

ZipWire Bit-pump Version 4.2 Channel Unit Version 6.1 Release Notes

Table of Contents

1	INTRODUCTION.....	2
2	KNOWN PROBLEMS	2
2.1	BIT-PUMP CODE.....	2
2.1.1	Un-initialized Pointer in _InitTxGain().....	2
2.2	APPLICATION AND CHANNEL UNIT CODE	2
2.2.1	API Command (_CU_SET_MFRAME) Does Not Process Zero Correctly	2
2.2.2	LEDs not modified correctly when SINGLE_LOOP and TWO_LOOPS defined.....	3
3	MEMORY REQUIREMENTS AND COMPILER INFORMATION	4
3.1	RAM AND ROM REQUIREMENTS	4
3.2	CHANGES TO KEIL CA51 PROJECT FILES	4
4	BIT-PUMP CODE	5
4.1	BUG FIXES.....	5
4.1.1	Incorrect Initial AAGC at Data Rates less than 256kbps.....	5
4.1.2	API Command (_SYSTEM_CONFIG, _NOT_PRESENT) Did Not Mask Interrupts.....	5
4.1.3	Power Up/Down AFE Section in SYSTEM_CONFIG API Command.....	5
4.1.4	Incorrect Range Check in _ReadNmr() Function.....	6
4.2	ENHANCEMENTS	6
4.2.1	80C32 Microcontroller Stack Increase	6
4.2.2	Updated FELM Table.....	6
4.2.3	Switch Capacitor Pole Activated for RS8973, Rev. B	7
4.2.4	HTU-R Watch Dog Timer.....	7
4.2.5	HTU-R Glitch Code.....	7
4.2.6	Increased Temperature/Environment Interrupt Rate	7
4.2.7	Increased Transmit Gain for Lower Data Rates	8
4.3	API DETAILS.....	8
4.3.1	Activation Timer Command.....	8
5	APPLICATION AND CHANNEL UNIT CODE	9
5.1	BUG FIXES.....	9
5.1.1	ETSI Specification for EOC Non-Conformance.....	9
5.2	ENHANCEMENTS	9
5.2.1	More Comments	9
5.2.2	ROM reductions	9
5.2.3	Combined Calls to _Reset_Pid_Validation() in ASM.....	9
5.2.4	Single Loop multi-rate.....	9
5.2.5	DPLL Status API.....	9
5.2.6	Repeater Support.....	12
5.3	API DETAILS.....	19
5.3.1	Miscellaneous Commands.....	19
5.3.2	Channel Unit Indicator Bit Commands.....	19
5.3.3	Single Loop Commands.....	20
5.3.4	EOC Commands.....	22
6	EOC APPLICATION EXAMPLES	28
6.1	READ EXAMPLE.....	28
6.1.1	Step 1: Select HTU-C Register Name.....	28
6.1.2	Step 2: Set HTU-C Register Size.	28
6.1.3	Step 3: Set up HTU-R Register Number and Size.....	28
6.1.4	Step 4: Load the HTU-R Read Register D.....	28
6.1.5	Step 5: Set Command for HTU-C to Read HTU-R Register D.....	29
6.1.6	Step 6: Read the New Data Flags.	29
6.1.7	Step 7: Set Index to 0 for Read Register D.....	29
6.1.8	Step 8: Read the Read Register D.....	29
6.2	WRITE EXAMPLE	30
6.2.1	Step 1: Set up HTU-C Register Number.....	30
6.2.2	Step 2: Set up HTU-C Register Size.	30
6.2.3	Step 3: Set up HTU-R Register Number and Size.....	30
6.2.4	Step 4: Load the HTU-C Write Register B.....	30
6.2.5	Step 5: Set HTU-C "Start Sending" Command to Write HTU-R Register B.....	31
6.2.6	Step 6: Read the Received Data Status.....	31
6.2.7	Step 7: Set Byte Number Location.....	31
6.2.8	Step 8: Read the B Data Register.....	31

6.3	HTU-C CRC CHECK COMMAND EXAMPLE	32
6.3.1	Step 1: HTU-C receives Corrupted CRC from HTU-R.	32
6.3.2	Step 2: Set the End Corrupted CRC Command.	32
6.4	HTU-R CRC CHECK COMMAND EXAMPLE	33
6.4.1	Step 1: Notify the HTU-R of Corrupted CRC.	33
6.4.2	Step 2: Send over Corrupted CRC.	33

1 Introduction

The ZipWire Bit-pump Version 4.2 Channel Unit Version 6.1 release includes changes to the 3 software components: Bit-pump, Channel Unit, and Application Code. The release notes discuss the bit-pump code in Section 4; the Channel Unit and Application Code in Section 5. This separation allows customers to easily understand the changes and incorporate them into their system.

ZipStartup customers should refer to ZipStartup Release Notes Version 1.3 for additional information. The files, *zipvalid.c* and *zipvalid.h*, are no longer provided in the non-ZipStartup release.

The new features added in the ZipWire Bit-pump Version 4.2 Channel Unit Version 6.1 (Single Pair API, EOC, Indicator Bits, DPLL status) are only supported in the Version 6.0 UIP release.

2 Known Problems

Verified problems with the release are identified in this section. Changes described here will be incorporated into the next software release. Whenever a problem is added to this section, these release notes will be re-issued and the document will be given a new Rev A suffix (A01, A02, etc.). Changes from one revision to the next of this document can be identified by change bars on the margin.

2.1 Bit-pump Code

2.1.1 Un-initialized Pointer in _InitTxGain()

As discussed in Section 4.2.7, the INIT_TX_GAIN macro was replaced by the Init_Tx_Gain() function. The pointer to the bitpump address space, bp_mode_ptr, is declared but it is not initialized. This causes the transmit gain to be un-initialized.

Modify UTIL.C - Starting at Line 810:

```
void _InitTxGain (BP_U_8BIT no)
{
    DECLARE_MODE_PTR;
    BP_S_8BIT cal_tx_gain;

    /*Addition starts*/
    INIT_BP_MODE_PTR;
    /*Addition ends*/

    /* Read Calibrated Tx Gain */
    cal_tx_gain = BP_READ_BIT(bp_mode_ptr, tx_calibrate, tx_gain);
```

2.2 Application and Channel Unit Code

2.2.1 API Command (_CU_SET_MFRAME) Does Not Process Zero Correctly

The _CU_SET_MFRAME API command sets the multi-frame length minus 1. The code does not properly handle a parameter value of zero that sets the MFRAME_SYNC pulse to be 125 μ s.

Change CU_API.C - Starting at Line 570:

```

case _CU_SET_MFRAME:
/*
 * parameter is valid from 0x00 to 0x2F ( 0 to 47 ).
 * parameter = MF_LEN;
 * TMF_LOC = MF_LEN - 1;
 * ( MF_CNT + 1 ) * ( MF_LEN + 1 ) = 48;
 */
if ( parameter < 0 || parameter > _6MS_MFRAME )
{
    break;
}
/*Addition starts*/
if ((parameter == 0x00)
{
    common_wr_ptr->tmf_loc = 0x00;
}
else
{
/*Addition ends*/
    common_wr_ptr->tmf_loc = parameter - 1;
/*Addition starts*/
}
/*Addition ends*/

    common_wr_ptr->mf_len = parameter;
    common_wr_ptr->mf_cnt = (48 / ( parameter + 1 )) - 1;
    break;

```

2.2.2 LEDs not modified correctly when _NO_OF_LOOPS is less than 3.

The definition of the size of the arrays, cu_bp_led_block[] and cu_led_block[], is based on _NO_OF_LOOPS, but they are initialized based on hard-coded array indices. When _NO_OF_LOOPS is less than 3, the initialization of these arrays corrupts other RAM data.

Change CU_LED.C - Starting at Line 35:

```
CU_BP_LED_BLOCK BP_XDATA cu_bp_led_block[_NO_OF_LOOPS];
```

To:

```
CU_BP_LED_BLOCK BP_XDATA cu_bp_led_block[2];
```

Change CU.H - Starting at Line 1259:

```
extern CU_BP_LED_BLOCK BP_XDATA cu_bp_led_block[_NO_OF_LOOPS];
```

To:

```
extern CU_BP_LED_BLOCK BP_XDATA cu_bp_led_block[2];
```

Change CU_INIT.C - Starting at Line 466:

```

cu_led_block[0].reg = 0;
cu_led_block[1].reg = 0;
cu_led_block[2].reg = 0;

```

To:

```

for ( loop_cntr = 0 ; loop_cntr < _NO_OF_LOOPS ; loop_cntr++ )
{
    cu_led_block[loop_cntr].reg = 0;
}

```

3 Memory Requirements and Compiler Information

3.1 RAM and ROM Requirements

Table 1 shows the RAM and ROM requirements for various build options. Unless specified otherwise, the code was compiled with the Keil CA51 compiler (Version 5.5A) using the Small Memory Model, the Large Code Size Limit, and the following compiler switches: SB, DB, OE, and NOAM. This list shows the DEFINE options used for each build option:

1. ZIPWIREC: HTU-C EVM Software with Channel Unit

C51,PDATA_MODE,ADD_DELAY,SER_COM,HTUC,BER_METER,ERLE,CHAN_UNIT

2. ZIPWIRER: HTU-R EVM Software with Channel Unit

C51,PDATA_MODE,ADD_DELAY,SER_COM,HTUR,BER_METER,ERLE,CHAN_UNIT

3. ZIPWIREC: Minimum HTU-C

C51,PDATA_MODE,ADD_DELAY,HTUC

4. ZIPWIRER: Minimum HTU-R

C51,PDATA_MODE,ADD_DELAY,HTUR

Table 1. ZipWire Memory Requirements

	Build Option	RAM	XDATA RAM	ROM
1	ZIPWIREC: HTU-C EVM with CU, 3E1	171	3166	61588
2	ZIPWIRER: HTU-R EVM with CU, 3E1	171	3170	62513
3	ZIPWIREC: Minimum HTU-C	116	85	27965
4	ZIPWIRER: Minimum HTU-R	117	85	27418

By default, build options that define CHAN_UNIT include support for 2E1, 3E1, and 2T1 configurations. While the EVM supports all these options, most end applications do not need to support all these options. Therefore, the RAM and ROM requirements can be significantly trimmed by deleting the unused configurations. The user should modify *typedefs.h* as required for their particular application and stage of development. For example, the ERLE option can be deleted after the verification of the hybrid circuitry.

If a compiler directive (e.g. SINGLE_LOOP) that specifies the number of lines is not defined, then *typedefs.h* defines THREE_LOOPS as the default number of loops. Defining more loops than are required wastes RAM and processing time.

The compiler directives, 1T1 and 1E1, have been superseded by a single directive, SP_API. See Section 5.2.4 for more information.

3.2 Changes to Keil CA51 Project Files

The release continues to include Keil CA51 project files that are utilized by the Keil μ Vision Integrated Developer's Environment (IDE). The release includes the following Keil project files:

1. ZIPWIREC: C51,PDATA_MODE,ADD_DELAY,SER_COM,HTUC,BER_METER,ERLE,CHAN_UNIT
2. ZIPWIRER: C51,PDATA_MODE,ADD_DELAY,SER_COM,HTUR,BER_METER,ERLE,CHAN_UNIT
3. TDEBUG: C51,PDATA_MODE,ADD_DELAY,SER_COM,HTUC,HTUR,BER_METER,ERLE
4. ZIPREG: C51,PDATA_MODE,ADD_DELAY,SER_COM,HTUC,HTUR, BER_METER,ERLE,CHAN_UNIT, TWO_LOOPS

The release includes a new Keil project file, *zipreg.prj*, to build the repeater software. The Keil project file *zipwire.prj*, is no longer provided. The following changes were made to the project files:

1. Updated to use a later version (Version 5.5A for Windows) of the Keil compiler.
2. Removed the file, *zipvalid.c*, from the source file lists.

4 Bit-pump Code

The following changes were made to the Bit-pump code found in the BIT-PUMP directory.

4.1 Bug Fixes

The following bugs were fixed in ZipWire Bit-pump Version 4.2 Channel Unit Version 6.1.

4.1.1 Incorrect Initial AAGC at Data Rates less than 256kbps

In ZipWire Version 4.1, the software incorrectly set the initial AAGC setting for data rates less than 256 kbps. This bug is limited to data rates \leq 256 kbps when the hybrid has poor analog echo cancellation on short loops. The bug could incorrectly adjust the zero length FELM threshold that would then prevent detecting the real far-end signal and thus would not successfully perform start-up.

Change SUC.C - Starting at Line 127 and SUR.C – Starting at Line 146:

```
/* At low data rate, use higher initial AAGC */
if ( _bp_vars[no].symbol_rate <= 32 )
{
    AAGC(AGAIN12DB);
}
```

To:

```
/* At low data rate, use higher initial AAGC */
if ( _bp_vars[no].symbol_rate <= 32 )
{
    _SetAdcControlAgain(no, AGAIN12DB);
}
```

4.1.2 API Command (_SYSTEM_CONFIG, _NOT_PRESENT) Did Not Mask Interrupts

In ALL previous bit-pump releases (Bt8952, Bt8960, Bt8970, and Bt8973), the _SYSTEM_CONFIG, _NOT_PRESENT API command did not mask off the bit-pump interrupts. Since the interrupts were not masked, the device would generate the *IRQ but the interrupt handler would not process the interrupt since the device's 'operational' bit was cleared (set to 0). This causes the microprocessor to get stuck in the bit-pump interrupt handler.

The following file is effected: API.C – Line 150

4.1.3 Power Up/Down AFE Section in SYSTEM_CONFIG API Command

As described in the RS8973 Product Bulletin (#N8973BL3B, April 12, 1999):

“The Analog Front-End portion of the RS8973 is not initialized after power-on reset, and during power down mode, as stated in the data sheet. However, the Conexant supplied device driver will correctly initialize this portion of the circuit. The device will operate normally after initialization by the device driver.”

The RS8973 power down mode is set by the _SYSTEM_CONFIG API command. This AFE power down bug causes the RS8973 AFE to remain in its current state, therefore the RS8973 device could still be transmitting a signal. This bug is only an issue if the application issues the _SYSTEM_CONFIG, _NOT_PRESENT command.

In ZipWire Bit-pump Version 4.2 Channel Unit Version 6.1, the `_SYSTEM_CONFIG`, `NOT_PRESENT` command manually disables the AFE section prior to setting the RS8973 power down mode. The `_SYSTEM_CONFIG`, `_PRESENT` command manually enables the AFE section after setting the RS8973 power up mode

The following file is effected: `API.C` – Line 150 and `API.C` – Line 159.

4.1.4 Incorrect Range Check in `_ReadNmr()` Function

In the HTU-R, the `_ReadNmr()` function verifies that the PLL is locked before returning the noise margin reading. This bug should not cause any performance problems. The HTU-R uses a different function, `_IsPhaseLocked()`, to determine whether the PLL is locked during startup.

Change `UTIL.C` – Line 1103:

```
if ( p_value >= MAX_PLL_VALUE || -p_value <= -MAX_PLL_VALUE )
```

To:

```
if ( p_value >= MAX_PLL_VALUE || p_value <= -MAX_PLL_VALUE )
```

4.2 Enhancements

4.2.1 80C32 Microcontroller Stack Increase

The global variables used by the bit-pump software were moved from internal RAM to external RAM. This change was required to support the channel unit single pair (`SP_API`) option (see Section 5.2.4).

Change `TYPEDEFS.H`:

```
#define BP_MSPACE          BP_IDATA
```

To:

```
#define BP_MSPACE          BP_XDATA
```

4.2.2 Updated FELM Table

As stated in the Bit-pump Version 3.0 Release Notes – Outstanding Issues Section:

“The Far-end Attenuation API command (0x82 - `_FELM`) is calibrated for 1168kbps data rates. When running at other data rates, the `_FELM` API command does not return an accurate result.”

In ZipWire Bit-pump Version 4.2 Channel Unit Version 6.1, the FELM Lookup Table is expanded to support multiple data rates. The FELM Lookup Table is dimensioned for 8 data rates: 144kbps, 272kbps, 400kbps, 528kbps, 784kbps, 1168kbps, 1552kbps, and 2320kbps. For other data rates, the table will use the closest data rate entry.

The Far-end Attenuation Table is calibrated (normalized) to a 150kHz Sine Wave while transmitting 13.5dBm of power.

The following files are effected:

`UTIL.C` – Line 136, new 2 dimensional FELM table, `_far_end_signal[][]`, defined.

`API.C` – Line 727, new table index, `_bp_vars[no].felm_lookup_index`, passed for table lookup.

`UTIL.C` – Line 575, new table index, `_bp_vars[no].felm_lookup_index`, initialized

`EXTERN.H` – Line 84, new table index, `felm_lookup_index`, defined

4.2.3 Switch Capacitor Pole Activated for RS8973, Rev. B

As described in the RS8973 Non-Conformance #2 (#N8973BL2, November 11, 1998):

“The RS8973 does not meet the Pulse Shape and Output Power specifications for 2320 kbps operation according to ETSI TS-101-135 (formerly ETR-152) and the RS8973 data sheet.”

The hardware non-conformance was corrected in RS8973, Revision B. An associated software change is required to enable the switch_cap_pole bit in the adc_control register (0x21). The software enables this bit only if the data rate is above 1,200 kbps and the RS8973 device is not revision A.

The following file is effected: UTIL.C – Line 356.

4.2.4 HTU-R Watch Dog Timer

The HDSL specifications specify that the HTU-R does not start its activation timer until after it detects a signal. In order to support features added to the ZipWire software in the future, it is beneficial to have a watch dog timer running in the HTU-R while the bit-pump is in the WAIT_FOR_SIGNAL state. This watch dog timer uses start-up timer 4 (SUT4) in the bit-pump and sets the timer to expire after 0xFFFF (65,535) symbols. The HTU-R watch-dog-timer values at various data rates are given in Table 2. The activation time-outs values as discussed in the Version 4.0 Release Notes are also included in Table 2.

Table 2. Activation Time-out and Watchdog Timer settings.

Data Rate (kbps)	Typical Start-up Time (s)	Activation Time-out (s)	HTU-R Watch Dog Timer (s)
144	64.4	128.8	932.1
288	35.3	70.5	466.0
416	26.3	52.6	322.6
784	16.8	33.7	171.2
1168	13.3	26.7	114.9
1552	11.5	23.1	86.5
2320	9.8	19.5	57.9

The following file is effected: SUR.C – Line 575. Start timer.

Note: The expiration of the SUT4 timer is already handled in the bit-pump interrupt.

4.2.5 HTU-R Glitch Code

When the HTU-R bit-pump activation state machine is in the WAIT_FOR_SIGNAL state, it may attempt to train on a line “glitch” (a temporary line connection and disconnection). In previous versions of the code, the HTU-R would fail eventually fail that start-up attempt. The activation-state-manager would detect the failure and successfully perform a re-train. The added HTU-R glitch code will detect this failure scenario sooner and cause the bit-pump to not react to the glitch and thus have a greater likelihood of a successful start-up on the first attempt.

The following file is effected: SUR.C – Line 249.

4.2.6 Increased Temperature/Environment Interrupt Rate

In previous ZipWire releases, the processing rate for the temperature/environmental compensation was set at once every meter interval (32,768 symbols). This setting allows for a maximum temperature variation of 30° C/hour. This variation is unnecessarily conservative and the default processing-rate was changed to once every 3 meter intervals (98,304 symbols). This rate compensates for a (more realistic) maximum temperature variation of 10 °C/hour. This

modification increases the processing rate from once every 84 msec. (for a 784 kbps DSL rate) to once every 252 msec. (for a 784 kbps DSL rate). Because this processing occurs endlessly in normal operation, the reduced processing load may noticeably improve system performance especially in applications with many bit-pumps. Note that the bit-pump interrupt rate remains unchanged.

The customer can change the processing rate using the following #define in *util.c*:

```
#define DEFAULT_TE_METER_INTERVAL    3
```

This value determines the number of 32,768 meter intervals that must expire before *_HandleTempEnv()* performs temperature/environment processing. On every 32,768 symbol interrupt, *HandleTempEnv()* increments a counter and compares it to variable that is set to *DEFAULT_TE_METER_INTERVAL*.

The following files are effected:

UTIL.C – Line 577, new counter, *_bp_vars[no].te_num_meter_intervals*, initialized.

BIT-PUMP.C – Line 249, comparison of counter to interval.

EXTERN.H – Line 85, *te_meter_interval_cntr* and *te_num_meter_intervals*, defined.

4.2.7 Increased Transmit Gain for Lower Data Rates

When using the RS8973 and a 2mH transformer, as recommend for the H101 hybrid, there is a roll-off in transmit power at lower data rates. To compensate for this decreased transmit power, the bit-pump driver automatically increases the value in the transmitter calibration register (0x28). The value is increased based on the data rate as shown in **Table 3**. These adjustments ensure that the transmit power is between 13.5 and 13.7 dBm. The adjusted calibrated transmit gain is then written to the *tx_gain* register (0x29). The macro *INIT_TX_GAIN* in *util.h* was replaced by the *_InitTxGain()* function in *util.c* to remove any compiler problem with a long macro. The software performs a limit check on the adjusted gain value.

Table 3. Transmit Gain Adjustment

Data Rate (kbps)	Increase to Calibrated Transmit Gain
0 < data rate <= 144	+5
144 < data rate <= 208	+3
208 < data rate <= 336	+2
336 < data rate <= 528	+1
528 < data rate	No change

The following files are effected:

UTIL.C – Line 368, *InitTxGain()* function called.

UTIL.C – Line 808, *InitTxGain()* function added.

UTIL.H – Line 111, *INIT_TX_GAIN* is removed with #If 0

4.3 API Details

4.3.1 Activation Timer Command

When the channel unit is present, the activation timer is set to 30 seconds per HDSL specifications. This command allows the user to stop or restart the activation timer in an application that exceeds the 30-second start-up. An example is the repeater application.

Opcode: 0x26 (_ACTIVATION_TIMER)		
Destination	Description	Parameter
_BIT_PUMP0-2	Modifies the activation timer.	0x00-Stop timer 0x01-Restart timer

5 Application and Channel Unit Code

The following changes were made to the application code and channel unit found in the root and CHANUNIT directories.

5.1 Bug Fixes

The following bugs were fixed in ZipWire Bit-pump Version 4.2 Channel Unit Version 6.1.

5.1.1 ETSI Specification for EOC Non-Conformance.

In ZipWire CU Version 6.0, the EOC protocol met ETSI standards, but the software did not have an application layer implementation. New sets of API commands were added to help the user modify and use the EOC channel for far-end communication. The API Details section has more specifics of what API commands were added.

These APIs include support for all the commands defined in the ETSI standards. The APIs support general purpose Write Register and Read Register processing. EOC Write Register processing is shown in [Figure 1](#); EOC Read Register processing is shown in [Figure 2](#).

5.2 Enhancements

5.2.1 More Comments

Many comments have been added to the application and channel unit software to make it easier to understand.

5.2.2 ROM reductions

1. Functions removed: _DisplayStart-upStageNumber() and _CuCheckForLoopReversal().
2. BT8953 Prototype support was removed from the software.

5.2.3 Combined Calls to _Reset_Pid_Validation() in ASM.

Added single call to _Reset_Pid_Validation() within DEACTIVATED_STATE. Removed 2 separate calls to _Reset_Pid_Validation() from the transitions from PENDING_DEACTIVATED_STATE and PID_VALIDATION_STATE to DEACTIVATED_STATE.

5.2.4 Single Loop multi-rate

For 2-wire (Single Loop) applications, a set of 5 API commands were added to allow the user to modify the HDSL rate between 208 and 2320kbps in (Nx64+16)kbps. The PCM can vary either 1544kbps or 2048kbps.

This feature requires that the SP_API compiler directive to be defined. There is no requirement to define the 1T1 or 1E1 compiler directives.

5.2.5 DPLL Status API

This returns the state of the current DPLL status. Details are below in the Read the DPLL State.

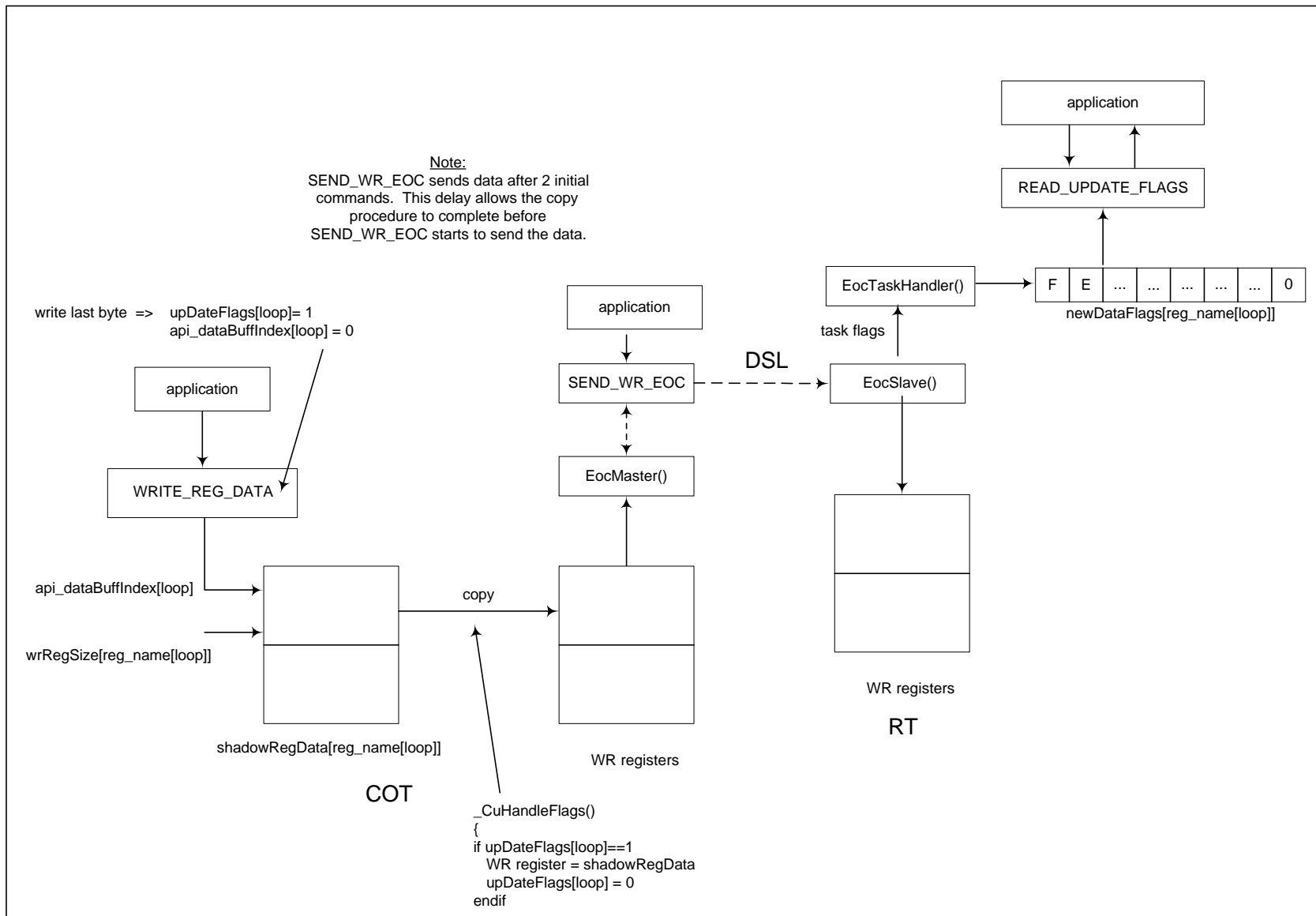


Figure 1. EOC Write Register Processing

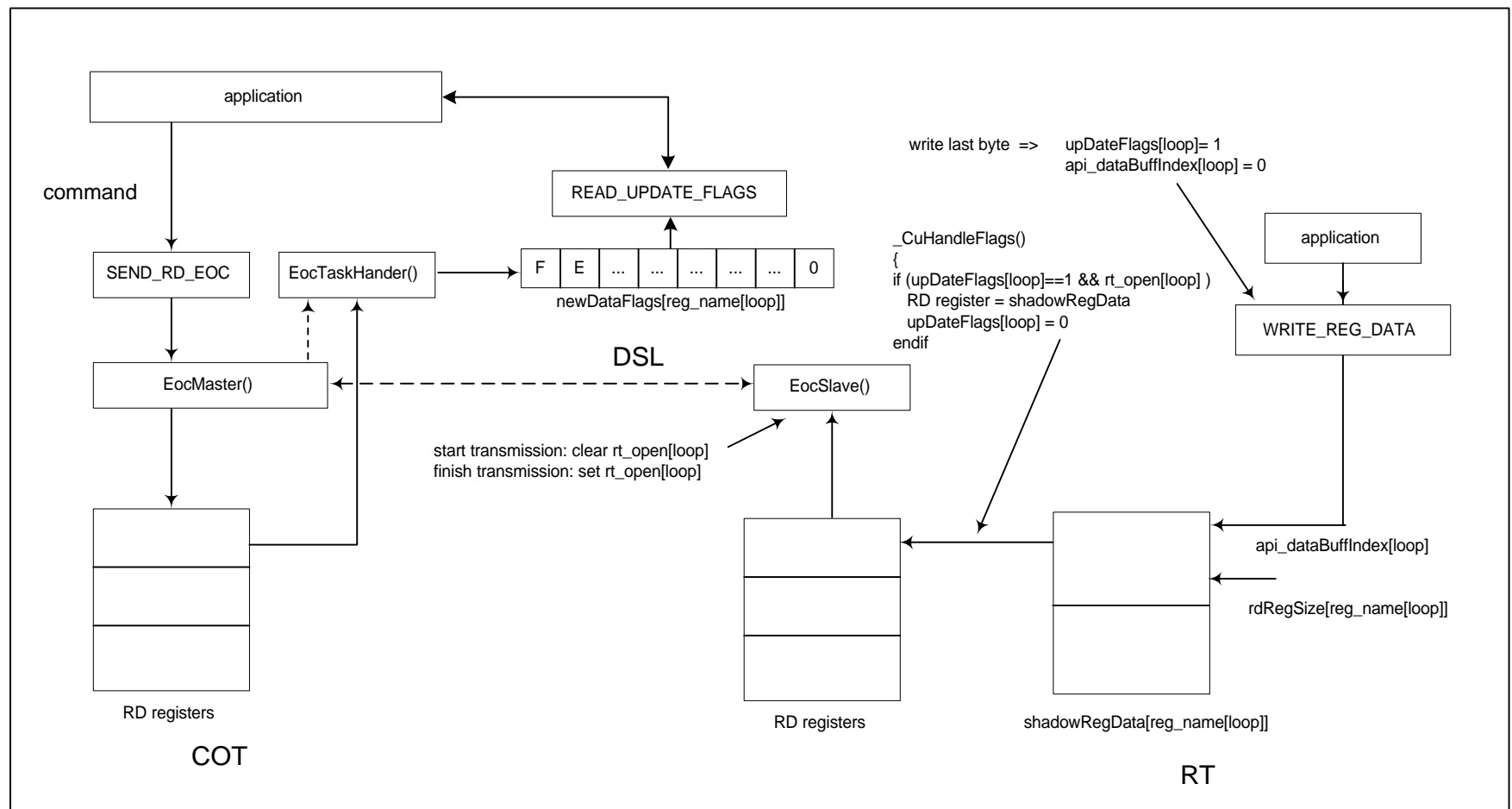


Figure 2. EOC Read Register Processing

5.2.6 Repeater Support

The software now supports a regenerator configuration that is compliant with the ETSI standards. This document and ZipWire software use the ANSI term “repeater” rather than the ETSI term “regenerator”. The terminal devices are referred to as “RegC” (regenerator configured as a COT) and “RegR” (regenerator configured as a RT). Because the PCM data is passed through the repeater, the DPLL in the channel unit does not need to be programmed or monitored.

To use the repeater software, the following compiler options must be defined:

REPEATER, CHAN_UNIT, HTUR, HTUC

In *typedefs.h*, the software defines and un-defines other directives based on the REPEATER directive. The repeater software defines CU_2E1 as the default configuration. The software also works for the E1 configuration, however, ROM constraints require that only one configuration be chosen at a time.

The DSL Loop Handler function, `_DSLLoopHandler()`, is not called in the repeater software for the repeater mode. The current implementation assumes that a single micro-controller controls a maximum of 2 bit-pumps.

5.2.6.1 Repeater Terminal Type

A repeater application requires a hardware work-around described in the latest RS8953B Data Sheet (#N8953BDSB), Appendix B. The hardware work-around requires that one bit-pump be hard-wired as a RegR and the other as the RegC. For the software to support the hardware, it likewise hard-codes the terminal types. The software requires that bit-pump 0 is the RegR and that bit-pump 1 is the RegC.

For the repeater application, the software checks the “bp” value rather than `htu_index` or `terminal_type` when configuring the RegR and RegC. For example, the descrambler and scrambler taps on the RS8953B are set based on the “bp” value.

5.2.6.2 Repeater Activation State Manager Implementation.

Diagrams of the activation state managers (ASMs) for the COT, RegR, RegC, and RT are given in Figure 3 through Figure 6. Specific changes to the ASMs to support the repeater application are now described.

The repeater passes through the Z-bits. Therefore, states related to E1 pair identification (PID) processing are not required in the repeater ASM.

The RegC uses the recovered HDSL clock from the RegR. Therefore, the COT-to-RegR link must be started first followed by the RegC-to-RT link. This requirement is implemented with these changes:

1. The RegC is placed into the `SYSTEM_IDLE` state at power-up.
2. The RegR places the RegC into `INACTIVE_STATE` when the RegR enters `GOTO_ACTIVE_RX_TX_STATE`.

Link failures on either link causes a deactivation of the other link. This is required by the HDSL specifications and implemented both in hardware and software. If the RegR or RegC enters the `DEACTIVATED_STATE`, the RegC is placed into `SYSTEM_IDLE` by calling the `SetLoopIdle()` function.

Further details of the start-up sequence for the repeater application are shown in Figure 7. The repeater application implements the normal PID negotiation between the COT and the RT. To implement this negotiation, the RegR sends an invalid PID to the COT. The COT is held in the `PID_VALIDATION_STATE` until the RegC-to-RT has been completely activated and the RegC activates the repeater mode in the RS8953B device. The RT then performs normal PID validation and echoes a valid PID back to the repeater. The repeater passes through the Z-bits and the COT then receives the valid PID bits. The activation timer within the COT is re-started to allow additional time for the delayed PID validation process.

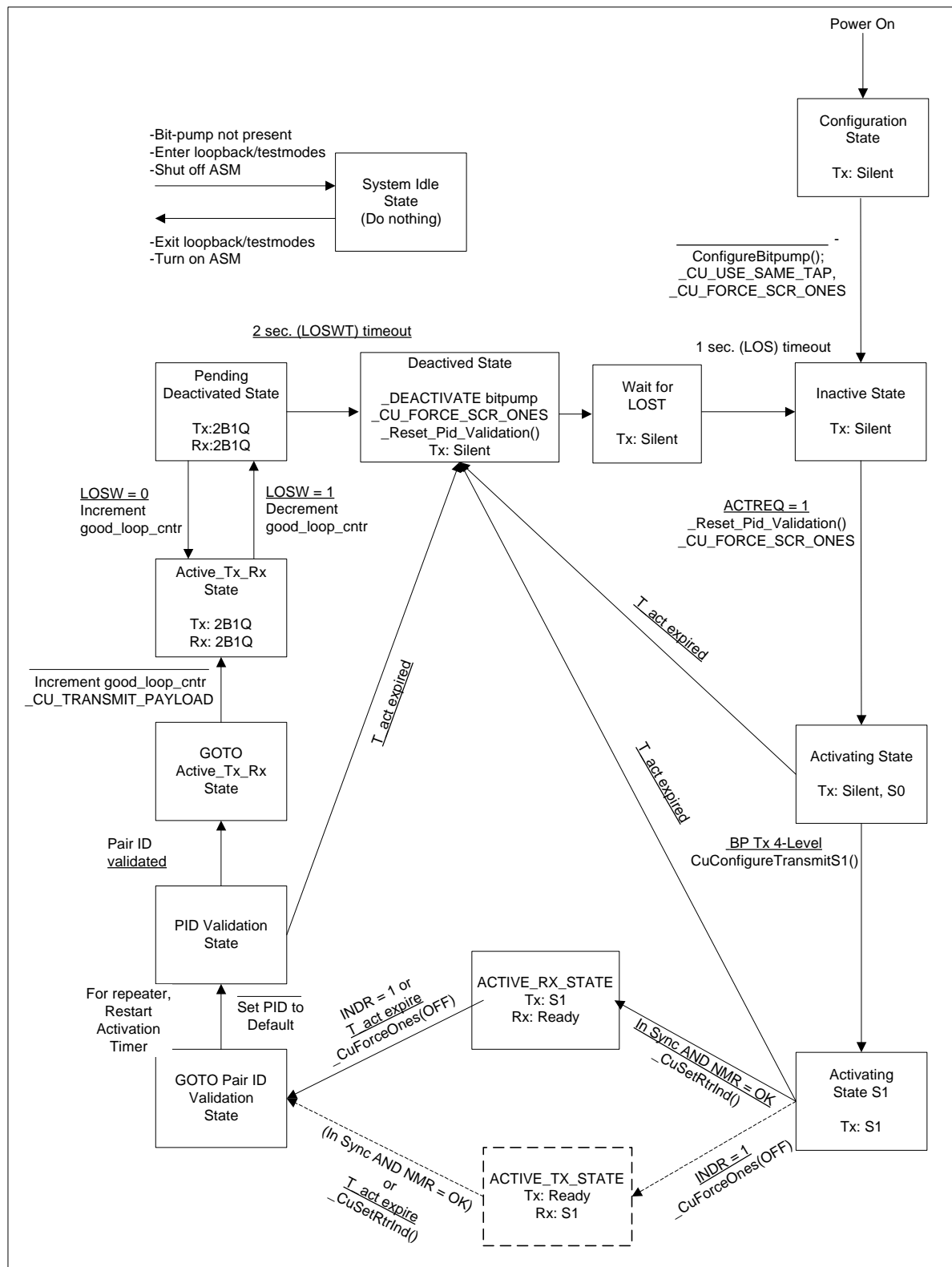


Figure 3. Activation State Manager – COT

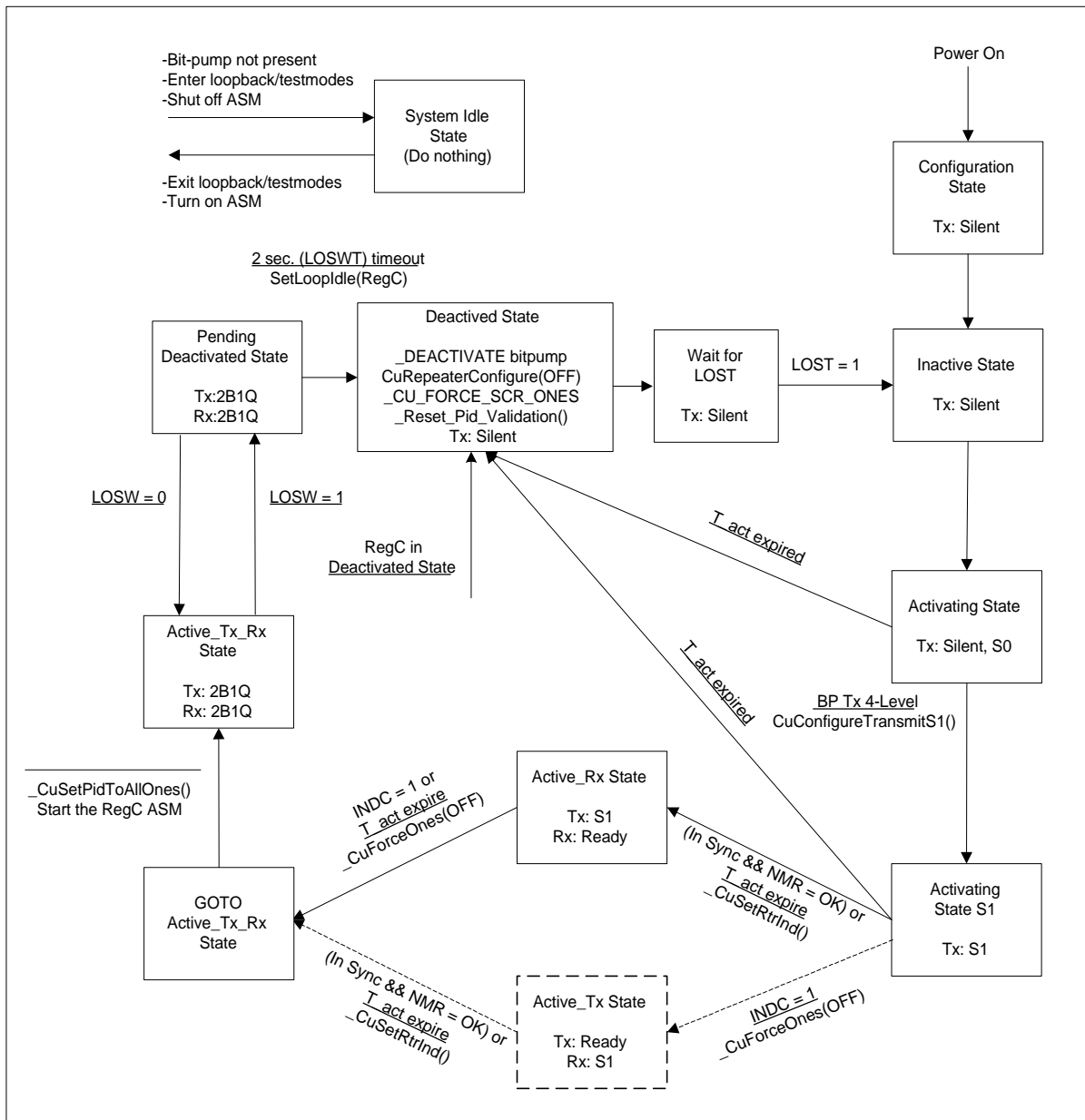


Figure 4. Activation State Manager – RegR (bp == 0)

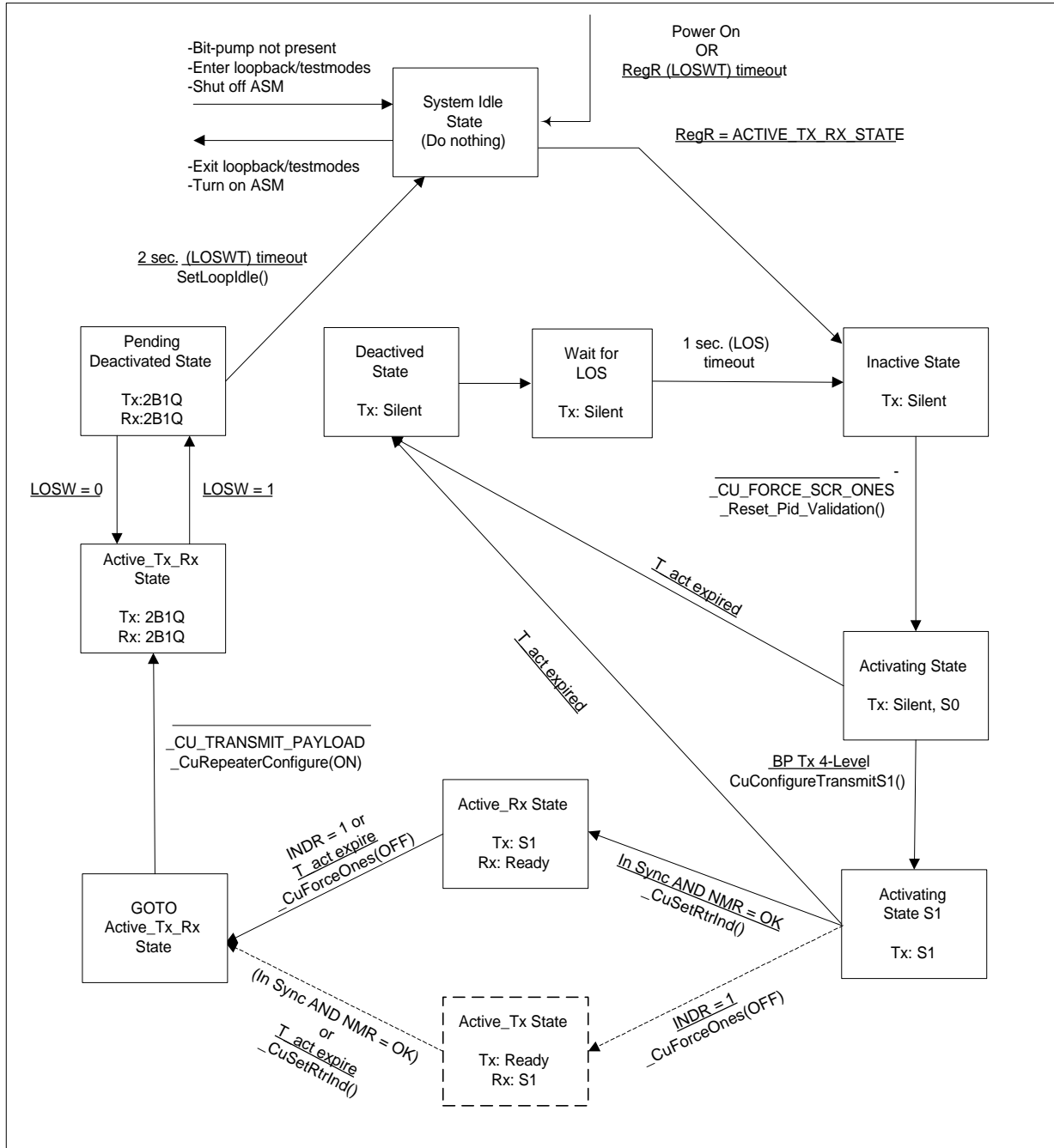


Figure 5. Activation State Manager – RegC (bp == 1)

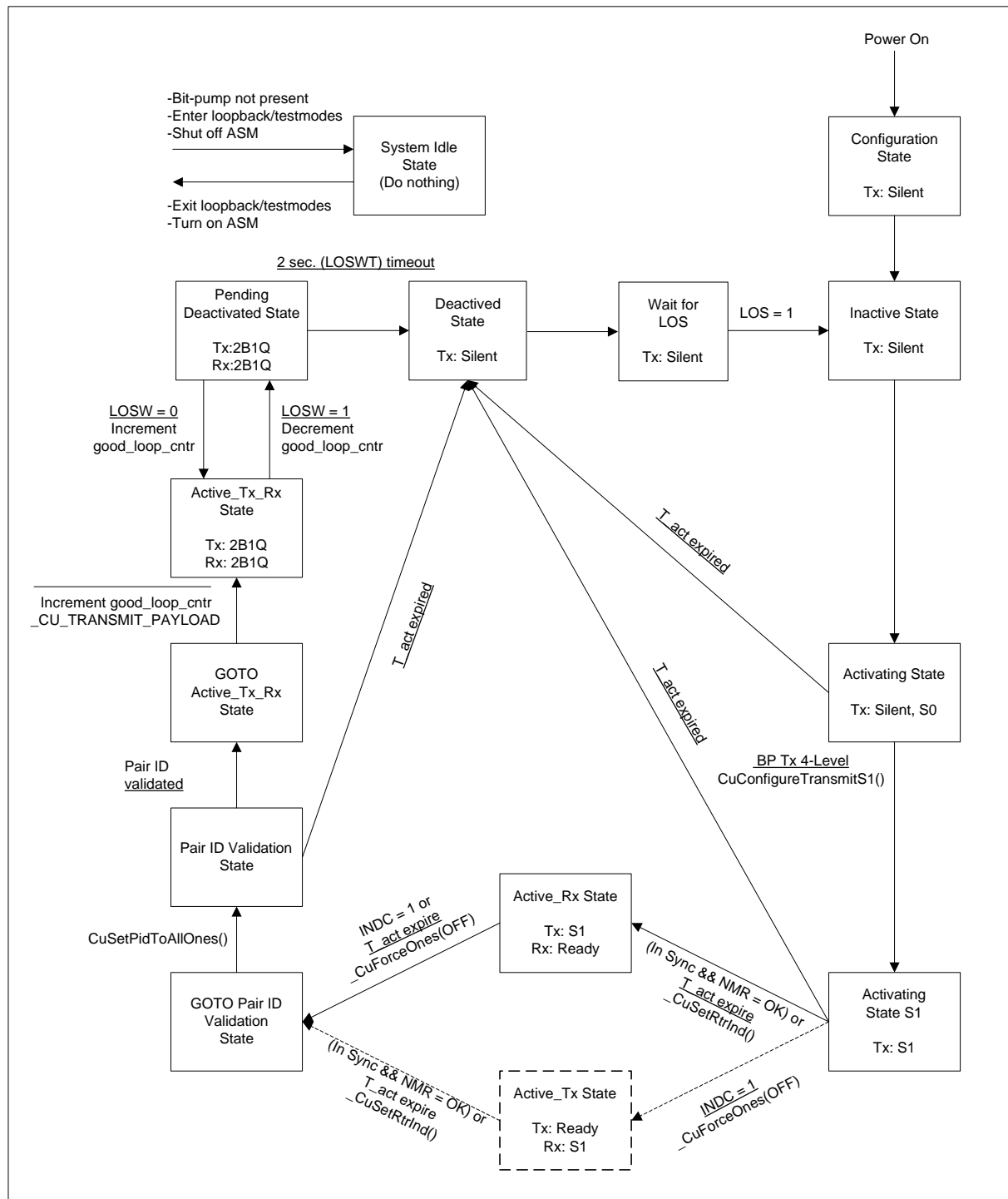


Figure 6. Activation State Manager – RT



5.2.6.3 Repeater EOC processing

EOC processing by the repeater is shown in Figure 8. This implementation follows the ETSI specifications. The repeater always passes through the EOC stream received from the COT to the RT. The repeater checks whether the destination of the received EOC stream matches its address of 0x01. If it matches, then the repeater processes the EOC stream and takes any action as required. Normally, the repeater also passes through the EOC stream received from the RT. However, while it is processing EOC information, it temporarily replaces the EOC stream from the RT with its own EOC output. This software processing is shown conceptually as a MUX in Figure 8. The RT sends the “Hold State” message if it receives a message that is not addressed to it.

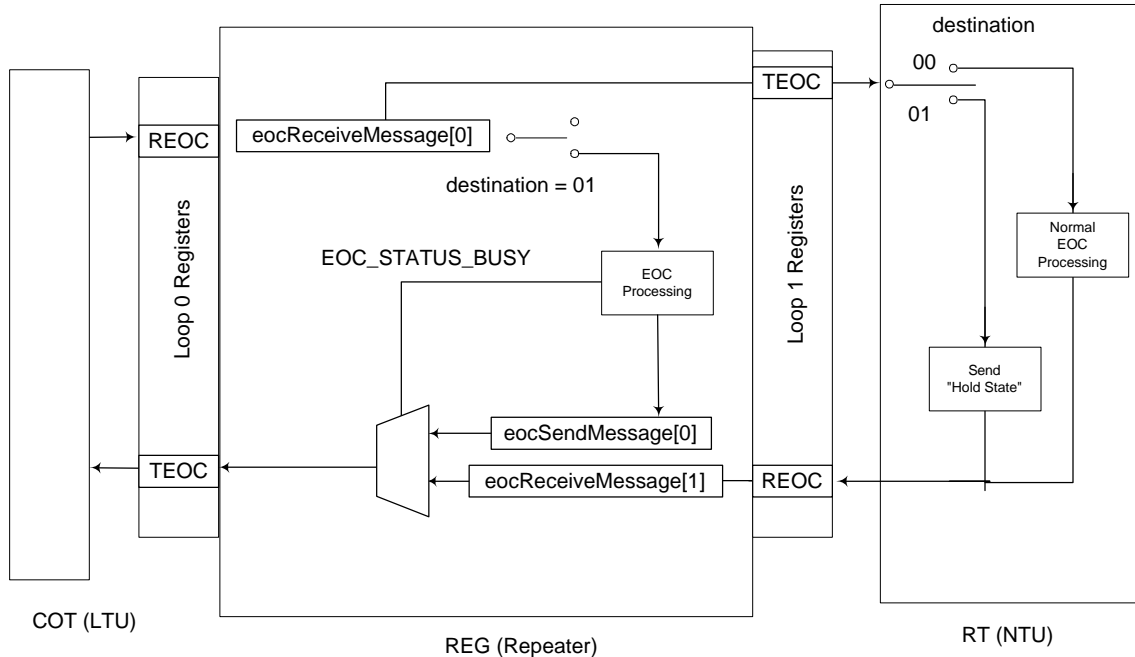


Figure 8. Repeater EOC Processing

5.2.6.4 Repeater DSL overhead processing

The RS8953B device passes through the Z-bits automatically in the hardware. The repeater software is responsible for the extraction and processing of the Indicator and EOC bits. The repeater software clears the hrp (HDSL repeater present) bit. If the RegR detects a CRC error, then it clears the rrbe bit to both the COT and RT. If the RegC detects a CRC error, then it clears the rcbe bit to both the COT and RT. The repeater extracts the febe and losd bits and only passes them through to the other link.

5.2.6.5 Repeater FIFO error processing

For the repeater, TFIFO and RFIFO errors are not checked in RxInterrupt() and TxInterrupt(). The PCM data is passed through and the FIFOs and DPLL within the RS8953B are not used.

5.2.6.6 Clock polarity interface between RS8973 and RS8953B for Repeater

The Repeater hardware work-around requires that TQ[1,0] be updated on the falling edge of TBCLK and that RQ[1,0] be sampled on the falling edge of RBCLK. The _PARALLEL_SLAVE API command was modified to set the tbclk_pol and rbclk_pol bits in the RS8973 cu_interface_modes register (0x06) only if REPEATER is defined.

5.3 API Details

All the following commands have been added in version CU61. All other API commands are functionally equivalent to the version CU60 code.

5.3.1 Miscellaneous Commands

5.3.1.1 Read the DPLL State

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

Opcode: 0x90 (_CU_READ_DPLL)		
Destination	Description	Parameter
_CU_COMMON	Reads the DPLL state.	0x00

Return Value

DPLL State 0 – Stable.
 1 – Minimum Gain setting
 2 – Medium Gain setting
 3 – Maximum Gain setting

The DPLL State numbers were modified from the CU6.0 release. Previously, the maximum gain was 1 and the minimum gain was 3. For more information see DPLL Error Handling in the Application and Channel Unit Software Programmer's Guide.

5.3.1.2 Insert CRC Errors

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

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.

5.3.2 Channel Unit Indicator Bit Commands

The following 4 API commands allow access to the 13 channel unit indicator bits.

5.3.2.1 Write Indicator Lo Byte

This command writes the first 8 indicators bits in the channel unit. These bits are sent to the far-end every 6 ms HDSL frame. Since the EVM code uses Losd, Febe, Rrbe and Rcbe, interrupt handler overwrites them.

Opcode 0x35 (_CU_WRITE_IND_LO)		
Destination	Description	Parameter
_CU_CHAN0-2	Returns indicator bits 0-7.	User defined

5.3.2.2 Write Indicator Hi Byte

This command writes the upper 5 indicators bits in the channel unit. These bits are sent to the far-end every 6 ms HDSL frame.

Opcode 0x36 (_CU_WRITE_IND_HI)		
Destination	Description	Parameter
_CU_CHAN0-2	Returns indicator bits 8-13.	User defined

5.3.2.3 Read Indicator Lo Byte

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

Opcode 0x91 (_CU_READ_IND_LO)		
Destination	Description	Parameter
_CU_CHAN0-2	Returns indicator bits 0-7.	0x00

Low Byte Return Status Bit Definitions

Definition	Hex-Code	Opcode Description
Losd:1	0x01	If no signal from application, LOSD set to 0.
Febe:1	0x02	Far-end bit error sets bit to 0.
Ps1:1	0x04	Primary power supply (unused)
Ps2:1	0x08	Secondary power supply (unused)
Bpv:1	0x10	Bipolar violation (unused)
Hrp:1	0x20	Regenerator present
Rrbe:1	0x40	Regenerator remote block error
Rcbe:1	0x80	Regenerator central block error

5.3.2.4 Read Indicator Hi Byte

This command reads the upper 5 indicators bits in the channel unit. These bits are received from the far-end every 6 ms HDSL frame. In bits 13-15 a channel unit manufacturing code is returned for each loop. The manufacturing code returns a 0x2 for loop 2, and returns 0x0 for loops 1 and 3.

Opcode 0x92 (_CU_READ_IND_HI)		
Destination	Description	Parameter
_CU_CHAN0-2	Returns indicator bits 8-15.	0x00

High Byte Return Status Bit Definitions

Definition	Hex-Code	Opcode Description
Rega:1	0x01	Remote Terminal Alarm (unused)
Rta:1	0x02	Remote alarm indicator (unused)
Rtr:1	0x04	Ready to Receive indicator
Unspecified:2	0x18	Reserved for future use by ETSI(unused)
Reserved:3	0xE0	Manufacture code

5.3.3 Single Loop Commands

The following five commands apply to single loop applications only. These commands allow the user to set up the system for any HDSL data rate between 208kbps and 2320kbps in ((Nx64)+16)kbps steps. These commands are only used with the RS8973 and RS8953 products. For information on how to use these commands with the 8970, contact local technical support.

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

5.3.3.1 Set Total Number of PCM Time Slots

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

$(\text{Parameter} * 64 \text{ kbps}) = \text{PCM Data rate.}$

Example: parameter of 24; $(32 * 64 \text{ kbps}) = 2048 \text{ kbps.}$

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)

5.3.3.2 Set Total Number of HDSL Payload Bytes

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

$((\text{Parameter} * 64) + 16) \text{ kbps} = \text{HDSL Data rate.}$

Example: parameter of 4; $((4 * 64) + 16) \text{ kbps} = 272 \text{ kbps.}$

Opcode: 0x41 (_SP_TOTAL_HDSL_TSLOT)		
Destination	Description	Parameter
_DSL_APPLICATION	Total number of HDSL payload bytes to send over the HDSL.	3-36

5.3.3.3 Set the Number of Occupied HDSL Payload Bytes and PCM Timeslots 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, then payload bytes 0,1,2,3 and 4 are used for payload.

This number cannot be greater than the set “Total Number of HDSL Payload Bytes” or “Total Number of PCM Timeslots”.

Opcode: 0x42 (_SP_USED_TSLOT)		
Destination	Description	Parameter
_DSL_APPLICATION	Total number of payload channels used.	3-36

5.3.3.4 Set F-bit Present

This command selects if the F-bit is present or not. When PCM 1544kbps is selected and no Fbit is inserted, the payload bandwidth is 1536kbps. With the current 8973/8953B EVM application, the Fbit choose which T1\E1 framer type is selected.

Opcode: 0x43 (_SP_FBIT_PRESENT)		
Destination	Description	Parameter
_DSL_APPLICATION	Sets the F-bit to be present.	0-Not Present 1-Present

5.3.3.5 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 goes through the following sequence of events:

- IDLE State: The transceiver is set to idle.
- Configure: The system is re-configured for the new data rate for the destination loop
- Restarts: Performs start-up process at the new data rate.

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

Opcode: 0x45 (_SP_CONFIGURE)		
Destination	Description	Parameter
_DSL_CHANNEL0-2	Configures channel unit and bit-pump and restarts system.	0x00

5.3.4 EOC Commands

There are two software levels in the EOC. The higher level API modifies the EOC register contents and control when messages are transferred. The lower level drivers are responsible for moving the data back and forth through the channel and error checking. The user doesn't need access to the lower level drivers to utilize the EOC channel.

The following 10 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. It is left to the customer to add any Vendor Defined specifics from the HDSL standard. All reserved bits should be set to 0.

5.3.4.1 EOC Register Select

This command selects an EOC register name and the 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.

For alternate feature registers such as the loop back, insert CRC, etc. See Set EOC Control Commands.

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	Write 0/ Read 1	Register Name

5.3.4.2 EOC Register Size

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

Example: “write register A” could have a register size of 16 while “read register A” has a register size of 1. All sizes are defaulted 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).

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 Size up to 16

Sets the rd/wrRegSize[] global variable.

5.3.4.3 EOC Byte Number Location

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

NOTE: When writing or reading from the first byte, the Byte Number Location is 0 and the last byte is _EOC_REG_SIZE minus 1.

Opcode: 0x37 (_EOC_BYTE_NUM_LOC)		
Destination	Description	Parameter
_DSL_CHANNEL0-2	Selects which byte of the register to modify.	0–rd/wrRegSize[] –1

Sets the api_dataBuffIndex[] global variable.

5.3.4.4 EOC Write Register Data

This command writes the 8-bit parameter value into the wrRegData or rdRegData arrays. 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 step was added to make it impossible to corrupt the EOC data registers while sending the EOC data.

The HTU-C (Master) only writes to the wrRegData[] (control) with the parameter information to be sent to the HTU-R (Slave). The HTU-R only writes to the rdRegData[] (status) with the parameter that returns to the HTU-C.

The rdRegData[] on the HTU-R needs to be loaded before the HTU-C can read it. This eliminates latency by the host processor to update the HTU-R's rdRegData[] after the EOC read command from the HTU-C.

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

Opcode: 0x38 (_EOC_WRITE_REG_DATA)		
Destination	Description	Parameter
_DSL_CHANNEL0-2	Writes an 8-bit value to the selected register.	0 – 0xFF

5.3.4.5 Start EOC Read/Write Operation

This command starts a read or write process. The HTU-C is the only side that can issue this command. When performing a write command the HTU-C sends over its wrRegData[]. If performing a read (status) command, the HTU-C will request HTU-R to return 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.

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 0/ Read 1	Register Name

5.3.4.6 Set EOC Control Commands

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

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

Definition	Hex-Code	Opcode Description
EOC_CMD_RTN	0x07	Disables all hold states (loopbacks).
EOC_CMD_LOOP_NTU	0x08	Loop back in channel unit HDSL to PCM.
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 to it.
EOC_CMD_REGC_CCRC_IND	0x50	Notify RegC CRC6 is being sent to it.
EOC_CMD_NTU_CRC_OK	0x5F	Notify RegR CRC6 is not being sent to it.
EOC_CMD_REGC_CRC_OK	0x60	Notify RegC CRC6 is not being sent to it.

Figure 9 shows a list of the HDSL reference sub-paragraph numbers, hex codes, and diagrams of the operation of each command. The sub-paragraph numbers appear in section 5.5.5 in any of the HDSL specifications (ETR-152, ETSI TS-101-135 or ITU G.991.1).

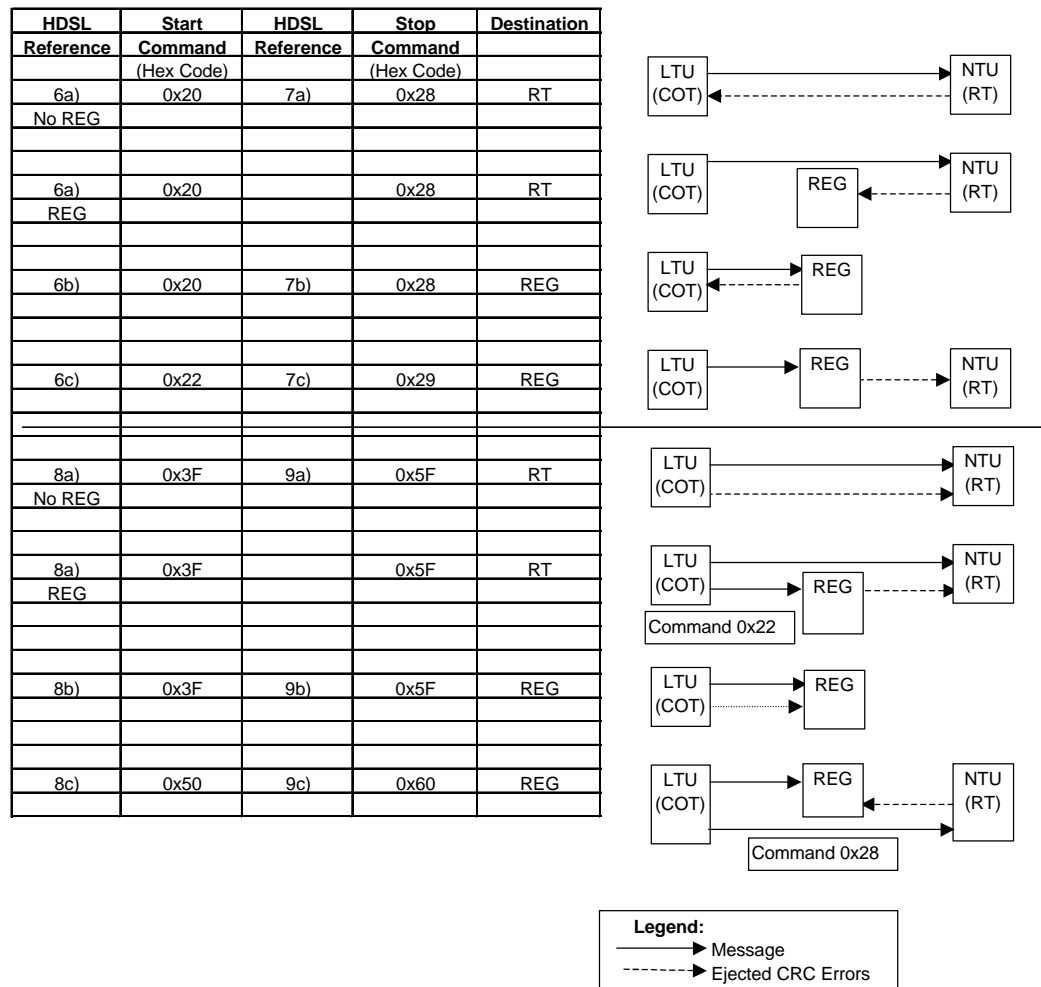


Figure 9. Cross Reference of HDSL Specification versus Hex Codes

5.3.4.7 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.

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.

5.3.4.8 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 command is sent and before another EOC Read command of the same register is sent.

For the HTU-R, the processor checks for any updated wrRegData from the HTU-C by this API.

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

The register is cleared after _EOC_RCVD_NEWDATA_STATUS is queried.

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
Hi 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 the lower 8-bits of the variable then clears it. The parameter of 0x01 returns the upper 8-bits of the 16-bit variable then clears it.

5.3.4.9 EOC Read Data Register

This command is used to extract the contents of the EOC buffer. The HTU-C will return the contents of the rdRegData[] buffer while the HTU-R will return the contents of the wrRegData[] buffer.

This command requires the parameter to contain the selected rd/wrRegister. The command increments the byte-number location after every read. The system has to repeatedly call this command for the same number of times as the rd/wrRegSize[]. If the byte-number location is greater than the rd/wrRegSize[], then this command will return 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.

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	Write 0/ Read 1	Register Name

5.3.4.10 Read EOC Status

This command is used to query the low level EOC protocol status.

Opcode: 0x88 (_EOC_STATUS)		
Destination	Description	Return Value
_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

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

6 EOC Application Examples

API commands have three parts: destination, opcode and parameter. Destinations for the following examples use _DSL_CHANNEL0 (0xF9).

6.1 Read Example

This section provides EOC read examples. Example steps 1-8 show how:

- To select HTU-C and HTU-R Register D
- To select HTU-C and HTU-R Register size for register D
- To load the register D value into the HTU-R.
- The HTU-C reads the HTU-R's register D value.

6.1.1 Step 1: Select HTU-C Register Name.

After the EOC available flag is set and start-up is complete the EOC channel is ready. This example sets the HTU-C register D. Register D is the EOC ETSI standard for (*Device ID*). It is left to the vendor to define the Device ID value. The HTU-C will select the register to modify by using the EOC Register Select API.

0xF9-DSL_CHANNEL0.

0x35-EOC Register Select

0x1D-The 1 is for read and D is for Register D.

6.1.2 Step 2: Set HTU-C Register Size.

Select the register size. For this example set the HTU-C register D (*Device ID*) to be 8 bytes long.

0xF9-DSL_CHANNEL0.

0x36-EOC Register Size

0x08-Read Register D is 8 bytes

6.1.3 Step 3: Set up HTU-R Register Number and Size.

Repeat steps 1 and 2 except on the HTU-R

6.1.4 Step 4: Load the 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: Since the read register D was just modified for size, the EOC Register Select doesn't need to be selected again.

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. 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 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.

Following is the API to write the data into the “read register D”.

0xF9-DSL_CHANNEL0.

0x38-EOC Write Register Data.

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

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

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

The HTU-C issues two commands to read this value. The first is Start EOC Read/Write Operation

0xF9-DSL_CHANNEL0.

0x39-Start EOC Read/Write Operation

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

This command will make the HTU-R send back all 8 bytes of “read register D”. These were entered in Step 4: Load the HTU-R Read Register D.

6.1.6 Step 6: Read the New Data Flags.

After the read command is sent, the HTU-C has to wait a variable amount of time 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 one, 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.

6.1.7 Step 7: Set Index to 0 for Read Register D.

For the HTU-C to query the value from the first byte, the user must use the following API.

0xF9-DSL_CHANNEL0.

0x37-Byte Number Location.

0x00-Byte 0 will be read next time.

6.1.8 Step 8: Read the Read Register D.

This command allows the HTU-C to read the returned value with the EOC Read Data Register.

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. Executing the same API again the byte-number location increments and the next read register D value is returned.

6.2 Write Example

This section provides a write EOC example and how they are implemented. The example steps 1-8 show how:

- To set-up the write register size for register B.
- To load the register value.
- To send the register.
- The HTU-R handles the command.

For the following examples API commands are used. API commands have three parts: destination, opcode and parameter. Destinations for the following examples use `_DSL_CHANNEL0` (0xF9).

6.2.1 Step 1: Set up HTU-C Register Number.

After EOC available flag is set and start-up is complete the EOC channel is ready. For this example set the HTU-C Write Register B (*NTU Configuration*). The HTU-C will select 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.

6.2.2 Step 2: Set up HTU-C Register Size.

Select the register size. For this example set the HTU-C Write Register B (*NTU Configuration*) to be 10 bytes long.

0xF9-DSL_CHANNEL0.

0x36-EOC Register Size.

0x0A-Set "write register B" to 10 bytes

6.2.3 Step 3: Set up HTU-R Register Number and Size.

Repeat steps 1 and 2 except on the HTU-R

6.2.4 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: Since the Write Register B was just modified for size, the EOC Register Select doesn't need to be set again.

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. After this 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 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 will be the first byte of the NTU-Configuration.

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

6.2.5 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.

6.2.6 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 (8-0xF).

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.

6.2.7 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.

6.2.8 Step 8: Read the B Data Register.

To read the byte at the byte location issue the EOC Read Data 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, the first data value written to the HTU-R in Step 4: Load the HTU-C Write Register B.

6.3 HTU-C CRC Check Command Example

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

For the following steps DSL_CHANNEL0 is used with a destination value of 0xF9.

6.3.1 Step 1: HTU-C receives Corrupted CRC from HTU-R.

To verify the HTU-C CRC detector unit is functioning. The HTU-C requests the HTU-R return 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 (0x28) is sent. See the *ETSI TS 152 Edition 4* under HDSL EOC opcode messages for more information.

The user monitors the HTU-C's CRC-6 errors counter to verify CRC detector functionality.

6.3.2 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.

6.4 HTU-R CRC Check Command Example

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

6.4.1 Step 1: Notify the HTU-R of Corrupted CRC.

The HTU-C can notify the HTU-R that it will be sending over 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 over corrupted CRC-6, disable any alarms.

6.4.2 Step 2: Send over Corrupted CRC.

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

0xF9-DSL_CHANNEL0.

0x3C-Insert CRC Errors

0x01-Enable CRC-6 injection.

To disable the insert CRC-6, change the Step 2 API parameter to a 0.