RS8953B

Application and Channel Unit Software Developer's Guide

100418C
May 2000

CONEXANT™

Reader Response: Conexant strives to produce quality documentation and welcomes your feedback. Please send comments and suggestions to tech.pubs@conexant.com. For technical questions, contact your local Conexant sales office or field applications engineer.

Table of Contents

List of Figures

List of Tables

1.0 Introduction

This document describes the application and channel unit code distributed with the Conexant Evaluation Module Systems (EVMs). The application and channel unit code and bit pump code provide a complete DSL application. Refer to the *ZipWire Software User Guide* (100417C) for details of the bit pump code.

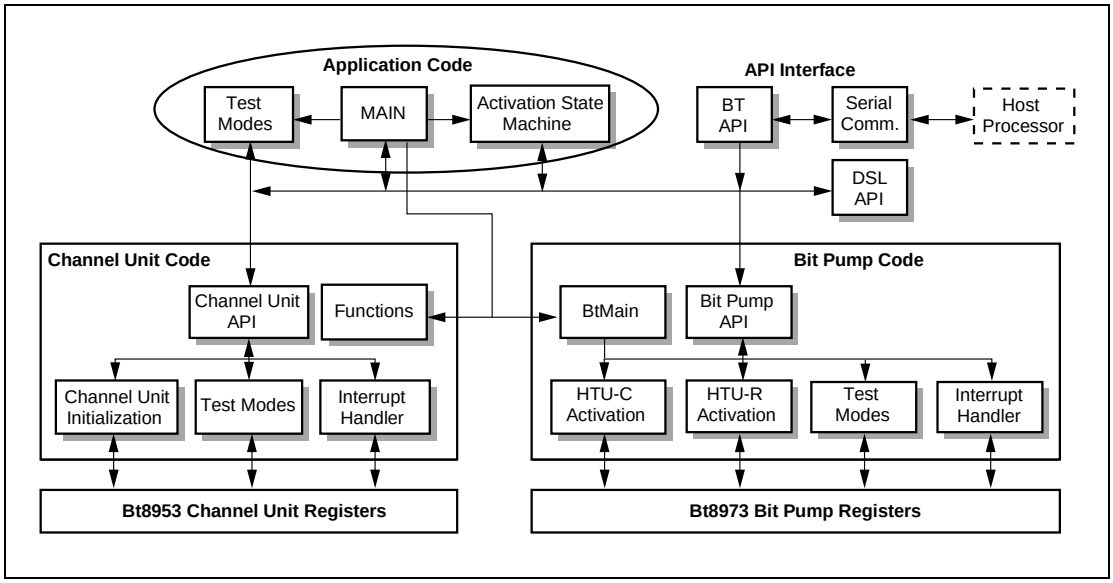
In Versions 5.3 and earlier, the channel unit code was primarily regarded as sample code for the evaluation boards. The code was written so the customer would be able to use it; however, it was incomplete and required extensive efforts by the end user.

Version 6.x is a more thorough software package. The main focus of the channel unit code is for the standard 1T1, 2T1, 1E1, 2E1, and 3E1 HDSL transport applications.

The channel unit is an extremely flexible device that can fit many applications. Even with all of the improvements, for many applications, the customer must modify the code.

Figure 1-1 illustrates the software overview and interface to the HDSL chip sets. The bit pump code manages the transceiver functionality and the channel unit code is responsible for the framing functionality. The application code lays on top of the bit pump code and channel unit code and calls them to activate and maintain the system. The serial interface provides customer control of the system through a host computer.

Figure 1-1. Software Overview



1.1 What's New in Channel Unit Version 6.x

The following features are added to the channel unit code version 6.x. See Chapter 6.0 for a detailed description of each feature.

1.1.1 Bt8953A Revision C and RS8953B Support

Channel unit code version 6.x supports the newest channel unit devices: Bt8953A Rev. C and RS8953B.

1.1.2 RS8973 versus Bt8970 Line Card Support

The channel unit PLL and stuffing values are based on the RS8973 XOUT values. The Bt8970 device will not work with any channel unit device (including the Conexant EVMs) because the Bt8970 XOUT frequency is different than the RS8973 XOUT frequency.

Customers who have both a Bt8970 and channel unit device must define the MCLK_1024 directive.

1.1.3 EOC Protocol

Previous versions of channel unit code did not support the ANSI/ETSI Embedded Operation Channel (EOC) protocol. In channel unit code version 6.x, the EOC protocol has been added and can be enabled by the CU_EOC compiler flag defined in TYPEDEFS.H.

1.1.4 Dynamic Loop Reversal and Master Loop

In channel unit code version 6.x, the master loop and the loop connections between the HTU-C and HTU-R are no longer fixed and can change dynamically.

1.1.5 Performance Monitoring

In channel unit code version 6.x, Performance monitoring has been added to keep a history of CRC and FEBE errors at different time intervals. Performance monitoring can be enabled with the PERF_MONITOR compiler flag defined in TYPEDEFS.H.

1.1.6 Channel Blocking

Standard 2T1/2E1 channel blocking is implemented in channel unit code version 6.x.

1.1.7 Bt8370 Support

The Bt8370 T1/E1 framer/LIU support is added to the channel unit code version 6.x. The framer code configures the Bt8370 for either T1 or E1 mode. Bt8370 support can be enabled with the BT8370_FRAMER compiler flag defined in TYPEDEFS.H.

1.2 Features and Functionality Not Supported

The following features or functionality are not by channel unit code version 6.x:

Bt8953A Evaluation Boards: Support for the Bt8510, Bt8360, and Bt8069 are compiler options in version 6.x, which minimizes waste on ROM space. To revert to the Bt8510/Bt8360/Bt8069 framers, the compiler options must be enabled in the "TYPEDEFS.H" file. When using the Bt8510 or Bt8360 channel unit EVMs, the Bt8069 option should be enabled.

Loop Relativity: The term "loop relativity" is used to describe the EVM motherboard slot where each line card is connected. Currently, if 1E1 or 1T1 is simulated, the line card must be in slot one. If the line card is in slot two or three, some application and channel unit code features do not function properly.

Multiple Channel Unit Devices: The current release does not support multiple channel unit devices. Contact Conexant for assistance.

2.0 EVM Specific

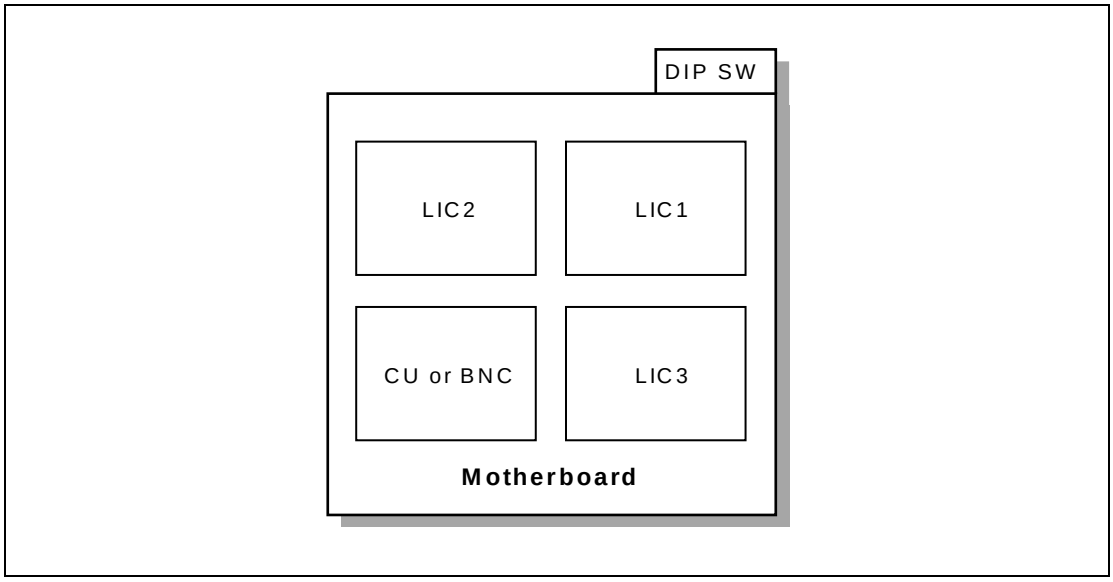
This section describes any interaction of the software that is specific to the HDSL EVMs. Customers may need to modify the EVM-specific code for their platform.

2.1 HDSL EVM Hardware

The modular HDSL EVM hardware is designed to accommodate the standard 1T1, 2T1, 1E1, 2E1, and 3E1 configurations. As illustrated in Figure 2-1, the HDSL EVM terminal unit consists of the following:

- Motherboard
- 1, 2, or 3 Line Interface Cards (LIC)
- Channel Unit (CU) card or BNC connector card

Figure 2-1. HDSL EVM Terminal Unit



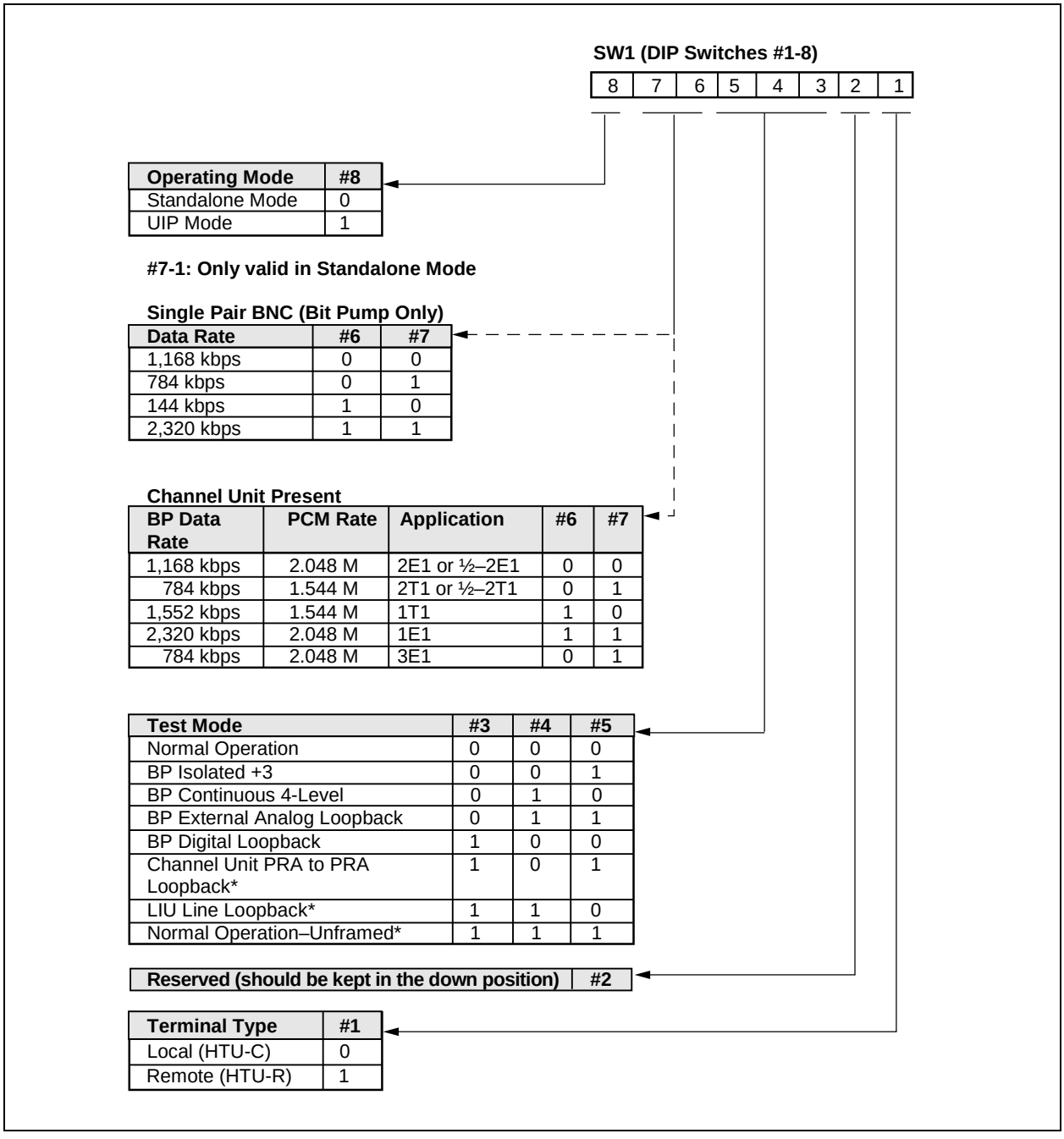
Although the code is written for the HDSL EVMs, it is also written with consideration of porting to other hardware environments.

2.1.1 DIP Switch

In the HDSL EVM, the Dual In-line Package (DIP) switch is used to determine the system configuration, test modes, etc. Customers must mimic this selection process based on their system architecture or hardware. Figure 2-2 illustrates the DIP switch mode configuration.

NOTE: The DIP switch numbers on the motherboard are transposed compared with the P1 port (DipSw #1 is connected to P1 #7, DipSw #2 is connected to P1 #6, etc.). The DIP switch is labeled 1–8, and the P1 port is labeled 0–7.

Figure 2-2. DIP Switch Mode Configuration



2.1.2 Channel Unit LEDs

The three CU_LED registers are 8-bit write only registers that display status information via LEDs the channel unit and line interface cards. The LEDs are lit when a 1 is present in the corresponding register bit. Table 2-1 lists the memory map for the CU_LED registers. Figure 2-3 through Figure 2-5 define the CU_LED Registers.

Table 2-1. CU_LEDn Memory Map

Address Range	Description
0xC600–0xC7FF	CU_LED1 (Write Only)
0xC800–0xC9FF	CU_LED2 (Write Only)
0xCA00–0xCBFF	CU_LED3 (Write Only)

Figure 2-3. CU_LED1 Register

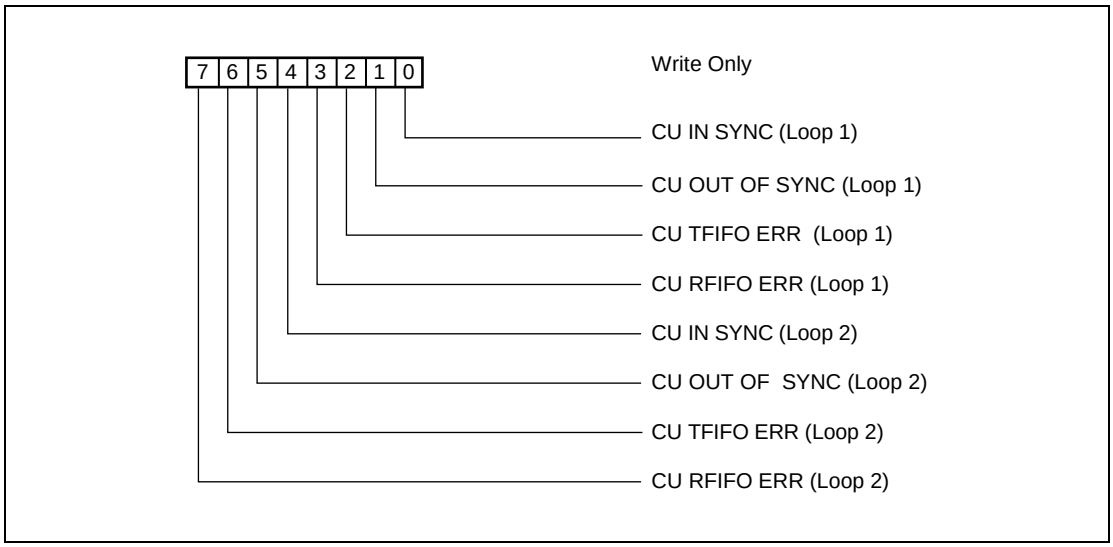


Figure 2-4. CU_LED2 Register

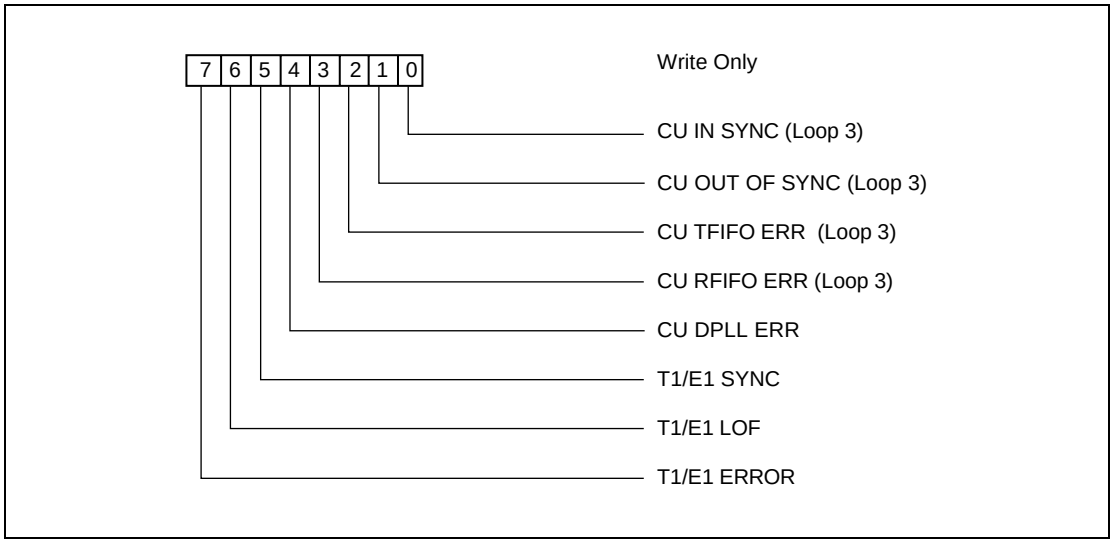
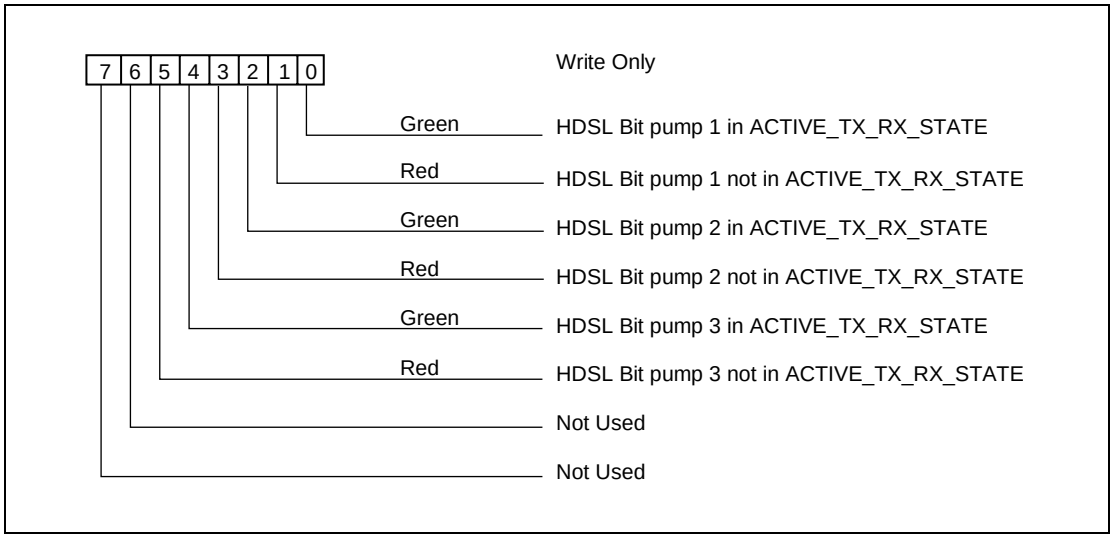


Figure 2-5. CU_LED3 Register



2.2 General Purpose Timers

The 80C32 timer #0 implements a general purpose timer function (timer #1 is used as a baud rate generator). The timer structure is a two-dimensional array to index multiple bit pumps and general purpose timers. Currently, one general purpose timer per bit pump is used for the LOSWT (loss of sync word timer). (See Section 11.13 for a detailed description.) See Table 2-2 lists General Purpose Status bits definitions.

```
/* Found in TIMER.H */
/* Timer indexes */
#define _NO_GEN_PURPOSE_TIMERS 1
#define PENDING_DEACTIVATE_TIMER 0

/* Timer Structure */
typedef union
{
    BP_U_8BIT reg;
    struct
        {
        BP_BIT_FIELD state:1;
        BP_BIT_FIELD complete:1;
        BP_BIT_FIELD continuous:1;
        BP_BIT_FIELD reserved:5;
        } bits;
} GEN_PURPOSE_TIMER_STATUS;

typedef struct
{
    BP_U_16BIT counter_value;
    GEN_PURPOSE_TIMER_STATUS status;
} GEN_PURPOSE_TIMER;

typedef struct
{
    BP_U_16BIT load_value;
    BP_U_32BIT elapsed_counter;
} GEN_PURPOSE_CONT_TIMER;

/* Found in TIMER.C */
static GEN_PURPOSE_TIMER
       gen_timer[_NO_OF_LOOPS][_NO_GEN_PURPOSE_TIMERS];
static GEN_PURPOSE_CONT_TIMER
       gen_cont_timer[_NO_OF_LOOPS] [_NO_GEN_PURPOSE_TIMERS];
```

Table 2-2. General Purpose Status Bit Definitions

Bit	Description	0 Value	1 Value
0	State	Disabled	Enabled
1	Complete (Expired)	Not complete	Complete
2	Continuous	Not continuous	Continuous
3–7	Reserved	—	—

NOTE(S):
1. To add more timers, increase the _NO_GEN_PURPOSE_TIMERS definition, and add the appropriate timer index definitions.
2. If the user employs a microcontroller other than 80C32 and wants a general purpose timer, similar functions should be implemented.

2.2.1 Continuous Mode

In this mode, the timer continuously counts the specified time interval. The elapsed_counter infinitely tracks the number of continuous timer expirations. The load_value sets the timer interval when the continuous timer is reloaded. The _ContinuousGenPurposeTimer() function enables or disables the continuous timer.

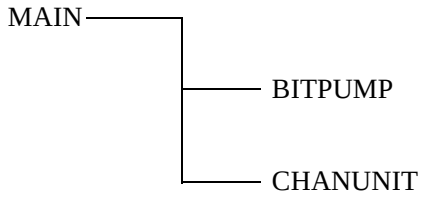
2.2.2 One Second Timer

The one second timer is a continuous mode timer, used by the application code to track when to:

- Estimate bit pump activation time.
- Run a watchdog timer in the WAIT_FOR_LOS and WAIT_FOR_LOST states.
- Determine when to update the ZipStartup coefficients.

3.0 Directory Structure

The directory structure is partitioned into three directories: the MAIN (application-specific) directory as the root, the bit pump code as a subdirectory labeled BITPUMP, and the channel unit code as a subdirectory labeled CHANUNIT.



The application examples, hex files, build script files, etc., are located in the MAIN directory. The bit pump source code and header files are in the BITPUMP subdirectory. The channel unit source code and header files are in the CHANUNIT subdirectory.

NOTE: It is recommended that the application have separate subdirectories for the bit pump and channel unit code to more easily manage the software and future upgrades. However, the user can specify their own directory and file structure.

3.1 MAIN (Application) Directory

Table 3-1 lists all C source files in the MAIN (application) directory.

Table 3-1. Source Files Under the Main Directory

Files	Description
DSL_MAIN.C	Main() routine for HDSL EVM system.
DSL_INIT.C	HDSL initialization functions.
DSL_ASM.C	HDSL activation state manager (ASM).
DSL_MAN.C	HDSL dynamic loop control functions.
DSL_MISC.C	HDSL miscellaneous functions.
DSL_API.C	HDSL application-level API commands.
BT_API.C	HDSL API command-wrapper.
ZIPVALID.C	ZIPSTARTUP validation function.(1)
TIMER.C	80C32 general purpose timer functions.

NOTE(S):
(1) ZIPSTARTUP is an add-on feature that offers an 8-fold reduction to start-up times. For ordering information contact Conexant.

3.2 BITPUMP Directory

Table 3-2 lists all C source files in the BITPUMP subdirectory.

Table 3-2. Source Files Under BITPUMP Directory

Files	Description
API.C	Bit pump APIs.
BTMAIN.C	Bit pump main routine.
BTINT.C	Bit pump interrupt handler.
BITPUMP.C	Handle bit pump interrupt status and temperature changes.
INIT51.C	Routines for initializing 80C32 interrupts, timers, and serial port.
MAIL.C	RS-232 communication support. Manage receive messages, transmit messages, and read/write to mail boxes.
MONITOR.C	RS-232 Printf support (used only with TDEBUG compiler flag).
SERINT.C	Serial communication interrupt handler.
SUC.C	Startup control process for HTU-C terminal.
SUR.C	Startup control process for HTU-R terminal.
SUUTIL.C	Startup utility routines common to HTU-C and HTU-R.
TESTMODE.C	Performs test modes: loopbacks, isolated transmit pulse, scrambled 1s, VCXO control voltage test. Performs bit pump self-test.
USER.C	User-modifiable code. Definitions of the absolute addresses of bit pump devices, and routines for initializing the bit pump pointers to these addresses.
UTIL.C	Bit pump utility routines.
ZIPSTART.C	Save and load the ZipStartup registers (requires ZipStartup software package).

3.3 CHANUNIT Directory

Table 3-3 lists all C source files in the CHANUNIT directory.

Table 3-3. Source Files Under CHANUNIT Subdirectory

Files	Description
CU_INIT.C	Channel unit initialization functions.(1)
CU_INT.C	Channel unit interrupt handlers.
CU_MAP.C	Transmit routing table, receive combination table, transmit payload map and receive payload map initialization functions.
CU_ASM.C	Channel unit functions called by ASM.
CU_LOOP.C	Channel unit functions for dynamic loop control.
CU_UTILS.C	Channel unit utility functions.
CU_EOC.C	Embedded operation channel protocol and task handlers.
CU_PERF.C	Performance monitoring functions and global variables.
CU_API.C	Channel unit APIs.
CU_LED.C	Channel unit LED functions.
FRAMER.C	T1/E1 framer functions.
FRMR_API.C	T1/E1 framer APIs.

NOTE(S):
(1) In channel unit V 5.3 and earlier, the channel unit initialization functions were located in CHANUNIT.C rather than CU_INIT.C.

3.4 Header File Dependencies

Table 3-4 lists the header file dependencies for the application, bit pump, and channel unit code.

Table 3-4. Header File Dependencies

Files	Source Files Depend on…
Application Files	Application Header Files: DSL_MAIN.H, DSL_INIT.H, DSL_INCL.H, DSL_MAN.H, DSL_MISC.H, DSL_API.H, BT_API.H, ZIPVALID.H, TIMER.H, TYPEDEFS.H Bit pump Header Files: BTMAIN.H, API.H, USER.H, BITPUMP.H, SUC.H, TESTMODE.H, EXTERN.H, ZIPSTART.H, MAIL.H, INIT51.H Channel Unit Header Files: CU_API.H, CU.H, FRMR_API.H, FRAMER.H, CU_TABLE.H, CU_ASM.H, CU_MAP.H, CU_LOOP.H, CU_UTILS.H, CU_INIT.H, CU_EOC.H, CU_PERF.H
Bit pump Files	Application Header Files: BT_API.H, TYPEDEFS.H Bit pump Header Files: API.H, BITPUMP.H, BTMAIN.H, EXTERN.H, FIFO.H, INIT51.H, MAIL.H, PTRDEF.H, SERINT.H, SUC.H, SUR.H, SUUTIL.H, TESTMODE.H, USER.H, UTIL.H, ZIPSTART.H Channel Unit Header Files: none
Channel Unit Files	Application Header Files: DSL_MAIN.H, BT_API.H, TYPEDEFS.H Bit pump Header Files: BITPUMP.H, BTMAIN.H, API.H, EXTERN.H, MAIL.H, SERINT.H, TESTMODE.H Channel Unit Header Files: CU_API.H, CU.H, FRMR_API.H, FRAMER.H, CU_TABLE.H, CU_ASM.H, CU_MAP.H, CU_LOOP.H, CU_UTILS.H, CU_INIT.H, CU_EOC.H, CU_PERF.H

3.5 TYPEDEFS.H

TYPEDEFS.H is located in the MAIN directory and contains the data type definitions for a Keil compiler. It also contains some compiler flag definitions (see Chapter 4.0 for a detailed description of each compiler flag). Table 3-5 lists the data types that must be specified.

Table 3-5. Data Type Definitions

Data Type	Description	Keil Equivalent(1)	GNU C Equivalent(2)
BP_BIT_FIELD	1-bit field	unsigned char	unsigned char
BP_S_8BIT	Signed 8-bit	char	char
BP_U_8BIT	Unsigned 8-bit	unsigned char	unsigned char
BP_S_16BIT	Signed 16-bit	short	short
BP_U_16BIT	Unsigned 16-bit	unsigned short	unsigned short
BP_S_32BIT	Signed 32-bit	long	int
BP_U_32BIT	Unsigned 32-bit	unsigned long	unsigned int
BP_TABLE	Signed 16-bit	short	short
BP_CONSTANT	Constant	code	const
BP_VOLATILE	Volatile	N/A	volatile
BP_DATA	Internal Memory	data(3)	N/A
BP_IDATA	Indirect Internal Memory	idata(3)	N/A
BP_PDATA	External Memory	pdata(3)	N/A
BP_XDATA	External Memory	xdata(3)	N/A

NOTE(S):
1. The code is targeted towards the Keil compiler.
2. Assuming 32-bit GNU C compiler.
3. The data, idata, pdata, and xdata types are unique to the Keil 80C32 compiler.

4.0 Compiling and Linking Application Examples

Table 4-1 lists the HEX files that can be built.

Table 4-1. HEX Files

Application Example	HEX File	Description
Single Processor HTU-C Terminal	ZIPWIREC.HEX	EVM version (standalone and User Interface Program control).
TDEBUG	TDEBUG.HEX	Bit pump Debug version (bit pump only).

NOTE(S):
1. It is assumed the user knows how to compile and link the source code with the necessary compiler and linker flags.
2. The HDSL EVM Bt8952-001 motherboard external RAM address begins at address 0x8000.

4.1 Using the Keil uVision Project Manager

The ZIPWIRE.PRJ and TDEBUG.PRJ are Keil uVision Project Files. The uVision tool includes an Integrated Development Environment (IDE) to edit, compile, and link the code. The Keil compiler is specific to the 8051 microcontroller family. Table 4-2 lists the compiler directives.

4.2 Using a Makefile

The SCRIPT.BLD file in the MAIN directory contains information to generate the different application examples. The SCRIPT.BLD file is an Intersolv® Configuration Builder 5.0 script file (makefile). The SCRIPT.BLD file provides the ability to generate additional HEX files not specified in Table 4-1.

4.3 Linker Flags

The LFLAGS variable specifies the linker flags. All HEX files use the same linker flags:

```
XDATA(8000) RAM(256)
```

The XDATA(8000) specifies the base address of the external RAM memory mapping. The RAM(256) specifies that the micro controller contains 256 bytes of internal RAM.

4.4 Compiler Flags

The CFLAGS variable specifies the compiler flags which are unique to each application. There are two aspects to the CFLAGS list: 8032 compiler options and Bt8970 bit pump compiler switches, denoted by the DF<directories>. Each bit pump switch is defined in the *ZipWire Software User Guide* (100417C).

The CHANNEL_UNIT flag in SCRIPT.BLD determines if the channel unit code is compiled in the build. The default is set to 1; set this flag to 0 to not compile in the channel unit code.

Table 4-2. Compiler Directives

Project File	Keil Project Options													typedefs.h[1]															btmain.h
	C51	ADD_DELAY	PDATA_MODE	SER_COM	HTUR	HTUC	BER_METER	ERLE	SINGLE_LOOP	TWO_LOOPS	REPEATER	CHAN_UNIT	ZIP_START	PERF_MONITOR	CHANNEL_BLOCK	CU_EOC	MCLK_1024	TEMP_ENV	CU_LED	SP_API	CU_2T1	CU_2E1	CU_3E1	CU_CUSTOM	T1E1_FRAMER	BT8370_FRAMER	BT8360_FRAMER	BT8510_FRAMER	BT8069_LIU	_NO_OF_LOOPS[2]
ZIPWIRER	X	X	X	X	X		X	X				X				X	X	X	X	X	X	X	X	X	X	X				3
ZIPWIREC	X	X	X	X		X	X	X				X				X	X	X	X	X	X	X	X	X	X	X				3
ZSTARTR	X	X	X	X	X		X	X				X	X			X	X	X	X	X	X				X	X				3
ZSTARTC	X	X	X	X		X	X	X				X	X			X	X	X	X	X	X				X	X				3
ZIPREG	X	X	X	X	X	X	X	X		X	X	X				X	X	X				X			X	X				2
TDEBUG	X	X	X		X	X	X	X										X												3

NOTE(S):

(1) The following are default options in typedefs.h:
BP_MASK_INTERRUPTS, NO_INDIRECT_RAM_VARS, PRINTF_NOT_SUPPORTED are not defined.
If CHAN_UNIT is defined, then MCLK_1024, CU_BUG is defined.
If ZIP_START is defined, then ZIP_START_EC_BUG is defined.

(2) The following are default options in btmain.h:
If ZIP_SOCKET is defined, then _NO_OF_LOOPS = 2.
If TWO_LOOPS, SINGLE_LOOP and ZIP_SOCKET are not defined, then _NO_OF_LOOPS = 3.

4.4.1 CHAN_UNIT

The CHAN_UNIT flag tells the compiler whether to compile in the channel unit code.

4.4.2 TWO_LOOPS

The TWO_LOOPS compiler flag, defined in the bit pump software, is also used by the channel unit code. If the TWO_LOOPS is declared, the channel unit code only allocates memory for the first two loops, and the developer is unable to access the third Loop.

4.4.3 CU_EOC

When the CU_EOC compiler flag is defined, the EOC-related operations are available in the channel unit code. To make ZIP_START work, CU_EOC should be defined, because it is used in the ZIP_STARTUP validation procedure.

4.4.4 PERF_MONITOR

When the PERF_MONITOR compiler flag is defined, performance monitoring-related functions, such as available seconds for each loop since power-on, CRC and FEBE history maintenance, are available in the channel unit code.

4.4.5 CU_2T1

The CU_2T1 compiler flag enables ANSI standard 2T1 operation in the channel unit code. Two HDSL links are established with each link carrying half the payload from T1. The data rate for each link is 784 kbps.

4.4.6 CU_2E1

The CU_2E1 compiler flag enables ETSI standard 2E1 operation in the channel unit code. Two HDSL links are established with each link carrying half the payload from E1. The data rate for each link is 1168 kbps.

4.4.7 CU_3E1

The CU_3E1 compiler flag enables ETSI standard 3E1 operation in the channel unit code. Three HDSL links are established with each link carrying one-third the payload from E1. The data rate for each link is 784 kbps.

4.4.8 CU_1T1

The CU_1T1 compiler flag enables the 1T1 operation in the channel unit code. When this compiler is used, only one HDSL loop is established. The data rate for the link is 1552 kbps.

4.4.9 CU_1E1

The CU_1E1 compiler flag enables the 1E1 operation in the channel unit code. When this compiler is used, only one HDSL loop is established. The data rate for the link is 2320 kbps.

4.4.10 CU_CUSTOM

When CU_CUSTOM is enabled, customers can experiment with different configurations for the system by modifying the CU_CUSTOM entries in the RATE_VALUES[][] array (see Section 10.5)

4.4.11 T1E1_FRAMER

The T1E1_FRAMER compiler flag enables the codes in the channel unit code that handles the T1 or E1 framer. It should always be defined when the CHAN_UNIT compiler flag is defined.

4.4.12 CU_LED

The CU_LED compiler flag enables the codes in the channel unit code that handles the channel unit LED updates. It should always be defined when the CHAN_UNIT compiler flag is defined.

4.4.13 BIT_REVERSE

The BIT_REVERSE compiler flag controls the bit field definitions. When defining a C structure containing bit fields, such as in the text below, the first bit is normally allocated to the least significant bit in the byte.

```
typedef struct {
BP_BIT_FIELD scr_en:1;
BP_BIT_FIELD two_level:1;
BP_BIT_FIELD icrc_err:1;
BP_BIT_FIELD sync_sel:1;
BP_BIT_FIELD hoh_en:1;
BP_BIT_FIELD force_one:1;
BP_BIT_FIELD tx_err_en:1;
BP_BIT_FIELD reserved:1;
} TCMD_1;
```

NOTE: This fact is critical when bit fields are used to access external addresses.

The BIT_REVERSE flag must be declared if the compiler allocates the first bit-field to the most significant bit in the byte.

4.4.14 ZIPSOCKET

The ZIPSOCKET compiler flag is for the Zipsocket product and is not used in the normal EVM system.

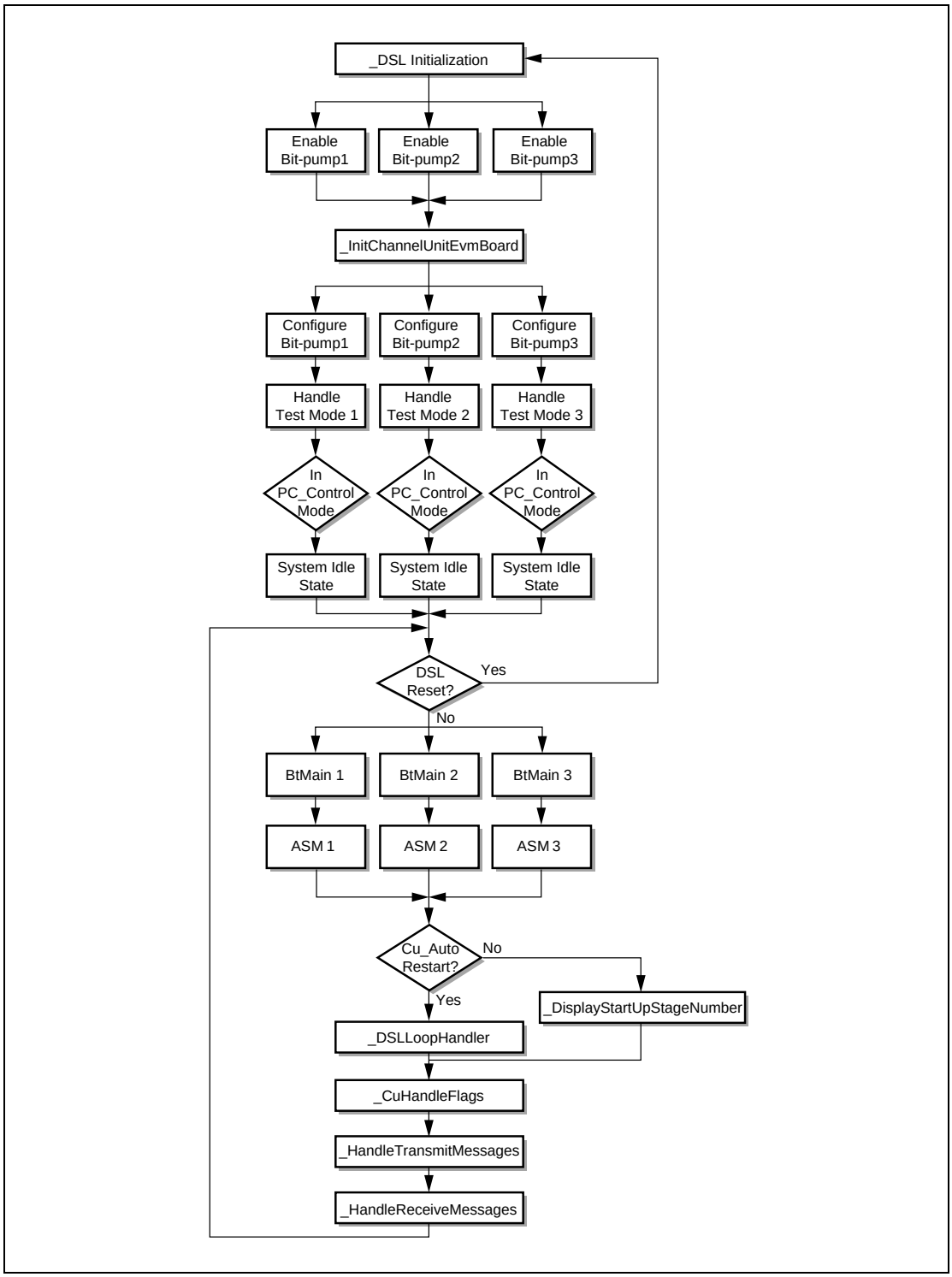
5.0 Application Code

This section describes an overview of the application code.

5.1 Software Flow

Figure 5-1 illustrates a detailed block diagram of the main program flow.

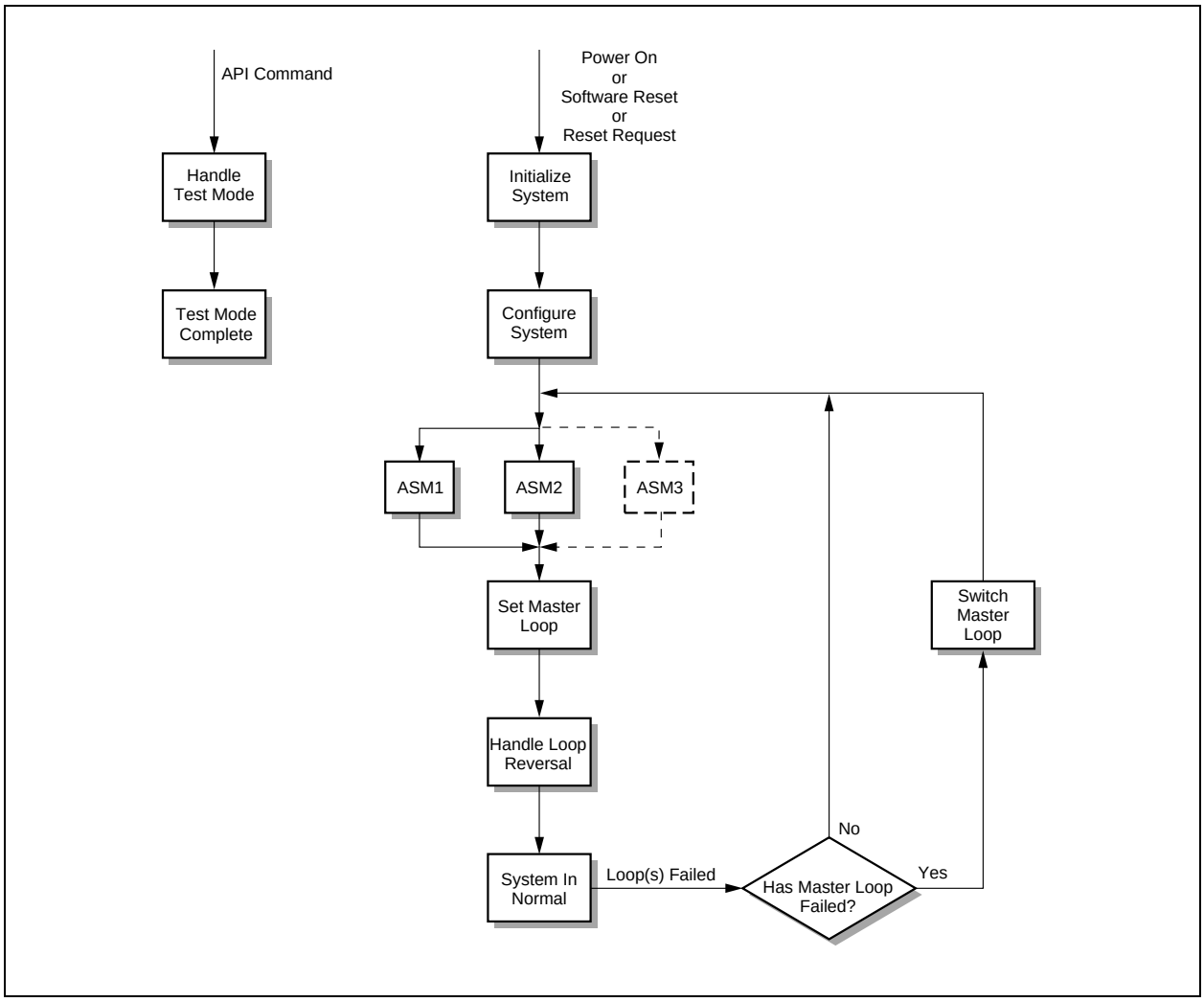
Figure 5-1. Main Program Flow



5.2 DSL Loop Manager

Figure 5-2 illustrates the functional overview of the DSL manager. The DSL manager administers system initialization, configuration, activation state management, HDSL link, and any test mode operation.

Figure 5-2. DSL Manager



5.3 Software and Devices Initializations

The functions listed in Table 5-1 initialize the software (hardware pointers, timers, etc.). These functions are called in the _DSLInitialization() function.

The API commands listed in Table 5-2 initialize the bit pump device. These functions are called in the EnableBitpump() and ConfigureBitpump() functions.

The API commands listed in Table 5-3 initialize the channel unit and framer devices. These functions are called in the _InitChannelUnitEvmBoard() function.

The configure channel unit command initializes all channel unit registers for the specified configuration. The configure framer and configure LIU commands initialize the framer and LIU for transparent operation.

Table 5-1. Software Initialization Functions

Functions	Descriptions
_BtSwPowerUp()	Software power-up initialization.
_MaskBtHomerInt()	Masks for all bit pumps, interrupts.
_Init8051()	8051 configuration initialization. Calls initialization routines for interrupts, internal timers, and serial port parameters (assuming 80C32 family processor).
_InitGenPurposeTimers()	Initialize the interrupt #1 which decrements the timer counters when timer #0 expires.
_CuInitAddress()	Initialize the channel unit EVM board base addresses.

Table 5-2. Bit Pump Initialization Commands

API Command Name	Parameter	Remarks
Bit pump ON/OFF	_PRESENT	Turn bit pump ON.
Symbol Rate	Symbol rate is set according to SW1.	API value = (Symbol Rate / 4000) i.e., 98 = 392k / 4000 (784 kbps data rate).
Terminal Type	_HTUC or _HTUR	Central office or remote terminal.
Start-up Sequence Source	_EXTERNAL	Use externally generated scrambled data from channel unit during activation.
Transmit Scrambler	_BYPASS	Bypass bit pump scrambler.(1)
Receive Scrambler	_BYPASS	Bypass bit pump descrambler.(1)
Framer Format	_SERIAL_SWAP	Sign bit followed by Magnitude bit.
Other Side Bit Pump	_BT	Assume other terminal is a Conexant bit pump.
LOST Time Period	10	Set LOST = 1 second.
Auto Tip/Ring Reversal	_AUTO_TIP_RING_OFF	Disable tip/ring reversal by software because channel unit has automatic tip/ring reversal logic.

NOTE(S):
(1) Channel unit provides necessary scrambled data.
2. The bit pump on and off, and symbol rate commands must be called first and second, respectively. The other API commands can be called in any order.

Table 5-3. Channel Unit Initialization Commands

API Command Name	Parameter	Remarks
Terminal Type	_HTUC or _HTUR	Central office or remote terminal.
Configure Channel Unit	_2T1, _2E1, _3E1, _1T1, or _1E1	Configuration is set according to SW1 (see Figure 2-2).
Configure Framer	_BT8360(T1) or _BT8510(E1) or _BT8370(T1/E1)	Configuration is determined by querying the framer type.
Framer PCM Mode	T1 or E1	Configuration is set according to SW1 (see Figure 2-2).
Configure LIU	_ELS_T1_0_110(T1) or _ELS_E1_PCM30(E1)	Configuration is based on the framer PCM mode.

5.4 Activation State Manager

This section describes how the Activation State Manager (ASM) is implemented using the bit pump and channel unit code. The ASM is based on the ETSI TS-101-135 and ANSI T1E1.4 HDSL specifications.

For each state, the functions of the application code, the bit pump code, and the channel unit code are unique. The application code accesses the bit pump code and channel unit code by calling the _BtControl() and _BtStatus() functions. These functions, in turn, call _BitpumpControl() and _BitpumpStatus() to handle bit pump-related requests, and _CuControl() and _CuStatus() to handle channel unit-related requests. See Section 11.10 for details on API function structure.

The ASM must issue only three API commands to control the bit pump portion, assuming the bit pump is properly configured using appropriate API commands. The bit pump API commands and their descriptions are as follows:

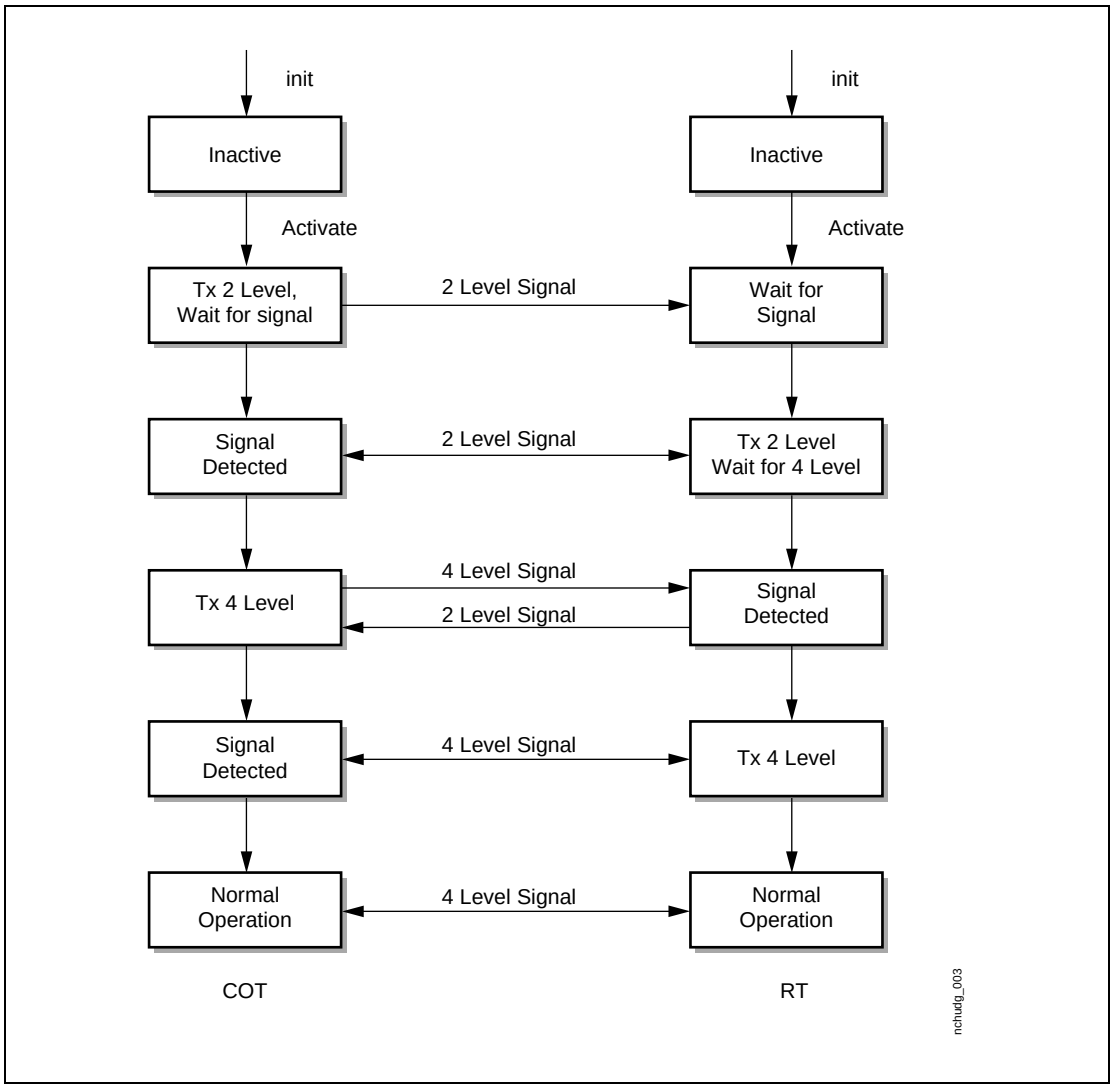
_STARTUP_STATUS	Monitors the bit pump activation status, which allows the application code to determine what state the bit pump is in and to detect any error conditions.
_ACTIVATE	Initiates the bit pump activation.
_DEACTIVATE	Deactivates the bit pump.

The ASM controls the channel unit by issuing the following list of API commands and functions. In this list, API commands are all capitals, and functions are denoted by the parentheses ():

_CU_USE_SAME_TAP	Configure Tx/Rx to use HTU-C SCR/DSCR tap #5 while in normal operation.
_CU_SYNC	Monitors the channel unit sync status.
_CU_FORCE_SCR_ONES	Configure the channel unit to transmit framed scrambled 1s; both the overhead and payload are set to all 1s.
_CuConfigureTransmitS1()	Configure the channel unit for S1 data; overhead enabled but the payload data is still all 1s.
_CuSetRtrInd()	Set the RTR (Ready To Receive) indicator bit.
_CuForceOnes()	Enable the payload data (by disabling the force payload 1s).
_CuSetPid()	Initialize the loops pair ID transmitted by the HTU-C.
_CuSetPidToAllOnes()	Initialize the loops pair ID transmitted by the HTU-R to all 1s.
_CU_TRANSMIT_PAYLOAD	Complete channel unit configuration for normal operation.

Figure 5-4 illustrates the state diagram for activating the HTU-C and HTU-R, respectively. Upon an activation request, the HTU-C side transmits a two-level signal to the far end. The HTU-R side, upon an activation request, waits for the HTU-C signal. Once the HTU-C two-level signal is detected, the HTU-R performs frequency lock, line characterization, and echo cancellation coefficient calculation. Upon completion, HTU-R transmits a two-level signal back to HTU-C and waits for a four-level signal. The HTU-C then performs characterization based on the HTU-R two-level signal. When HTU-C completes its characterization, it sends a four-level 2B1Q signal. At this stage, normal operation is reached with transmission of a 2B1Q signal across the link.

Figure 5-3. Bit Pump Start-up Sequence



The bit pump activation is controlled by a separate state machine. The application code must repeatedly call the _BtMain() function to proceed through the bit pump state machine. Refer to the *ZipWire Software User Guide* (100417C) for further details about the bit pump state machine.

Table 5-4 lists DSL functions and the corresponding ASM state or states. Each activation state is illustrated in Figure 5-4 and Figure 5-5. Except where noted as "HTU-C only" or "HTU-R only," the states are applicable to both the HTU-C and HTU-R. The software processing within each state and between each state (i.e., the state transitions) is discussed in the following two sections.

Table 5-4. Cross Reference of DSL Functions vs. ASM States

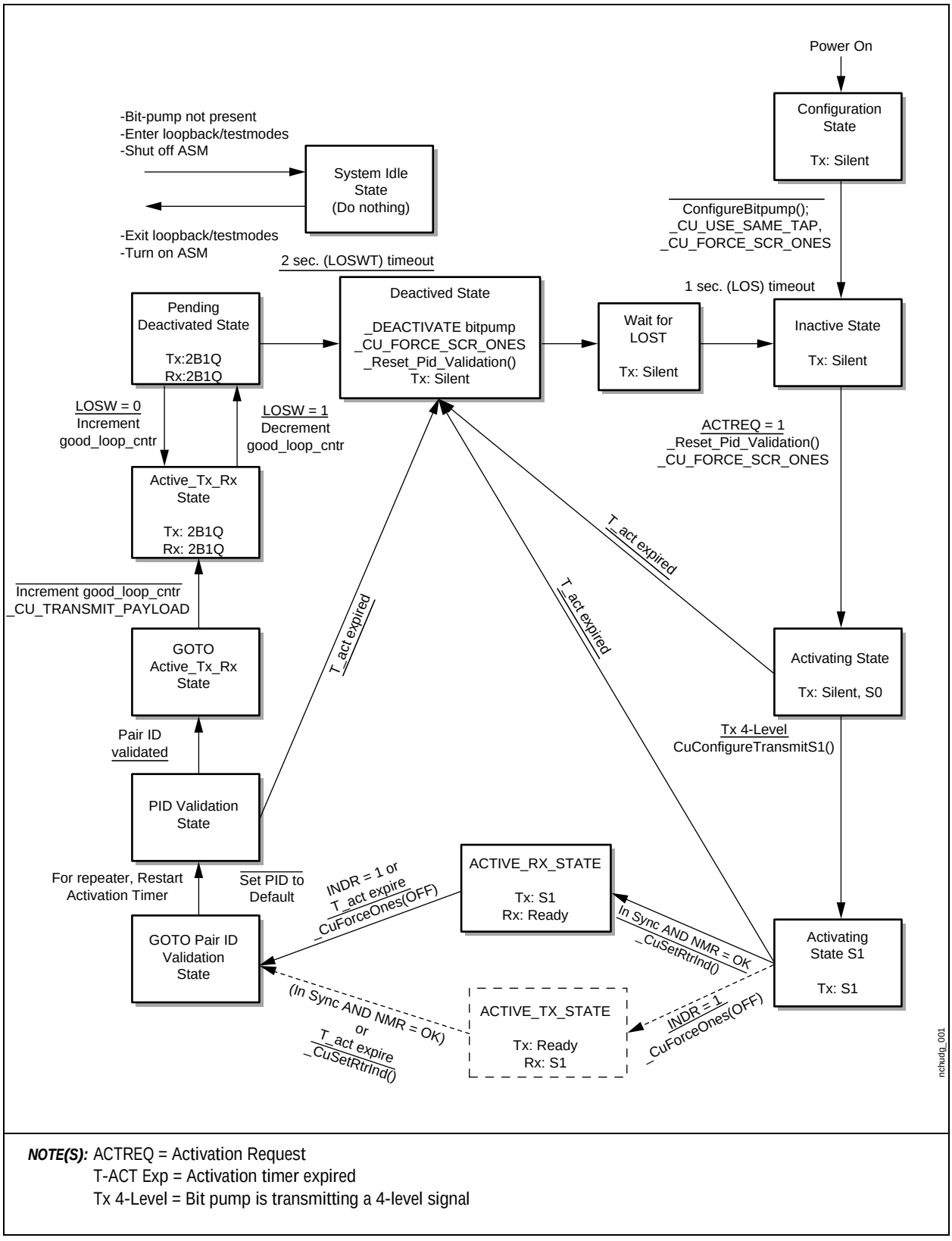
DSL Function	ASM States
Standby	SYSTEM_IDLE
Configuration and initialization	CONFIGURATION_STATE, INACTIVE_STATE
Start-up of the bit pump device (e.g., RS8953B)	ACTIVATING_STATE, ACTIVATING_STATE_S1
Start-up of the channel unit device (e.g., RS8953B)	ACTIVE_RX_STATE, ACTIVE_TX_STATE, GOTO_PID_VALIDATION_STATE, PID_VALIDATION_STATE, GOTO_ACTIVE_TX_RX_STATE
Normal operation (both bit pump and channel unit)	ACTIVE_TX_RX_STATE
Error processing and shutdown	PENDING_DEACTIVATED_STATE, DEACTIVATED_STATE, WAIT_FOR_LOST (HTU-C only), WAIT_FOR_LOS (HTU-R only)

5.4.1 HTU-C Activation

Figure 5-4 illustrates the state diagram for activation at the HTU-C side.

NOTE: There are several states that are not explicity stated in the TS-101-135 HDSL standard. These extra states were added to ease implementation of the HDSL standard for the bit pump and channel unit devices.

Figure 5-4. Activation State Machine at HTU-C



NOTE(S): ACTREQ = Activation Request
T-ACT Exp = Activation timer expired
Tx 4-Level = Bit pump is transmitting a 4-level signal

5.4.1.1 Configuration State

In the configuration state, the bit pump is reconfigured as depicted in Table 5-2. The activation state is changed to inactive after preliminary channel unit configuration.

Bit pump code: The bit pump code processes the API commands.

Channel unit code: The channel unit code issues _CU_USE_SAME_TAP to configure Tx/Rx to use HTU-C SCR/DSCR tap #5 while in normal operation. _CU_FORCE_SCR_ONES is issued to configure the channel unit for startup.

NOTE: There is some redundancy in the configuration state, and some APIs or functions are duplicated in other states. This is because the configuration state serves as an exit point for test modes and loopback operations. The redundancy guarantees that, after exiting from test modes or loopbacks, the bit pumps and channel unit are properly configured.

5.4.1.2 Inactive State

ASM: In the inactive state, the _ACTIVATE and _CU_FORCE_SCR_ONES API commands are issued to initiate the activation process. The transmitter is initially silent. The activation request signal is always considered to be true (ACTREQ = 1). The activation state variable changes to the activating state.

Bit pump code: The bit pump activation state machine is idle during the inactive state. The bit pump waits for the _ACTIVATE API command. When _ACTIVATE is detected, the activation state machine initializes the bit pump for the activation process.

Channel unit code: The channel unit code waits for the _CU_FORCE_SCR_ONES API command. When the command is received, the channel unit configures the HDSL transmitter to framed scrambled 1s where both the overhead and payload data are all 1s. The Tx/Rx 6 ms interrupts are masked and the DPLL is set to open mode for the master loop (See Section 11.6.2).

5.4.1.3 Activating State

ASM: In the activating state, the application code monitors the bit pump activation status. If the bit pump T-Act timer expires, the activation state changes to the deactivated state. If the bit pump Tx 4-level flag is detected, the channel unit enables the S1 signal (enable overhead). The activation state changes to the activating S1 state.

Bit pump code: The bit pump code begins transmitting the S0 signal and monitors the received signal for S0 (LOS = 0). When the signal is detected, the bit pump performs the optimized phase search and adapts filters, etc. The bit pump code then transmits the S1 signal.

Channel unit code: When the Tx 4-level is enabled, the channel unit enables the Tx/Rx 6 ms interrupts and the overhead data (see Section 11.6.3).

5.4.1.4 Activating State S1

ASM: In the activating state S1, the ASM monitors the bit pump activation status and channel unit status bits. If the bit pump T-Act timer expires, the activation state changes to the deactivated state. If the channel unit InSync flag is detected, and the noise margin reading (NMR) is ok, the RTR indicator bit is set and the activation state changes to the active Rx state. If the INDR flag is detected, the payload data is enabled, and the activation state changes to the active Tx state.

The INDR flag is an HDSL overhead bit received from HTU-R (refer to the rtr flag in Table 6-1). The rtr flag is transmitted by both the HTU-C and HTU-R and indicates the other side is in sync.

Bit pump code: The bit pump transmits the S1 and monitors the received signal for S1. When the S1 signal is detected, the bit pump code performs a final adaptation.

Channel unit code: The channel unit code monitors the sync word and indicator bits. When the sync word is detected, the channel unit sync status is set to CU_IN_SYNC. If the RTR indicator bit is detected, the INDR bit is set. After a status is detected, the channel unit code sets the appropriate configuration bits as determined by the application code.

5.4.1.5 Active Rx State

ASM: In the active Rx state, the ASM monitors bit pump activation and channel unit status. If the bit pump TS101 Activation Timer expires or the INDR bit is detected, the payload data is enabled. The activation state is changed to the GOTO Pair ID validation state.

Bit pump code: The bit pump ASM finalizes its activation process.

Channel unit code: The channel unit code monitors the indicator bits. If the RTR indicator bit is detected, the INDR bit is set. When the enable payload command is issued, the following occurs:

- the payload data is enabled
- the pair ID is set (per *ETSI* specification)
- the Tx/Rx FIFO buffers are reset
- the DPLL is closed and the DPLL interrupt is enabled for the master loop (see Section 11.6.5)

5.4.1.6 Active Tx State

The active Tx state path is typically not taken; therefore, it is shown as dashed lines.

ASM: In the active Tx state, the ASM monitors bit pump activation and channel unit status. If the bit pump TS101 activation timer expires or the combination of the InSync flag is set and NMR ok, the payload data is enabled. The activation state is changed to GOTO Pair ID validation state.

Bit pump code: The bit pump activation state machine finalizes its activation process.

Channel unit code: The channel unit code monitors the sync status. If the InSync bit is detected, the INDC bit is set. When the enable payload command is issued, the following occurs:

- the payload data is enabled
- the pair ID (ETSI) is set
- the Tx/Rx FIFOs are reset
- if the master loop, the DPLL is closed and the DPLL interrupt is enabled (see Section 11.6.5).

5.4.1.7 GOTO Pair ID Validation State

ASM: In the GOTO pair ID validation state, if it is the E1 application, the loop's pair ID is initialized to the default values. The expected pair loop ID and corresponding received valid pair ID counter are reset to start the pair ID validation procedure. The activation state then changes to Pair ID validation state.

If it is not an E1 application, the activation state goes directly to Pair ID validation state without doing anything.

Bit pump code: The bit pump activation state machine remains in normal operation.

Channel unit code: The channel unit code processes the Tx/Rx 6 ms interrupts and monitors whether or not the pair ID validation procedure should start.

5.4.1.8 PID Validation State

ASM: In the PID validation state, if it is an E1 application, the ASM monitors whether or not the loop's pair ID is validated. If the loop's pair ID is validated before the activation timer expires, the activation state is changed to GOTO active Tx/Rx state; otherwise, the activation state is changed to the deactivated state and the loop's pair ID validation procedure is reset.

If it is not an E1 application, the activation state is changed to GOTO active Tx/Rx state.

Bit pump code: The bit pump activation state machine remains in normal operation.

Channel unit code: The channel unit code processes the Tx/Rx 6 ms interrupts and monitors the received pair ID for pair ID validation.

5.4.1.9 GOTO Active Tx/Rx State

ASM: In the GOTO active Tx/Rx state, the ASM initializes timers, performance monitoring records, and the EOC state, and updates the loop's sync LED. The channel unit is configured to transmit payloads and the activation state is changed to active Tx/Rx state.

Bit pump code: The bit pump activation state machine remains in normal operation.

Channel unit code: The channel unit code processes the Tx/Rx 6 ms interrupts.

5.4.1.10 Active Tx/Rx State

ASM: In the active Tx/Rx state, the ASM monitors the channel unit sync status. If the LOSW = 1 flag is detected, the activation state is changed to the pending deactivated state.

Bit pump code: The bit pump code handles any temperature and environmental changes. Different status responses are continuously monitored and can be probed by issuing the corresponding status request commands.

Channel unit code: The channel unit code processes the Tx/Rx 6 ms interrupts. The appropriate status bits are set according to the status registers.

5.4.1.11 Pending Deactivated State

ASM: In the pending deactivated state, the ASM monitors the channel unit status. If the LOSW = 0 (return of sync word) flag becomes valid again within 2 seconds, the activation state is changed back to the active Tx/Rx state. If the LOSWT = 1 (loss of sync word timer) is detected, the activation state is changed to the deactivated state.

Bit pump code: The bit pump activation state machine behaves similarly to the active Tx/Rx state.

Channel unit code: The channel unit code behaves similarly to the active Tx/Rx state.

5.4.1.12 Deactivated State

ASM: In the deactivated state, the ASM issues the bit pump _DEACTIVATE API command to turn off the bit pump transmitter and _CuConfigureBeginStartup() function to mask the channel unit Tx/Rx 6 ms interrupts. The ASM then polls for LOST = 1. When LOST = 1 is true, the activation state changes to the inactive state.

Bit pump code: The bit pump interrupt handler starts the LOS Timer (LOST) when the _DEACTIVATE API command is received and when the LOS flag is TRUE (LOS = 1). The bit pump interrupt handler sets the LOST flag (LOST = 1) when the LOST expires.

Channel unit code: The channel unit code performs the same tasks as before it entered this state.

5.4.1.13 System Idle State

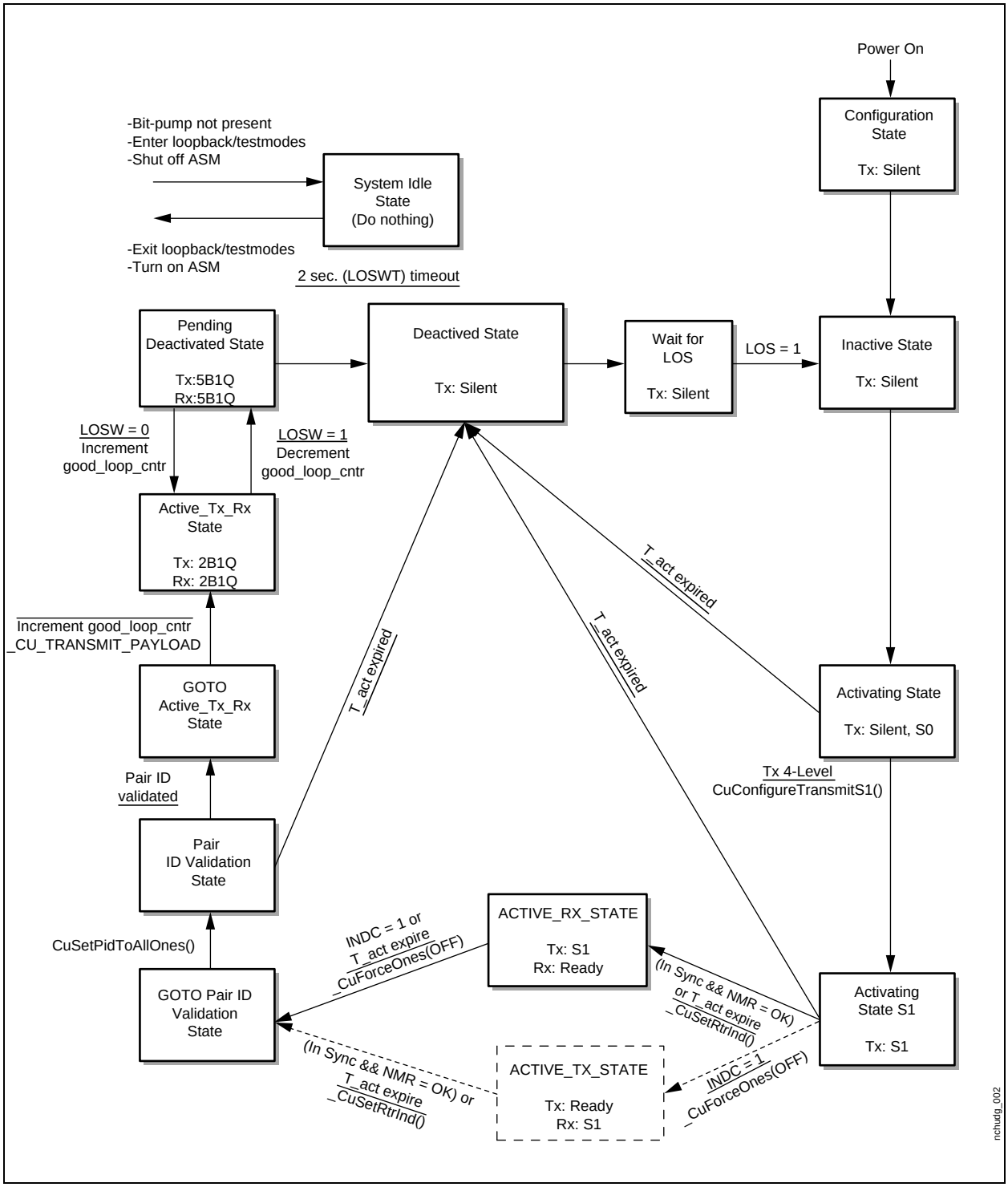
ASM: In the system idle state, the ASM is idle. This allows the system to perform test modes, loopbacks, etc. The activation state should be changed back to the configuration state after test modes or loopbacks are done.

Bit pump code: The bit pump code does what it is requested to do, i.e., test modes and loopbacks.

Channel unit code: The channel unit code does what it is requested to do, i.e., test modes and loopbacks.

5.4.2 HTU-R Activation

Figure 5-5. Activation State Machine at HTU-R



5.4.2.1 Configuration State

ASM: In the configuration state, the ASM is configured as indicated in Table 5-2. The activation state changes to inactive after preliminary channel unit configuration.

Bit pump code: The bit pump code processes the API commands.

Channel unit code: The channel unit code issues the _CU_USE_SAME_TAP to API sets the HTU-C scrambler and descrambler taps to #5 for normal operation. The _CuConfigureBeginStartup() function is called to configure the channel unit for startup.

5.4.2.2 Inactive State

ASM: In the inactive state, the _ACTIVATE and _CU_FORCE_SCR_ONES API commands are issued to initiate the activation process. The transmitter is initially silent. If the bit pump LOS = 1 is detected, the activation state is changed to the activating state.

Bit pump code: The bit pump activation state machine does nothing during the inactive state. The bit pump waits for the _ACTIVATE API command. When _ACTIVATE is detected, the activation state machine initializes the bit pump for the activation process. The bit pump code waits until an S0 signal is detected (LOS = 0).

Channel unit code: The channel unit code waits for the _CU_FORCE_SCR_ONES API command. When the command is received, the channel unit configures the HDSL transmitter to framed scrambled 1s, where both the overhead and payload data are all 1s. The Tx/Rx 6 ms and interrupts are masked and the DPLL is set to open mode for the master loop (see Section 11.6.2).

5.4.2.3 Activating State

ASM: In the activating state, the ASM monitors the bit pump activation status. If the bit pump T-Act timer expires, the activation state is changed to deactivated. If the bit pump Tx 4-level flag is detected, the channel unit enables the S1 signal (enable overhead). The activation state is changed to the activating S1 state.

Bit pump code: The bit pump code transmits the S0 signal and monitors the received signal for S0 (LOS = 0). When the signal is detected, the bit pump performs frequency lock, optimizes phase search, adapts filters, etc. The bit pump code transmits the S0 signal and monitors the received signal for S1. When the S1 signal is detected, the bit pump code performs adaptations and transmits the S1 signal; the Tx 4-level flag is set.

Channel unit code: When the Tx 4-level is enabled, the channel unit enables the Tx/Rx 6 ms interrupts and the overhead data (see Section 11.6.3).

5.4.2.4 Activating State S1

This state is not part of the ETR-152 HDSL standard, but is required for the implementation of the bit pump and channel unit devices.

ASM: In the activating state S1, the ASM monitors the bit pump activation status and channel unit status bits. If the bit pump T-Act timer expires, the activation state is changed to the deactivated state. If the channel unit InSync flag is detected, and the NMR is ok, the RTR indicator bit is set and the activation state is changed to the active Rx state. If the INDC flag is validated, the payload data is enabled and the activation state is changed to the active Tx state.

Bit pump code: The bit pump transmits the S1 and monitors the received signal for S1. When the S1 signal is detected, the bit pump code will perform some final adaptation.

Channel unit code: The channel unit code monitors the sync word and indicator bits. When the sync word is detected, the channel unit sync status is set to CU_IN_SYNC. If the RTR indicator bit is detected for six consecutive frames, the RTR valid flag is set.

5.4.2.5 Active Rx State

ASM: In the active Rx state, the ASM monitors the bit pump activation and channel unit status. If the bit pump T-Act expires or the channel unit INDC bit is detected, the payload data is enabled. The activation state is changed to the active Tx/Rx state.

Bit pump code: The bit pump activation state machine finalizes its activation process.

Channel unit code: The channel unit code monitors the indicator bits. If the RTR indicator bit is detected, the INDC bit is set. When the enable payload command is issued, the following occurs:

- the payload data is enabled
- the pair ID is set (per *ETSI* specification)
- the Tx/Rx FIFO buffers are reset
- if the master loop, the DPLL is closed and the DPLL interrupt is enabled (see Section 11.6.5)

5.4.2.6 Active Tx State

The active Tx state path is typically not taken; therefore, it is shown as dashed lines.

ASM: In the active Tx state, the ASM monitors the bit pump activation and channel unit status. If the bit pump T-Act timer expires or the combination of the InSync flag is set and NRM ok, the payload data is enabled. The activation state is changed to the GOTO Pair ID validation state.

Bit pump code: The bit pump activation state machine finalizes its activation process.

Channel unit code: The channel unit code monitors the sync status. If the InSync bit is detected, the INDR bit is set. When the enable payload command is issued, the following occurs:

- the payload data is enabled
- the pair ID (ETSI) is set
- the Tx/Rx FIFOs are reset
- if the master loop, the DPLL is closed and the DPLL interrupt is enabled (see Section 11.6.5)

5.4.2.7 GOTO Pair ID Validation State

ASM: In the GOTO pair ID validation state, if it is the E1 application, the loop's pair ID is initialized to all 1s. The expected pair ID and the corresponding received valid pair ID counter are reset to start the pair ID validation procedure. The activation state is then changed to the pair ID validation state.

If it is not an E1 application, the activation state goes directly to the pair ID validation state.

5.4.2.8 PID Validation State

Bit pump code: The bit pump activation state machine remains in normal operation.

Channel unit code: The channel unit code processes the Tx/Rx 6 ms interrupts and monitors the received pair ID for pair ID validation. If the pair ID is validated, the validated pair ID is transmitted to the HTU-C.

5.4.2.9 GOTO Active Tx/Rx State

ASM: In the GOTO active Tx/Rx state, the ASM initializes timers, performance monitoring records, and the EOC state and updates the loop's sync LED. The channel unit is configured to transmit payloads, and the activation state is changed to the active Tx/Rx state.

Bit pump code: The bit pump activation state machine remains in normal operation.

Channel unit code: The channel unit code processes the Tx/Rx 6 ms interrupts.

5.4.2.10 Active Tx/Rx State

ASM: In the active Tx/Rx state, the ASM monitors the channel unit sync status. If the LOSW = 1 flag is detected, the activation state is changed to the pending deactivated state.

Bit pump code: The bit pump code handles any temperature and environmental changes. Different status responses are continuously monitored and can be probed by issuing the corresponding status request commands.

Channel unit code: The channel unit code processes the Tx/Rx 6 ms interrupts. The code reads status registers in the device and sets the corresponding software bits.

5.4.2.11 Pending Deactivated State

ASM: In the pending deactivated state, the ASM monitors the channel unit status. If the LOSW = 1 (loss of sync word) flag becomes false within 2 seconds, the activation state is changed back to the active Tx/Rx state. If the LOSWT = 1 (loss of sync word timer) is detected, then the activation state changes to the deactivated state.

Bit pump code: The bit pump activation state machine behaves similarly to the active Tx/Rx state.

Channel unit code: The channel unit code behaves similarly to the active Tx/Rx state.

5.4.2.12 Deactivated State

ASM:	In the deactivated state, the ASM issues the bit pump _DEACTIVATE API command to turn off the bit pump transmitter and channel unit _CuConfigureBeginStartup() function to mask the Tx/Rx 6 ms interrupts. The application code waits for LOS = 1. When LOS = 1 is true, the activation state is changed to the inactive state.
Bit pump code:	The bit pump interrupt handler sets the LOS = 1 when the loss of signal is detected.
Channel unit code:	The channel unit code performs the same tasks as before it entered this state.

5.4.2.13 System Idle State

ASM:	In the system idle state, the ASM does nothing. This allows the system to perform test modes, loopbacks, etc. The activation state should be changed back to the configuration state after test modes or loopbacks are done.
Bit pump code:	The bit pump code does what it is requested to do, i.e., test modes and loopbacks.
Channel unit code:	The channel unit code does what it is requested to do, i.e., test modes and loopbacks.

6.0 Channel Unit Code

This chapter discusses the details of the channel unit code found in the CHANUNIT subdirectory.

6.1 Configurations

The system supports several kinds of configurations for T1 and E1 transmission using HDSL technology. The basic structure of an HDSL frame is listed in Table 6-1, where each frame is nominally 6 ms in length and consists of 48 payload blocks. Each payload block contains a single F- or Z-bit plus an application-specific number of payload bytes. Groups of 12 payload blocks are concatenated and separated by an ordered set of HDSL overhead bits, where a 14-bit SYNC word pattern identifies the starting location of the HDSL frame. Fifty overhead bits are defined in one HDSL frame, but the last 4 STUFF bits are nominally present in alternate frames. Therefore, one frame contains an average of 48 overhead bits. Figure 6-1 illustrates the frame structure. The payload block structures for different applications are illustrated in the following subsections.

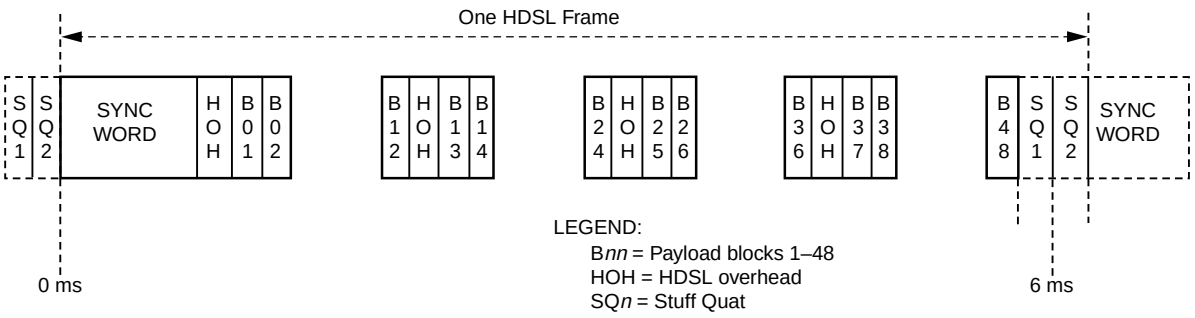
Table 6-1. HDSL Frame Structure and Overhead Bit Allocation *(1 of 2)*

HOH Bit	Symbol	Bit Name	HOH Register Bit
1–14	sw1–sw14	SYNC Word	—
15	losd	Loss of Signal	IND[0]
16	febe	Far-End Block Error	IND[1]
Payload Blocks 1–12			
17–20	eoc1–eoc4	Embedded Operation Channel	EOC[0]–EOC[3]
21–22	crc1–crc2	Cyclic Redundancy Check	—
23	ps1	HTU-R Power Status	IND[2]
24	ps2	Power Status Bit 2	IND[3]
25	bpv	Bipolar Violation	IND[4]
26	eoc5	Embedded Operation Channel	EOC[4]
Payload Blocks 13–24			
27–30	eoc6–eoc9	Embedded Operation Channel	EOC[5]–EOC[8]
31–32	crc3–crc4	Cyclic Redundancy Check	—
33	hrp	HDSL Repeater Present	IND[5]

Table 6-1. HDSL Frame Structure and Overhead Bit Allocation *(2 of 2)*

HOH Bit	Symbol	Bit Name	HOH Register Bit
34	rrbe	Repeater Remote Block Error	IND[6]
35	rcbe	Repeater Central Block Error	IND[7]
36	rega	Repeater Alarm	IND[8]
Payload Blocks 25–36			
37–40	eoc10–eoc13	Embedded Operation Channel	EOC[9]–EOC[12]
41–42	crc5–crc6	Cyclic Redundancy Check	—
43	rta	Remote Terminal Alarm	IND[9]
44	rtr	Ready To Receive	IND[10]
45	uib	Unspecified Indicator Bit	IND[11]
46	uib	Unspecified Indicator Bit	IND[12]
Payload Blocks 37–48			
47	sq1	Stuff Quat Sign	STUFF[0]
48	sq2	Stuff Quat Magnitude	STUFF[1]
49	sq3	Stuff Quat Sign	STUFF[2]
50	sq4	Stuff Quat Magnitude	STUFF[3]

Figure 6-1. HDSL Frame Structure



NOTE(S): In the EVM system, the bit pumps and channel unit are configured based on the DIP switch settings illustrated in Figure 2-2.

6.1.1 CU_2T1

The CU_2T1 compiler flag enables standard 2-loop T1 mapping at 784 kbps with each loop carrying one-half of the payloads from T1. Each payload block contains 1 F-bit followed by 12 payload bytes (Figure 6-2). The relation between the payload bytes and PCM time slot is listed in Table 6-2.

Figure 6-2. Payload Block Structure for 2T1 Application

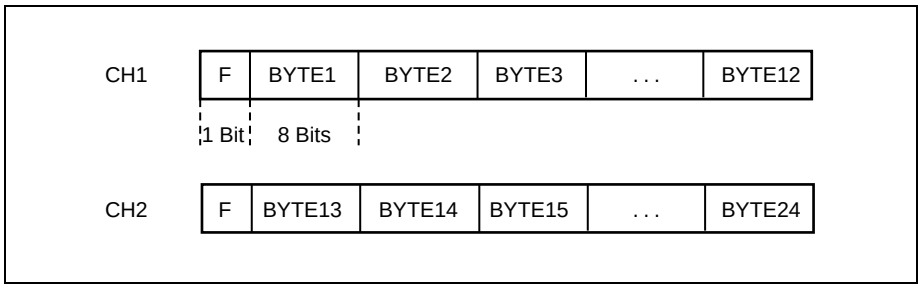


Table 6-2. 2T1 Framing

Channel 1

Byte	1	2	3	4	5	6	7	8	9	10	11	12
Time slot	1	2	3	4	5	6	7	8	9	10	11	12

Channel 2

Byte	13	14	15	16	17	18	19	20	21	22	23	24
Time slot	13	14	15	16	17	18	19	20	21	22	23	24

6.1.2 CU_2E1

The CU_2E1 compiler flag enables standard two-loop E1 mapping at 1168 kbps with each loop carrying one-half of the E1 payload. Each payload block contains one Z-bit followed by 18 payload bytes (Figure 6-3). The relation between the payload bytes and PCM time slot is listed in Table 6-3.

Figure 6-3. Payload Block Structure for 2E1 Application

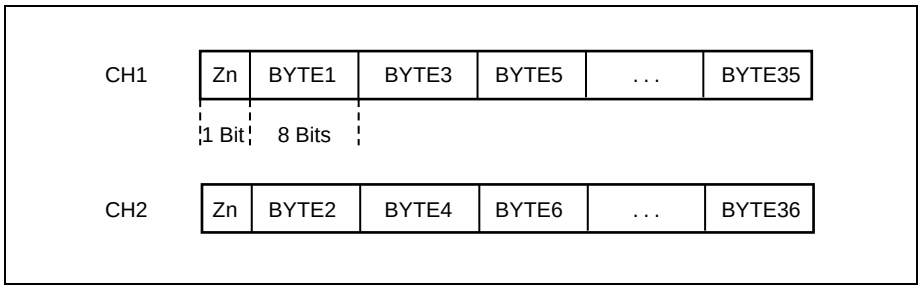


Table 6-3. 2E1 Framing

Channel 1

Byte	1	3	5	7	9	11	13	15	17	19	21	23	25	27	29	31	33	35
Time slot	0	1	3	5	7	9	11	13	15	16	18	20	22	24	26	28	30	DBANK

Channel 2

Byte	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34	36
Time slot	0	2	4	6	8	10	12	14	16	17	19	21	23	25	27	29	31	DBANK

6.1.3 CU_3E1

The CU_3E1 compiler flag enables standard 3-loop E1 mapping at 784 kbps with each loop carrying one-third of the E1 payload. Each payload block contains 1 Z-bit followed by 12 payload bytes (Figure 6-4). The relation between the payload bytes and the PCM time slot is listed in Table 6-4.

Figure 6-4. Payload Block Structure for 3E1 Application

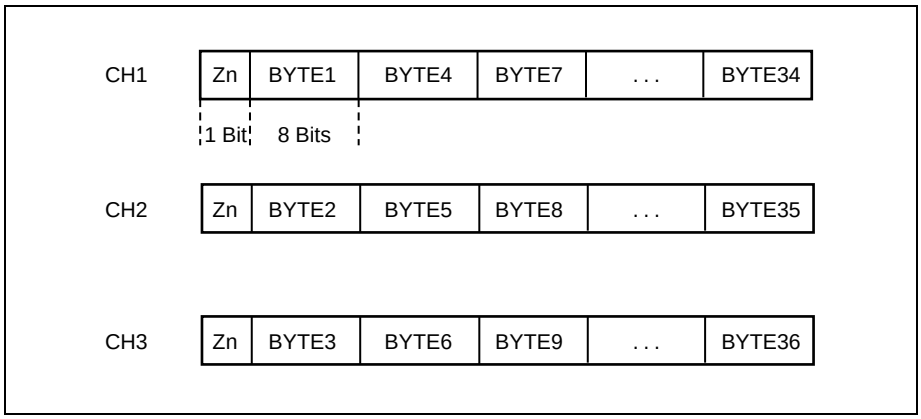


Table 6-4. 3E1 Framing

Channel 1

Byte	**1**	**4**	**7**	**10**	**13**	**16**	**19**	**22**	**25**	**28**	**31**	**34**
Time slot	0	1	4	7	10	13	16	17	20	23	26	29

Channel 2

Byte	**2**	**5**	**8**	**11**	**14**	**17**	**20**	**23**	**26**	**29**	**32**	**35**
Time slot	0	2	5	8	11	14	16	18	21	24	27	30

Channel 3

Byte	**3**	**6**	**9**	**12**	**15**	**18**	**21**	**24**	**27**	**30**	**33**	**36**
Time slot	0	3	6	9	12	15	16	19	22	25	28	31

6.1.4 CU_1T1

The CU_1T1 runs the standard 1-loop T1 mapping at 1,552 kbps with 1 loop carrying all the T1 payload. Each payload block contains 1 F-bit followed by 24 payload bytes (Figure 6-5). The relation between the payload bytes and PCM time slot is listed in Table 6-5.

Figure 6-5. Payload Block Structure for 1T1 Application

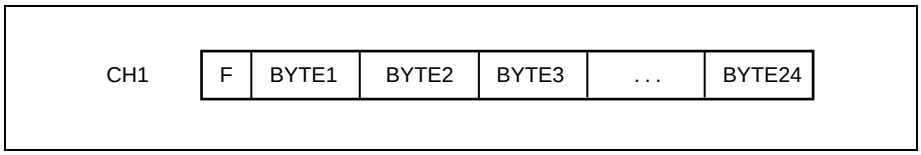


Table 6-5. 1T1 Framing

Byte	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Time slot	1	2	3	4	5	6	7	8	9	10	11	12
Byte	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**
Time slot	13	14	15	16	17	18	19	20	21	22	23	24

6.1.5 CU_1E1

The CU_1E1 runs the standard one-loop E1 mapping at 2320 kbps with one loop carrying all the E1 payload. Each payload block contains one Z-bit followed by 36 payload bytes (Figure 6-6). The relation between the payload bytes and PCM time slot is listed in Table 6-6.

Figure 6-6. Payload Block Structure for 1E1 Application

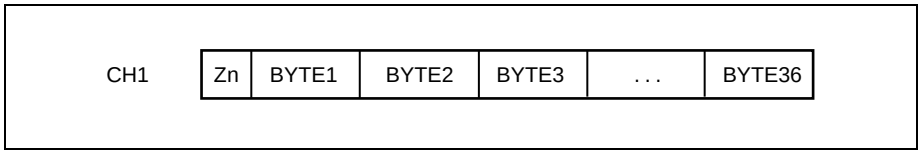


Table 6-6. 1E1 Framing

Byte	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**
Time slot	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Byte	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**	**32**	**33**	**34**	**35**	**36**
Time slot	18	19	20	21	22	23	24	25	26	27	28	29	30	31	*	*	*	*
NOTE(S): * = DBANK																		

6.1.6 Modifying the Code for Custom Applications (_CU_CUSTOM)

To customize the channel unit code for applications other than standard 1T1, 2T1, 1E1, 2E1, and 3E1, the following parts of the channel unit code must be modified.

6.1.6.1 Rate_values[][]

The rate_values[][] array is a matrix of register values for the various rate configurations, i.e., 1T1, 2T1, 1E1, 2E1, 3E1, and _CU_CUSTOM. The rate_values[][] array is defined in CU_INIT.C. Only registers that require different programming values between the various configurations are included in this matrix. Other registers that are programmed to the same value between various configurations are hard coded into the various initialization functions. The first dimension of the array indexes the rate, and the second dimension indexes the channel unit register. See CU_TABLE.H for the list of registers. The channel unit initialization functions index this array to determine the desired register value for the current configuration.

6.1.6.2 CU_MAP.C

Three functions in the CU_MAP.C file initialize the transmit routing table (see Section 10.9), receive combination table (see Section 10.10), transmit payload map registers (see Section 10.11) and receive payload map registers (see Section 10.12) based on the channel unit configuration.

- _CuInitMapper()
- _CuInitRouteTable()
- _CuInitCombineTable()

The customer must modify the portions of the code in the above functions that are surrounded by CU_CUSTOM compiler flag.

6.2 Interrupt Handler

The interrupt handlers in CU_INT.C process the HDSL 6 ms Tx/Rx and T1/E1 framer interrupts. Sync status, error status, and indication bits are checked and updated; Tx/Rx FIFO buffer and DPLL errors are handled when they occur; any EOC requests are processed, and the pair ID is validated during the start-up stage.

6.2.1 Sync Status

Loop's synchronization status is checked and updated every 6 ms.

6.2.2 Error Status Reporting

The Tx/Rx Interrupt functions check the status registers or indication bits for errors. When an error occurs, the corresponding error counter increments. The following error counters are maintained by the system:

- _CU_OUT_OF_SYNC_CTR
- _CU_CRC_ERR_CTR
- _CU_RFIFO_FULL_CTR
- _CU_RFIFO_EMPTY_CTR
- _CU_RFIFO_SLIP_CTR
- _CU_TFIFO_FULL_CTR
- _CU_TFIFO_EMPTY_CTR
- _CU_TFIFO_SLIP_CTR
- _CU_TFIFO_STUFF_CTR
- _CU_DPLL_ERROR_CTR
- _CU_FEBE_ERROR_CTR
- _CU_LOSD_ERROR_CTR

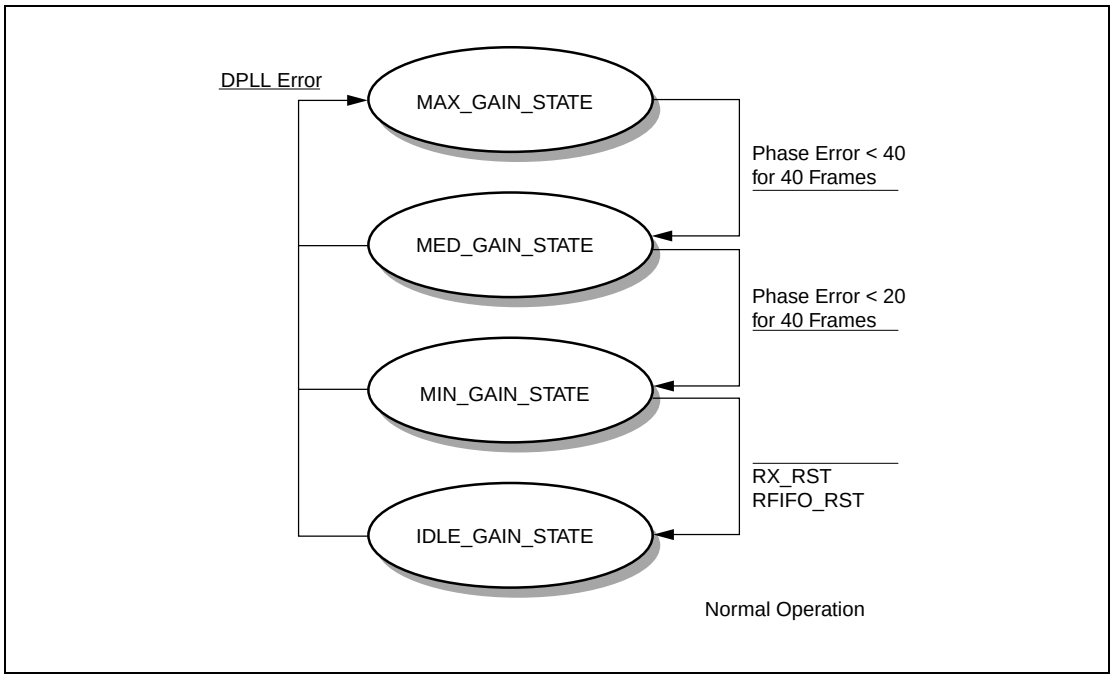
6.2.3 Tx/Rx FIFO Error Handling

In the Tx/Rx Interrupt functions, when a FIFO error occurs, the corresponding FIFO buffer is reset. Resetting the FIFO corrupts the next three HDSL frames which cause additional Tx/Rx FIFO errors. However, the FIFO error check is bypassed when processing the next three Tx/Rx interrupts.

6.2.4 DPLL Error Handling

The _CuDpllInterrupt() function processes DPLL errors. Figure 6-7 illustrates the DPLL state machine. The DPLL Error Interrupt is triggered when the DPLL Phase Error (0x38) exceeds the –128 or +127 threshold. When no DPLL error is detected, the DPLL state machine is called every 6 ms. The 6 ms interval is based on the master loop's Rx 6 ms interrupt.

Figure 6-7. DPLL State Machine



The DPLL state machine increases (opens up) the DPLL bandwidth when an error occurs, to allow the DPLL to increase its capture range. Then as the DPLL locks onto the far-end PCM clock, the DPLL bandwidth is decreased to minimize the PCM RCLK jitter. The DPLL bandwidth is set by register 0xD8.

MAX_GAIN_STATE: When a DPLL error occurs, the DPLL state is set to MAX_GAIN_STATE. The DPLL bandwidth is set to 0x34 (MAX_DPLL_GAIN_VALUE). The DPLL Error LED is set. When the DPLL Phase Error reads less than 40 for 40 consecutive frames, the DPLL state is set to MED_GAIN_STATE.

MED_GAIN_STATE: The DPLL bandwidth is set to 0x22 (MED_DPLL_GAIN_VALUE). When the DPLL Phase Error reads less then 20 for 40 consecutive frames, the DPLL state is set to MIN_GAIN_STATE.

MIN_GAIN_STATE: The DPLL bandwidth is set to 0x10 (MIN_DPLL_GAIN_VALUE). A RX_RST (0xF1), and a RFIFO_RST (0x62, 0x82, 0xA2) are issued which resynchronizes the PCM timebase using the stable RCLK. The DPLL Error LED is cleared. The DPLL state is set to IDLE_GAIN_STATE.

IDLE_GAIN_STATE: The DPLL is now assumed to be stable, and the DPLL state machine does nothing.

6.2.5 EOC Termination

TEOC registers are updated with eocSendMessage every 6 ms in the HDSL transmit interrupt handler. eocSendMessage contains the EOC messages sent by the HTU-C and HTU-R. It is updated by EOC protocol. eocReceiveMessage is updated every 6 ms in the HDSL receive interrupt handler by extracting the information field from REOC registers. It contains the messages received from the other side. In the HDSL receive interrupt handler, EocMaster() and EocTaskHandler_CO() are called to handle the EOC protocol on the HTU-C, while EocSlave() and EocTaskHandler_RT() are called to handle the EOC protocol on the HTU-R.

6.2.6 Pair ID Termination (E1 Mode)

In the HDSL receive interrupt handler, if a loop's pair ID (E1 application only) has not been validated, the E1 pair ID validation function is called. A loop's pair ID is considered as validated only after a valid, unique pair ID is received consecutively for 6 frames.

6.2.7 Indicator Bit Termination

If a CRC error is detected, the FEBE indication bit is cleared; if an E1/T1 LOS or OOF is detected, the LOSD indication bit is cleared. The updated indication bits are written to the TIND register in the HDSL transmit interrupt handler.

6.2.8 T1/E1 Framer Interrupt Handling—6 ms Polling

This function polls the Framer Status registers, and the LEDs are updated accordingly. The framer interrupt handler is called every 6 ms, an interval based on the master loop's Rx 6 ms interrupt.

6.3 BER Meter

The internal BER meter is used to test the HDSL link without an external BER tester or data. PCM time slots from TSER or RSER can be examined for test patterns on a per time slot basis, or the entire framed or unframed PCM channel from TSER can be examined; choose from four Pseudo-Random Bit Sequence (PRBS) patterns or an 8-bit fixed pattern. The MPU configures BER_SCALE to determine the test measurement by issuing BER_RST, then monitors the test results (BER_METER) and test status (BER_STATUS).

6.4 Dynamic Master Loop

The master loop is responsible for extracting the framing and signaling information (F-bit for T1 or TS0/TS16 for E1) from the HDSL frame into the PCM data. The DSL loop handler monitors the number of loops in normal operation as well as the loop's pair ID or sync word. When a master loop failure is detected, the system switches the master loop to the next available loop in normal operation.

The master loop provides the extraction of time slots 0 and 16. The rmap table for the master loop is programmed to extract these time slots. The master loop also provides the frame sync signal to the DPLL and PCM formatter in the channel unit device.

6.5 Tip/Ring Reversal

Tip/ring reversal is the reversal of a twisted pair of wires. The channel unit device automatically handles any tip/ring reversal so the software does not need to implement this feature. There is no way to disable the channel unit tip/ring reversal feature.

6.6 Loop Reversal

Loop reversal is the reversal of two or more pairs of wires. The HTU-R monitors the Sync Word (T1) or the loop's pair ID (E1) and configures the HTU-R Route, Combine, TMAP, and RMAP tables accordingly. The loop reversal does not apply in single pair systems (1T1 and 1E1).

6.7 EOC Operation

The Embedded Operations Channel (EOC) is defined in the ANSI/ETSI standard. It uses 13 out of 50 HDSL overhead bits every 6 ms as a communication channel between the HTU-C and HTU-R in normal operation. EOC protocol is implemented in the channel unit code and can be enabled using the CU_EOC compiler flag.

EOC operation is handled by two sets of functions:

1. EOC protocol handler functions
2. EOC task handler functions

The EOC protocol handler is designed to handle the EOC protocol independent of the application. This enables a reasonable portability to other applications and delivers a strong separation between the EOC protocol and application-specific requirements. The EOC protocol is implemented by the functions EocMaster on the HTU-C side and EocSlave on the HTU-R side.

The EOC task handler coordinates the application-specific access to the EOC channel and performs some application-specific code that is related to the EOC commands. New applications using EOC may require modifications to the EOC Task Handler.

There is a well-defined interface between the EOC protocol handler and EOC task handler that enables controlling the EOC protocol handler with no restrictions.

The EOC tasks on the HTU-C and the HTU-R differ considerably. The HTU-C serves as the EOC Master, and the HTU-R is the slave that only responds to commands from the HTU-C. On the HTU-C side, the EOC task handler responds to requests from other tasks. On the HTU-R side, the EOC task handler only has to set or reset flags or to read or write registers.

6.7.1 EOC Data Format

EOC channel uses 13 of the available 50 HDSL overhead bits every 6 ms. The definitions of the 13 bits are listed in Table 6-7.

Table 6-7. HDSL EOC Frame Structure

Bit Position	# of Bits	Description	Remarks
1–2	2	Destination Address	Can address 4 locations
3	1	Data(0)/Message(1) indicator	—
4	1	Odd(1)/Even(0) bit	For multibyte transmission
5	1	Unused	—
6–13(1)	8	Information Field	256 opcodes, 8 bit data

NOTE(S):
(1) EOC6 contains the MSB, and EOC13 contains the LSB of the opcode/data.

6.7.2 EOC-Related Data

According to ANSI/ETSI standards, a maximum of 16 registers (Registers 0–F) can be read and written using the EOC protocol. Each register can contain one or more bytes. channel unit code 6.0 only supports registers A–F, and registers 0–9 are not implemented.

The data structure for the EOC registers is defined below:

```
typedef union
{
   BP_U_8BIT buffer[8]; /* Linear address space of EOC registers
      */
    struct
    {
       BP_U_8BIT eocRegA;    /* API Status Result*/
       BP_U_8BIT eocRegB;    /* Not used    */
       BP_U_8BIT eocRegC;    /* Not used   */
       BP_U_8BIT eocRegD;    /* HTUR ZIP Status   */
       BP_U_8BIT eocRegE;    /* Not used   */
       BP_U_8BIT eocRegF[3]; /* API commands contents   */
    }fields;
}Bt8953_EOC_DATA;
```

6.7.3 Supported EOC Commands

Channel unit code 6.0 implements a generic method of sending API commands through the EOC channel. It uses eocRegF[1]–eocRegF[3] to store the API's destination, opcode, and parameter. An EOC write register request is issued to send eocRegF[1]–[3] to the HTU-R. After the HTU-R successfully receives the API and correctly interprets it, the API command is executed. If the API is a control command, the HTU-R takes the requested action; if the API is a status checking command, the HTU-R stores the status checking result in eocRegA. Another EOC request can then be issued to read eocRegA to get the API result.

Channel unit code 6.0 also uses EOC internally to check the HTU-R ZIP status. The requested ZIP status is stored in eocRegD.

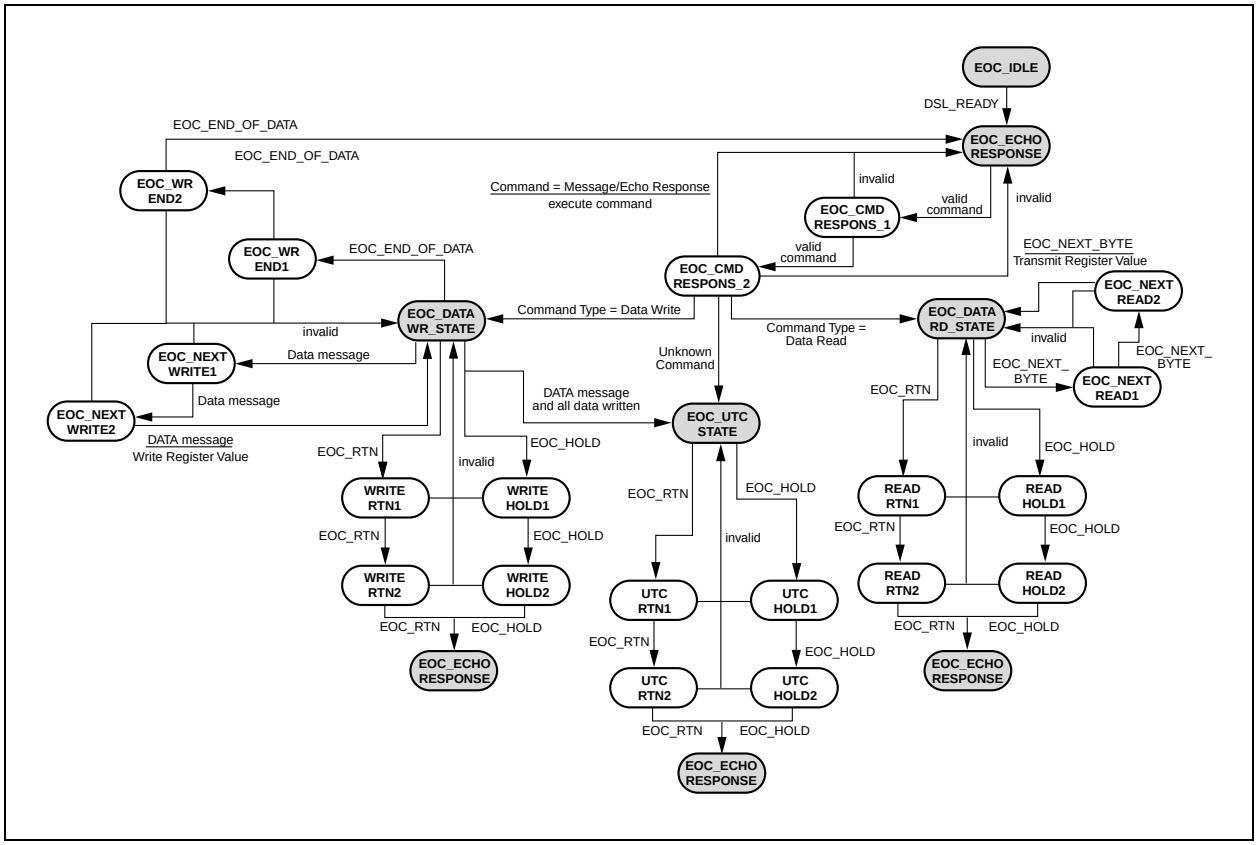
6.7.4 EOC Handling on the HTU-R Side

This section discusses the operation of the EOC slave software.

6.7.4.1 EOC Slave

Figure 6-8 illustrates the state transition diagram for the EOC slave. The diagram includes the three main states, EOC_DATA_WR_STATE, EOC_DATA_RD_STATE, and EOC_UTC_STATE. These three states transition to the EOC_ECHO_RESPONSE state after three consecutive EOC_RTN or EOC_HOLD commands are received from the HTU-C. The four temporary states (xx_RTN1/2 and xx_HOLD1/2) for each of the main states verify the reception of these three consecutive commands. Unable To Comply (UTC) is entered when the received command is not supported or the HTU-C sends a data write command after the HTU-R register is completely written.

Figure 6-8. State Transition Diagram for EOC Slave



EOC slave contains five main states, marked as grayed ovals. All other (white) states are temporary states that track the consecutive reception of three valid commands or data messages. A temporary state is entered if a valid EOC command is received. If the next frame receives a repetition of the valid command, the state machine transitions forward to the next state or returns to the original main state. These transitions back to the main state are labeled invalid in Figure 6-8.

Usually, the second of the two temporary states decides on the action to perform. In these states, if the third valid command or data message is received, an action is taken.

All messages received from the HTU-C must be verified three consecutive times. This includes data messages and commands such as EOC_RTN and EOC_HLD.

EOC_IDLE

EOC_IDLE is the EOC slave's reset state which is entered on power-on or if the DSL is not ready. Because EOC communication should also be possible if the RT transmits scrambled 1s but the HTU-C does not, transparency in this special case is not assumed in both directions but is dependent on the framer being synchronized.

EOC_ECHO_RESPONSE

EOC_ECHO_RESPONSE can be regarded as a ready state. If there is no EOC transmission in progress, the HTU-C sends EOC_RTN or EOC_HOLD, causing the EOC slave to remain in EOC_ECHO_RESPONSE. If a different message is received from the EOC channel, the message is saved and the EOC_CMD_RESPONS_1 state is entered. Changes of the received message within the next two frames cause a return to EOC_ECHO_RESPONSE.

If the message remains constant for three consecutive frames, it is treated as a command and interpreted using a switch instruction. For all commands, the RT transmits a response to the HTU-C to inform the EOC master that the command has been received. The HTU-C treats this response as an echo.

Table 6-8 lists the five types of commands processed in the EOC_CMD_RESPONS_2 state and the next state.

Table 6-8. EOC Command Processing in EOC_CMD_RESPONS_2

Command Type	Next State
Message/Echo Response	EOC_ECHO_RESPONSE
Data Read	EOC_DATA_RD_STATE
Data Write	EOC_DATA_WR_STATE
Unknown command	EOC_UTC_STATE
Invalid command	EOC_ECHO_RESPONSE

For message and echo response commands, an EOC task flag is set to notify the EOC task handler. In certain applications, the EOC_CMD_RESPONS2 state could become application-dependent.

For data read and data write commands, a data transfer is required. Based on which register is being read or written, the EOC slave sets the values for the array index of eocData.buffer[] and the buffer size as listed in Table 6-9. These variables are called dataBuffIndex and dataBuffSize.

Table 6-9. Buffer Values for EOC Registers (RT side)

EOC Register	dataBuffIndex	DataBuffSize
EocRegA	0	1
EocRegD	3	1
EocRegF[3]	7	3

EOC_UTC_STATE

EOC_UTC_STATE is a lock state that is entered when an unknown command is received three times. This state can only be left by receiving an EOC_RTN or EOC_HOLD command from the EOC master.

EOC_DATA_WR_STATE

The EOC_DATA_WR_STATE is a starting state for a sequence of one or more write operations. A write operation is started with an EOC message that contains data and has the odd-bit set to 1. If the message is received consecutively three times, the value is written to the register specified by the EOC command (e.g., EOC_WRITE_REGISTER_F) and the EOC_DATA_WR_STATE is re-entered.

The EOC_DATA_WR_STATE distinguishes between an EOC_END_OF_DATA command and a data byte by checking the message/data indicator bit (eoc3). If eoc3 is high, then EOC bits 6 to 13 represent a command.

In the EOC_NEXT_WRITE2 state, the dataRecCtr counter records the number of data bytes received. In EOC_DATA_WR_STATE, dataRecCtr must be less than dataBuffSize before more data is accepted. If dataRecCtr equals or exceeds dataBuffSize, the next state is EOC_UTC_STATE.

The write operation is completed if the HTU-R receives three consecutive EOC_END_OF_DATA messages. This sequence of states is shown as EOC_WR_END1 and EOC_WR_END2. When the third EOC_END_OF_DATA is received, the state machine transitions from EOC_WR_END2 to EOC_ECHO_RESPONSE and notifies the EOC task handler that the write operation is complete. EOC_WRITE_END2 is the second state that may contain application-specific code.

EOC_DATA_RD_STATE

A read operation is similar to a write operation from the HTU-R's point of view. The data read command specifies which register to read, and the HTU-C sends the EOC_NEXT_BYTE command with the odd-bit set to 1. After receiving three consecutive EOC_NEXT_BYTE requests, the EOC slave reads the register and transmits the value to the HTU-C.

As illustrated in Figure 6-8, the read operation is stopped when the HTU-C sends EOC_RTN or EOC_HOLD.

6.7.4.2 EOC Task Handler Related to the EOC Slave

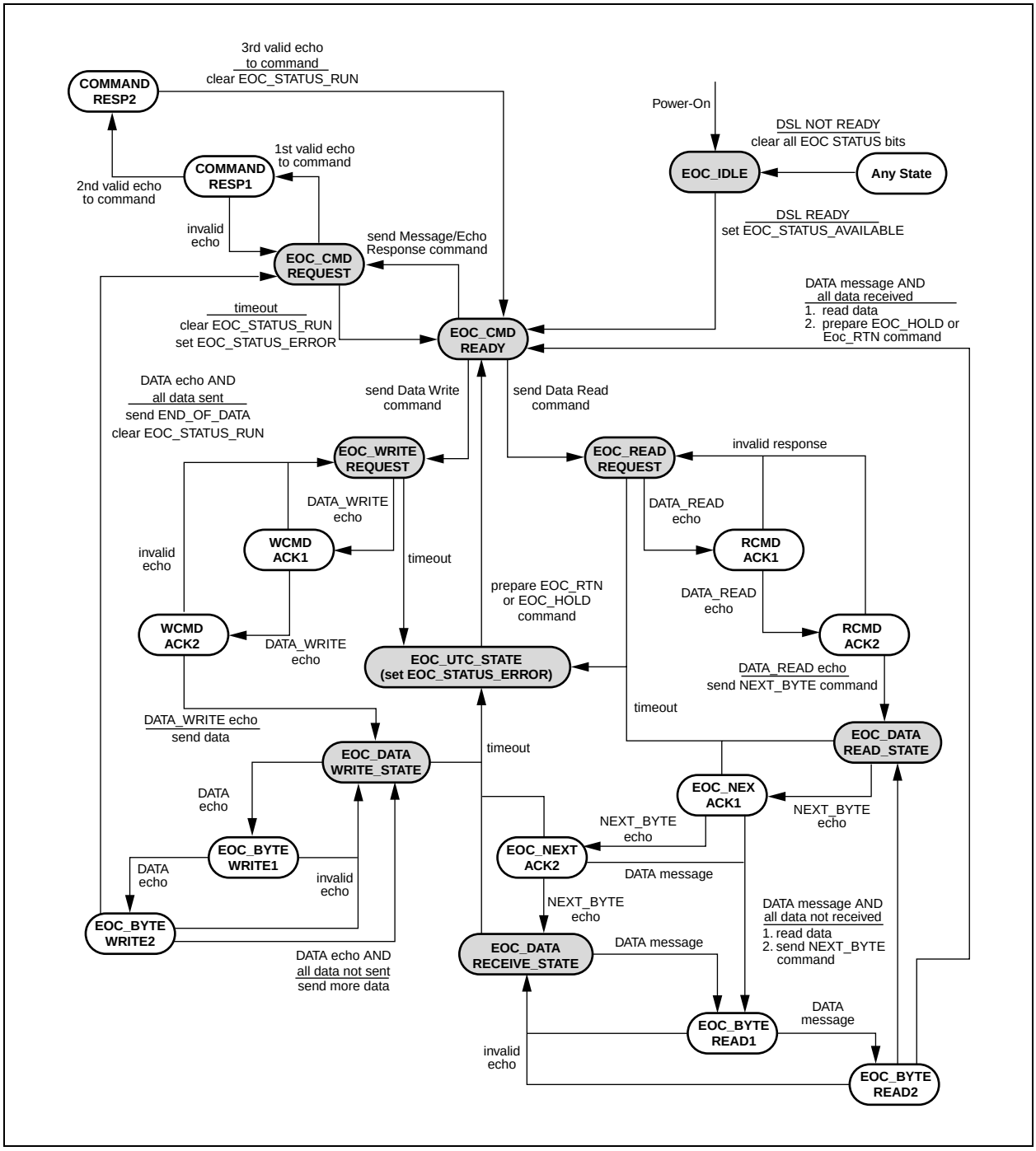
The EOC task handler is a function that performs application-specific actions after an EOC command is received or a data transfer is completed. It is called each time the EOC slave is called, but takes action only if an appropriate flag is set. After the API command is successfully written to eocRegF[1]–eocRegF[3], the eocTasks.bits.eocWrRegF flag is set to inform the EOC task handler that the EOC operation is complete, and API contents are now available. The EOC task handler can then informs the application to execute the requested API command.

6.7.5 EOC Handling on the HTU-C Side

6.7.5.1 EOC Master

Figure 6-9 illustrates the state transition diagram for the EOC master.

Figure 6-9. State Transition Diagram for the EOC Master



EOC_IDLE

Similar to the HTU-R, this state is the EOC master reset state and is entered at power-on. For all states, if DSL_NOT_READY becomes true, EOC_IDLE state is also entered. In this state, the DSL_READY flag is checked and if true, it indicates that the EOC channel is available. In this case, the EOC master enters the EOC_CMD_READY state and sets the EOC_STATUS_AVAILABLE flag. For a complete description of the EOC status flags, refer to Table 6-11.

EOC_CMD_READY

The EOC_CMD_READY state indicates that the EOC master is ready to perform EOC activities. When an application requests the EOC task handler to send an EOC message, the EOC task handler sets the EOC_STATUS_RUN flag to request the EOC master to start the EOC message transfer. In the EOC_CMD_READY state, the EOC master determines which command to transmit. If the eocCtrl.command byte is a data read or data write command, the EOC master transfers the command to the TEOC register. It then enters either the EOC_READ_REQUEST or EOC_WRITE_REQUEST state and waits for the first echo from the HTU-R. If the value in the eocCtrl.command byte is a Message/Echo Response command, the command is sent and a first response is awaited in the EOC_CMD_REQUEST state. The EOC master does not check the validity of the command, which allows for the addition of future commands without requiring modifications to the EOC master.

EOC_UTC_STATE

The EOC master does not directly check for the receipt of an EOC_UTC response; however, many states will time-out if the desired response is not received. The time-out period is 20 HDSL frames. Receipt of an EOC_UTC response eventually causes these states to time-out and enter the EOC_UTC_STATE.

In the EOC_UTC_STATE, the EOC master notifies the EOC task handler of an error condition by setting the EOC_STATUS_ERROR flag.

If the EOC_STATUS_HOLD flag is set, the EOC master sends an EOC_HOLD command; otherwise, it sends an EOC_RTN command. If the EOC slave is in EOC_UTC_STATE, either of these commands cause it to transition into EOC_ECHO_RESPONSE. From the EOC_UTC_STATE, the EOC master transitions to EOC_CMD_READY. The EOC_STATUS_RUN flag remains set indicating that the communication is not completed.

EOC_CMD_REQUEST

In this state, the EOC master requires the first echo response from the EOC slave to the previous command. If it receives two echoes, it goes through COMMAND_RESP1 and COMMAND_RESP2. On the third echo, the EOC master clears the EOC_STATUS_RUN flag. This flag informs the EOC task handler to start the next action or to perform some application-specific activities as a result of the completed EOC action.

EOC_WRITE_REQUEST

The EOC_WRITE_REQUEST state is similar to EOC_CMD_REQUEST. The EOC master waits for the first echo response to the previous Data Write command. If three consecutive echoes are received, the EOC master transitions to the EOC_DATA_WRITE_STATE. If no response is received within 20 HDSL frames, the Data Write sequence is aborted and the EOC master enters EOC_UTC_STATE.

EOC_READ_REQUEST

Similar to EOC_CMD_REQUEST, the EOC master waits for the first echo response to the previous Data Read command. If three consecutive echoes are received, the EOC master transitions to EOC_DATA_READ_STATE. If an echo is not received within 20 HDSL frames, the Data Read command is aborted and the EOC master enters the EOC_UTC_STATE.

EOC_DATA_WRITE _STATE

On the transition from WCMD_ACK2 to EOC_DATA_WRITE_STATE, the EOC master transmits a data message that contains a value from the eocData.buffer[] array. The index of the array is preset by the EOC task handler and is incriminated each time a byte is sent to the HTU-R.

If the EOC master receives three consecutive data echoes, it enters EOC_BYTE_WRITE2, which checks if all register data was sent. If all data was not sent, EOC_DATA_WRITE_STATE is re-entered; otherwise, an END_OF_DATA command is sent and the EOC master enters EOC_CMD_REQUEST.

EOC_DATA_READ_STATE

On the transition from RCMD_ACK2 to EOC_DATA_READ_STATE, the EOC task handler transmits an EOC_NEXT_BYTE command to request a data response from the HTU-R.

The transmitted odd-even bit is set to the current value of eocCtrl.oddEvenCtr. On the transition from RCMD_ACK2 to EOC_DATA_READ_STATE, this bit is set by default to an odd value set on the EOC_CMD_READY to EOC_READ_REQUEST transition. On the transition from EOC_BYTE_READ2 to EOC_DATA_READ_STATE, the eocCtrl.oddEvenCtr bit is toggled.

If the HTU-R receives three consecutive EOC_CMD_NEXT_BYTE commands, it sends the requested value in a data message to the HTU-C. If the EOC master receives three EOC_NEXT_BYTE commands, it enters EOC_DATA_RECEIVE_STATE. If a data message is received in EOC_NEXT_ACK1 or EOC_NEXT_ACK2, EOC_BYTE_READ1 is entered.

EOC_DATA_RECEIVE _STATE

The EOC_DATA_RECEIVE_STATE waits for the reception of data messages if three consecutive EOC_NEXT_BYTE messages are received. It also provides a re-try procedure if three consecutive data messages are not received. The re-try occurs if the EOC master transitions from EOC_BYTE_READ1 or EOC_BYTE_READ2 to EOC_DATA_RECEIVE_STATE.

If three consecutive data messages are received, the received data is written to the eocData.buffer[] array on the transition from EOC_BYTE_READ2 to either EOC_DATA_READ_STATE or EOC_CMD_READY. If all data was received, the EOC master prepares to send an EOC_RTN or EOC_HOLD command and enters EOC_CMD_READY. If all data was not received, the EOC master enters EOC_DATA_READ_STATE and sends EOC_NEXT_BYTE to request more data from the HTU-R.

6.7.5.2 EOC Task Handler Related to the EOC Master

The EOC task handler on the HTU-C coordinates requests for EOC services that may come from different parts of the application. Similar to the HTU-R, the application sets flags to inform the EOC task handler which tasks to perform, as listed in Table 6-10.

Table 6-10. Tasks of the HTU-C-EOC Task Handler

EOC Task Flag	Action
EocTask.bits.eocRdRegA	Request the API result after sending a status checking API command through EOC channel.
EocTask.bits.eocZipStatus	Request Zip status of HTU-R (eocRegD is used to store the Zip status).
EocTask.bits.eocWrRegF	Send an API command stored in eocRegF[1]–eocRegF[3] to HTU-R.

If the EOC_STATUS_AVAILABLE flag is set, the EOC task handler knows that the EOC master is ready and prepares the appropriate EOC action. For this preparation, it uses a special data structure that controls the EOC master and looks like the text below:

```
typedef struct
{
BP_U_8BIT status;         /* current status of EOC master*/
BP_U_8BIT command;        /* requested command to perform*/
BP_U_8BIT dataSizeCtr;    /* number of bytes to xmit/receive*/
BP_U_8BIT dataBuffIndex;  /* current index within eocData.buffer*/
BP_U_8BIT eocSlave;       /* EOC-Slave to access (0=RT,1=Repeater)*/
BP_U_8BIT oddEvenCtr;     /* buffer for current ODD/EVEN-flag*/
}Bt8953_EOC_CTRL;
```

This structure is used only on the HTU-C side and is assigned to the variable eocCtrl. It is used by both the EOC master and the EOC Task Handler.

The elements of the eocCtrl data structure are explained below:

Status

The status byte contains information about the EOC master and EOC task handler activities. Individual bits are cleared as described in Table 6-11; all status bits are cleared upon initialization and when the EOC master enters the EOC_IDLE state. The bits are defined in the file CU_EOC.H and listed in Table 6-11.

Table 6-11. Status Flags within the EOC Master

Bit	Status Flag	Description
0	EOC_STATUS_AVAILABLE	Set by EOC master if its state is not EOC_IDLE; otherwise, the flag is cleared. Checked by the EOC task handler before it checks for EOC task flags set by applications.
1	EOC_STATUS_BUSY(1)	Only one EOC message sequence at a time can be started. The EOC task handler uses this flag as a semaphore for EOC services. It checks the flag before starting a EOC message sequence, sets it while waiting for the EOC message sequence to complete, and clears it after completing the EOC service requested by an application.
2	EOC_STATUS_RUN	Set by the EOC task handler to request the EOC master to start an EOC transmission. Checked by the EOC master in the EOC_CMD_READY state. Cleared by the EOC master to inform the EOC task handler that the EOC transmission is complete.
3	EOC_STATUS_HOLD	Indicates a hold condition in the HTU-R. This flag is set if a latched EOC command is sent to the HTU-R. The EOC task handler assumes that these EOC commands are latched by the HTU-R: -EOC_LOOP_NTU -EOC_NTU_CCRC_REQ The EOC master clears the flag if it sends either a EOC_HTU-RN command or a EOC_NTU_CCRC_END command and receives three echoes. The EOC master checks the flag to determine whether to send a EOC_HTU-RN or an EOC_HOLD command as part of the Data Read command sequence.
4	EOC_STATUS_ERROR	Set by the EOC master if it enters the EOC_UTC_STATE state. Checked by the EOC task handler to determine if the EOC transmission was successful or not.

NOTE(S):
(1) A task that uses the EOC master is responsible for releasing it after finishing the EOC communication by clearing the EOC_STATUS_BUSY semaphore flag. Furthermore, the application must not use a spin loop to wait for the completion of the EOC action, as indicated by the EOC_STATUS_RUN flag. A spin loop would delay the processing of other tasks by at least 30 ms.

Command

The command byte tells the EOC master which command type to transmit. The EOC master only distinguishes among three different command types (Message/Echo Response, Data Read, and Data Write). For each particular Data Read and Data Write command, the EOC task handler loads the dataSizeCtr and dataBuffIndex variables with the correct values.

dataSizeCtr

The dataSizeCtr byte specifies the number of bytes to transmit from the HTU-C to the HTU-R in a Data Write command, or the number of bytes to receive in a Data Read command. The value corresponds to the size (in bytes) of the EOC register that is written or read.

dataBuffIndex

The dataBuffIndex byte specifies the start index of the eocData.buffer[] array. The index points to the first byte transmitted as data in a Data Write command sequence or the first location assigned when data is received in a Data Read command sequence. Table 6-12 lists the buffer values for the EOC registers.

Table 6-12. Buffer Values for EOC Registers (HTU-C side)

EOC Register	DataBuffIndex	DataSizeCtr
EocRegA	0	1
EocRegD	3	1
EocRegF[3]	5	3

eocSlave

The eocSlave byte corresponds to the destination address in Table 6-7. This byte is provided to allow accessing two EOC slaves with different EOC addresses. Because there is no repeater in the system, this byte is always set to the HTU-R's address.

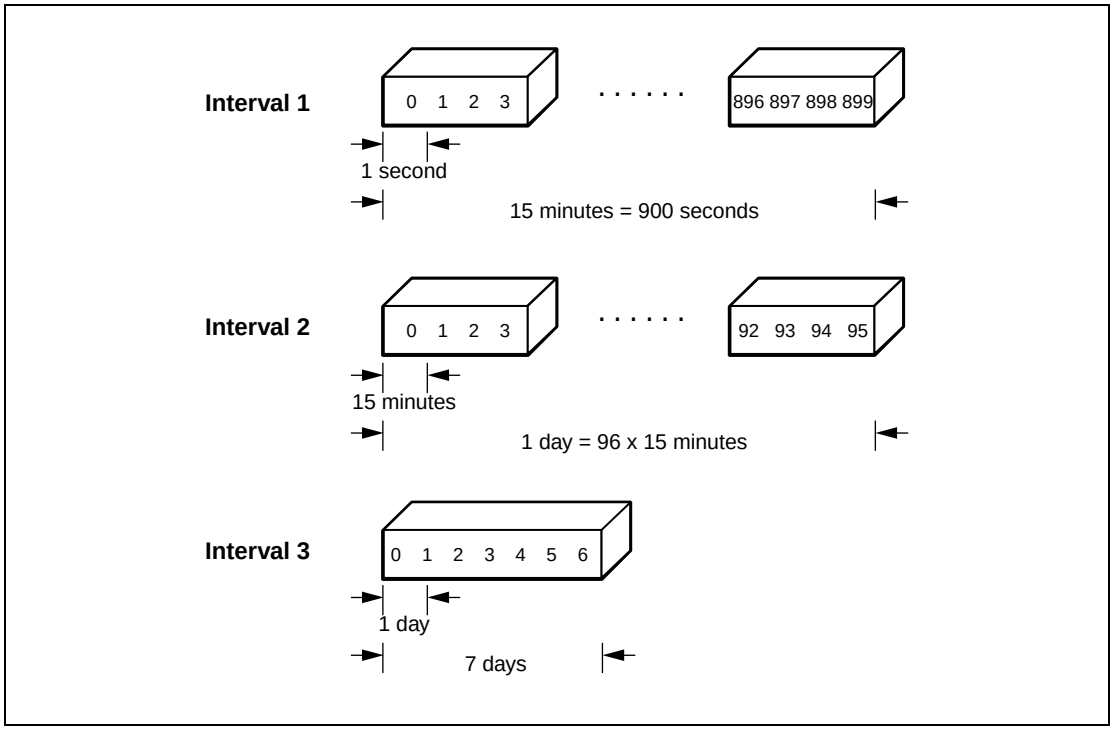
oddEvenCtr

The oddEvenCtr byte corresponds to the odd-even bit in Table 6-7. The bit toggles each time a data byte has been transferred over the EOC channel. The software implements the toggle operation using an addition operation, which explains the use of "Ctr" in the variable name. The element is only used and initialized by the EOC master and should not be used by other tasks.

6.8 Performance Monitoring

Performance monitoring maintains a history of CRC and FEBE errors at different time intervals. Figure 6-10 illustrates the structure of the records.

Figure 6-10. CRC and FEBE Error Records at Three Time Intervals



Interval1 records the number of CRC and FEBE errors occurring every second during a 15-minute interval and are updated every second. Interval2 records the accumulated CRC and FEBE errors occurring in each 15-minute interval for 24 hours and are updated every 15 minutes. Interval3 records the accumulated CRC and FEBE errors occurring every 24 hours for 7 days and are updated daily.

6.9 Channel Blocking

Channel blocking allows selected PCM time slots to be used or replaced with DBANK_1 values. The number of time slots is constant, but the active number of time slots can vary. Channel blocking should be done at HTU-C only; it causes temporary data corruption. To simplify the channel blocking operation, APIs are provided to select or disable a PCM time slot. Once the PCM time slot usage is determined, another API command can be issued to reconfigure the transmit routing table, receive combination table, transmit payload mapper, and receive payload mapper based on the time slot selection.

6.10 T1/E1 Framer and LIU Support

The channel unit code 6.0 provides support for a T1/E1 Framer and LIU (Line Interface Unit). The framer and LIU code contain the minimum drivers necessary to configure the framer and LIU for a transparent mode. The code supports the Bt8360 (T1), Bt8370(T1/E1), and Bt8510 (E1) framers. The code only supports the Bt8069 LIU. In addition, a minimal set of API commands is supported to configure the framer and LIU from a host processor.

7.0 Serial Communication Interface

7.1 Communication Protocol

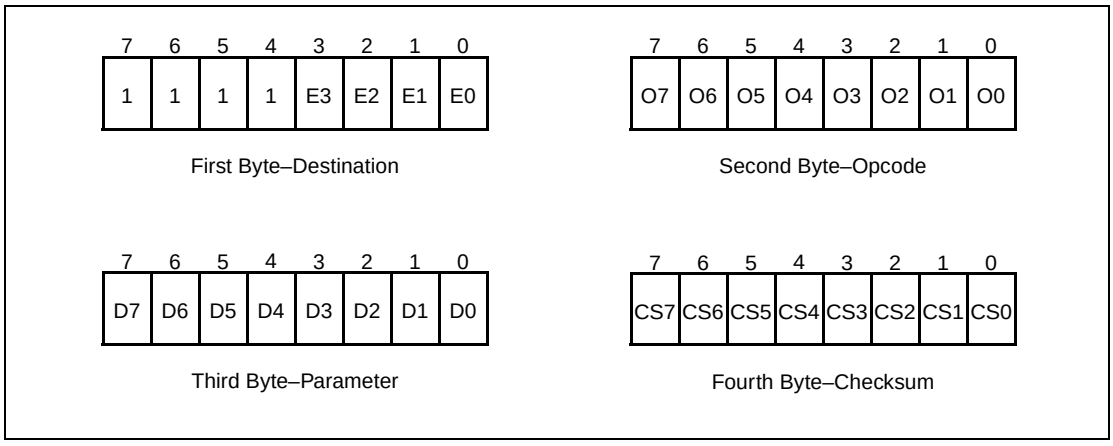
The 8032 communicates with the host processor using a standard UART interface. The physical connection includes two lines: RXD (pin 24) and TXD (pin 25). The data is transferred in asynchronous format: 19200 baud, 1 start bit, 8 data bits, 1 stop bit, no parity.

7.2 Message Structure

All messages are 4 bytes long. Figure 7-1 illustrates the structure of a message sent by the host processor to the 8032. The first 3 bytes are the command bytes, the fourth byte (last transmitted byte) contains checksum information that is a function of the first 3 bytes. See Section 7.4 for checksum details. The checksum considerably reduces the probability of the 8032 misinterpreting an incoming message.

The commands are interpreted according to the destination, opcode and parameter fields.

Figure 7-1. Host Processor to 8032 Message Structure



7.2.1 Destination Field (Bits E3–E0)

Table 7-1 lists the destination field to which the command is targeted.

Table 7-1. Destination Field Specification

E3	E2	E1	E0	Destination
0	0	0	0	_BIT_PUMP0
0	0	0	1	_BIT_PUMP1
0	0	1	0	_BIT_PUMP2(1)
0	0	1	1	_CU_COMMON
0	1	0	0	_ CU_CHAN1
0	1	0	1	_CU_CHAN2
0	1	1	0	_CU_CHAN3(1)
1	0	0	0	_DSL_APPLICATION
1	0	0	1	_DSL_CHANNEL0
1	0	1	0	_DSL_CHANNEL1
1	0	1	1	_DSL_CHANNEL2(1)

NOTE(S):
(1) Reserved for future designs.

7.2.2 Opcode Field (Bits O7–O0)

The opcode field specifies the command or status request to be executed.

7.2.3 Parameter Field (Bits P7–P0)

The parameter field uses commands where additional data or parameter selection is required. In commands with no need for additional data, zeros are placed as the data byte to ensure future compatibility.

7.3 Message Transfer Protocol

The application sends a command to any of the bit pumps in the system by transmitting a message over the serial communication channel. Every command that is correctly received and decoded by the 8032 is acknowledged by sending an acknowledge message back to the application. In response to a status request command, the 8032 sends a status response message containing the requested information.

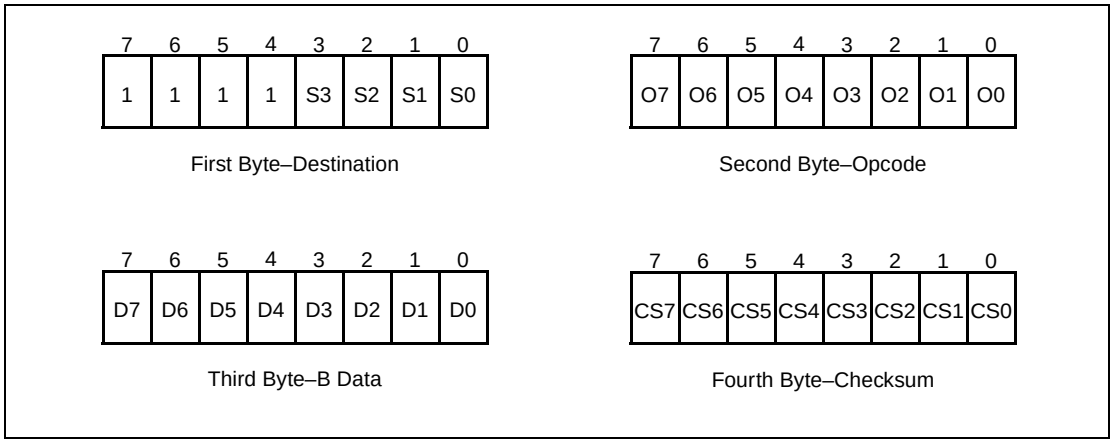
The 8032 acknowledges a received message within 200 ms, except during activation where larger delays (up to 2 seconds) may be present. The host processor retransmits a message not acknowledged within this time limit. No new message from the host processor should be sent before the previous one is acknowledged, unless the time limit has been exceeded.

When the 8032 receives a status request command, it responds (after acknowledging the command) by sending a status message to the host processor. The structure of a status message is similar to the structure of a host-to-8032 message, as illustrated in Figure 7-2.

The first 2 bytes are identical to the first 2 bytes of the corresponding status request command. Bits S3–S0 of the first byte are interpreted, according to Table 7-1, as the source bit pump for the status response. The second opcode byte (bits O7–O0) contains the opcode of the command requesting the information.

The third byte (bits D7–D0) contains the requested information. The fourth byte (bits CS7–CS0) is the checksum value, calculated according to the equation described in Section 7.4.

Figure 7-2. 8032 to Host Processor Message Structure



7.4 Checksum Function

For every command sent by the host processor, a checksum function value is calculated and sent as the fourth byte of the message. This value is calculated using the following equation:

$$CS = (\text{Byte \#1}) \oplus (\text{Byte \#2}) \oplus (\text{Byte \#3}) \oplus (0xAA)$$

Where "⊕" denotes a bit-wise exclusive OR operation, and 0xAA is the binary byte 10101010.

The 8032 uses the same rule to calculate the checksum byte of status message sent to the host processor.

7.5 Acknowledge Message

The format of the acknowledge message is as follows:

Table 7-2. Acknowledge Message

Byte 1	Byte 2	Byte 3	Checksum
0xFF	0xFF	0xFF	0x55

8.0 API Command Set

Channel Unit Code 6.0 provides the complete software suite in order to simplify the integration of the module into the system. A modular API-based command set allows a seamless initialization of the application layer and Channel Unit. Once in operation, a comprehensive set of diagnostic and testing commands assist in performance monitoring and fault detection and isolation tasks. The command set is grouped at 3 levels:

- Level 1 API commands
  API commands at this level are used at normal operation, and are provided to access the system at the application layer. They can be used to check the system version, control the activation state, and monitor the performance for each individual loop.
- Level 2 API commands
  API commands at this level are used to configure the system (such as channel blocking), diagnose problems and check DSL status.
- Level 3 API commands
  API commands at this level are lower level Channel Unit-related APIs. They can be used to access the Channel Unit directly.

Table 8-1 through Table 8-118 describes the commands for all three levels.

8.1 Level 3 API Commands

Level 1 API commands are used during normal operation.

8.1.1 DSL Control

8.1.1.1 Reset the DSL

Table 8-1. Opcode: 0x01 (_DSL_RESET)

Destination	Description	Parameter
_DSL_APPLICATION	Restart the program, reset all internal registers, and set all programmable options to default values.	0x01

8.1.1.2 Enable or Disable Activation State Manager

Table 8-2. Opcode: 0x02 (_DSL_ASM_ENABLE)

Destination	Description	Parameter
_DSL_APPLICATION	Enable or disable the activation state manager.	0x00 (disable) 0x01 (enable)

When the Activation State Machine (ASM) is enabled, HTU-C and HTU-R will go through the activation process (Figures 5-4 and 5-5) to reach the normal operation. When the ASM is disabled, the link is not activated.

8.1.2 DSL Status

8.1.2.1 History of Link in Sync Status

Table 8-3. Opcode: 0x85 (_DSL_AVAILABLE_SECONDS)

Destination	Description	Parameter
_DSL_CHANNEL0–2	Return the available seconds for specific loop.(1)	0x00 (Byte 1) 0x01 (Byte 2) 0x02 (Byte 3) 0x03 (Byte 4)
NOTE(S): (1) Available seconds is a 32-bit variable.		

Available seconds for a specific loop is the accumulated time that the loop is in normal operation since power-on.

8.1.2.2 Overall DSL Status

Table 8-4. Opcode: 0x82 (_DSL_STATUS)

Destination	Description	Parameter
_DSL_APPLICATION	Return the overall DSL status.	0x00 (Status register 0) 0x01 (Status register 1)

Table 8-5. Status Register 0

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Terminal type	Reserved	Alarm	Sync_1	Sync_2	Test mode	T1_E1_mode	Reserved

T1_E1_mode: 0 – E1
1 – T1

Test Mode: 0 – Normal Operation
1 – Test Mode

Sync_1: 0 – Loop 1 out of sync
1 – Loop 1 in sync

Sync_2: 0 – Loop 0 out of sync
1 – Loop 0 in sync

Alarm: 0 – Normal Operation
1 – Bit pump or Channel Unit not present

Terminal Type: 0 – HDSL Terminal Unit Central (HTU-C)
1 – HDSL Terminal Unit Remote (HTU-R)

Table 8-6. Status Register 1

Bit 7	Bit 5–6	Bit 3–4	Bit 2	Bit 1	Bit 0
Reserved	Good_loop_cnt	Master_loop	Loop_reversal	Cu_present	Reserved

Cu_present: 0 – Channel Unit not present
1 – Channel Unit present

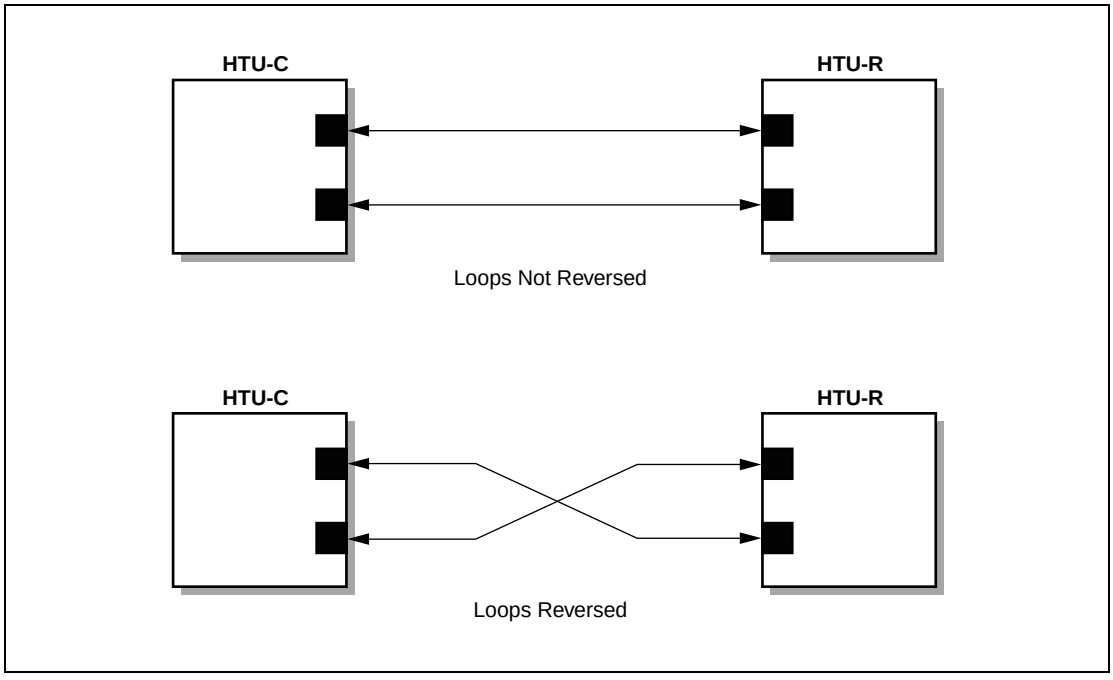
Loop_reversal: 0 – Loops not reversed
1 – Loops reversed

Master_loop: 00 – Loop 0 is the master loop
01 – Loop 1 is the master loop
10 – Reserved for future use
11 – Reserved for future use

Good_loop_cnt: 0 – None of the loops are in normal operation
1 – 1 loop is in normal operation
2 – 2 loops are in normal operation

NOTE: Loop_reversal returns a value representing whether or not the physical loops connections are reversed. When a loop reversal occurs, only the HTU-R reverses the loops. The HTU-C never detects any loop reversal; therefore, the user only needs to check the loop reversal at the HTU-R.

Figure 8-1. Loop Reversal



8.1.2.3 DSL Loop Status

Table 8-7. Opcode: 0x83 (_DSL_LOOP_STATUS)

Destination	Description	Parameter
_DSL_CHANNEL0–2	Return the loop status.	0x00
Bit 3–7	**Bit 1–2**	**Bit 0**
Activation_state	Zip_start_attempt	Present

Present: 0 – Bit pump not present
1 – Bit pump present

Zip_start_attempt: 0 – Cold startup
>0 – ZipStartup

Activation_state: 0 – SYSTEM_IDLE
1 – CONFIGURATION_STATE
2 – INACTIVE_STATE
3 – ACTIVATING_STATE
4 – ACTIVATING_STATE_S1
5 – ACTIVE_RX_STATE
6 – ACTIVE_TX_STATE
7 – GOTO_PID_VALIDATION_STATE
8 – PID_VALIDATION_STATE
9 – GOTO_ACTIVE_TX_RX_STATE
10 – ACTIVE_TX_RX_STATE
11 – PENDING_DEACTIVATED_STATE
12 – DEACTIVATED_STATE
13 – WAIT_FOR_LOST
14 – WAIT_FOR_LOS
20 – BACKGROUND_TESTMODE
21 – ERLE_TESTMODE
22 – AAGC_TESTMODE
23 – AAGC2_TESTMODE
25 – ANALOG_LB_TESTMODE
26 – ANALOG_LB2_TESTMODE

8.1.2.4 DSL ZipStartup Status

Table 8-8. Opcode: 0x84 (_DSL_ZIP_STATUS)

Destination	Description	Parameter
_DSL_CHANNEL0–2	Return the ZipStartup status.	0x00 (_LOCAL)
Bit 4–7	**Bit 1–3**	**Bit 0**
Reserved	Zip_start_state	Enable

Enabled: 0 – ZipStartup is disabled
1 – ZipStartup is enabled

Zip_start_state: 0 – ZIP_START_IDLE
1 – ZIP_START_VALIDATION_STATE0
2 – ZIP_START_VALIDATION_STATE1
3 – ZIP_START_UPDATE_STATE
4 – ZIP_START_WAIT_STATE

8.1.2.5 DSL Version

Table 8-9. Opcode: 0x81 (_DSL_VERSION)

Destination	Description	Parameter
_DSL_APPLICATION	Return the system version.	_DSL_SW_VERSION (0x00) _DSL_HW_VERSION (0x01)

The actual software version is the return value divided by 10. For instance, if the return value is 12, the software version is 1.2.

8.1.3 Performance Monitoring

Channel Unit Code 6.0 maintains CRC and FEBE error history at three different intervals. See Section 6.8 for more information.

8.1.3.1 Enable or Disable Performance Monitoring Update

Table 8-10. Opcode: 0x10 (_SET_PERFMONITOR_STATE)

Destination	Description	Parameter
_DSL_CHANNEL0–2	Enable or disable the performance monitoring update.	0x00 (disable) 0x01 (enable)

8.1.3.2 Set Starting Address to Check Performance Record at Interval 1

Table 8-11. Opcode: 0x11 (_INTERVAL1_ADDR_LO)

Destination	Description	Parameter
_DSL_CHANNEL0–2	Set the lower byte of the starting index to check network management records at Interval1.	0x00

Table 8-12. Opcode: 0x12 (_INTERVAL1_ADDR_HI)

Destination	Description	Parameter
_DSL_CHANNEL0–2	Set the higher byte of the starting record index to check network management records at Interval1.	0x00

8.1.3.3 Set Starting Address to Check Performance Record at Interval 2

Table 8-13. Opcode: 0x13 (_INTERVAL2_ADDR)

Destination	Description	Parameter
_DSL_CHANNEL0–2	Set the starting record index to check network management records at Interval2.	0x00

8.1.3.4 Set Starting Address to Check Performance Record at Interval 3

Table 8-14. Opcode: 0x14 (_INTERVAL3_ADDR)

Destination	Description	Parameter
_DSL_CHANNEL0–2	Set the starting record index to check network management records at the Interval3.	0x00

8.1.3.5 Performance Records at Different Intervals

Interval1 performance records are 8 bits each, and Interval2 and 3 records are 32 bits each. After setting the starting record index, the internal pointer is pointed to read the first byte of the record. For the 32-bit records, the internal pointer increments after each call; the record index is incremented only when the fourth byte of the record is read. For 8-bit records, the record index increments after each call.

Table 8-15. Opcode: 0x90 (_CRC_ERR_AT_INTERVAL1)

Destination	Description	Parameter
_DSL_CHANNEL0–2	Return the CRC error at Interval1 at the record index. The record index increments after each call.	0x00

Table 8-16. Opcode: 0x91 (_CRC_ERR_AT_INTERVAL2)

Destination	Description	Parameter
_DSL_CHANNEL0–2	Return the CRC error at Interval2 at the record index.	0x00

Table 8-17. Opcode: 0x92 (_CRC_ERR_AT_INTERVAL3)

Destination	Description	Parameter
_DSL_CHANNEL0–2	Return the CRC error at Interval3 at the record index.	0x00

Table 8-18. Opcode: 0x93 (_FEBE_ERR_AT_INTERVAL1)

Destination	Description	Parameter
_DSL_CHANNEL0–2	Return the FEBE error at Interval1 at the record index. The record index increments after each call.	0x00

Table 8-19. Opcode: 0x94 (_FEBE_ERR_AT_INTERVAL2)

Destination	Description	Parameter
_DSL_CHANNEL0–2	Return the FEBE error at Interval2 at the record index.	0x00

Table 8-20. Opcode: 0x95 (_FEBE_ERR_AT_INTERVAL3)

Destination	Description	Parameter
_DSL_CHANNEL0–2	Return the FEBE error at Interval3 at the record index.	0x00

8.1.3.6 Latest Performance Record at Different Intervals

The last CRC/FEBE performance records show the latest errors for each interval. Interval1 is updated every second and returns the CRC/FEBE error for the previous second. The Interval2 is updated every 15 minutes and shows the errors for the previous 15 minutes. Interval3 is updated every 24 hours and shows the errors for the previous 24 hours.

NOTE: These API commands do not require Performance Monitoring Update to be disabled.

Table 8-21. Opcode: 0x96 (_LAST_CRC_ERR_INTERVAL1)

Destination	Description	Parameter
_DSL_CHANNEL0–2	Return the number of CRC errors during the last second for the specific loop.	0x00

Table 8-22. Opcode: 0x97 (_LAST_CRC_ERR_INTERVAL2)

Destination	Description	Parameter
_DSL_CHANNEL0–2	Return the number of CRC errors during the last 15 minutes for the specific loop.	0x00 (Byte 1) 0x01 (Byte 2) 0x02 (Byte 3) 0x03 (Byte 4)

Table 8-23. Opcode: 0x98 (_LAST_CRC_ERR_INTERVAL3)

Destination	Description	Parameter
_DSL_CHANNEL0–2	Return the number of CRC errors during the last hour for the specific loop.	0x00 (Byte 1) 0x01 (Byte 2) 0x02 (Byte 3) 0x03 (Byte 4)

Table 8-24. Opcode: 0x99 (_LAST_FEBE_ERR_INTERVAL1)

Destination	Description	Parameter
_DSL_CHANNEL0–2	Return the number of FEBE errors during the last second for the specific loop.	0x00 (Byte 1)

Table 8-25. Opcode: 0x9A (_LAST_FEBE_ERR_INTERVAL2)

Destination	Description	Parameter
_DSL_CHANNEL0–2	Return the number of FEBE errors during the last 15 minutes for the specific loop.	0x00 (Byte 1) 0x01 (Byte 2) 0x02 (Byte 3) 0x03 (Byte 4)

Table 8-26. Opcode: 0x9B (_LAST_FEBE_ERR_INTERVAL3)

Destination	Description	Parameter
_DSL_CHANNEL0–2	Return the number of FEBE errors during the last hour for the specific loop.	0x00 (Byte 1) 0x01 (Byte 2) 0x02 (Byte 3) 0x03 (Byte 4)

8.2 Level 2 API Commands

Level 2 API commands configure the system for other than standard 1T1 and 1E1, isolate and identify problems, and check important system status parameters.

8.2.1 Channel Blocking

8.2.1.1 Channel Blocking Time Slot Location

Table 8-27. Opcode: 0x30 (_CB_TIMESLOT_LOCATION)

Destination	Description	Parameter
_DSL_APPLICATION	Set the time slot number that needs to be blocking.	0 ~ 31 for 2E1 blocking 0 ~ 23 for 2T1 blocking

8.2.1.2 Channel Blocking Time Slot Enable/Disable

Table 8-28. Opcode: 0x31 (_CB_TIMESLOT_STATE)

Destination	Description	Parameter
_DSL_APPLICATION	Enable or disable selected time slot.	0 _TIMESLOT_BLOCKED 1 _TIMESLOT_IN_USE

8.2.1.3 Channel Blocking Configuration

Table 8-29. Opcode: 0x32 (_CONFIGURE_CHANNEL_BLOCKING)

Destination	Description	Parameter
_DSL_APPLICATION	Configure the Channel Unit based on the time slot selection.	0x00
NOTE(S): Transmit routing table, receive combine table, transmit payload mapper, and receive payload mapper will be modified based on the time slot selection. Issuing the API command may cause corrupted data for about one second.		

8.2.1.4 Set all Time Slots

Table 8-30. Opcode: 0x33 (_SET_ALL_TIMESLOTS)

Destination	Description	Parameter
_DSL_APPLICATION	Block or unblock all the timeslots.	0 Block all timeslots 1 Unblock all timeslots

8.2.1.5 Channel Blocking Time Slot Usage

Table 8-31. Opcode: 0xA3 (_CB_TIMESLOT_USAGE)

Destination	Description	Parameter
_DSL_APPLICATION	Return the value indicating the current time slot usage.	0x00 Timeslots 0–7 0x01 Timeslots 8–15 0x02 Timeslots 16–23 0x03 Timeslots 24–31

The returned value:

Parameter	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0x00	TS7	TS6	TS5	TS4	TS3	TS2	TS1	TS0
0x01	TS15	TS14	TS13	TS12	TS11	TS10	TS9	TS8
0x10	TS23	TS22	TS21	TS20	TS19	TS18	TS17	TS16
0x11	TS31	TS30	TS29	TS28	TS27	TS26	TS25	TS24

TS0–TS31 0 – Time slot blocked and replaced with DBANK1 values
1 – Time slot in use

8.2.2 Diagnostic

8.2.2.1 DSL Loopbacks

Four loopback tests are supported by the HTU-C and the HTU-R. The API commands place the system in the invoked loopback state. A single _LOOPBACKS_OFF API returns the device to "normal operation" mode. PCM to PCM and HDSL to PCM loopbacks are addressed to _DSL_APPLICATION; analog loopbacks are addressed to the intended _DSL_CHANNEL.

Table 8-32. Opcode: 0x20 (_DSL_LOOPBACK)

Destination	Description	Parameter	
_DSL_APPLICATION	Turn off loopback test for all loops.	_LOOPBACKS_OFF	0
	Turn on loopback test for all loops.	_CU_PCM_ON_PCM _CU_HDSL_ON_PCM _BP_ISOLATED_ANALOG_LOOPBACK _BP_EXTERNAL_ANALOG_LOOPBACK	1 2 3 4

Table 8-33.

Destination	Description	Parameter	
_DSL_CHANNEL0–2	Turn off loopback test for selected loop.	_LOOPBACKS_OFF	0
	Turn on loopback test for selected loop.	_BP_ISOLATED_ANALOG_LOOPBACK _BP_EXTERNAL_ANALOG_LOOPBACK	3 4

8.2.2.2 DSL Test Modes

The DSL Testmode API command supports eight test modes. Parameters 1–4 measure pulse templates for internal use. Parameters 5 and 6 measure the transmit signal power. Parameters 7 and 8 can be used for manufacturing test and analog front-end verification.

Table 8-34. Opcode: 0x21 (_DSL_TESTMODE)

Destination	Description	Parameter	
_DSL_CHANNEL0–2	Operates the bit pump in special test modes.	_BP_TESTMODE_OFF _BP_ISOLATED_PULSE_PLUS3 _BP_ISOLATED_PULSE_PLUS1 _BP_ISOLATED_PULSE_MINUS1 _BP_ISOLATED_PULSE_MINUS3 _BP_FOUR_LEVEL_SCR _BP_TWO_LEVEL_SCR _BP_ERLE_TEST _BP_MEASURE_AAGC	0 1 2 3 4 5 6 7 8

8.2.2.3 Sending API Commands Through EOC Channel

This set of API commands is provided so the HTU-C can send a defined API command through the EOC channel to control the HTU-R or check its status. Because the HTU-C is the EOC master according to the HDSL ANSI/ETSI standard, only the HTU-C can use these API commands. The EOC channel is only available when the associated loop is in normal operation. There are two EOC channels when the system is in normal operation, and the user can choose which channel to use by specifying the destination field.

To implement this feature, first use APIs _API_DEST, _API_OPCODE, and _API_DATA to specify the API command that needs to be sent, then use API command _API_SEND to send it through the chosen EOC channel. If the API sent is a control command, the user sees the expected action at HTU-R; if it is a status command, use _API_RESULT to check the return results from the HTU-R. The reason that an additional API is needed to get the status checking result is that EOC protocol needs time to handle each EOC request.

For example, to send API command _DSL_ZIP_STATUS to HTU-R to determine if ZIP_STARTUP is enabled or not, choose an available EOC channel. Assuming _DSL_CHANNEL0 is available, perform the following steps:

1. Use API _API_DEST with the destination = _DSL_CHANNEL0 (0x09) and parameter = 0x00.
2. Use API _API_OPCODE with the destination = _DSL_CHANNEL0 (0x09) and parameter = _DSL_ZIP_STATUS (0x84).
3. Use API _API_DATA with the destination = _DSL_CHANNEL0 (0x09) and parameter = 0x00.
4. Use API _API_SEND with the destination = _DSL_CHANNEL0 (0x09) and parameter = 0x00.
5. Use API _API_RESULT with the destination = _DSL_CHANNEL0 (0x09) and parameter = 0x00 to check the returned result from HTU-R.

Table 8-35. Opcode: 0x22 (_API_DEST)

Destination	Description	Parameter
_DSL_CHANNEL0–2	Set the destination of the API command.	0–11

Table 8-36. Opcode: 0x23 (_API_OPCODE)

Destination	Description	Parameter
_DSL_CHANNEL0–2	Set the opcode of the API command.	Valid API opcode.

Table 8-37. Opcode: 0x24 (_API_DATA)

Destination	Description	Parameter
_DSL_CHANNEL0–2	Set the parameter of the API command.	Necessary parameter or 0x00 if no parameter is needed.

Table 8-38. Opcode: 0x25 (_API_SEND)

Destination	Description	Parameter
_DSL_CHANNEL0–2	Send the API command through EOC channel.	0x00

Table 8-39. Opcode: 0xA2 (_API_RESULT)

Destination	Description	Parameter
_DSL_CHANNEL0–2	Check the result of the API command (status API only) sent through EOC.	0x00

8.2.3 DSL Status

8.2.3.1 Far End Signal Attenuation

The return value is a 1-byte unsigned integer x (0–255) which indicates the overall signal power attenuation in units of 0.5 dB. For example, a value of 60 means total cable attenuation of 30 dB.

The result is calibrated to represent the overall signal power attenuation over the cable in dB. The result is calibrated for the nominal 13.5 dBm transmit power at the far_end.

Table 8-40. Opcode: 0xB0 (_DSL_FELM)

Destination	Description	Parameter
_DSL_CHANNEL0–2	Request a value of the far_end signal attenuation. This value is based on measuring the average far_end signal level after echo cancellation.	0x00

8.2.3.2 Noise Margin

The return value is a 1-byte signed integer x (–128 through 127) which indicates the noise margin in units of 0.5 dB. For example, a value of –8 means a noise margin of –4 dB.

Table 8-41. Opcode: 0xB1 (_DSL_NMR)

Destination	Description	Parameter
_DSL_CHANNEL0–2	Request a value of the noise margin of the receiver. The noise margin is defined as the maximum tolerable increase in external noise power that still allows for BER of less than 1×10^{-7}.	0x00

8.2.4 ERLE Test

8.2.4.1 Background and ERLE Test Mode

NOTE: The Background Noise Test and ERLE Test are typically run with the NLEC bypassed, transmitter set to 4-level, and AAGC set to 12 db. A 150 Ω termination between the Tx and Rx signals on the line side of the transformer is used for benchmarking.

Table 8-42. Opcode 0x18 (_ERLE_TEST_MODE)

Destination	Description	Parameter
BITPUMP0–2	Allows user to customize ERLE test setup.	Pass in desired ERLE Bit definitions.

Table 8-43. _ERLE_TEST_MODE Parameter

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
NLEC Mode	Transmit Level	Analog Gain Setting			Reserved	Reserved	Transmitter State

NLEC Mode: 0 – Disable the NLEC
1 – Enable the NLEC

Transmit Level: 0 – Set transmitter to 4-level
1 – Set transmitter to 2-level

Analog Gain Setting: 0 – Set Analog gain to 0 db
1 – Set Analog gain to 3 db
2 – Set Analog gain to 6 db
3 – Set Analog gain to 9 db
4 – Set Analog gain to 12 db
5 – Set Analog gain to 15 db

Transmitter State: 0 – Disables transmitter, used to measure the background noise
1 – Enables transmitter, used to measure ERLE

The ERLE test takes approximately 5 seconds to complete. Query the _STARTUP_STATUS to determine when the ERLE test is complete.

Table 8-44. Opcode 0x85 (_STARTUP_STATUS)

Destination	Description	Parameter
BITPUMP0–2	Status of the ERLE test.	0x00

Table 8-45. _STARTUP_STATUS Return Value

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Normal Operation	Tx 4-level	Reserved	NMR_OK	Activation Timer	Tip/Ring	LOST	LOS

When the ERLE test is complete, the Normal Operation (bit 7) is set to 1, indicating the _ERLE_RESULTS are valid. If the ERLE test times out, the Activation Timer (bit 3) is set to 1, indicating that the ERLE test failed, and the results are invalid.

When the ERLE test has successfully completed, the _ERLE_RESULTS are valid and the analog and digital ERLE can be calculated.

8.2.4.2 ERLE Results

This command queries for the Background and ERLE Test Mode results. Both the Background and ERLE Test execute the same code (with one difference—transmitter OFF vs. transmitter ON, respectively) and thus use the same result fields. However, when analyzing the results, Table 8-46 lists the meaningful values.

Table 8-46. Meaningful Values Returned for Different Tests

Test	Meaningful Values
Background	Only SLM and FELM
ERLE Test	All Results

Table 8-47. Opcode 0x93 (ERLE_RESULTS)

Destination	Description	Parameter
BIT_PUMP0–2	Low Byte SLM result	0x00
	High Byte SLM result	0x01
	Low Byte FELM result	0x02
	High Byte FELM result	0x03
	Low Byte, SLM2, bypassing hybrid input result	0x04
	High Byte, SLM2, bypassing hybrid input result	0x05
	DC Offset Low Byte result	0x06
	DC Offset High Byte result	0x07

Use the following formula to convert the low and high bytes into a 16-bit value:

$$\text{16-bit value} = (\text{high byte} << 8) + (\text{low byte})$$

The ERLE and Analog ERLE measurements are determined by the following formulas:

$$\text{ERLE} = 20 \times \log\left(\frac{\text{SLM}}{\text{FELM}}\right)$$

$$\text{AERLE} = 20 \times \log\left(\frac{\text{SLM2}}{\text{SLM}}\right)$$

8.2.4.3 AAGC Results

This command queries for the AAGC Test results.

Table 8-48. Opcode 0x94 (_AAGC_RESULTS)

Destination	Description	Parameter
BIT_PUMP0–2	SLM @ 0 db Setting Low Byte	0x00
	SLM @ 0 db Setting High Byte	0x01
	SLM @ 3 db Setting Low Byte	0x02
	SLM @ 3 db Setting High Byte	0x03
	SLM @ 6 db Setting Low Byte	0x04
	SLM @ 6 db Setting High Byte	0x05
	SLM @ 9 db Setting Low Byte	0x06
	SLM @ 9 db Setting High Byte	0x07
	SLM @ 12 db Setting Low Byte	0x08
	SLM @ 12 db Setting High Byte	0x09
	SLM @ 15 db Setting Low Byte	0x0A
	SLM @ 15 db Setting High Byte	0x0B

Use the following formula to convert the low and high bytes into a 16-bit value:

$$\text{16-bit value} = (\text{high byte} << 8) + (\text{low byte})$$

The AAGC measurements are normalized to the 0db SLM reading and determined by the following formula:

$$\text{AAGC(n)} = 20 \times \log\left(\frac{\text{SLM(n)}}{\text{SLM(0)}}\right)$$

where *n* is 0 db to 15 db.

8.2.5 DPLL Status Command

8.2.5.1 Read the DPLL State

This command returns the current state of the DPLL state machine.

Table 8-49. Opcode: 0x90 (_CU_READ_DPLL)

Destination	Description	Parameter
_CU_COMMON	Reads the DPLL state machine.	0x00

DPLL State	Return Value
Stable	0
Minimum Gain setting	1
Medium Gain setting	2
Maximum Gain setting	3

8.2.6 Channel Unit Indicator Bit Commands

The following four API commands allow access to the 13 channel unit indicator bits.

8.2.6.1 Write Indicator Low Byte

This command writes the indicator low byte, which is sent to the far-end HDSL unit every 6 ms HDSL frame. The EVM code may overwrite the LOSD, FEBE, RRBE and RCBE bits.

Table 8-50. Opcode: 0x35 (_CU_WRITE_IND_LO)

Destination	Description	Parameter
_CU_CHAN0-2	Returns indicator bits 0–7.	User-defined

8.2.6.2 Write Indicator High Byte

This commands writes the upper 5 indicator bits in the indicator high byte. These bits are sent to the far-end HDSL unit every 6 ms HDSL frame.

Table 8-51. Opcode: 0x36 (_CU_WRITE_IND_HI)

Destination	Description	Parameter
_CU_CHAN0-2	Returns indicator bits 8–13.	User-defined

8.2.6.3 Read Indicator Lo Byte

This command reads the first 8 indicator bits in the channel unit. These bits are received from the far-end every 6 ms HDSL frame.

Table 8-52. Opcode: 0x91 (_CU_READ_IND_LO)

Destination	Description	Parameter
_CU_CHAN0-2	Returns indicator bits 0–7.	0x00

Table 8-53. Low Byte Return Status Bit Definitions

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
rcbe	rrbe	hrp	bpv	ps2	ps1	febe	losd

NOTE: All indicator bits are active low.

8.2.6.4 Read Indicator Hi Byte

This command reads the upper 5 indicator bits in the channel unit. These bits are received from the far-end every 6 ms HDSL frame.

Table 8-54. Opcode: 0x92 (_CU_READ_IND_HI)

Destination	Description	Parameter
_CU_CHAN0-2	Returns indicator bits 8–15.	0x00

Table 8-55. High Byte Return Status Bit Definitions

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Manufacturing code			Reserved		rtr	rta	rega

Manufacturing code: 0 – Loop 1 or loop 3
1 – Loop 2

rega, rta, rtr: Active low

8.2.7 Single Loop Commands

The following six commands apply to single loop applications only. These commands allow the user to set up the system for any HDSL data rate between 208 kbps and 2320 kbps in 64 kbps steps. These commands are used only with the RS8973 and RS8953 products. For information on how to use these commands with the Bt8970, contact local technical support.

NOTE: The following single loop API's have only been verified with the _SP_TOTAL_PCM_TSLOT equal to _SP_TOTAL_HDSL_TSLOT.

8.2.7.1 Set Number of PCM Time Slots Used

This command selects the total number of PCM time slots on the PCM bus. For a PCM rate of 2.048 Mbps the parameter is 32 (0x20). For a PCM rate 1544 kbps (or 1536 kbps if Fbit_present is 0) the parameter is 24 (0x18). The PCM data rate is calculated with the following formula:

$$(\text{Parameter} \times 64 \text{ kbps}) = \text{PCM data rate}$$

Example: parameter of 32; (32 x 64 kbps) = 2048 kbps

Table 8-56. Opcode: 0x40 (_SP_TOTAL_PCM_TSLOT)

Destination	Description	Parameter
_DSL_APPLICATION	Sets the number PCM time slots to calculate the PCM rate.	E1–32 (0x20) T1–24 (0x18)

8.2.7.2 Set Number of HDSL Payload Bytes

This command selects the number of HDSL payload bytes used and the HDSL data rate. To calculate the HDSL data rate use this formula:

$$((\text{Parameter} \times 64) + 16) \text{ kbps} = \text{HDSL data rate}$$

Example: parameter of 0x04; ((4 x 64) + 16) kbps = 272 kbps.

If a parameter of 2 is used, the stuff thresholds must be changed. Contact Conexant for more details.

Table 8-57. Opcode: 0x41 (_SP_TOTAL_HDSL_TSLOT)

Destination	Description	Parameter
_DSL_APPLICATION	Total number of HDSL payload bytes to send over the HDSL.	2–36

8.2.7.3 Set Number of Occupied HDSL Payload Bytes and PCM Time Slots Used

This command selects the number HDSL payload bytes out of the "total number of HDSL payload bytes" that transmit over the HDSL link. This command selects the first *n* number of bytes of the parameter. If the parameter is 5, payload bytes 0, 1, 2, 3, and 4 are used for payload.

This number cannot be greater than the "Total Number of HDSL Payload Bytes" or "Total Number of PCM Time Slots".

Table 8-58. Opcode: 0x42 (_SP_USED_TSLOT)

Destination	Description	Parameter
_DSL_APPLICATION	Total number of payload channels used.	3–36

8.2.7.4 Set F-bit Present

This command sets whether or not the F-bit is present. When PCM rate of 1544 kbps is selected and no F-bit is inserted, the payload bandwidth is 1536 kbps. With the current 8973/8953B EVM application, when the F-bit is present, the framer mode is set as T1; otherwise, the framer mode is E1.

Table 8-59. Opcode: 0x43 (_SP_FBIT_PRESENT)

Destination	Description	Parameter
_DSL_APPLICATION	Sets whether or not the F-bit is present.	0–Not Present 1–Present

8.2.7.5 Set Derived MClk Value

This command sets the derived MClk. To calculate the parameter use the formula:

$$\text{Parameter} = \text{MClk frequency} \div 128\ \text{kHz}$$

On the EVM, XOUT from the RS8973 drives the MCLK signal to the channel unit. Because XOUT equals 10.24 Mhz, the parameter is set to 80 (0x50).

Table 8-60. Opcode: 0x44 (_SP_DERIVED_MCLK)

Destination	Description	Parameter
_DSL_APPLICATION	The parameter is multiplied by 128 kHz to equal MCLK input frequency.	30–145

8.2.7.6 Configure Single Loop

This command takes all previous single loop API command parameters and calculates the required channel unit and bit pump registers.

The system transits as follows:

- Initialize: Sets bit pump to idle.
- Configure: Reconfigures the system for the new data rate.
- Restart: Performs a DSL activation at the new data rate.

This command only supports water level adjustments for PCM 24 and 32.

Table 8-61. Opcode: 0x45 (_SP_CONFIGURE)

Destination	Description	Parameter
_DSL_CHANNEL0–2	Configures channel unit and bit pump and restarts system.	0x00

8.2.8 EOC Commands

The Embedded Operations Channel (EOC) has two software levels:

1. The higher level API modifies the EOC register contents and control when messages are transferred.
2. The lower level drivers move the data back and forth through the channel and perform error checking.

The user does not need access to the lower level drivers to use the EOC channel.

The following ten commands are additional higher level API commands for the channel unit's EOC channel. This channel is used to communicate with the far-end channel unit and host processor. The customer must add any vendor-defined specifics from the HDSL standard. All reserved bits should be set to 0.

8.2.8.1 EOC Register Select

This command selects an EOC register name and the register type (write or read). This API must be selected before modifying any different registers (0–F). This command also resets the byte number location to 0.

The value of the register type is stored and affects the operation of the _EOC_REG_SIZE, _EOC_BYTE_NUM_LOC, and _EOC_WRITE_REG_DATA commands.

Table 8-62. Opcode: 0x35 (_EOC_REG_SELECT)

Destination	Description	Parameter
_DSL_CHANNEL 0–2	Selects the register to modify with the following API commands.	See Parameter Field Definitions

Parameter Field Definitions

Byte 7–5	Bit 4	Bits 3–0
Reserved	Register type	Register Name

Register type: 0 – Write register
1 – Read register

8.2.8.2 EOC Register Size

This command sets the register byte size, from 0–16 bytes. The EOC_REG_SELECT must be set prior to modifying the register size. Because there are read and write registers for each register name, registers with the same name may have two different sizes.

Example: "write register A" can have a register size of 16 while "read register A" has a size of 1. All sizes default to 0. When the size is 0 and a command is performed, the return value from the HTU-R will be Unable To Comply (UTC).

Table 8-63. Opcode: 0x36 (_ EOC_REG_SIZE)

Destination	Description	Parameter
_DSL_CHANNEL0–2	Selects the size of the selected EOC register.	See Parameter Field Definitions

Parameter Field Definitions

Byte 7–5	Bits 4–0
Reserved	Register Name

8.2.8.3 EOC Byte Number Location

This command sets the location within the rdRegSize[] or wrRegSize[] array initially accessed by the _EOC_WRITE_REG_DATA and _EOC_READ_REG_DATA API commands.

NOTE: When writing to or reading from the register, the first byte number location is 0 and the last byte number is the register size minus 1.

Table 8-64. Opcode: 0x37 (_EOC_BYTE_NUM_LOC)

Destination	Description	Parameter
_DSL_CHANNEL0–2	Selects which byte of the register to modify.	0–(register size–1)

8.2.8.4 EOC Write Register Data

This command writes the 8-bit parameter value into the wrRegData[] or rdRegData[] arrays based on the current register type setting. The EOC Write Register Data API command copies this value into the first byte of the shadow buffer bytes. After the API is executed, it increments the byte number location by one. When the next EOC Write Register Data API is executed, the parameter is written to the second byte number location of the shadow buffer. This continues until all bytes are filled, then the shadow buffer is copied into the correct register when the EOC channel is inactive. This shadow buffer prevents corruption of the EOC data registers while sending data.

The HTU-C (the EOC master) should only write to the wrRegData[] (control) with the parameter information to be sent to the HTU-R (the EOC slave). The HTU-R should only write to the rdRegData[] (status) with the parameter information that is returned to the HTU-C. The API command returns a fail if a write operation to the improper register type is attempted.

The rdRegData[] on the HTU-R must be loaded before the HTU-C reads it.

NOTE: For previous compatibility, the HTU-R wrRegData D byte 0 should be updated only after a system reset, once NORMAL OPERATION is met.

Table 8-65. Opcode: 0x38 (_EOC_WRITE_REG_DATA)

Destination	Description	Parameter
_DSL_CHANNEL0–2	Writes an eight-bit value to the selected register.	0–0xFF

8.2.8.5 Start EOC Read/Write Operation

This command starts a read or write process. Only the HTU-C can issue this command. When performing a write command the HTU-C sends its wrRegData[]. If performing a read (status) command, the HTU-C requests HTU-R to send its rdRegData[].

NOTE: Register Read E is in the HDSL standard as a special purpose register. To read noise margin, the user only has to issue this command and read the returned value. (See *ETSI TS 152 Edition 4* under EOC Coding of the noise margin.)

Table 8-66. Opcode 0x39 (_EOC_SEND_RD_WR)

Destination	Description	Parameter
_DSL_CHANNEL0–2	Performs the write/read process based on parameter.	See Parameter Field Definitions

Parameter Field Definitions

Byte 7–5	Bit 4	Bits 3–0
Reserved	Write or read process.	Register Name

Write or read process: 0 – Write process
1 – Read process

8.2.8.6 Set EOC Control Commands

This command requires one of the following parameters in the table. Once this API is executed, the appropriate EOC command is executed. (See *ETSI TS 152 Edition 4* under HDSL EOC requirements for control definitions.)

Table 8-67. Opcode: 0x3A (_EOC_SET_CONTROL)

Destination	Description	Parameter
_DSL_CHANNEL0–2	Controls the HTU-R via the EOC to perform one of the functions.	See Parameter Field Definitions

Parameter Field Definitions

C Constant	Hex Code	Description
EOC_CMD_RTN	0x07	Disables all hold states (loopbacks).
EOC_CMD_LOOP_NTU	0x08	Enables loopback (HDSL to PCM) in channel unit.
EOC_CMD_HOLD	0x10	Sets a hold state.
EOC_CMD_ALOOP_REG	0x19	Sets analog loop back. User-defined.
EOC_CMD_NTU_CCRC_REQ	0x20	RegR sends inverted CRC6 to HTU-C.
EOC_CMD_REGC_CCRC_REQ	0x22	RegC sends inverted CRC6 to HTU-R.
EOC_CMD_NTU_CCRC_END	0x28	RegR stops inverted CRC6 to HTU-C.
EOC_CMD_REGC_CCRC_END	0x29	RegC stops inverted CRC6 to HTU-R.
EOC_CMD_NTU_CCRC_IND	0x3F	Notify RegR CRC6 is being sent.
EOC_CMD_REGC_CCRC_IND	0x50	Notify RegC CRC6 is being sent.
EOC_CMD_NTU_CRC_OK	0x5F	Notify RegR CRC6 is not being sent.
EOC_CMD_REGC_CRC_OK	0x60	Notify RegC CRC6 is not being sent.

8.2.8.7 Set EOC Address Destination

For the HTU-C, this command selects the destination of the EOC command or message. The command or message can be routed to either the repeater or HTU-R.

Table 8-68. Opcode 0x3B (_EOC_ADD_DEST)

Destination	Description	Parameter
_DSL_CHANNEL0–2	Sets where command is sent. Defaults to HTU-R.	0–Send to Repeater 1–Send to HTU-R

8.2.8.8 Insert CRC Errors

This command continuously injects CRC-6 errors by inverting the CRC 6-bit calculation. Errors continue every 6 ms HDSL frame until the command is disabled. This is used to test far-end CRC detection capabilities.

Table 8-69. Opcode: 0x3C (_INSERT_CRC6)

Destination	Description	Parameter
_DSL_CHANNEL0-2	Enables CRC6 errors to be sent to the far-end.	0–disable CRC errors 1–enable CRC errors

8.2.8.9 EOC Query Received New Data

This command is used to determine when either the HTU-C or HTU-R has received an EOC read/write message from the far-end. The flags are then cleared after the status is returned. This command does not return the contents of the EOC rd/wrRegData buffer.

For the HTU-C, these flags should be queried after an EOC Read Register command is sent and before another EOC Read Register command of the same register is sent.

For the HTU-R, this command checks for updated wrRegData from the HTU-C.

When a newDataFlag bit is set, the host processor can read the EOC register contents by using the EOC Read Register command.

The register is cleared after _EOC_RCVD_NEWDATA_STATUS is queried.

Table 8-70. Opcode: 0x86 (_EOC_RCVD_NEWDATA_STATUS)

Destination	Description	Parameter
_DSL_CHANNEL0–2	Return the received new data status then clears the register.	0–Request Lo Byte 1–Request Hi Byte

Return Status Bit Definitions

	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
Lo Byte 0	Reg 7	Reg 6	Reg 5	Reg 4	Reg 3	Reg 2	Reg 1	Reg 0
High Byte 1	Reg F	Reg E	Reg D	Reg C	Reg B	Reg A	Reg 9	Reg 8

The newDataFlags is a 16-bit variable. A parameter of 0x00 returns and clears the lower 8-bits of the variable. The parameter of 0x01 returns and clears the upper 8-bits of the 16-bit variable.

8.2.8.10 EOC Read Register

This command extracts the contents of the EOC buffer. The HTU-C returns the contents of the rdRegData[] buffer; the HTU-R returns the contents of the wrRegData[] buffer.

This command requires that the parameter contain the selected rd/wrRegister. The command increments the byte number location after every read. The system must repeatedly call this command for the same number of times as the register size. If the byte number location is greater than the register size, this command returns a failed response and the data is invalid. In addition, the EOC Byte Number location API command can be used at any time to reset the buffer index pointer.

Table 8-71. Opcode: 0x87 (_EOC_READ_REG_DATA)

Destination	Description	Parameter
_DSL_CHANNEL0–2	Returns 8-bit value in the register selected.	See Parameter Field Definitions

Parameter Field Definitions

Byte 7–5	Bit 4	Bits 3–0
Reserved	Register type	Register Name

Register type: 0 – Write
1 – Read

8.2.8.11 Read EOC Status

This command queries the low level EOC protocol status.

Table 8-72. Opcode: 0x88 (_EOC_STATUS)

Destination	Description	Return Value HTU-C
_DSL_CHANNEL0–2	Returns the EOC status based on parameter.	See HTU-C and HTU-R Return Status Bit Definitions.

For the HTU-C the parameter is 0 and returns the eocCtrl.status. The API allows the user to query the HTU-C status activity.

HTU-C Return Status Bit Definitions

C Constant	Hex Code	Description
EOC_STATUS_AVAILABLE	0x01	EOC channel is transparent
EOC_STATUS_BUSY	0x02	EOC handler is reserved
EOC_STATUS_RUN	0x04	EOC handler performs EOC action
EOC_STATUS_HOLD	0x08	Latched EOC command sent out
EOC_STATUS_ERROR	0x10	EOC action failed

For the HTU-R, the parameter is 1 and returns the eocHoldStates[]. This API allows the user to query which EOC commands are currently latched (in-progress).

HTU-R Return Status Bit Definitions

Opcode Description	Hex Code	Definition
EOC_REQ_LOOP_RT EOC_REQ_CCRC_RT Reserved _NMR EOC_REQ_ALOOP_RT EOC_REQ_CCRC_CO EOC_REQ_NOT_CCRC_RT EOC_REQ_NOT_CCRC_CO	0x01 0x02 0x04 0x08 0x10 0x20 0x40	HP loopback HTU-R inserts corrupted CRC Reserved for NMR Analog loopback Reg-C inserts corrupted CRC HTU-R is notified of CRC HTU-C in notified of CRC

8.3 Application Examples

API commands have three parts:

1. Destination
2. Opcode
3. Parameter

Destinations for the following examples use _DSL_CHANNEL0 (0xF9).

8.3.1 Read Example

This section provides EOC read examples. The steps below show how:

1. Select HTU-C and HTU-R Register D.
2. Select HTU-C and HTU-R Register size for register D.
3. Load the register D value into the HTU-R.
4. Read the HTU-Rs register D value from the HTU-C.

8.3.1.1 Step 1: Select HTU-C Register Name

After the EOC available flag is set and startup is complete the EOC channel is ready. This example sets the HTU-C register D. The ETSI standard specifies that register D contains the device ID value. The vendor must define the Device ID value. The HTU-C selects the register to modify by using the EOC Register Select API.

0xF9–DSL_CHANNEL0.
0x35–EOC Register Select.
0x1D–The 1 is for read; D is for Register D.

8.3.1.2 Step 2: Select HTU-C Register Size

Select the register size. This example sets the HTU-C read register D size to 8 bytes.

0xF9–DSL_CHANNEL0.
0x36–EOC Register Size.
0x08–Set Read register D to 8 bytes.

8.3.1.3 Step 3: Set Up HTU-R Register Name and Size

Repeat steps 1 and 2, except on the HTU-R.

8.3.1.4 Step 4: Load HTU-R Read Register D

The HTU-R must load values into the read registers. The HTU-R registers must be loaded before the HTU-C can read them.

NOTE: The EOC Register Select command is not needed because read register D is the current register.

After the EOC Register Select API, the EOC byte number location is set to 0. The EOC Write Register Data API command copies the 8-bit parameter value into the first byte of the shadow buffer bytes and increments the byte number location by one. When the next EOC Write Register Data API is executed, the parameter is written to the second byte number location of the shadow buffer. This continues until all 8 bytes are filled; then the shadow buffer is copied into the correct register when the EOC channel is inactive. This shadow buffer step was added to make it impossible to corrupt the EOC data registers while sending the EOC data.

The following API writes the data into read register D.

0xF9–DSL_CHANNEL0.

0x38–EOC Write Register Data.

0x55–This is user-defined, in this case 0x55 is the first byte of the Device ID.

This API, with user-defined parameters, is repeated seven times until all eight bytes are filled.

8.3.1.5 Step 5: Set Command for HTU-C to Read HTU-R Register D

The HTU-C issues the Start EOC Read/Write Operation to read HTU-R Register D.

0xF9–DSL_CHANNEL0.

0x39–Start EOC Read/Write Operation.

0x1D–1 is for read and D is for register D.

This command forces the HTU-R to return all 8 bytes of read register D.

8.3.1.6 Step 6: Read New Data Flags

After the read command is sent, the HTU-C waits until the HTU-R returns the information. The HTU-C can read the _EOC_RCVD_NEWDATA_STATUS API. This command's result indicates which EOC register has valid receive data from the EOC channel. When the _EOC_RCVD_NEWDATA_STATUS result for register D is 1, the data is valid and the flag is cleared.

0xF9–DSL_CHANNEL0.

0x86–Read the Received Data Status.

0x01–Read the higher 8-bits of the 16-bit _EOC_RCVD_NEWDATA_STATUS register.

8.3.1.7 Step 7: Set Index to 0 for Read Register D

For the HTU-C to query the value from the first byte, use the following API:

0xF9–DSL_CHANNEL0.

0x37–Byte Number Location.

0x00–Byte 0 will be read next time.

8.3.1.8 Step 8: Read Register D

The EOC Read Register command allows the HTU-C to read the returned value.

0xF9–DSL_CHANNEL0.

0x87–Read EOC Register.

0x1D–The 1 is for read; the D is for register D.

When the API is executed the return value is 0x55. Re-executing the API increments the byte number location and returns the next read register D value.

8.3.2 Write Example

This section provides a write EOC example. The following steps show how to:

- Set-up the write register size for register B.
- Load the register value.
- Send the register value.
- Inform the HTU-R about the new data.
- Read the register value at the HTU-R.

8.3.2.1 Step 1: Set Up HTU-C Register Number

After the EOC available flag is set and startup is complete, the EOC channel is ready. The ETSI standard specifies that register B contains the Network Termination Unit (NTU) configuration. The HTU-C selects the register to modify by using the EOC Register Select API.

0xF9–DSL_CHANNEL0.

0x35–EOC register select.

0x0B–The 0 is for write register, the B is for register B.

8.3.2.2 Step 2: Set Up HTU-C Register Size

Select the register size. This example sets the HTU-C write register B size to 10 bytes.

0xF9–DSL_CHANNEL0.

0x36–EOC register size.

0x0A–Set Write register B to 10 bytes.

8.3.2.3 Step 3: Set Up HTU-R Register Number and Size

Repeat steps 1 and 2, except on the HTU-R.

8.3.2.4 Step 4: Load the HTU-C Write Register B

The HTU-C must load the values to send to the HTU-R's Write Register B.

NOTE: The EOC Register Select command does not need to be set again because Write Register B is the current register.

After the EOC Register Select API, the EOC Byte Number Location is set to 0. The EOC Write Register Data API writes the 8-bit parameter into the first byte of the shadow buffer bytes and increments the byte number location by one. When the next EOC Write Register Data API is executed, the parameter is written to the second byte number location of the shadow buffer. This continues until all 10 bytes are filled; then the shadow buffer is copied into the correct register when the EOC channel is inactive. This shadow buffer step was added to make it impossible to corrupt the EOC data registers while sending the EOC data.

The following API writes the data into Read Register B:

0xF9–DSL_CHANNEL0.

0x38–EOC Write Register Data.

0xAA–This is user-defined; in this case, 0xAA is the first byte of the NTU-Configuration.

This API is repeated nine times until all bytes are filled. After the last byte is written and the EOC channel is available, the shadow buffer is written to Write Register B.

8.3.2.5 Step 5: Set HTU-C Start Sending Command to Write HTU-R Register B

For the HTU-C to write the Write Register B value it issues the Start EOC Read/Write Operation.

0xF9–DSL_CHANNEL0.

0x39–Start EOC Read/Write Operation.

0x0B–The 0 is for write the B is for register B.

This command sends 10 NTU-Configuration bytes to the HTU-R.

8.3.2.6 Step 6: Read the Received Data Status

This command allows the HTU-R to query if any write registers have been updated. If it returns a zero, no registers have been updated via the EOC channel. If it returns a number, that number will correspond to the register number with new data.

0xF9–DSL_CHANNEL0.

0x86–New Data Flag.

0x01–Reads New data flags for registers 8–F.

The return value is 0x08, which corresponds to register B. If register 9 was updated, the value would be 0x02, A would be a 0x04 and so on. Once the API command has read the New Data Flags they are cleared.

8.3.2.7 Step 7: Set Byte Number Location

Set the api_dataBuffIndex[] to 0.

0xF9–DSL_CHANNEL0.

0x37–EOC Byte Number Location.

0x0B–The 0 is for write and the B is Register B.

The byte location is pointing to the first byte.

8.3.2.8 Step 8: Read the B Data Register

To read the byte at the byte location issue the EOC Read Register API command.

0xF9–DSL_CHANNEL0.

0x87–EOC Read Data Register.

0x0B–0 is for write B is Register B.

The return value is 0xAA.

8.3.3 HTU-C CRC Check Command Example

Example steps 1 and 2 explain how to test the HTU-C CRC detector.

8.3.3.1 Step 1: HTU-C Receives Corrupted CRC from HTU-R.

To verify the HTU-C CRC detector unit is functioning, the HTU-C requests that the HTU-R send corrupted CRC-6 errors.

0xF9–DSL_CHANNEL0.

0x3A–Set EOC Control Commands.

0x20–Request corrupted CRC-6 from the HTU-R.

This command sends corrupted CRC-6 every multi-frame until the End of Corrupted CRC-6 command (0x28) is sent. (See the *ETSI TS 152 Edition 4* under HDSL EOC opcode messages for more information.)

The user monitors the HTU-Cs CRC-6 errors counter to verify CRC detector functionality.

8.3.3.2 Step 2: Set the End Corrupted CRC Command

To disable the HTU-R sending corrupted CRC-6 issue the following values.

0xF9–DSL_CHANNEL0.

0x3A–Set EOC Control Commands.

0x28–Request End of Corrupted CRC-6 from the HTU-R.

Stops the HTU-R from sending corrupted CRC-6.

8.3.4 HTU-R CRC Check Command Example

The example steps 1 and 2 explain how to test the HTU-R CRC detector.

8.3.4.1 Step 1: Notify the HTU-R of Corrupted CRC

The HTU-C can notify the HTU-R that it will be sending CRC-6 errors. This will be used in the HTU-R to disable alarm indication circuitry activated by the detection of corrupted CRC-6.

0xF9–DSL_CHANNEL0.

0x3A–Set EOC Control Commands.

0x3F–Notify HTU-R the HTU-C will send corrupted CRC-6, disable any alarms.

8.3.4.2 Step 2: Send Corrupted CRC

The HTU-C can insert corrupted CRC-6 to every HDSL 6 ms frame with the following API command.

0xF9–DSL_CHANNEL0.
0x3C–Insert CRC Errors.
0x01–Enable CRC-6 injection.

To disable the insert CRC-6:
0xF9–DSL_CHANNEL0.
0x3C–Insert CRC Errors.
0x00–Disable CRC-6 injection.

8.4 Level 1 API Commands

Use these API commands to access the bit pump or channel unit directly. They are typically not used in the ZipSocket system.

8.4.1 Bit Pump APIs

8.4.1.1 Input Signal Level

Table 8-73. Opcode: 0x80 (_SLM)

Destination	Description	Parameter
_BIT_PUMP0–2	Requests the level of average signal level at the ADC input.	0x00

The return value is an integer (0–255) that is relative to the average absolute value of the ADC input signal. The measurement scale is such that a value of 255 corresponds to the ADC positive full scale value.

8.4.1.2 Input DC Offset

Table 8-74. Opcode: 0x81 (_DC_METER)

Destination	Description	Parameter
_BIT_PUMP0–2	Requests the value of the average DC level at the ADC input.	0x00

The return value is a signed integer (–128 through 127) that is relative to the average DC offset per ADC sample. The measurement scale is such that the actual DC offset in units of ADC LSB is 32x. If the DC offset is outside the representable range (–4096 through 4095), the nearest representable value is used.

8.4.1.3 Bit Pump BER Meter

Table 8-75. Opcode: 0x15 (_BER_METER_START)

Destination	Description	Parameter
_BIT_PUMP0–2	Start BER meter.	0x00

Activates the BER meter. The bit pump is set to transmit an internal 4-level scrambled 1s pattern. The enabled bit is set and the bit_errors and meter_intervals variables are reset to 0. This command should only be called during the bit pump's normal operation.

Table 8-76. Opcode: 0x16 (_BER_METER_STOP)

Destination	Description	Parameter
_BIT_PUMP0–2	Stop BER meter.	0x00

Deactivates the BER meter. The bit pump is set to transmit external 4-level data. The enabled bit is set to OFF. The bit_errors and meter_intervals variables are unmodified so they can still be read.

Table 8-77. Opcode: 0x92 (_BER_METER_STATUS)

Destination	Description	Parameter
_BIT_PUMP0–2	Reads the BER meter status.	0x00
	Reads the low byte of the number of bit errors.	0x01
	Reads the high byte of the number of bit errors.	0x02
	Reads the low byte of the number of meter intervals elapsed.	0x03
	Reads the high byte of the number of meter intervals elapsed.	0x04

8.4.1.4 Self-test

Table 8-78. Opcode: 0x8C (_SELF_TEST)

Destination	Description	Parameter
_BIT_PUMP0–2	Execute a bit pump self test. Verifies read/write operations.	0x00 0x01

The return value 0x00 is a bit pump self-test pass; 0x01 is a bit pump self-test fail.

8.5 Channel Unit API Commands

These API commands can be used to access the Channel Unit directly.

8.5.1 Set the PCM Multiframe Length

Table 8-79. Opcode: 0x2A (_CU_SET_MFRAME)

Destination	Description	Parameter
_CU_COMMON	Changes the multi-frame sync to the desired length between 1–48 frames.	0x00–0x2F

The parameter value should equal the desired multi-frame length minus 1. For example, if one frame per multi-frame is desired, set the parameter to 0. The following settings are used for various Conexant devices.

- Bt8370 (T1/E1) 48 frames/multi-frame
- Bt8360 (T1) 24 frames/multi-frame
- Bt8510 (E1) 16 frames/multi-frame

8.5.2 Channel Unit Error Counters

API commands _CU_ERROR_COUNTERS_LO and _CU_ERROR_COUNTERS_HI together can be used to query a specific error counter in the Channel Unit.

Table 8-80. Opcode: 0x86 (_CU_ERROR_COUNTERS_LO)

Destination	Description	Parameter
_CU_CHAN1–3	Return lower byte of the specified error counter value.	0–11
NOTE(S): Refer to Table 8-82 for a description of each error counter.		

Table 8-81. Opcode: 0x87 (_CU_ERROR_COUNTERS_HI)

Destination	Description	Parameter
_CU_CHAN1–3	Return higher byte of the specified error counter value.	0–11
NOTE(S): See Table 8-82 for the purpose of each error counter.		

Table 8-82. Opcode: 0x0A (_CU_CLEAR_ERROR_COUNTERS)

Destination	Description	Parameter
_CU_COMMON	Clear all the errors counters.	0xFF
_CU_CHAN1–3	Clear _CU_OUT_OF_SYNC_CTR error counter.	0
	Clear _CU_CRC_ERR_CTR error counter.	1
	Clear _CU_RFIFO_FULL_CTR error counter.	2
	Clear _CU_RFIFO_EMPTY_CTR error counter.	3
	Clear _CU_RFIFO_SLIP_CTR error counter.	4
	Clear _CU_TFIFO_FULL_CTR error counter.	5
	Clear _CU_TFIFO_EMPTY_CTR error counter.	6
	Clear _CU_TFIFO_SLIP_CTR error counter.	7
	Clear _CU_TFIFO_STUFF_CTR error counter.	8
	Clear _CU_DPLL_ERROR_CTR error counter.	9
	Clear _CU_FEBE_ERROR_CTR error counter.	10
	Clear _CU_LOSD_ERROR_CTR error counter.	11

If the destination of the API is _CU_COMMON, the specified error counter for all loops is cleared; if the destination is a specific loop, this command clears the specified error counter for that loop only.

8.5.3 Modify Receive Combination Table

Refer to Section 10.10 for a detailed description of receive combination table.

Table 8-83. Opcode: 0x13 (_CU_COMBINE_ADDR)

Destination	Description	Parameter
_CU_COMMON	Change the combine table pointer (combine_address) to the value specified by the parameter. This is the starting address for the modification.	Desired combine table address (0–63)

Table 8-84. Opcode: 0x11 (_CU_COMBINE_VALUE)

Destination	Description	Parameter
_CU_COMMON	Change the combine table value at combine_address. Combine_address increments at each call.	Desired combine table value.

Table 8-85. Opcode: 0x12 (_CU_COMBINE_WRITE)

Destination	Description	Parameter
_CU_COMMON	Update the modified combine table.	0x00

Table 8-86. Opcode: 0x8B (_CU_READ_COMBINE)

Destination	Description	Parameter
_CU_COMMON	Read the combine table value at the address specified by the parameter.	Combine table address.

Use _CU_READ_COMBINE to check the current combine table value. To modify the combine table for fractional T1/E1 application, follow these steps:

1. Use _CU_COMBINE_ADDR to set the starting address in the combine table that needs modification.
2. Use _CU_COMBINE_VALUE to change the combine table values that needs modification.
3. Use _CU_COMBINE_UPDATE to update the modified combine table.

8.5.4 Modify Transmit Routing Table

See Section 10.9 for detailed description of transmit routing table.

Table 8-87. Opcode: 0x16 (_CU_ROUTE_ADDR)

Destination	Description	Parameter
_CU_COMMON	Change the route table pointer (route_address) to the value specified by the parameter.	Desired route table entry address (0–63)

Table 8-88. Opcode: 0x14 (_CU_ROUTE_VALUE)

Destination	Description	Parameter
_CU_COMMON	Change the route table value at entry combine_address to the parameter.	Desired route table value.

Table 8-89. Opcode: 0x15 (_CU_ROUTE_WRITE)

Destination	Description	Parameter
_CU_COMMON	Update the modified route table.	0x00

Table 8-90. Opcode: 0x8C (_CU_READ_ROUTE)

Destination	Description	Parameter
_CU_COMMON	Read the route table value specified by the parameter.	Route table address.

Use _CU_READ_ROUTE to check the current route table value. To modify the routing table for fractional T1/E1 application, follow these steps:

1. Use _CU_ROUTE_ADDR to set the starting address in the route table that needs modification.
2. Use _CU_ROUTE_VALUE to change the route table values that needs modification.
3. Use _CU_ROUTE_UPDATE to update the modified route table.

8.5.5 Modify Transmit Payload Mapper (TMAPS)

See Section 10.11 for a detailed description of transmit payload mapper.

Table 8-91. Opcode: 0x1B (_CU_TMAP1_VALUE)

Destination	Description	Parameter
_CU_CHAN1–3	Changes loop specific TMAP1 register value.	Desired TMAP1 value.

Table 8-92. Opcode: 0x1C (_CU_TMAP2_VALUE)

Destination	Description	Parameter
_CU_CHAN1–3	Changes loop specific TMAP2 register value.	Desired TMAP2 value.

Table 8-93. Opcode: 0x1D (_CU_TMAP3_VALUE)

Destination	Description	Parameter
_CU_CHAN1–3	Changes loop specific TMAP3 register value.	Desired TMAP3 value.

Table 8-94. Opcode: 0x1E (_CU_TMAP4_VALUE)

Destination	Description	Parameter
_CU_CHAN1–3	Changes loop specific TMAP4 register value.	Desired TMAP4 value.

Table 8-95. Opcode: 0x1F (_CU_TMAP5_VALUE)

Destination	Description	Parameter
_CU_CHAN1–3	Changes loop specific TMAP5 register value.	Desired TMAP5 value.

Table 8-96. Opcode: 0x20 (_CU_WRITE_TMAP)

Destination	Description	Parameter
_CU_COMMON	Updates the modified transmit mapping table.	0x00

Table 8-97. Opcode: 0x8D (_CU_READ_TMAP)

Destination	Description	Parameter
_CU_CHAN1–3	Reads specified TMAP register value.	TMAP register number.

Use _CU_READ_TMAP to check the current transmit payload map value. To modify any of the five TMAP registers for fractional T1/E1 application, follow these steps:

1. Use _CU_TMAPx_VALUE to change the corresponding TMAP register value.
2. Use _CU_WRITE_TMAP to update the Transmit Payload Mapper.

8.5.6 Modify Receive Payload Mapper (RMAPS)

See Section 10.12 for detailed description of receive payload mapper.

Table 8-98. Opcode: 0x21 (_CU_RMAP1_VALUE)

Destination	Description	Parameter
_CU_CHAN1–3	Changes loop specific RMAP1 register value.	Desired RMAP1 value.

Table 8-99. Opcode: 0x22 (_CU_RMAP2_VALUE)

Destination	Description	Parameter
_CU_CHAN1–3	Changes loop specific RMAP2 register value.	Desired RMAP2 value.

Table 8-100. Opcode: 0x23 (_CU_RMAP3_VALUE)

Destination	Description	Parameter
_CU_CHAN1–3	Changes loop specific RMAP3 register value.	Desired RMAP3 value.

Table 8-101. Opcode: 0x24 (_CU_WRITE_RMAP)

Destination	Description	Parameter
_CU_COMMON	Updates the modified receive mapping table.	0x00

Table 8-102. Opcode: 0x8E (_CU_READ_RMAP)

Destination	Description	Parameter
_CU_CHAN1–3	Reads specified RMAP register values.	RMAP register number.

Use _CU_READ_RMAP to check the current receive payload map value. To modify any of the three RMAP registers for fractional T1/E1 application, follow these steps:

1. Use _CU_RMAPx_VALUE to change the corresponding RMAP register value.
2. Use _CU_WRITE_RMAP to update the receive payload mapper.

8.5.7 Modify Data Bank Patterns (DBANKs)

Table 8-103. Opcode: 0x17 (_CU_DBANK_1)

Destination	Description	Parameter
_CU_COMMON	Change the DBANK1 register to the desired pattern specified by the parameter.	Desired pattern for DBANK1

Table 8-104. Opcode: 0x18 (_CU_DBANK_2)

Destination	Description	Parameter
_CU_COMMON	Change the DBANK2 register to the desired pattern specified by the parameter.	Desired pattern for DBANK2

Table 8-105. Opcode: 0x19 (_CU_DBANK_3)

Destination	Description	Parameter
_CU_COMMON	Change the DBANK3 register to the desired pattern specified by the parameter.	Desired pattern for DBANK3

1. DBANK 1–3, each holds an 8-bit programmable pattern that can be used to replace transmit HDSL payload bytes or receive PCM time slots according to the transmit payload map (Section 10–11) and receive combination table (Section 10–10) selections.
2. Multiple DBANK registers may be needed to fill transmit HDSL payload bytes reserved by ETSI standards for future applications.

8.5.8 Set Channel Unit Frame Format

Table 8-106. Opcode: 0x09 (_CU_FRAME_FORMAT)

Destination	Description	Parameter
_CU_COMMON	Framed: MSYNC accepts TMSYNC as transmit sync reference.	0x00
_CU_COMMON	Unframed: MSYNC ignores TMSYNC.	0x01

8.5.9 Reset Transmit/Receive FIFOs

If the destination of the API is _CU_COMMON, transmit FIFOs for all loops will be reset; if the destination is a specific loop, this command resets the transmit FIFO for that loop only.

Table 8-107. Opcode: 0x0B (_CU_RESET_TX_FIFO)

Destination	Description	Parameter
_CU_COMMON _CU_CHAN1–3	Reset the transmit FIFO.	0x00

If the destination of the API is _CU_COMMON, receive FIFO for all loops will be reset; if the destination is a specific loop, this command resets the receive FIFO for that loop only.

Table 8-108. Opcode: 0x04 (_CU_RESET_RX_FIFO)

Destination	Description	Parameter
_CU_COMMON _CU_CHAN1–3	Reset the receive FIFO.	0x00

8.5.10 Set Transmit/Receive FIFO Water Levels

TFIFO water level specifies the phase offset between the PCM and HDSL 6 ms frames. It is programmed as the number of TCLK cycle delays from the start of PCM 6 ms sync to the start of HDSL 6 ms frame. This phase offset determines the amount of PCM data written to the TFIFO before the HDSL transmitter begins extracting data from the TFIFO, which also defines each transmitter's data throughput delay and subsequently the differential delay with respect to other HDSL channels.

RFIFO water level determines the PCM and HDSL receiver's phase error tolerance and receive throughput data delay by establishing a fixed phase offset between the master HDSL channel's receive 6 ms frame and the PCM 6 ms sync. It selects the number of RCLK delays from HDSL to PCM 6 ms frames and controls the amount of time available for the HDSL receiver to map data into the RFIFO before the PCM receiver begins extracting data from RFIFO.

Table 8-109. Opcode: 0x0E (_CU_TFIFO_WL)

Destination	Description	Parameter
_CU_CHAN1–3	Set transmit FIFO water level.	Desired water level.

Table 8-110. Opcode: 0x0F (_CU_RFIFO_WL_LO)

Destination	Description	Parameter
_CU_COMMON	Sets receive FIFO water level.	Desired water level (low).

Table 8-111. Opcode: 0x10 (_CU_RFIFO_WL_HI)

Destination	Description	Parameter
_CU_COMMON	Sets receive FIFO water level.	Desired water level (high).

8.5.11 Set Master Loop

Table 8-112. Opcode: 0x27 (_CU_SET_MASTER_LOOP)

Destination	Description	Parameter
_CU_COMMON	Sets the specified loop to be master.	Desired master loop.

8.5.12 Channel Unit SYNC Status

Table 8-113. Opcode: 0x81 (_CU_SYNC)

Destination	Description	Parameter
_CU_COMMON _CU_CHAN1–3	Channel Unit synchronization status.	0x00

If the destination is _CU_COMMON, the return value:

1 – all the loops are IN_SYNC

0 – not all the loops are IN_SYNC

If the destination is a specific loop, the return value:

0 – CU_OUT_OF_SYNC

1 – CU_ACQUIRING_SYNC

2 – CU_IN_SYNC

3 – CU_LOSING_SYNC

8.5.13 Channel Unit BER Meter

Table 8-114. Opcode: 0x25 (_CU_BER_START)

Destination	Description	Parameter
_CU_COMMON	Start Channel Unit BER meter.	0x00

Table 8-115. Opcode: 0x26 (_CU_BER_CONFIGURE)

Destination	Description	Parameter
_CU_COMMON	Configure Channel Unit BER meter.	BER parameter (see Table 8-116)

Table 8-116. BER Parameter

Bit 7	Bit 5–6	Bit 3–4	Bit 2	Bit 0–1
Reserved	PRBS mode (test pattern)	BER scale (test interval)	Reserved	BER select

BER select:
00 – Normal
10 – Reserved
01 – PCM Framed
11 – PCM Serial

BER scale:
00 – 2^{31} bits
01 – 2^{28} bits
10 – 2^{25} bits
11 – 2^{21} bits

PRBS mode:
00 – 2^{23}
01 – 2^{20}
10 – 2^{15}
11 – 2^{4}

NOTE: Refer to Command Register 3 (CMD_3) and Command Register 6 (CMD_6) in the [N8953BDSA] for additional information.

References:

1. [N8953BDSA] *RS8953B/RS8953SPB HDSL Channel Unit Data Sheet*, August, 1998.
2. [G.991.1] International Telecommunication Union. *High Speed Digital Subscriber Line (HDSL) Transceivers*. October, 1998.

Table 8-117. Opcode: 0x8A (_CU_MEASURE_BER)

Destination	Description	Parameter
_CU_COMMON	Check Channel Unit BER meter.	0x00

Table 8-118. Opcode: 0x89 (_CU_BER_STATUS)

Destination	Description	Parameter
_CU_COMMON	Check Channel Unit BER meter status.	0x00

1. Refer to the Bt8953 data sheet for detailed information on command register 6 BER_SEL options.
2. These BER meter-related APIs are not supported in Channel Unit 6.0.

9.0 Structures

9.1 CU_WR

CU_WR is a global structure that holds several register structures. It maps all the write registers in the channel unit. All related structures are defined in CU.H, and definitions register can be found in the channel unit data sheet.

```
typedef struct
{
    TX_WR tx_wr_loop1; /* 0x00 - 0x1F */
    TX_WR tx_wr_loop2; /* 0x20 - 0x3F */
    TX_WR tx_wr_loop3; /* 0x40 - 0x5F */
    RX_WR rx_wr_loop1; /* 0x60 - 0x6B */
    BP_U_8BIT reserved1[4]; /* 0x6C - 0x6F */
    PRA_TX_WR pra_tx_wr; /* 0x70 - 0x74 */
    BP_U_8BIT reserved2[11]; /* 0x75 - 0x7F */
    RX_WR rx_wr_loop2; /* 0x80 - 0x8B */
    BP_U_8BIT reserved3[20]; /* 0x8C - 0x9F */
    RX_WR rx_wr_loop3; /* 0xA0 - 0xAB */
    BP_U_8BIT reserved4[4]; /* 0xAC - 0xAF */
    PRA_RX_WR pra_rx_wr; /* 0xB0 - 0xB4 */
    BP_U_8BIT reserved5[11]; /* 0xB5 - 0xBF */
    COMMON_WR common_wr_regs; /* 0xC0 - 0xFF */
} CU_WR;
```

9.2 CU_RD

CU_RD is a global structure that holds several register structures. It maps all the read registers in the channel unit. All related structures are defined in CU.H, and definitions register can be found in the channel unit data sheet [N8953BDSB].

```
typedef struct
{
    RX_RD rx_rd_loop1;
    TX_RD tx_rd_loop1;
    RX_RD rx_rd_loop2;
    TX_RD tx_rd_loop2;
    RX_RD rx_rd_loop3;
    TX_RD tx_rd_loop3;
    COMMON_RD common_rd_regs;
    PRA_TX_RD pra_tx_rd;
    BP_U_8BIT reserved1[56];
    PRA_RX_RD pra_rx_rd;
} CU_RD;
```

9.3 CU_FLAGS

CU_FLAGS structure holds six important channel unit status flags.

```
typedef struct
{
    BP_BIT_FIELD _CuMasterLoop:2;
    BP_BIT_FIELD _CuLoopsReversed:1;
    BP_BIT_FIELD _CuFrameFormat:1;
    BP_BIT_FIELD _CuUpdateLeds:1;
    BP_BIT_FIELD _CuAutoRestart:1;
    BP_BIT_FIELD :2;
} CU_FLAGS;
```

_CuMasterLoop	Current master loop 00—Loop 0 is the master loop 01—Loop 1 is the master loop 10—Loop 2 is the master loop 11—Reserved for future use
_CuLoopsReversed	Current loop connection status 0—No loop reversal 1—Loops reversed
_CuFrameFormat	Current channel unit frame format 0—Framed data 1—Unframed data
_CuUpdateLeds	Enable/Disable the channel unit LED update 0—Disable the channel unit LED update 1—Enable the channel unit LED update
_CuAutoRestart	Activation State Machine (ASM) status 0—ASM is disabled 1—ASM is enabled

9.4 CU_REG_COPY

CU_REG_COPY structure keeps copies of values written to a subset of all the write registers that are not Write/Readable. The structures within this structure are defined in CU.H. To keep track of a write register value, update the corresponding register in CU_REG_COPY before writing the value to the chip. This structure can be expanded to keep track of more write registers when necessary.

```
typedef struct
{
    COMMON_CMD_1 cmd_1;
    COMMON_CMD_2 cmd_2;
    COMMON_CMD_3 cmd_3;
    COMMON_CMD_5 cmd_5;
    COMMON_CMD_6 cmd_6;
    COMMON_CMD_7 cmd_7;
    TCMD_1 tcmd_1[_NO_OF_LOOPS];
    TCMD_2 tcmd_2[_NO_OF_LOOPS];
    RCMD_1 rcmd_1[_NO_OF_LOOPS];
    RCMD_2 rcmd_2[_NO_OF_LOOPS];
    IRR imr;
    DPLL_GAIN dpll_gain;
    BP_U_8BIT dpll_factor;
    CU_IND_LO tind_lo[_NO_OF_LOOPS];
    CU_IND_HI tind_hi[_NO_OF_LOOPS];
    CU_ZBIT_PID tzbit_pid[_NO_OF_LOOPS];
    BP_U_8BIT zbits_common[5];
} CU_REG_COPY;
```

9.5 IRR

IRR structure holds the bits indicating any of the eight interrupt events. When a bit is set to 1, the corresponding interrupt happens.

```
typedef union
{
    BP_U_8BIT reg;
    struct
    {
        BP_BIT_FIELD tx1:1;
        BP_BIT_FIELD tx2:1;
        BP_BIT_FIELD tx3:1;
        BP_BIT_FIELD rx1:1;
        BP_BIT_FIELD rx2:1;
        BP_BIT_FIELD rx3:1;
        BP_BIT_FIELD tx_err:1;
        BP_BIT_FIELD rx_err:1;
    } bits;
} IRR;
```

tx1–tx3 Transmit HDSL 6 ms Frame Interrupt—reported coincident with the start of the transmit 6 ms frame for the respective HDSL channel.

rx1–rx3 Receive HDSL 6 ms Frame Interrupt—reported coincident with the start of the receive 6 ms frame for the respective HDSL channel.

tx_err Transmit Error Interrupt—the transmit stuffing and TFIFO errors from all enabled error sources are logically OR'd to form TX_ERR.

rx_err Receive Error Interrupt—Framer state transitions, RFIFO errors, CRC and FEBE counter overflows, and DPLL errors from all enabled error sources are logically ORed to form RX_ERR.

10.0 Global Variables

10.1 *cu_wr

Type: CU_WR
File: CU_INIT.C

*cu_wr is a pointer to the write register structure for the channel unit. See Section 9-1 for the definition of structure CU_WR.

10.2 *cu_rd

Type: CU_RD
File: CU_INIT.C

*cu_rd is a pointer to the read register structure for the channel unit. See Section 9-2 for the definition of structure CU_RD.

10.3 num_bit_pumps

Type: BP_U_8BIT
File: CU_UTILS.C

The num_bit_pumps variable contains the number of Line Interface Cards (LIC) found on the EVM motherboard. The value is from 0 to 3. The application code determines the num_bit_pumps value by issuing the _BITPUMP_PRESENT API command to all motherboard slots and counting the number of successful responses.

10.4 bp_position[]

Type: BP_U_8BIT
File: CU_UTILS.C

The bp_position[NUM_LOOPS] array contains the physical position of the LICs as they are located on the motherboard. The application code determines the bp_position[] values by issuing the _BITPUMP_PRESENT API command to all motherboard slots. Table 10-1 lists the possible location values of the LIC.

Table 10-1. Possible Location Values of the Line Interface Cards

Motherboard Slot No.	Value
1	0
2	1
3	2

bp_position[0] contains the location of the first LIC found.
bp_position[1] contains the location of the second LIC found.
bp_position[2] contains the location of the third LIC found.

If a bp_position[] index is unused because there was not a sufficient number of LICs for the current application to fill all EVM motherboard slots, the unused bp_position[] values are set to 0xFF.

Examples:

In a 2E1 system with the LICs located in slots #1 and #3, the bp_position[] array will look as follows:

bp_position[0] = 0
bp_position[1] = 2
bp_position[2] = 0xFF

In a one pair system with the LIC located in slot #2, the bp_position[] array will look as follows:

bp_position[0] = 1
bp_position[1] = 0xFF
bp_position[2] = 0xFF

10.5 rate_values[][]

Type: BP_U_8BIT
File: CU_INIT.C

The first index of the rate_values[][] arrays is specified by the rate_index variable.

The rate_values[][] array contains register values for the standard rate configurations, i.e., 2T1, 2E1, 3E1, and 1E1 (see Section 10.6).

The second dimension of the rate_values[][] determines which channel unit register is being indexed. See CU_TABLE.H for the list of registers.

The channel unit initialization functions index this array to determine the desired register value for the current configuration.

Those registers requiring different programming values among the configurations are included in this matrix. The other registers, programmed identically between the various configurations, are hard-coded.

NOTE: In the EVM system, the rate_values[][] array is maintained in external RAM and consumes a large amount of memory. For many systems with limited external RAM, this array could cause memory problems. Developers should hard-code the register values for their specific application configuration into the appropriate initialization function. The rate_values[][] and rate_index variables should then be deleted.

10.6 rate_index

Type: BP_U_8BIT
File: CU_INIT.C

The rate_index variable contains the channel unit configuration index. The rate_index indexes the rate_values[][] array when programming the channel unit registers. The channel unit configuration index is defined in Table 10-2.

Table 10-2. Channel Unit Configuration Index

Configuration	Value	Macro Definition
2T1	0	_2T1
2E1	1	_2E1
3E1	2	_3E1
Custom	3	_CU_CUSTOM
1T1	4	_1T1
1E1	5	_1E1

The channel unit _CU_CONFIGURE API command sets the rate_index variable.

10.7 htu_values[][]

Type: BP_U_8BIT
File: CU_INIT.C

The htu_values[][] array contains register values for HTU types, i.e., HTU-C and HTU-R (see Section 10.8).

The second dimension of htu_values[][] determines which channel unit register is being indexed. See CU_TABLE.H for the list of registers.

The channel unit initialization functions index this array to determine the desired register value for the current HTU type.

Only registers that require different programming values between the various HTU types are included in this matrix. The registers that are programmed the same between the various configurations are hard-coded.

10.8 htu_index

Type: BP_U_8BIT
File: CU_INIT.C

The htu_index variable contains the channel unit HTU type index. The htu_index indexes the htu_values[][] array when programming the channel unit registers. Table 10-3 lists the channel unit terminal index.

Table 10-3. Channel Unit htu_index Values

HTU Type	Value	Macro Definition
HTU-C	0	_HTUC
HTU-R	1	_HTUR

The channel unit _CU_TERMINAL_TYPE API command sets the htu_index variable.

10.9 route_table[64]

Type: BP_U_8BIT
File: CU_MAP.C

The Route table array has up to 64 entries, each containing the routing information for the corresponding time slot. Standard E1 requires 32 table writes, corresponding to 32 time slots. Standard T1 requires 25 table writes, where the F-bit location is treated as the 25th time slot. Table 10-4 lists each entry in the transmit routing table.

Table 10-4. Route Table Entry Definition

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
—	INSERT_EN	ROUTE[1:0] CH3		ROUTE[1:0] CH2		ROUTE[1:0] CH1	

ROUTE[1:0] Routing Code—Three identical routing codes are present in each table entry to select which data source is routed to each of three HDSL channel destinations (CH1–CH3). Route data is available from three sources: PCM Transmit Serial Data (TSER), PCM Insert Serial Data (INSDAT), and PRBS generator data. In addition, TSER data is available from an 8-bit delay buffer to allow routing codes to use the same TSER byte twice as a data source. PCM time slot data can also be discarded by selecting no destination channels. INSDAT is available only from the 8-bit delay buffer, and cannot be repeated in the same manner as TSER. INSDAT occupies delay buffer space and prevents routing of previous TSER data during the time slot following INSERT_EN. For example, if INSERT_EN is active in the time slot 1 table entry, during time slot 2 the delay buffer contains INSDAT, not the previous TSER. The PRBS generator is active only during time slots that select PRBS data, which allows discontinuous time slots to be tested with a single continuous PRBS test pattern. Sequential time slot routing is performed from inputs to destination channels without time slots reordering.

ROUTE[1:0]	Source of Transmit HDSL Channel Data
00	Discard, do not route time slot data
01	TSER
10	PRBS (or FILL_PATT, if PRBS_DIS = 1)
11	Previous TSER (or INSDAT) from delay buffer

INSERT_EN	Enable INSERT—Controls the state of the internal mux and the INSERT output pin during the corresponding PCM time slot's sample time. The next table entry is programmed to select the previous time slot (ROUTE = 11) to place INSDAT data from the previous time slot into the TFIFO.
0	INSERT output pin remains inactive (low)
1	INSERT output pin active (high)

10.10 combine_table[64]

Type: BP_U_8BIT
File: CU_MAP.C

The Combine_table array has up to 64 entries, each containing the combination information for the corresponding time slot. Standard E1 requires 32 table writes, corresponding to 32 time slots. Standard T1 requires 25 table writes, where the F-bit location is treated as the 25th time slot. Table 10-5 lists each entry in the receive combination table.

Table 10-5. Combine Table Entry Definition

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
—	—	DBANK		DROP_EN	BER_EN	COMBINE[1:0]	

COMBINE[1:0]	Source of RSER Output Data Combine Code—Selects one of four data sources for output on RSER during the respective receive PCM time slot destination. The data source is selected from one of three HDSL receive channels or the DBANK register. The first combine code that selects data from a HDSL channel receives the first payload byte mapped from that channel's payload block, as determined by the payload map. When COMBINE[1:0] is not 00, DBANK_SEL[1:0] must be 00.
00	Determined by DBANK_SEL[1:0]
01	HDSL receive channel1
10	HDSL receive channel2
11	HDSL receive channel3
BER_EN	BER Meter Enable—Places a copy of the respective PCM time slot's data into the BER meter. Any number of time slots can be copied without affecting throughput.
0	BER meter ignores PCM time slot
1	BER meter receives copy of PCM time slot data from RSER
DROP_EN	Enable DROP—Controls the state of the DROP output pin which marks the respective time slot coincident with data output on RSER.
0	DROP output pin remains inactive (low)
1	DROP output pin active (high)

DBANK_SEL[1:0]	Source of RSER Output Data Data Bank Select (applicable only if COMBINE = 00)—Selects one of three DBANK registers to output on RSER during the respective time slot.
00	Determined by COMBINE[1:0]
01	DBANK_1
10	DBANK_2
11	Determined by RSIG_EN

DBANK_3 (when RSIG_EN = 0)

RSIG_TBL (when RSIG_EN = 1)

10.11 tmap_table[9][_NO_OF_LOOPS]

Type: BP_U_8BIT
File: CU_MAP.C

The transmit payload map (TMAP_1–TMAP_9, defined in Table 10-6 through Table 10-14), determines whether HDSL payload bytes (bytes 1–36) are supplied from PCM time slots, DBANK registers, or the HDSL auxiliary channel data. All routed time slots to a given channel's TFIFO buffer must also be mapped out of the buffer. If PCM transmit data is input-aligned to MSYNC, the first TMAP byte to select PCM receives the first routed PCM time slot from the transmit PCM multiframe (i.e., PCM frame 0 maps to HDSL payload block 1). If PCM data is not aligned to MSYNC, payload bytes mapped from the TFIFO buffer are not aligned to PCM time slots and HDSL payload blocks are not aligned to PCM frames.

Table 10-6. Transmit Payload Map (TMAP_1)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
BYTE 4 TMAP[1:0]		BYTE 3 TMAP[1:0]		BYTE 2 TMAP[1:0]		BYTE 1 TMAP[1:0]	

Table 10-7. Transmit Payload Map (TMAP_2)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
BYTE 8 TMAP[1:0]		BYTE 7 TMAP[1:0]		BYTE 6 TMAP[1:0]		BYTE 5 TMAP[1:0]	

Table 10-8. Transmit Payload Map (TMAP_3)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
BYTE 12 TMAP[1:0]		BYTE 11 TMAP[1:0]		BYTE 10 TMAP[1:0]		BYTE 9 TMAP[1:0]	

Table 10-9. Transmit Payload Map (TMAP_4)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
BYTE 16 TMAP[1:0]		BYTE 15 TMAP[1:0]		BYTE 14 TMAP[1:0]		BYTE 13 TMAP[1:0]	

Table 10-10. Transmit Payload Map (TMAP_5)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
BYTE 20 TMAP[1:0]		BYTE 19 TMAP[1:0]		BYTE 18 TMAP[1:0]		BYTE 17 TMAP[1:0]	

Table 10-11. Transmit Payload Map (TMAP_6)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
BYTE 24 TMAP[1:0]		BYTE 23 TMAP[1:0]		BYTE 22 TMAP[1:0]		BYTE 21 TMAP[1:0]	

Table 10-12. Transmit Payload Map (TMAP_7)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
BYTE 28 TMAP[1:0]		BYTE 27 TMAP[1:0]		BYTE 26 TMAP[1:0]		BYTE 25 TMAP[1:0]	

Table 10-13. Transmit Payload Map (TMAP_8)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
BYTE 32 TMAP[1:0]		BYTE 31 TMAP[1:0]		BYTE 30 TMAP[1:0]		BYTE 29 TMAP[1:0]	

Table 10-14. Transmit Payload Map (TMAP_9)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
BYTE 36 TMAP[1:0]		BYTE 35 TMAP[1:0]		BYTE 34 TMAP[1:0]		BYTE 33TMAP[1:0]	

TMAP[1:0]	Transmit HDSL Payload Source
00	PCM data from TFIFO
01	DBANK_1
10	DBANK_2
11	DBANK_3

10.12 rmap_table[6][_NO_OF_LOOPS]

Type: BP_U_8BIT
File: CU_MAP.C

The receive payload map (RMAP_1–RMAP_6), defined in Table 10-15 through Table 10-20, controls placement of HDSL payload bytes (bytes 1–36) into the RFIFO. It accomplishes this by instructing the mapper to place or discard payload bytes from the received payload block (the RFIFO outputs to reconstruct the PCM channel). RMAP is programmed to discard bytes within the payload block that are not needed for PCM reconstruction. In T1 mode, RMAP must be programmed to choose which HDSL channel supplies F-bits, by enabling one extra byte of payload at the end of the payload block.

Table 10-15. Receive Payload Map (RMAP_1)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
—	—	RMAP[5:0]					

Table 10-16. Receive Payload Map (RMAP_2)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
—	—	RMAP[11:6]					

Table 10-17. Receive Payload Map (RMAP_3)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
—	—	RMAP[17:12]					

Table 10-18. Receive Payload Map (RMAP_4)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
—	—	RMAP[23:18]					

Table 10-19. Receive Payload Map (RMAP_5)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
—	—	RMAP[29:24]					

Table 10-20. Receive Payload Map (RMAP_6)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
—	—	RMAP[35:30]					

RMAP[35:0] Receive Payload Map—Six registers hold a 36-bit value to define which of the received HDSL payload bytes (bytes 1–36) are placed into the RFIFO. RMAP[0] corresponds with the first HDSL payload byte (byte 1). In T1 mode, F-bits are mapped by enabling one extra byte after the last payload-mapped byte. For example, RMAP[12] controls F-bit mapping to the RFIFO in 2T1 applications.

If RMAP[x] = 0, discard payload byte (x + 1)
If RMAP[x] = 1, map payload byte (x + 1) to RFIFO

10.13 _CuFlags

Type: CU_FLAGS
File: CU_INIT.C

The _CuFlags variable contains channel unit status flags, which dictate the flow of the channel unit code.

10.14 cu_reg_copy

Type: CU_REG_COPY
File: CU_INIT.C

The cu_reg_copy variable maintains a copy of registers that could change during typical channel unit operating conditions. This is different from the rate_values and htu_values arrays in that those arrays are only programmed during channel unit initialization.

The cu_reg_copy variable is used primarily by the bit-wise registers (CMD_1, CMD_2, etc.) because the channel unit does not have any write or read capability. When a register or bit-field is changed, the specific register is modified in the cu_reg_copy variable. The entire contents of the cu_reg_copy variable are then written to the part.

11.0 Functions

11.1 DSL Initialization Functions

This section discusses DSL or application level initialization functions are discussed. These functions can be found in DSL_INIT.C in the MAIN directory.

11.1.1 void _DSLInitialization(void)

This function initializes the HDSL system in this order:

1. Initialize software upon power-up
2. Initialize DIP switch
3. Initialize 8051
4. Initialize Global parameter
5. Check the presence of bit pumps and channel unit

Called by:

Application code
Application code

11.1.2 BP_U_8BIT _IsChannelUnitEvmPresent (void)

This function checks whether or not the channel unit card is present. If present, channel unit-related LEDs are turned off.

The return value is:

1 = _PRESENT
2 = _NOT_PRESENT

This function only needs to be called once during system initialization.

Called by:

Application code

11.1.3 BP_U_8BIT _InitChannelUnitEvmBoard (void)

This function initializes the channel unit EVM board. Sequence of events:

1. Determine framer type
2. Determine the EVM configuration
3. Initialize channel unit using the _CU_CONFIGURE command
4. Initialize framer using the _FRAMER_CONFIGURE command
5. Initialize LIU using the _LIU_CONFIGURE command

The return values are:

0 = _PASS	successfully completed operation
1 = _FAIL	an error occurred

This function only needs to be called once during system initialization.

Called by:

Application code

11.1.3.1 BP_U_8BIT _CuInitFramer (void)

This function initializes the framer (Bt8510, Bt8360, or Bt8370).

11.2 Channel Unit Initialization Functions

This section discusses channel unit initialization functions found in CU_INIT.C in the CHANUNIT subdirectory.

11.2.1 BP_U_8BIT _CuInitChannelUnit (void)

This function initializes all channel unit registers for a specific configuration and terminal unit. After initialization, the channel unit is in idle mode.

This function must be called during the initial configuration and when the channel unit is reset.

Called by:

CU_API.C (_CU_CONFIGURE API command)

11.2.2 void _CuInitAddresses (void)

This function initializes the channel unit EVM base addresses and channel unit, framer, and LIU pointers and masks all framer interrupts. In addition, _CuChipVersion.present is initialized to Not Present, the master loop is set to the location of the first LIC found, and the EOC handler state is initialized to be EOC_IDLE.

This function only needs to be called once during system initialization.

Called by:

Application code

11.2.3 void _CuInitCommonRegisters

This function initializes the channel unit's common registers.

Called by:

_CulnitChannelUnit()

11.2.4 void _CuInitHdslLoops

This function initializes the HDSL transmitter- and receiver-specific registers for all current loops. Any loop that does not connect to a LIC is not initialized.

Called by:

_CulnitChannelUnit()

11.3 Channel Unit Mapping Functions

This section discusses the channel unit mapping functions. These functions are responsible for initializing the transmit routing table, receive combination table, transmit payload mapper, receive payload mapper, and water levels based on the system configuration. They can be found in CU_MAP.C in the CHANUNIT subdirectory.

11.3.1 void _CuDefaultRouteLoops(void)

This function sets the default route_loop[] values. The default routes the channel unit channels 1, 2, and 3 to the LIC 1, 2, and 3 respectively.

Called by:

_CulnitChannel()

11.3.2 void _CuDefaultCombineLoops(void)

This function sets the default combine_loop[] values. The default combines the channel unit channels 1, 2, and 3 from the LIC 1, 2, and 3 respectively.

Called by:

_CulnitChannel()

11.3.3 void _CuInitMapper(void)

This function initializes the HDSL transmitter and receiver payload mappers (TMAP and RMAP) global arrays.

Called by:

_CulnitChannel()
_CuReverseLoops()

11.3.4 _CuInitRouteTable

This function initializes the route table. The global array route_loop[] determines how to route.

Called by:

_CulnitChannel()
_CuReverseLoops()

11.3.5 _CuInitCombineTable

This function initializes the combination table. The global array combine_loop[] determines how to set the combine table.

Called by:

_CulnitChannel()
_CuReverseLoops()

11.3.6 _CuWriteMapRouteCombine

This function writes the channel unit map, and route and combine tables, and resets the Rx receiver and Tx and Rx FIFOs.

Called by:

_CulnitChannel()
_CuReverseLoops()

11.4 void _ActivationStateManager(BP_U_8BIT bp)

This function manages the system state flow for the specified loop and is located in DSL_ASM.C. Figures 2-3 and 2-4 illustrate how the system transits from one state to another for HTU-C and HTU-R, respectively.

Called by:

Application code

11.5 void _ZipStartValidationManager(BP_U_8BIT bp)

This function validates the ZipStart after the specified loop reaches normal operation, and is located in DSL_MAIN.C. It is only compiled and linked when the ZIP_START directive is used.

Called by:

_ActivationStateManager

11.6 Channel Unit ASM-Related Functions

This section discusses channel unit functions called by the ASM. These functions are located in CU_ASM.C in the CHANUNIT subdirectory.

11.6.1 void _CuForceOnes(BP_U_8BIT state, BP_U_8BIT loop)

This function programs the HDSL to transmit forced 1s.

Called by:

_ActivationStateManager

11.6.2 void _CuConfigureBeginStartup (BP_U_8BIT loop)

This function configures the specified channel unit for startup in this order:

1. Set HDSL overhead and ZBits to all 1s
2. Set payload to all 1s
3. Disable Tx and Rx error interrupts
4. Disable Tx and Rx 6 ms interrupts
5. Disable DPLL error interrupt (if master loop)
6. Open DPLL NCO (if master loop)

Called by:

_ActivationStateManager()
_CU_FORCE_SCR_ONES API command

11.6.3 void _CuConfigureTransmitS1(BP_U_8BIT loop)

This function configures the specified channel unit for S1 Transmission.

Called by:

_ActivationStateManager()

11.6.4 void _CuHohEn(BP_U_8BIT state, BP_U_8BIT loop)

This function enables and disables the HDSL overhead state. Setting the state to 0 disables the HDSL overhead by forcing the overhead to all 1s. Setting the state to 1 enables the HDSL overhead.

Called by:

_CuConfigureTransmitS1()

11.6.5 void _CuConfigureStartupComplete (BP_U_8BIT loop)

This function configures the specified channel unit for startup. The following is performed:

1. Set HDSL overhead normal
2. Set Z-Bit PID (E1)
3. Set Payload to normal
4. Enable Tx and Rx error interrupts (should already be done)
5. Enable DPLL error interrupt (if Master Loop)
6. Close the DPLL (if master loop)
7. Reset Rx receiver (if master loop)
8. Reset Tx and Rx FIFOs

Called by:
Channel unit API command _CU_TRANSMIT_PAYLOAD

11.6.6 void _CuSetRtrInd(BP_U_8BIT state, BP_U_8BIT loop)

This function sets the transmit Ready To Receive (RTR) indicator bit.
Called by:
_ActivationStateManager()
_CuConfigureBeginStartup()
_CuConfigureStartupComplete()

11.6.7 void _CuSetPid(BP_U_8BIT loop)

This function initializes the first three Z-bits, also called the pair identification bits (PID) at the HTU-C. Table 11-1 lists initial pair IDs.

Table 11-1. Initial PID Values (E1 Mode)

	Z1	Z2	Z3
Loop 1	1	0	0
Loop 2	0	1	0
Loop 3	0	0	1

Called by:
_CulnitHdslLoop()
_ActivationStateManager()

11.6.8 void _CuSetPidToAllOnes(BP_U_8BIT loop)

This function initializes PID values at the HTU-R to all 1s.
Called by:
_ActivationStateManager()

11.7 Channel Unit Interrupt Handlers

This section discusses channel unit interrupt handlers. These functions are located in CU_INT.C in the CHANUNIT subdirectory.

11.7.1 void _CuInterruptHandler (void) interrupt 2

This function determines what caused the interrupt and handles the event appropriately. Depending on the type of interrupt, the corresponding interrupt handler (_CuTxInterrupt, _CuRxInterrupt, _CuDpllInterrupt, and _CuFramerInterrupt) is called. The DPLL interrupt handler and framer interrupt handler are called every 6 ms for the master loop.

11.7.2 void _CuTxInterrupt (BP_U_8BIT loop)

This function handles the transmitter interrupts.

Called by:

_CuInterruptHandler()

11.7.3 void _CuRxInterrupt (BP_U_8BIT loop)

This function handles the receiver interrupts.

Called by:

_CuInterruptHandler()

11.7.4 void _CuDpllInterrupt (void)

This function handles the DPLL interrupts.

Called by:

_CuInterruptHandler()

11.7.5 void _CuFramerInterrupt (void)

This function handles the transmitter interrupts.

Called by:

_CuInterruptHandler()

11.7.6 void E1_Pairid_Validation(BP_U_8BIT loop)

This function implements the E1 pair ID validation mechanism.

Called by:

_CuRxInterrupt()

11.8 DSL Dynamic Loop Managing Functions

This section discusses DSL dynamic loop managing functions. These functions are located in DSL_MAN.C in the MAIN directory.

11.8.1 void _DSLLoopHandler(void)

This function handles master loop switch and loop reversal dynamically.

Called by:

Application code

11.8.2 void _Set_2E1_PairID(BP_U_8BIT bp)

This function sets the 2E1 pair IDs after one loop reaches normal operation and its pair ID is validated.

Called by:

_DSLLoopHandler()

11.8.3 void _Set_2T1_SyncWord(BP_U_8BIT bp)

This function sets the 2T1 sync words after one loop reaches normal operation.

Called by:

_DSLLoopHandler()

11.8.4 void _Set_3E1_PairID1(BP_U_8BIT bp)

This function sets the 3E1 pair ID after one loop reaches normal operation and its pair ID is validated.

Called by:

_DSLLoopHandler()

11.8.5 void _Set_3E1_PairID2(BP_U_8BIT bp)

This function sets the 3E1 pair ID after two of the three loops reach normal operation and their pair IDs are validated.

Called by:

_DSLLoopHandler()

11.8.6 void _Reset_Pid_Validation(BP_U_8BIT bp)

This function resets the PID reservation for E1 applications.

Called by:

Application code

11.9 Channel Unit Dynamic Loop Managing Functions

This section discusses channel unit loop manager functions located in CU_LOOP.C in the CHANUNIT subdirectory.

11.9.1 void _CuSetMasterLoop(BP_U_8BIT loop)

This function sets the master loop to the specified loop.

Called by:

_DSLLoopHandler()

11.9.2 void _CuReverseLoops(void)

This function handles any possible loop reversal conditions by modifying transmit routing table, receive combination table, transmit payload mapper, and receive payload mapper based on the loops' pair IDs (E1 application) or synchronization words (T1 application).

Called by:

_DSLLoopHandler()

11.9.3 void _Configure_Channel_Blocking(void)

This function is used to configure the fractional T1 or E1 operation based on the time slot selection. The function modifies the:

- Transmit routing table
- Receive combine table
- Transmit payload mapping register
- Receive payload mapping register

Called by:

_DSLLoopHandler()
DSL API _CONFIGURE_CHANNEL_BLOCKING

11.9.4 void _CuCheckForLoopReversal(void)

This function checks to see if any loops are reversed. The global flag (_CuLoopReversed) is set.

Called by:

DSL API _DSL_STATUS

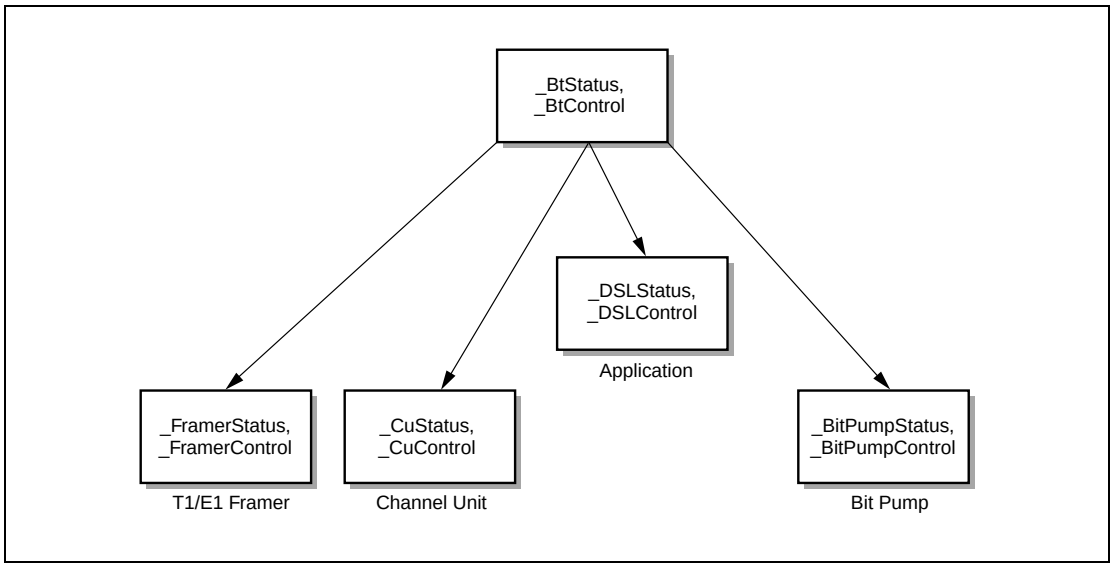
11.10 API Functions

This section discusses API functions. Table 11-2 lists all API functions and their origin. Figure 11-1 illustrates the API set structure.

Table 11-2. API Functions

API Functions	Source File	Directory
_BtStatus, _BtControl	BT_API.C	MAIN
_DSLStatus, _DSLControl	DSL_API.C	MAIN
_CuStatus, _CuControl	CU_API.C	CHANUNIT
_BitpumpStatus, _BitpumpControl	API.C	BITPUMP
_FramerStatus, _FramerControl	FRMR_API.C	CHANUNIT

Figure 11-1. API Command Parsing Structure



*11.10.1 _BtStatus(no, opcode, parameter, *indication)*

This function executes any status-checking API command. Depending on the destination of the API request, _DSLStatus, _CuStatus, _FramerStatus, or _BitpumpStatus is called.

Input Variables:
- BP_U_8BIT no
- BP_U_8BIT opcode
- BP_U_8BIT parameter

Returned Value:

_PASS	Command successfully interpreted, and will be executed.
_FAIL	Illegal command, command not executed. Cause may be illegal control command opcode, destination, or parameter.
BP_S_8BIT	*indication (contains the requested status information)

11.10.1.1 _DSLStatus(no, opcode, parameter, *indication)

This function executes any application-level status checking.

Called by:
- _BtStatus()

11.10.1.2 _CuStatus(no, opcode, parameter, *indication)

This function executes any channel unit status checking.

Called by:
- _BtStatus()

11.10.1.3 _FramerStatus(no, opcode, parameter, *indication)

This function executes any T1/E1 framer status checking.

Called by:
- _BtStatus()

11.10.1.4 _BitpumpStatus(no, opcode, parameter, *indication)

This function executes any bit pump status checking.

Called by:
- _BtStatus()

11.10.2 _BtControl(no, opcode, parameter)

This function executes any control API command. Depending on the destination of the API request, _DSLControl, _CuControl, _FramerControl, or _BitpumpControl is called.

Input Variables:
- BP_U_8BIT no
- BP_U_8BIT opcode
- BP_U_8BIT parameter

Returned Value:

_PASS	Command successfully interpreted, and will be executed.
_FAIL	Illegal command, command not executed. Cause may be illegal control command opcode, destination, or parameter.

11.10.2.1 _DSLControl(no, opcode, parameter)

This function executes any application-level control API command.

Called by:
- _BtControl()

11.10.2.2 _CuControl(no, opcode, parameter)

This function executes any channel unit control API command.

Called by:
- _BtControl()

11.10.2.3 _FramerControl(no, opcode, parameter)

This function executes any T1/E1 framer control API command.

Called by:
- _BtControl()

11.10.2.4 _BitpumpControl(no, opcode, parameter)

This function executes any bit pump control API command.

Called by:
- _BtControl()

11.11 Channel Unit EOC Functions

This section discusses EOC-related functions. These functions can be found in CU_EOC.C in the CHANUNIT subdirectory.

11.11.1 EOC Protocol Handler

11.11.1.1 void EocMaster(BP_U_8BIT loop)

This function handles the EOC protocol on the HTU-C side.

Called by:

_CuRxInterrupt() every 6 ms when CU_EOC compiler flag is used

11.11.1.2 void EocSlave(BP_U_8BIT loop)

This function handles the EOC protocol on the HTU-R side.

Called by:

_CuRxInterrupt() every 6 ms when CU_EOC compiler flag is used

11.11.2 EOC Task Handler

11.11.2.1 void EocTaskHandler_CO(BP_U_8BIT loop)

This function serves as an interface between EocMaster and application requests on the HTU-C.

Called by:

_CuRxInterrupt() every 6 ms when CU_EOC compiler flag is used.

11.11.2.2 void EocTaskHandler_RT(BP_U_8BIT loop)

This function serves as an interface between EocMaster and application requests on the HTU-C.

Called by:

_CuRxInterrupt(C) every 6 ms when CU_EOC compiler flag is used

11.11.3 Other EOC Related Functions

11.11.3.1 BP_U_8BIT EocGetData(BP_U_8BIT lByte, BP_U_8BIT hByte)

This function extracts the information field (8 bits) from the REOC registers.

Called by:

EocMaster() and EocSlave()

11.11.3.2 BP_U_16BIT EocSendWord(BP_U_8BIT command, BP_U_8BIT header)

This function passes EOC command/data and EOC header and writes them to TEOC registers.

Called by:

EocMaster() and EocSlave()

11.12 Channel Unit Utility Functions

This section discusses channel unit utility functions. These functions can be found in CU_UTILS.C in CHANUNIT subdirectory.

11.12.1 void _CuHandleFlags(void)

This function handles any flags that are set during an interrupt routine.

Called by:

Application code

11.12.2 void _CuWriteMasterCmd5(void)

This function writes the CMD_5 register based on the current master loop.

Called by:

_CuIntiCommonRegisters()

_CuSetMasterLoop()

11.12.3 void _CuClearCounters(BP_U_8BIT loop, BP_U_8BIT cntr)

This function clears the specified Error Counters for the specified loop.

Called by:

_CuConfigureStartupComplete()

_CuRxInterrupt()

Channel unit API command:

_CU_CLEAR_ERROR_COUNTERS

11.12.4 void _CuResetTxFIFO(void)

This function resets the transmitter FIFO for all loops. FIFOs are emptied, and the HDSL transmitter is forced to resample the transmit water level and realign the HDSL channel's transmit 6 ms frame to the PCM 6 ms frame.

Called by:

Channel unit API commands:

_CU_RESET_TX_FIFO

_CU_RESET_PCM

_CuWriteMapRouteCombine()

11.12.5 void _CuResetRxFIFO(void)

This function resets the receiver FIFO for all loops. FIFOs are emptied and the payload mapper is realigned with HDSL bytes with respect to the receive HDSL 6 ms frame.

Called by:

Channel unit API commands:

_CU_RESET_RX_FIFO
_CU_RFIFO_WL_LO
_CU_RFIFO_WL_HI
_CU_RESET_PCM

_CuWriteMapRouteCombine()

11.12.6 void _CuResetReceiver(void)

This function forces the PCM formatter to align the PCM receive timebase with respect to the master HDSL channel's receive 6 ms frame by reloading the RFIFO_WL value.

Called by:

_CuConfigureStartupComplete()
_CuWriteMapRouteCombine()
Channel unit API commands:
_CU_RFIFO_WL_LO
_CU_RFIFO_WL_HI
_CU_RESET_PCM

*11.12.7 TX_RD *get_tx_rd_ptr(BP_U_8BIT loop)*

This function returns the TX_RD (transmit status, read register) pointer address for the specified channel unit loop. The only transmit status register is the STATUS-3 register. For loop 1 it is at address 0x07; for loop 2, 0x0F; for loop 3, 0x17.

Return:

pointer to address (success)
0—invalid LOOP (fail)

Called by:

_CuStatus()

*11.12.8 RX_RD *get_rx_rd_ptr(BP_U_8BIT loop)*

This function returns the RX_RD (receive status, read register) pointer address for the specified channel unit loop.

Return:

pointer to address (success)
0—invalid LOOP (fail)

Called by:

_CuStatus()

11.13 General Purpose Timer Functions

This section discusses functions for general purpose timers. These functions are located in TIMER.C in the MAIN directory.

11.13.1 void _InitGenPurposeTimer(void)

This function initializes interrupt #1, which decrements the timer counters when timer #0 expires. The Keil compiler configures timer 0 as interrupt 1. (This function only applies if INT_BUG is not specified.)

Called by:

Application code

11.13.2 void _LoadGenPurposeTimerInterval(void)

This function loads the microprocessor timer counter value. The counter is set to an ~50 ms interval.

Called by:

_InitGenPurposeTimer()
_Timer0_ISR()

11.13.3 void _EnableGenPurposeTimer(bp, timer, value)

This function enables the timer #0 interrupt and initializes the counter.

Called by:

Application code
_ActivationStateManager()

11.13.4 void _ContinuousGenPurposeTimer(bp, timer, state)

This function sets the specified timer to continuous or discrete.

Called by:

Application code

11.13.5 void _DisableGenPurposeTimer(bp, timer)

This function disables the timer #0 interrupt.

Called by:

Application code
_ActivationStateManager()

11.13.6 BP_U_8BIT _GetGenPurposeTimerStatus(bp, timer)

This function returns the timer status.

Bit #	#define	Description
0	TIMER_STATE	Timer State—0=Off, 1=On
1	TIMER_COMPLETE	Timer Complete—0=No, 1=Yes

Return:
status (success)
0xFF—invalid timer specified (fail)

Called by:
Application code
_ActivationStateManager()

11.13.7 BP_U_32BIT _GetGenPurposeContCount(bp, timer)

This function returns the continuous-timer elapsed count.
Return:
status (success)
0xFF—invalid timer specified (fail)

Called by:
_ActivationStateManager()
_ZipStartValidationManager()

11.13.8 _Timer0_ISR

This interrupt is called when timer #0 overflows.
Called by:
Interrupt 1

11.14 Performance Monitoring Functions

This section discusses performance monitoring functions. These functions are located in CU_PERF.C in the CHANUNIT subdirectory.

11.14.1 void InitPMRecord(BP_U_8BIT loop)

This function clears the CRC and FEBE records at different intervals, initializes the record pointers and counters, and resets the running seconds counter.

Called by:

Application code when the specified loop reaches GOTO_ACTIVE_TX_RX_STATE

11.14.2 void UpdatePMRecord(BP_U_8BIT loop)

This function updates the CRC and FEBE records for the specified loop at different time intervals when the loop is in normal operation.

Called by:

_CuRxInterrupt every 6 ms when PERF_MONITOR compiler flag is used

11.14.2.1 void UpdateInterval1(BP_U_8BIT loop)

This function updates the CRC and FEBE records for the specified loop every second.

Called by:

UpdatePMRecord

11.14.2.2 void UpdateInterval2(BP_U_8BIT loop)

This function updates the CRC and FEBE records for the specified loop every 15 minutes.

Called by:

UpdatePMRecord

11.14.2.3 void UpdateInterval3(BP_U_8BIT loop)

This function updates the CRC and FEBE records for the specified loop every 24 hours.

Called by:

UpdatePMRecord

11.15 void _Configure_Channel_Blocking(void)

This function configures the channel blocked T1 or E1 operation based on the time slot selection. The function will modify these tables:

- Transmit routing
- Receive combine
- Transmit payload map
- Receive payload map

11.16 DSL Miscellaneous Functions

This section discusses DSL miscellaneous functions. These functions can be found in DSL_MISC.C in the MAIN directory.

11.16.1 void _Cu_Led_Update(bp, state)

This function updates channel unit-related LEDs. (See Figures 2-3 and 2-4 for the definition of the LED registers.)

Called by:

Application code

11.16.2 void _Bp_Led_Update(bp, state)

This function updates bit pump–related LEDs. (See Figure 2-5 for the definition of the LED register.)

Called by:

Application code

Appendix A: Acronyms and Abbreviations

AIS	Alarm Indication Signal
2B1Q	2 Binary, 1 Quaternary
BER	Bit Error Rate
CMOS	Complementary Metal-Oxide Semiconductor
CRC	HDSL Cyclic Redundancy Check
DPLL	Digital Phase Lock Loop
EOC	HDSL Embedded Operations Channel
ESF	Extended Superframe
FEBE	HDSL Far-End Block Error
JTAG	Joint Test Action Group
HDSL	High-Bit-Rate Digital Subscriber Line
HOH	HDSL Overhead
HRP	HDSL Repeater Present
HTU-C	HDSL Terminal Unit at the Central Office
HTU-R	HDSL Terminal Unit at the Remote Distribution
LIU	Line Interface Unit
LOSD	Loss of Signal - DS1
LOSW	HDSL Loss-of-Sync Word
LSB	Least Significant Bit
LFSR	Linear Feedback Shift Register
MSB	Most Significant Bit
PQFP	Plastic Quad Flat Pack
PLCC	Plastic Leaded Chip Carrier
PRBS	Pseudo-Random Binary Sequence
QUAT	Quaternary Symbol
QRSS	Quasi-Random Sequence Signal
SF	Super Frame
UIB	Unspecified Indicator Bit
VCXO	Voltage-Controlled Crystal Oscillator

Appendix B: References

RS8953B/RS8953SPB HDSL Channel Unit Data Sheet, April 1999 (N8953BDSB)

International Telecommunication Union. *High Speed Digital Subscriber Line (HDSL) Transceivers*. October, 1998 (G.991.1)

ZipWire Software User Guide, May 2000 (100417C)

Further Information
literature@conexant.com
(800) 854-8099 (North America)
(949) 483-6996 (International)
Printed in USA

World Headquarters
Conexant Systems, Inc.
4311 Jamboree Road
Newport Beach, CA
92660-3007
Phone: (949) 483-4600
Fax 1: (949) 483-4078
Fax 2: (949) 483-4391

Americas

U.S. Northwest/
Pacific Northwest – Santa Clara
Phone: (408) 249-9696
Fax: (408) 249-7113

U.S. Southwest – Los Angeles
Phone: (805) 376-0559
Fax: (805) 376-8180

U.S. Southwest – Orange County
Phone: (949) 483-9119
Fax: (949) 483-9090

U.S. Southwest – San Diego
Phone: (858) 713-3374
Fax: (858) 713-4001

U.S. North Central – Illinois
Phone: (630) 773-3454
Fax: (630) 773-3907

U.S. South Central – Texas
Phone: (972) 733-0723
Fax: (972) 407-0639

U.S. Northeast – Massachusetts
Phone: (978) 367-3200
Fax: (978) 256-6868

U.S. Southeast – North Carolina
Phone: (919) 858-9110
Fax: (919) 858-8669

U.S. Southeast – Florida/
South America
Phone: (727) 799-8406
Fax: (727) 799-8306

U.S. Mid-Atlantic – Pennsylvania
Phone: (215) 244-6784
Fax: (215) 244-9292

Canada – Ontario
Phone: (613) 271-2358
Fax: (613) 271-2359

Europe

Europe Central – Germany
Phone: +49 89 829-1320
Fax: +49 89 834-2734

Europe North – England
Phone: +44 1344 486444
Fax: +44 1344 486555

Europe – Israel/Greece
Phone: +972 9 9524000
Fax: +972 9 9573732

Europe South – France
Phone: +33 1 41 44 36 51
Fax: +33 1 41 44 36 90

Europe Mediterranean – Italy
Phone: +39 02 93179911
Fax: +39 02 93179913

Europe – Sweden
Phone: +46 (0) 8 5091 4319
Fax: +46 (0) 8 590 041 10

Europe – Finland
Phone: +358 (0) 9 85 666 435
Fax: +358 (0) 9 85 666 220

Asia – Pacific

Taiwan
Phone: (886-2) 2-720-0282
Fax: (886-2) 2-757-6760

Australia
Phone: (61-2) 9869 4088
Fax: (61-2) 9869 4077

China – Central
Phone: 86-21-6361-2515
Fax: 86-21-6361-2516

China – South
Phone: (852) 2 827-0181
Fax: (852) 2 827-6488

China – South (Satellite)
Phone: (86) 755-5182495

China – North
Phone: (86-10) 8529-9777
Fax: (86-10) 8529-9778

India
Phone: (91-11) 692-4789
Fax: (91-11) 692-4712

Korea
Phone: (82-2) 565-2880
Fax: (82-2) 565-1440

Korea (Satellite)
Phone: (82-53) 745-2880
Fax: (82-53) 745-1440

Singapore
Phone: (65) 737 7355
Fax: (65) 737 9077

Japan
Phone: (81-3) 5371 1520
Fax: (81-3) 5371 1501