqGetStatus

Description	This function retrieves status and alarms information for either the T1 or E1 framers as appropriate based on the current operating mode. Loss of signal, loss of frame, AIS and yellow alarm status information is available for T1 and E1. For E1, loss of CRC-4 multiframe, loss of signaling multiframe, and timeslot 16 RAI indication is also available.
Map to API	INT4 cometqGetStatus(sCMQ_HNDL deviceHandle, sCMQ_FRM_STATUS *pData);
Input field(s)	none
Output	framer and alarm information
Example	cometqGetStatus

5.10.4 cometqLineClkStatGet <Chan>

Description	This function provides the clock inputs status as well as the synchronization status of the clock service unit.
Map to API	INT4 cometqLineClkStatGet(sCMQ_HNDL deviceHandle, UINT2 chan, sCMQ_CLK_STATUS *pclkStat);
Input field(s)	<Chan> any number from 0 to 3
Output	clock input status and synchronization status
Example	cometqLineClkStatGet 0

5.10.5 cometqPRGDCntGet < chan >

Description	`This function retrieves the bit error count that is maintained in the pseudo-random generator/detector within the device.`
Map to API	`INT4 cometqPRGDCntGet(sCMQ_HNDL deviceHandle, UINT2 chan, UINT4 *pcount)`
Input field(s)	`< chan > : channel from 0 to 3`

Output	`print count if success`
	`total bit count or bit error count`

Example	`cometqPRGDCntGet 1`

5.10.6 cometqPRGDGetBitCnt

Description	`This function provides the user with the bit count maintained in the PRBS within a COMET device.  This function is supported by the COMET only.`
Map to API	`INT4 cometqPRGDGetBitCnt(sCMQ_HNDL deviceHandle, UINT4 *pcount);`
Input field(s)	`none`
Output	`print count if success`
Example	`cometqPRGDGetBitCnt`

5.10.7 cometqPmonSet <Chan> <enable>

Description	`Enable/Disable one second update of Auto Performance Report Monitoring (APRM).  This function is valid only in T1 ESF framing mode.`
Map to API	`INT4 cometqPmonSet(sCMQ_HNDL deviceHandle, UINT2 chan, eCMQ_APRM_ACTION action);`

Input field(s)	`<Chan> any number from 0 to 3`
	`<action>: CMQ_AUTO_PMON_UPDATE_DISABLE or 0,`
	`          CMQ_AUTO_PMON_UPDATE_ENABLE or 1,`
	`          CMQ_AUTO_PMON_UPDATE_MAN or 2`

Output	`none if success`
Example	`cometqPmonSet  0 1`

5.10.8 cometqPmonReportGet <Chan>

Description	`This API returns the current one second performance report.  This function is valid only in T1 ESF framing mode.`

Map to API	`INT4 cometqPmonReportGet(sCMQ_HNDL deviceHandle, UINT2 chan, sCMQ_STAT_APRM *pPmonReport);`
Input field(s)	`<Chan> any number from 0 to 3`

Output	`print pMonReport if success`
	`performance report buffer`

Example	`cometqPmonReportGet 3`

5.11 Device Diagnostics

5.11.1 cometqTestReg

Description	This function verifies hardware register integrity by writing and reading back values.
Map to API	`INT4 cometqTestReg(sCMQ_HNDL deviceHandle);`
Input field(s)	none
Output	none if success
Example	`cometqTestReg`

5.11.2 cometqLoopFramer<Chan><type>

Description	Clears / Sets a loopback of type line, payload, or digital within the E1/T1 framer section of the device. Note that when performing a line loopback, the transmit jitter attenuators reference and output clock divisors are set to 0x2F, the transmit timing options register is modified to jitter attenuated loop timing, and the transmit elastic store is enabled.
Map to API	`INT4 cometqLoopFramer(sCMQ_HNDL deviceHandle, UINT2 framer, eCMQ_LOOPBACK_TYPE type);`

Input field(s)	`<Chan> any number from 0 to 3`
	`<type>: CMQ_LOOPBACK_NONE or 0,`
	`CMQ_LOOPBACK_DIGITAL or 1,`
	`CMQ_LOOPBACK_LINE or 2,`
	`CMQ_LOOPBACK_PAYLOAD or 3`

Output	none if success
Example	`cometqLoopFramer 0 1`

5.11.3 cometqLoopTslots<Chan><timeSlot><enable>

Description	This function enables or disables DS0 loopback on specified DS0s. To facilitate DS0 loopback, the receive elastic store must be bypassed. As such, the current receive backplane configuration must be clock master otherwise DS0 loopback is not possible.
Map to API	INT4 cometqLoopTslots(sCMQ_HNDL deviceHandle, UINT2 framer, UINT4 timeSlot, UINT2 enable);
Input field(s)	<Chan> any number from 0 to 3 <timeSlot> timeslot <enable> 0 or 1
Output	none if success
Example	cometqLoopTslots 0 1 1

5.11.4 cometqTxAnalogByp<Chan><enable>

Description	This function enables or disables bypass of the TXRING and TXTIP outputs to use the digital TDAT and TCLKO lines.
Map to API	INT4 cometqTxAnalogByp(sCMQ_HNDL deviceHandle, UINT2 chan, UINT1 enable);
Input field(s)	<Chan> any number from 0 to 3 <enable> 0 or 1
Output	none if success
Example	cometqTxAnalogByp 0 1

5.11.5 cometqRxAnalogByp<Chan><enable>

Description	This function enables or disables bypass of the RXRING and RXTIP outputs to use the digital RDAT and RCLK lines.
Map to API	INT4 cometqRxAnalogByp(sCMQ_HNDL deviceHandle, UINT2 chan, UINT1 enable);
Input field(s)	<Chan> any number from 0 to 3 <enable> 0 or 1
Output	none if success
Example	cometqRxAnalogByp 1 1

6 REFERENCES

1. PMC-Sierra, Inc., PMC-1991237, "COMET-QUAD Evaluator Board", December 2000, Issue 2.

2. PMC-Sierra, Inc., PMC-2000151, "COMET-QUAD Programming Guide", August 2001, Issue 2.

3. PMC-Sierra, Inc., PMC-1990315, "COMET-QUAD Data Sheet", May 2001, Issue 6.

4. PMC-Sierra, Inc., PMC-2001401, "COMET and COMET-QUAD Driver Manual", June 2001, Issue 2.

CONTACTING PMC-SIERRA, INC.

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

Tel: (604) 415-6000

Fax: (604) 415-6200

Technical Support:	http://www.pmc-sierra.com/techSupport/requestSupport.html
	apps@pmc-sierra.com
	(604) 415-4533
Document Ordering:	document@pmc-sierra.com
Corporate Information:	info@pmc-sierra.com
Web Site:	http://www.pmc-sierra.com

PMC-2001854 (R2) Issue date: August 2001