CN8980

ZipWire2 HDSL2/SDSL Transceiver and Framer

The Conexant ZipWire2 chip set is a DSL transceiver which provides enhanced performance and better maximum reach at a given data rate than existing symmetric transport systems based on 2B1Q or CAP modulation. ZipWire2 is designed to be fully compliant with the OPTIS-based ANSI standard for HDSL2 T1 transport and meets all the current requirements of the emerging ETSI standards for SDSL E1 transport. In particular, flexible control of the transmitted signal power spectral density results in enhanced spectral compatibility with other services such as ADSL, T1, E1, HDSL, and ISDN.

The ZipWire2 chip set goes beyond providing modems for T1 or E1 transport by offering on-chip circuits to facilitate variable data rate operation. These circuits allow the user to trade off data rate for reach performance. In addition, ZipWire2 devices provide a mode of operation supporting legacy HDSL1 (2B1Q) transport and framing so that system OEMs can offer Central Office (CO) equipment capable of operation with 2B1Q-based (e.g., RS8973/8953B) remote terminals. These CO terminals can be later upgraded to OPTIS-based HDSL2 through software modifications.

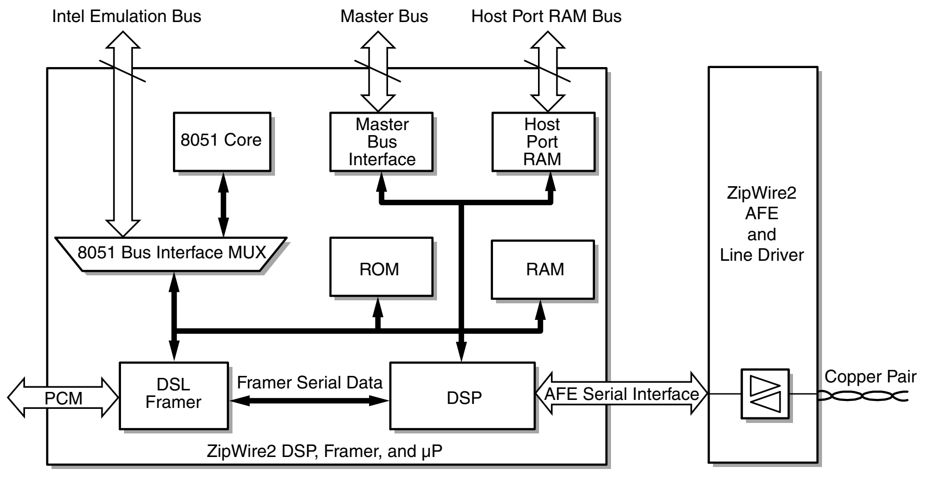
The ZipWire2 device has a two- or three-chip architecture: analog-front-end and line driver in one chip, and all digital functions in one or two other chips. Major functional blocks in the digital chip include:

- A bypassable framer/mapper function
- A rate-adaptive bit pump DSP with Trellis coding
- A high performance microprocessor core

The framer/mapper features a highly flexible bit-processing engine capable of almost any frame format. In particular, it supports the ANSI HDSL2 and ETSI HDSL1 frame formats. It performs EOC, overhead and payload insertion and extraction, data scrambling, bit stuffing, and sync detection. The framer supports T1/E1 Primary Rate framed or unframed modes, synchronous or asynchronous payload mapping, and per-time-slot random or fixed data insertion. In addition, it has programmable external time slot add/drop controls, bit error metering, and programmable payload mapping which supports 1-, 2-, 4- or 8-bit time slots.

The rate-adaptive DSP is responsible for echo cancellation, line equalization, and data coding. It is capable of 2-, 4-, 8- and 16-PAM coding and contains an integrated software-controlled clock recovery and synthesis function. The on-chip 8051-compatible microprocessor core provides DSP control and sequencing, but can also be used as a general purpose controller for peripheral components such as codecs or T1/E1 framers and to host network management software. The functional block diagram illustrates an overview of the ZipWire2 architecture.

Functional Block Diagram



Distinguishing Features

- A highly integrated solution including framer, controller, DSP, AFE, and line driver in two or three small packages.
- Fully compliant with the OPTIS-based ANSI standard for HDSL2 T1 transport and ITU G.shdsl transport.
- An integrated, bypassable, HDSL2 framer/mapper function. The bypass function provides direct access to the DSP interface.
- An on-chip, high performance µP core which provides the internal DSP bit pump sequencing and control. This core can also be used as a general purpose controller for peripheral components, such as codecs or T1/E1 framers, and to host network management software.
- Data rates between 144 kbps and 4,640 kbps. The data rate is software-selectable via an on-board software programmable clock synthesizer.
- The ability to function as a 2B1Q transceiver and interoperate with legacy HDSL 2B1Q terminals.
- A device architecture that supports multichannel line cards through the following features:
  - Single boot ROM loading.
  - Fully autonomous startup sequencing per channel.
  - A high-speed PCM interface that allows a maximum of eight devices which can share a common PCM bus.
  - Integrated framer for arbitrary time slot assignments per channel.
  - Point-to-Multipoint support with signaling channel grooming.

Applications

- T1 or E1 transport systems
- Variable data rate access systems
- Internet connectivity
- Voice and/or data pair gain systems
- NX64 data transport
- Cellular base station data links
- Campus modems

Ordering Information

Model Number	Package	Ambient Temperature Range
DSNP-L300-001	80-pin TQFP 27 mm × 27 mm BGA	0 °C to + 70 °C
DSNP-L301-001	80-pin TQFP 27 mm × 27 mm BGA	–40 °C to + 85 °C
DSNP-L301-011	80-pin TQFP 9 × 9 mm CABGA 15 × 15 mm CABGA	–40 °C to + 85 °C

Revision History

Revision	Level	Date	Description
A	Advanced	April 1999	Created
B	Advanced	April 1999	—
C	Advanced	April 2000	—

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 xiii

List of Tables xvii

1.0 Introduction 1-1

1.1 References 1-1

2.0 System Overview 2-1

2.1 ZipWire2 Transceiver/Framer Functional Summary 2-2

2.2 ZipWire2 Transceiver/DSP Functional Summary 2-4

2.3 ZipWire2 DSL Framer Functional Summary 2-5

2.4 ZipWire2 AFE Functional Summary 2-7

2.5 ZipWire2 Transmit Path 2-8

2.6 ZipWire2 Receive Path 2-9

3.0 Application Interfaces 3-1

3.1 Using Internal 8051 Processor Only 3-1

3.2 Using an External Embedded Host Processor 3-3

3.3 Multi-Pair DSL Framer Configuration (Cascade Mode) 3-6

3.4 ZipWire2 Transceiver/Framer to Bt8370 T1/E1 Interface 3-9

3.5 DSL Framer to CN8228 (ATM Phy) Interface 3-10

3.6 DSL Framer Bypass to CN8228 (ATM Phy) Interface 3-11

3.7 Dual Mode CN8228 (ATM Phy) Interface 3-12

3.8 DSL Framer to MUSYCC Interface 3-13

3.9 DSL Framer Bypass to MUSYCC Interface 3-14

3.10 Dual Mode MUSYCC Interface 3-15

3.11 Multiple ZipWire2 Devices to CN8228 or MUSYCC Interface 3-16

3.12 Deliverables 3-18

3.12.1 Customers who do not wish to modify 8051 code 3-18

3.12.2 Customers who wish to modify 8051 code to control other devices 3-18

3.12.3 Customers who wish to modify low-level DSL Framer code 3-18

4.0 Built-In 8051 Core Detailed Description 4-1

4.1 Internal 8051 Data Space Memory Map 4-2

4.2 Internal 8051 Interrupt Mapping 4-3

4.3 ZipWire2 Transceiver Function Registers 4-4

4.4 Configuration Pins 4-4

4.4.1 8051 Timer/Counter Description 4-5

4.5 Internal 8051 Communication Interfaces. 4-6

4.5.1 Master Bus Microprocessor Interface 4-6

4.5.2 Host Port RAM Interface. 4-6

4.5.3 RS232 Serial Interface 4-7

4.5.4 Group Talk Interface 4-7

4.6 Program RAM Download 4-8

4.6.1 Summary. 4-8

4.6.2 Download Description 4-9

4.6.3 Dip Switch #2—DEVADR and BOOTOP Pins 4-9

4.6.4 Download Protocol Overview 4-10

4.6.5 Download Times 4-10

4.6.5.1 Host Port RAM Interface 4-10

4.6.5.2 Group Talk Interface 4-10

4.6.6 Download and Device Validation 4-11

4.6.7 Download Group Master Device 4-11

4.6.8 Download Group Slave Device 4-11

4.6.9 Download Single Processor Configuration. 4-12

4.6.10 Download Dual Processor Configuration 4-13

4.6.10.1 Redundant Download Group Master 4-14

4.6.10.2 Device Uniqueness 4-14

4.6.10.3 Group Uniqueness 4-14

4.6.11 Download API Commands 4-15

4.6.11.1 Boot Code API Commands 4-15

4.6.11.2 Operational API Commands. 4-15

4.6.11.3 Program RAM Checksum 4-15

4.6.12 Download Examples 4-16

5.0 ZipWire2 DSP Detailed Description 5-1

5.1 ZipWire2 Clocks 5-1

6.0 ZipWire2 Framer Detailed Description 6-1

6.1 Distinguishing Features 6-1

6.2 Common Functions 6-2

6.2.1 DATA FIFO 6-2

6.3 HDSL Section 6-2

6.3.1 General HDSL Functions. 6-2

6.3.1.1 CRC Generator 6-2

6.3.1.2 Scrambler/ Descrambler 6-3

6.3.1.3 Auxiliary Channel 6-3

6.3.1.4 RX DSL Reference Phase Measurement. 6-3

6.3.2 HDSL Receiver Functionality. 6-4

6.3.2.1 DSL Sync Detector (DSD) 6-5

6.3.2.2 Tip/Ring Reversal Detection. 6-5

6.3.2.3 RX HDSL Payload Table. 6-5

6.3.3 HDSL Transmitter Functionality. 6-6

6.3.3.1 Stuffing Generator. 6-6

6.4 PCM Section 6-7

6.4.1 PCM Interface 6-7

6.4.2 General PCM Functions 6-7

6.4.2.1 CRC Generator 6-7

6.4.2.2 Insert/Drop 6-7

6.4.2.3 Overhead Handling 6-8

6.4.2.4 E1 Grooming 6-8

6.4.2.5 MF Phase Measurement 6-8

6.4.3 PCM Receiver 6-8

6.4.4 PCM Transmitter 6-10

6.4.4.1 PCM Sync Detector. 6-10

6.5 Test and Diagnostics. 6-11

6.5.1 Performance Monitoring. 6-11

6.5.2 PRBS and BER Meter 6-11

7.0 Hardware Interfaces 7-1

7.1 ZipWire2 Transceiver/Framer to AFE Interface 7-1

7.2 Transmission Line Interface 7-2

7.2.1 Continuous Time Filter and Line Driver Control 7-3

7.2.2 Compromise Hybrid, Matching Resistors, and Transformer 7-4

7.2.2.1 Compromise Hybrid 7-5

7.2.2.2 Impedance Matching Resistors 7-5

7.2.2.3 Transformer 7-5

7.2.2.4 Anti-Alias Filters 7-5

7.2.3 Surge Protection 7-5

7.3 Voltage Reference and Compensation Circuitry 7-6

7.4 Framer Bypass Interface (ZipWire2 Transceiver DSL Interface). 7-7

7.5 Test and Diagnostic Interface (JTAG). 7-8

8.0 Pin Descriptions 8-1
8.1 ZipWire2 Pin Assignments 8-1
8.1.1 ZipWire2 Transceiver/Framer Pin Assignments 8-2
8.1.2 ZipWire2 Transceiver Pin Assignments 8-3
8.1.3 ZipWire2 Framer Pin Assignments 8-4
8.1.4 ZipWire2 AFE Pin Assignments 8-5
8.2 ZipWire2 Signal Descriptions 8-6
8.2.1 ZipWire2 Transceiver/Framer Signal Descriptions 8-6
8.2.2 ZipWire2 AFE Signal Descriptions 8-15

9.0 EVM Specific 9-1
9.1 Bt8370 E1/T1 Framer 9-2
9.2 EVM LEDs and Miscellaneous Output 9-2
9.3 EVM DIP Switches 9-3

10.0 Software Overview 10-1
10.1 Software Features 10-2

11.0 Embedded 8051 Software Features 11-1
11.1 Activating the ZipWire2 Modem 11-1
11.1.1 Activation State Manager (ASM) 11-1
11.1.2 Pre-Activation 11-2
11.1.2.1 OPTIS (HDSL2 1T1) Pre-Activation 11-2
11.1.2.2 AutoBaud 11-2
11.1.2.3 G.hs 11-2
11.1.3 DSP Training 11-3
11.1.4 DSL Line Coding 11-3
11.1.5 Frame Format 11-3
11.2 Loopbacks 11-4
11.3 BER Meters 11-6
11.3.1 DSL Framer Tx PCM BER Meter 11-6
11.3.2 DSL Framer Rx PCM BER Meter 11-6
11.3.3 Bit-Pump-Only BER Meter 11-6
11.4 Performance Monitoring (Error History) 11-7
11.5 DSL Framer Interrupt Handler 11-9
11.5.1 Sync Status 11-9
11.5.2 Error Status Reporting 11-9
11.5.3 Tx/Rx FIFO Error Handling 11-9
11.5.4 DPLL Error Handling 11-10
11.5.5 Pair ID Termination (E1 Mode) 11-11
11.5.6 Indicator Bit Termination 11-11
11.6 Dynamic Master Loop 11-12
11.7 Tip/Ring Reversal 11-12

11.8 Loop Reversal 11-12
11.9 Embedded Operation Channel (EOC) Operation 11-13
11.9.1 Feature Overview 11-13
11.9.2 Does Not Support 11-13
11.9.3 EOC General Overview 11-13
11.9.4 EOC Frame Format 11-14
11.9.5 EOC Unit Addresses 11-15
11.9.6 EOC Message IDs 11-16
11.10 EOC Implementation Details 11-18
11.10.1 EOC Transmit 11-18
11.10.2 EOC Receive 11-20
11.10.3 EOC Transaction Time 11-22
11.10.4 EOC Transaction Time-Out 11-22
11.10.5 EOC Message Control 11-22
11.10.5.1 Auto-Response To Request Message 11-22
11.10.5.2 Auto-Trigger a Request Message 11-22
11.10.5.3 Notify Host That a Message Was Received 11-23
11.10.5.4 Notify Host When Error Is Detected 11-23
11.10.6 EOC Transmit Queue 11-23
11.10.7 EOC Receive Queue 11-24
11.10.8 EOC Proprietary Messages 11-24
11.10.8.1 User-Defined Message Request – Message ID 112 11-24
11.10.8.2 User-Defined Message Response— Message ID 240 11-25
11.10.8.3 API Over EOC Request—Message ID 113 11-25
11.10.8.4 API Over EOC Response—Message ID 241 11-26
11.10.8.5 Redefining Proprietary Messages 11-26
11.10.9 EOC Application State Machine 11-26
11.10.10 EOC API Commands 11-28
11.11 T1/E1 Framer and LIU Support 11-28

12.0 Embedded 8051 Code 12-1
12.1 Boot Code State 12-3
12.2 DSL Initialization State 12-3
12.3 Out-Of-Service Check 12-3
12.4 Configure ZipWire2 State 12-3
12.5 Handle Test Mode States 12-4
12.6 DSL Reset Check 12-4
12.7 API Manager 12-4
12.8 Bit Pump Manager 12-7
12.9 DSL Framer Manager 12-7
12.10 DSL Loop Manager 12-7
12.11 HDSL2 Activation State Manager 12-8

12.12 HDSL1 Activation State Manager 12-10
12.12.1 HTU-C Activation 12-10
12.12.2 HTU-R Activation 12-12

13.0 HDSL2 Standards Compliance 13-1
13.1 Bit-Level Mapping 13-1
13.1.1 Four-Level 2B1Q Mapping (HDSL1) 13-1
13.1.2 Sixteen-Level Optis Mapping (HDSL2) 13-2

14.0 DSL Frame Structure 14-1
14.1 Bypass DSL Frame Structure 14-1
14.2 HDSL2 Configurations 14-1
14.2.1 HDSL2 _1T1 14-3
14.2.2 HDSL2 _1E1 14-3
14.2.3 HDSL2—Single Pair 14-4
14.3 HDSL1 Configurations 14-5
14.3.1 HDSL1 _2T1 14-7
14.3.2 HDSL1_2E1 14-8
14.3.3 HDSL1_3E1 14-9
14.3.4 HDSL1_1T1 14-10
14.3.5 HDSL1_1E1 14-10
14.3.6 HDSL1_DSL_CUSTOM 14-10

15.0 API: Microprocessor Communicator Channel Protocol 15-1
15.1 API Message Time-Out 15-3
15.2 API Master and Slave Implementation 15-3
15.2.1 No Peer-to-Peer Protocol 15-3
15.3 API Message Structure 15-4
15.3.1 Incoming Message Structure 15-4
15.3.2 Outgoing Message Structure 15-4
15.3.3 Header Section–Destination Field 15-5
15.3.3.1 Header Section–Opcode Field 15-5
15.3.4 Header Section—Reserved Byte or ACK Status 15-6
15.3.4.1 Acknowledge Status Byte 15-6
15.3.5 Header Section–Message Length Field 15-8
15.3.6 Data Section–Data Parameter Field 15-8
15.4 Host Port RAM Interface Protocol 15-9
15.4.1 INTR_HOST and INTR_8051 Registers 15-10
15.4.2 Host Port Acknowledge Register 15-11
15.4.3 Host Port Status Registers 15-11
15.5 Host Port RAM Interface Sequence of Events 15-13
15.5.0.1 Host Processor Polling Method 15-14
15.5.1 Multi Device System 15-14

15.6 RS232 Serial Interface Protocol 15-15
15.6.1 Host Processor to 8051 Processor Message Structure 15-15
15.6.2 RS232 Acknowledge Message Structure 15-16
15.6.3 ZipWire2 8051 Processor to Host Processor Status Message Structure 15-16
15.6.4 RS232 Message Transfer Protocol 15-17
15.7 RS232 Checksum Function 15-19
15.7.1 RS232 Multi-Device System 15-19
15.7.2 Group Talk Serial Interface Protocol 15-19
15.7.3 Boot RAM Software Download 15-19
16.0 ZipWire2 API Configuration 16-1
16.1 API Command Sequencing 16-1
16.2 Indirect Configuration 16-3
16.2.1 Scrambler/Descrambler Taps 16-3
16.2.2 CRC Tap 16-3
16.2.3 Sync Word 16-4
16.2.4 Pair ID (Z-Bits) 16-4
16.3 Single Pair Configuration 16-5
17.0 ZipWire2 API Commands 17-1
17.1 API Commands: Quick Reference 17-2
17.2 API Command Set Documentation Convention 17-6
17.2.1 API Command Names 17-6
17.3 Level 1 API Commands 17-7
17.3.1 DSL System Enable 17-7
17.3.2 DSL System Configuration 17-9
17.3.3 DSL Activation 17-11
17.3.4 Bit Pump Training Mode 17-14
17.3.5 DSL Pre-Activation Mode 17-15
17.3.6 DSL Startup Mode 17-16
17.3.6.1 Bit Pump Auto Tip/Ring Reversal 17-16
17.3.7 LOST Time (Tsilent) Period 17-17
17.3.8 DSL Reset 17-18
17.3.9 DSL Framer—PCM Configuration 17-19
17.3.10 PCM Multi-Frame Length 17-20
17.3.11 Single Pair Configuration 17-20
17.3.12 DSL Data Rate 17-21
17.3.13 DSL Status—Dynamic 17-22
17.3.14 DSL Status—Static 17-25
17.3.15 Versions 17-26
17.3.16 Line Attenuation 17-29
17.3.17 Noise Margin 17-30
17.3.18 EOC Send Command 17-31
17.3.19 EOC Get Message Status 17-31

17.3.20 EOC Set Database Data 17-32
17.3.21 EOC Get Database Data 17-32
17.3.22 EOC Set Message Control 17-33
17.3.23 EOC Read Receive Queue 17-34
17.3.24 EOC Set Proprietary Length 17-35
17.3.25 Download Start (Length) 17-35
17.3.26 Download Data 17-36
17.3.27 Download End (Checksum) 17-36
17.3.28 Download Slave 17-37

17.4 Level 2 API Commands 17-38

17.4.1 Read DSL Control Commands 17-38
17.4.2 Stage Number 17-39
17.4.3 Read Configuration Pins 17-40
17.4.4 AFE Setting 17-41
17.4.5 Analog Front End (AFE) Configuration 17-42
17.4.6 DSL Force Deactivate 17-43
17.4.7 Transmit External Data 17-43
17.4.8 Test Modes 17-44
17.4.9 Bit Pump Transmit Isolated Pulses Test Mode 17-45
17.4.10 Bit Pump Transmit Fixed Pattern Test Mode 17-46
17.4.11 Loopbacks 17-47
17.4.12 Bit Pump BER Meter State 17-49
17.4.13 Bit Pump BER Meter Results 17-50
17.4.14 ERLE Test Mode 17-52
17.4.15 ERLE Results 17-53
17.4.16 Auxiliary CLK Select 17-54
17.4.17 PCM Clock Configuration 17-54
17.4.18 DSL Framer Transmit PCM BER Meter Results 17-55
17.4.19 DSL Framer Receive PCM BER Meter Results 17-57
17.4.20 Transmit PCM BER State 17-58
17.4.21 Receive PCM BER State 17-58
17.4.22 PRBS Configure 17-59
17.4.23 Fill Pattern (CONST_FILL) 17-60
17.4.24 Data Bank Contents 17-61
17.4.25 Transmit PCM Mapper Value 17-62
17.4.26 Transmit PCM Mapper Write 17-63
17.4.27 Receive PCM Mapper Value 17-63
17.4.28 Receive PCM Mapper Write 17-64
17.4.29 Transmit HDSL Mapper Value 17-65
17.4.30 Transmit HDSL Mapper Write 17-66
17.4.31 Receive HDSL Mapper Value 17-66
17.4.31.1 Number of Time Slots 17-67
17.4.32 Receive HDSL Mapper Write 17-67
17.4.33 Clear ZipWire2 Error Counters 17-68
17.4.34 Read ZipWire2 Operational Error Counters 17-69
17.4.35 Read ZipWire2 HDSL Performance Error Counters 17-70

17.4.36 Read ZipWire2 PCM Performance Error Counters 17-71
17.4.37 Read ZipWire2 System Performance Error Counters 17-72
17.4.38 Available Seconds and Total Seconds 17-73
17.4.39 Inject DSL CRC Error 17-74
17.4.40 Set CRC/FEBE Error History State 17-75
17.4.41 Query CRC/FEBE History Interval In-Progress 17-76
17.4.42 CRC Error History at Interval 1 17-77
17.4.43 CRC Error History at Interval 2 17-78
17.4.44 CRC Error History at Interval 3 17-79
17.4.45 FEBE Error History at Interval 1. 17-80
17.4.46 FEBE Error History at Interval 2. 17-81
17.4.47 FEBE Error History at Interval 3. 17-82

17.5 Level 3 API Commands 17-83

17.5.1 Signal Level Meter 17-83
17.5.2 Timing Recovery Offset 17-84
17.5.3 Bit Pump Reverse Tip/Ring. 17-85
17.5.4 Scrambler/Descrambler Configuration. 17-85
17.5.5 Write AFE Transmit Gain 17-87
17.5.6 Read AFE Transmit Gain 17-88
17.5.7 DSL Framer Transmit Path Reset 17-89
17.5.8 DSL Framer Receive Path Reset 17-90
17.5.9 DSL Framer—HDSL Configuration 17-91
17.5.10 Mask Host Port Interrupt (INTR_HOST) 17-92
17.5.11 DSL Framer Auto Water Level 17-93
17.5.12 DSL Framer Transmit Water Level. 17-93
17.5.13 DSL Framer Receive Water Level 17-94
17.5.14 DSL Framer DPLL Clock Generator 17-95

17.6 Read/Write Register Commands 17-97

17.6.1 Write Register 17-97
17.6.2 Read Register 17-98
17.6.3 Write AFE Register 17-98
17.6.4 Read AFE Register 17-99

17.7 T1/E1 Framer Commands 17-100

17.7.1 T1/E1 Framer Configure 17-100
17.7.2 T1/E1 Framer Frame Format 17-100
17.7.3 T1/E1 Framer Transmit AIS. 17-101
17.7.4 T1/E1 Framer Output Mode. 17-101
17.7.5 T1/E1 Framer Receive Termination 17-102
17.7.6 T1/E1 Framer Loopbacks 17-103
17.7.7 T1/E1 Read Framer Control Commands. 17-104
17.7.8 T1/E1 Framer Versions 17-105

17.8 EVM Specific Commands 17-106

17.8.1 EVM Set LED Bank. 17-106
17.8.2 EVM Set Miscellaneous Output 17-107
17.8.3 EVM LED Update State 17-107
17.8.4 EVM LED and DIP Switch Status. 17-108

18.0 Electrical and Mechanical Specifications 18-1

18.1 Specifications for the ZipWire2 Transceiver/Framer and ZipWire2 AFE 18-1

18.1.1 Recommended Operating Conditions 18-1

18.1.2 Absolute Maximum Ratings 18-2

18.2 Thermal Characteristics 18-3

18.2.1 ZipWire2 AFE 18-3

18.2.2 ZipWire2 Transceiver 18-3

18.2.3 ZipWire2 Framer 18-3

18.2.4 ZipWire2 Transceiver/Framer 18-3

18.3 Specifications for ZipWire2 Transceiver/Framer Only 18-4

18.3.1 Power Dissipation 18-4

18.3.2 DC Characteristics 18-4

18.3.3 Host Port RAM Interface Timing 18-5

18.3.4 Master Bus Interface Timing 18-7

18.3.5 DSL Framer Timing Requirements 18-7

18.3.6 DSL Framer Switching Characteristics. 18-8

18.4 Specifications for ZipWire2 AFE Only 18-9

18.4.1 Power Dissipation 18-9

18.4.2 DC Characteristics 18-9

18.4.3 PSD Specifications. 18-10

18.4.3.1 18-10

18.4.3.2 18-11

18.4.3.3 18-11

18.4.4 Pulse Template Specifications. 18-12

18.5 Mechanical Specifications 18-14

Appendix A: Acronyms and Abbreviations. A-1

List of Figures

Figure 2-1. High-Level Functional Diagram 2-1
Figure 2-2. ZipWire2 Transceiver/Framer Detailed Block Diagram. 2-3
Figure 2-3. ZipWire2 Transceiver/DSP Detailed Block Diagram 2-4
Figure 2-4. DSL Framer Detailed Block Diagram 2-6
Figure 2-5. ZipWire2 AFE Block Diagram. 2-7
Figure 2-6. Detailed Transmit Data Path Block Diagram 2-8
Figure 2-7. Detailed Receive Data Path Block Diagram 2-9
Figure 3-1. Single Pair Hardware Configuration 3-1
Figure 3-2. Multi-Pair Hardware Configuration 3-2
Figure 3-3. Group Master/Group Slave Multi-Pair Hardware Configuration. 3-3
Figure 3-4. Master Multi-Pair Hardware Configuration 3-4
Figure 3-5. Redundant Group Master Multi-Pair Hardware Configuration 3-5
Figure 3-6. Multi-Pair Configuration—PCM Bused 3-7
Figure 3-7. Multi-Pair Configuration—PCM Cascade 3-8
Figure 3-8. ZipWire2 Transceiver/Framer to Bt8370 T1/E1 Interface 3-9
Figure 3-9. DSL Framer to CN8228 (ATM Phy) Interface Diagram 3-10
Figure 3-10. DSL Framer Bypass to CN8228 (ATM Phy) Interface Diagram 3-11
Figure 3-11. Dual Mode CN8228 (ATM Phy) Interface Diagram 3-12
Figure 3-12. DSL Framer to MUSYCC Interface Diagram 3-13
Figure 3-13. DSL Framer Bypass to MUSYCC Interface Diagram 3-14
Figure 3-14. Dual Mode MUSYCC Interface Diagram 3-15
Figure 3-15. Multiple ZipWire2 Devices to CN8228 or MUSYCC Interface Diagram 3-16
Figure 3-16. Multiple ZipWire2 Devices to CN8228 or MUSYCC Interface Diagram 3-17
Figure 4-1. ZipWire2 PRAM Download Overview 4-9
Figure 4-2. Single Processor Configuration. 4-12
Figure 4-3. Host Processor Configuration 4-13
Figure 5-1. ZipWire2 Clocks 5-1
Figure 5-2. Crystal Interface 5-2
Figure 6-1. Generic CRC Generator 6-2
Figure 6-2. Generic Scrambler Generator 6-3
Figure 6-3. HDSL Auxiliary Channel Timing. 6-3
Figure 6-4. HDSL Receive Section Block Diagram. 6-4
Figure 6-5. DSD Synchronization State Machine 6-5
Figure 6-6. HDSL TX Section Block Diagram. 6-6
Figure 6-7. Insert/Drop Timing Diagram 6-7
Figure 6-8. PCM Receiver Block Diagram 6-9
Figure 6-9. PCM Transmitter Block Diagram 6-10
Figure 6-10. Generic PRBS Generator 6-11
Figure 6-11. PRBS and BER Meter Timing 6-12

Figure 7-1. ZipWire2 Transceiver/Framer to AFE Interface. 7-1
Figure 7-2. DSL Transmission Line Interface. 7-2
Figure 7-3. Continuous Time Filter and Line Driver Control 7-3
Figure 7-4. Hybrid Topology 7-4
Figure 7-5. ZipWire2 AFE Bias Current Network 7-6
Figure 7-6. ZipWire2 Transceiver DSL Interface 7-7
Figure 8-1. ZipWire2 Transceiver/Framer Pin Assignments 8-2
Figure 8-2. ZipWire2 Transceiver Pin Assignments 8-3
Figure 8-3. ZipWire2 Framer Pin Assignments 8-4
Figure 8-4. ZipWire2 AFE Pin Diagram 8-5
Figure 9-1. EVM Block Diagram. 9-1
Figure 10-1. Software Overview 10-1
Figure 11-1. Activation State Manager (ASM) Overview 11-2
Figure 11-2. ZipWire2 Loopbacks 11-4
Figure 11-3. Detailed AFE Loopbacks 11-5
Figure 11-4. CRC and FEBE Error Records at Three Time Intervals 11-7
Figure 11-5. DPLL State Diagram 11-10
Figure 11-6. Loop Reversal Definition 11-12
Figure 11-7. EOC Implementation Details—Transmit 11-19
Figure 11-8. EOC Implementation Details—Receive 11-21
Figure 11-9. EOC Application State Machine 11-27
Figure 12-1. Main Program Flow 12-2
Figure 12-2. API Manager Flow 12-6
Figure 12-3. HDSL2 HTU-C Activation State Diagram 12-8
Figure 12-4. HDSL2 HTU-R Activation State Diagram 12-9
Figure 12-5. HDSL1 Activation State Machine at HTU-C 12-11
Figure 12-6. HDSL1 Activation State Machine at HTU-R 12-12
Figure 14-1. HDSL2 Frame Structure 14-3
Figure 14-2. Payload Block Structure for 1T1 Application 14-3
Figure 14-3. Payload Block Structure for 1E1 Application 14-4
Figure 14-4. HDSL1 Frame Structure 14-6
Figure 14-5. Payload Block Structure for 2T1 Application 14-7
Figure 14-6. Payload Block Structure for 2E1 Application 14-8
Figure 14-7. Payload Block Structure for 3E1 Application 14-9
Figure 14-8. Payload Block Structure for 1T1 Application 14-10
Figure 14-9. Payload Block Structure for 1E1 Application 14-10
Figure 15-1. Communication Channel Protocols 15-2
Figure 16-1. API Command Sequencing 16-2
Figure 18-1. Host Port Interface Timing Diagrams 18-5
Figure 18-2. Transmit Pulse Template for Two- and Three-Pair Systems; Normalized Pulse Mask (Source ETSI TS 101 135, Formerly ETR 152) 18-10
Figure 18-3. Transmit Pulse Template for One-Pair Systems (Source ETSI TS 101 135, Formerly ETR 152) 18-11
Figure 18-4. Upper Bound of the Average PSD of a 392 kbaud System (Source ETSI TS 101 135, Formerly ETR 152) 18-12
Figure 18-5. Upper Bound of the Average PSD of a 584 kbaud System (Source ETSI TS 101 135, Formerly ETR 152) 18-12

Figure 18-6. Upper Bound of the Average PSD of a 1,160 kbaud System (Source ETSI TS 101 135, Formerly ETR 152) 18-13
Figure 18-7. Package Outline, 27 x 27 mm, Two-Layer Chip, 314-Pin Ball Grid Array (BGA) 18-14
Figure 18-8. Package Outline, 15 x 15 mm, Two-Layer Chip, 208-Pin CABGA 18-15
Figure 18-9. Package Outline, 9 x 9 mm Two-Layer Chip, 81-Pin CABGA 18-16
Figure 18-10. Package Outline for the 80-Pin Thin Quad Flat Pack (TQFP) 18-17

List of Tables

Table 4-1. Internal 8051 Memory Map. 4-2
Table 4-2. Internal 8051 Interrupt Mapping 4-3
Table 4-3. START Bit Definitions 4-4
Table 4-4. Internal 8051 Timers/Counters 4-5
Table 4-5. 8051 Communication Interfaces 4-6
Table 4-6. Dip Switch #2 Definition 4-9
Table 4-7. DEVADR Bit Definitions 4-9
Table 4-8. BOOTOP Bit Definitions. 4-10
Table 4-9. Boot Code API Commands 4-15
Table 4-10. Operational API Commands 4-15
Table 5-1. ZipWire2 Clocks 5-2
Table 7-1. ZipWire2 AFE Compensation Capacitor Values 7-6
Table 7-2. ZipWire2 AFE Bias Current Network Values. 7-6
Table 8-1. ZipWire2 Transceiver/Framer Signal Definitions 8-6
Table 8-2. ZipWire2 AFE Signal Descriptions. 8-15
Table 9-1. Generic Chip Select CS7 Memory Map 9-1
Table 9-2. DSL Status LED #1 Register—Write Only. 9-2
Table 9-3. T1/E1 Framer Output / LED #2 Register—Write Only 9-2
Table 9-4. DIP Switch 3 Bit Definitions 9-3
Table 9-5. DIP Switch 4 Bit Definitions 9-3
Table 11-1. Activation Phases 11-1
Table 11-2. EOC Message Types 11-14
Table 11-3. Frame Format for HDSL2 EOC. 11-14
Table 11-4. EOC Device Address 11-15
Table 11-5. Summary of EOC Request Message IDs 11-16
Table 11-6. EOC TxQueue Status Bits 11-24
Table 11-7. User-Defined Message Request Information Field. 11-24
Table 11-8. User-Defined Message Response Information Field 11-25
Table 11-9. API Over EOC Request Information Field. 11-25
Table 11-10. API Over EOC Response Information Field 11-26
Table 11-11. EOC API Commands Summary 11-28
Table 12-1. API Manager Flag Description 12-5
Table 13-1. 2B1Q PAM4 Bit-to-Level Mapping. 13-1
Table 13-2. Optis PAM16 Bit-to-Level Mapping 13-2
Table 14-1. HDSL2 Frame Structure and Overhead Bit Allocation 14-2
Table 14-2. 1T1 Framing 14-3
Table 14-3. 1E1 Framing 14-4
Table 14-4. HDSL1 Frame Structure and Overhead Bit Allocation 14-5
Table 14-5. 2T1 Framing 14-7

Table 14-6. 2E1 Framing 14-8
Table 14-7. 3E1 Framing 14-9
Table 14-8. 1T1 Framing 14-10
Table 14-9. 1E1 Framing 14-10
Table 15-1. Incoming Messages from the Host Processor 15-4
Table 15-2. Outgoing Messages from the 8051 Processor 15-4
Table 15-3. Destination Field Specification 15-5
Table 15-4. API Opcode Type 15-5
Table 15-5. Acknowledge Status Codes 15-6
Table 15-6. Host Port RAM Mapping 15-9
Table 15-7. Acknowledge Status Register (Interrupt Source Register) 15-11
Table 15-8. Host Port RAM Status Mapping 15-12
Table 15-9. Host Port RAM Message Protocol Events 15-13
Table 15-10. Host Processor to 8051 Processor RS232 Message Structure 15-15
Table 15-11. RS232 Acknowledge Response Message Structure 15-16
Table 15-12. 8051 Processor to Host Processor RS232 Message Structure 15-16
Table 15-13. Example 1—Incoming RS232 Message 15-17
Table 15-14. Example 1—Outgoing RS232 Message 15-17
Table 15-15. Example 2—Incoming RS232 Message 15-17
Table 15-16. Example 2—Outgoing RS232 Message 15-17
Table 15-17. Example 3—Incoming RS232 Message 15-18
Table 15-18. Example 3—Outgoing RS232 Message 15-18
Table 16-1. Scrambler/Descrambler Taps 16-3
Table 16-2. CRC Tap 16-3
Table 16-3. Sync Word—HDSL2 16-4
Table 16-4. Sync Word—HDSL1 16-4
Table 16-5. Pair ID of the Z-bit Field 16-4
Table 16-6. Single Pair Configuration API Commands 16-5
Table 17-1. API Command Summary 17-2
Table 17-2. HDSL1 Variable Rate Startup Times 17-12
Table 17-3. STATUS_1: DSL Status Bit Definitions 17-23
Table 17-4. STATUS_3: Startup Failure Status Bit Definitions 17-23
Table 17-5. STATUS_4: DSL Framer Status Bit Definitions 17-24
Table 17-6. STATUS_8: Acknowledge Status (ISR) Bit Definitions 17-24
Table 17-7. Fatal Error Bit Definitions 17-25
Table 17-8. Tx State Bit Definitions 17-25
Table 17-9. DEVADR / BOOPOP Bit Definitions 17-40
Table 17-10. AFE Bit Definitions 17-41
Table 17-11. Test Mode Options 17-44
Table 17-12. Loopback Options 17-47
Table 17-13. DSL Framer BER Status Bits 17-56
Table 17-14. PRBS Data Pattern 17-60
Table 17-15. Clear Error Counter Options 17-68
Table 17-16. Inject CRC Error Options 17-74
Table 17-17. T1/E1 Framer Loopback Options 17-103
Table 17-18. Framer Silicon Types 17-105

Table 18-1. Recommended Operating Conditions 18-1
Table 18-2. Absolute Maximum Ratings 18-2
Table 18-3. ZipWire2 Transceiver/Framer Power Dissipation. 18-4
Table 18-4. Transceiver/Framer DC Characteristics 18-4
Table 18-5. Host Port Ram Interface Timing Table. 18-6
Table 18-6. ZipWire2 AFE Power Dissipation 18-9
Table 18-7. AFE DC Characteristics 18-9
Table 18-8. Transmit Pulse Template for Two- and Three-Pair Systems (Source ETSI TS 101 135, Formerly ETR 152) 18-10
Table 18-9. Transmit Pulse Template for One-Pair Systems (Source ETSI TS 101 135, Formerly ETR 152) 18-11

1.0 Introduction

This data sheet provides information for using the Conexant CN8980 ZipWire2 HDSL2/SDSL Transceiver and Framer, and ZipWire2 Analog Front-End (AFE) devices. This document discusses application and hardware interfaces. It provides detailed descriptions of the devices and pins, software information, configuration information, implementation of communication protocols, commands, and electrical and mechanical specifications.

Due to the flexibility of the ZipWire2 chip set, not all applications are addressed in this data sheet. Please contact the local sales office or technical support to determine how the ZipWire2 device can be used in your DSL application.

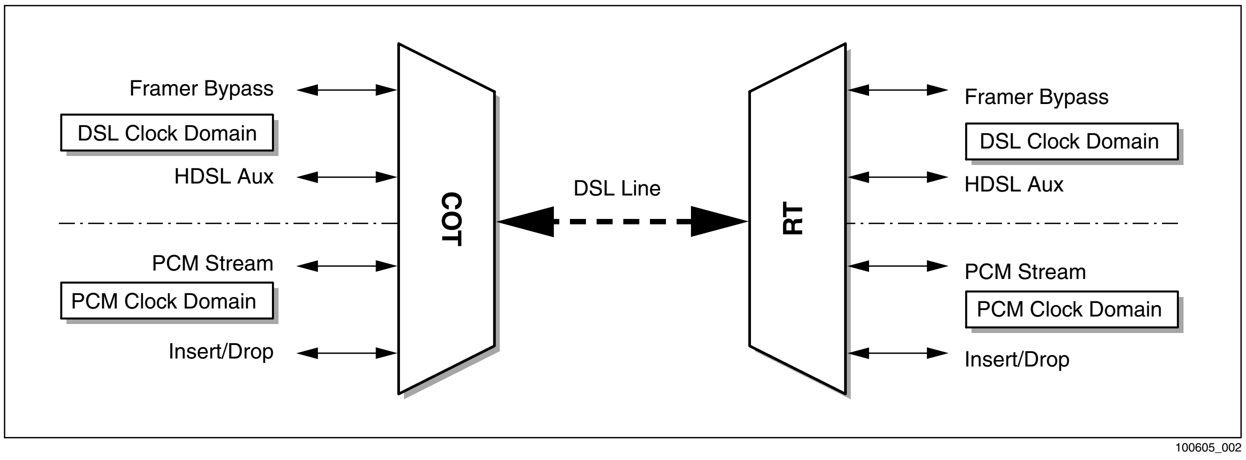
1.1 References

T1/E1.4 (T1E1.4/99-006)—Draft for HDSL2 Standard
RE/TM-06011-1—Draft for SDSL

2.0 System Overview

For most applications, the ZipWire2 chip set can be viewed as a pair of wires: What comes in on one terminal unit will go out the far-end terminal unit. Figure 2-1 illustrates the ZipWire2 data interfaces. The Framer Bypass and HDSL auxiliary interfaces operate at the DSL line rate. The PCM and Insert/Drop operate at the PCM clock rate. The DSL line interfaces to the physical twisted pairs.

Figure 2-1. High-Level Functional Diagram



2.1 ZipWire2 Transceiver/Framer Functional Summary

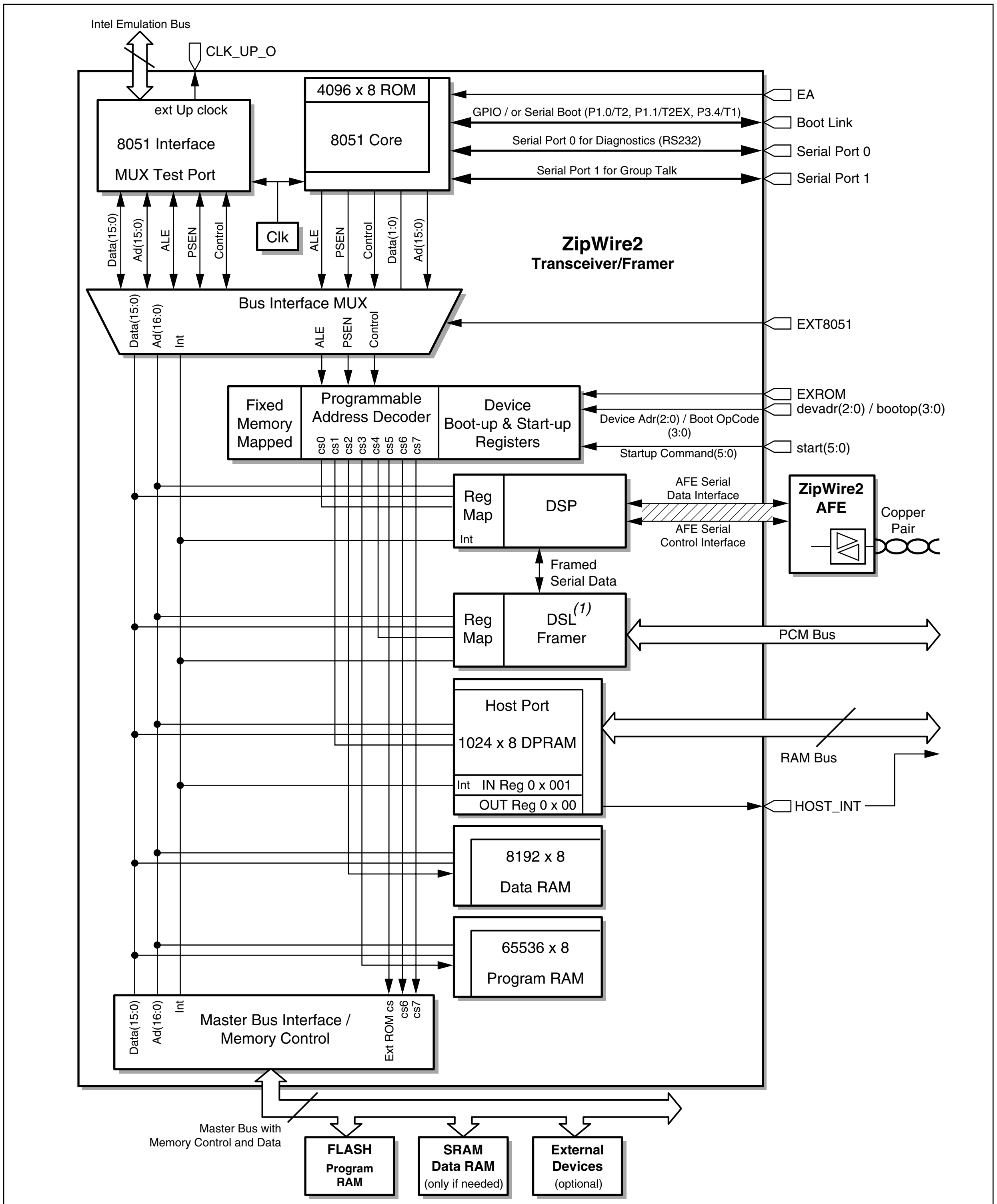
Figure 2-2 illustrates a detailed block diagram of the ZipWire2 Transceiver/Framer. The 8051 microprocessor sections contain an internal boot-up ROM, execution Program RAM (PRAM), Data Storage RAM, and address decoding. The internal 8051 is responsible for performing the transceiver startup, DSL Framer overhead management, interrupt handling, etc.

A full-featured API command set allows the user to configure the ZipWire2 system, query for status, execute loopbacks and test modes, and to dictate the program flow.

In addition to controlling the internal sections of the ZipWire2 device, the internal 8051 can be used to control other external devices, i.e., T1/E1 Framer, LEDs, and so on.

The CN8980 is available in a 2- or 3-package option. In both options, the AFE is the same. In the 2-package option the Transceiver and Framer are in a single package. In the 3-package option, the Transceiver and Framer are in separate packages.

Figure 2-2. ZipWire2 Transceiver/Framer Detailed Block Diagram



NOTE(S):
(1) This figure illustrates a 2-package option. In a 3-package option, the DSL Framer would be in a separate package.

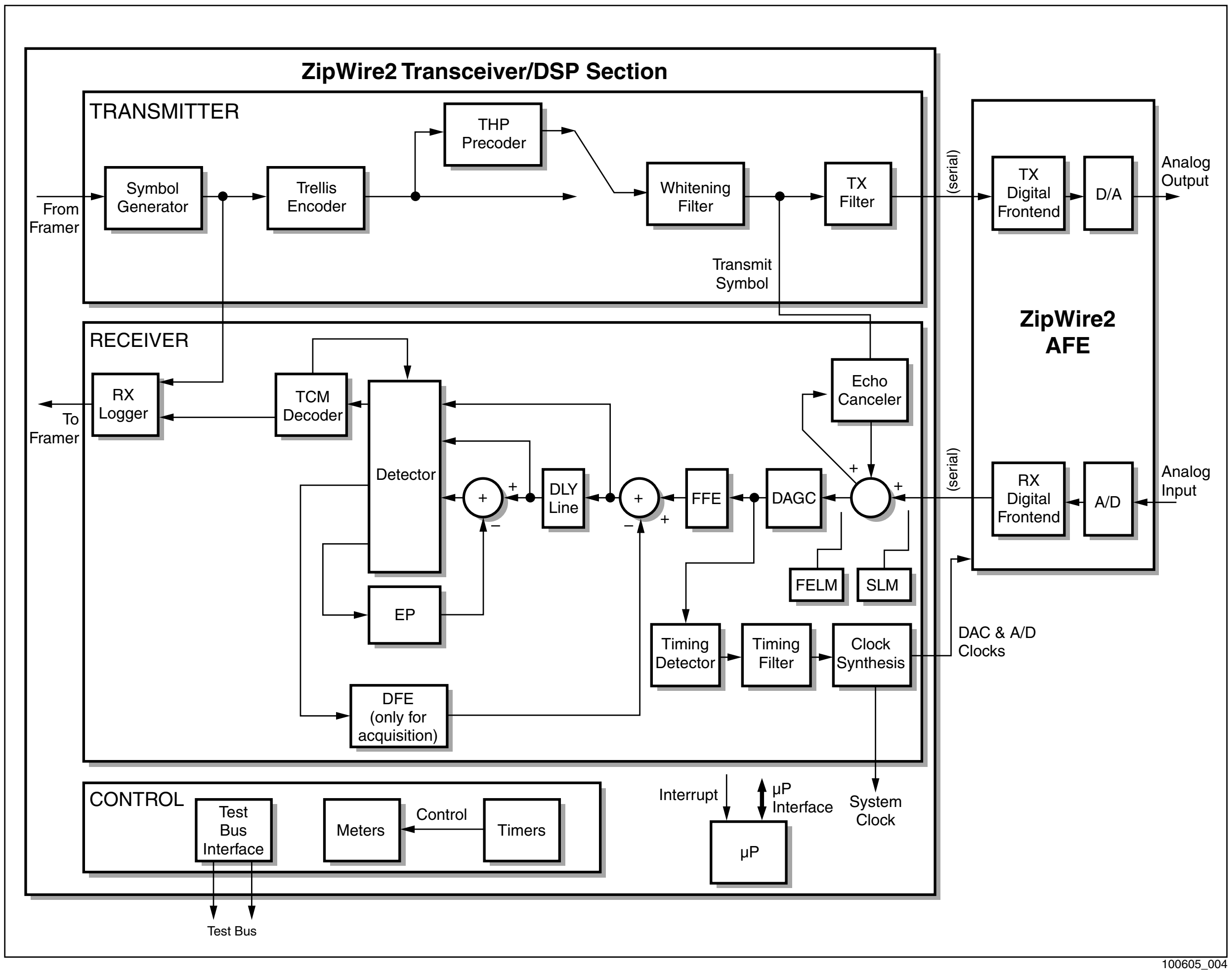
100605_003

2.2 ZipWire2 Transceiver/DSP Functional Summary

Figure 2-3 illustrates a detailed block diagram of the ZipWire2 Transceiver/DSP section. The transmitter receives a bit stream from the DSL Framer and maps the data bits to the appropriate PAM symbols. An optional precoding block which supports both Tomlinson-Harashima precoding as well as Tomlinson-Harashima precoding combined with Trellis shaping follows the PAM mapper. The signal is then processed by the transmit filter to achieve the desired time and/or frequency domain characteristics before being forwarded to the Analog Front-End (AFE).

The receiver receives serialized data from the AFE device and from the precoded symbols from the bit pump transmitter. The precoded symbols are fed into an Echo Canceler (EC) which estimates the echo response and subtracts it from the AFE samples. The signal is then equalized using a Feed Forward Equalizer (FFE) and a Decision Feedback Equalizer (DFE). Finally, a TCM decoder is used to recover the information bits. The DFE is used only during startup. An error predictor is also used as a part of the startup algorithm and as a precoder coefficient adaptation machine during normal operation.

Figure 2-3. ZipWire2 Transceiver/DSP Detailed Block Diagram

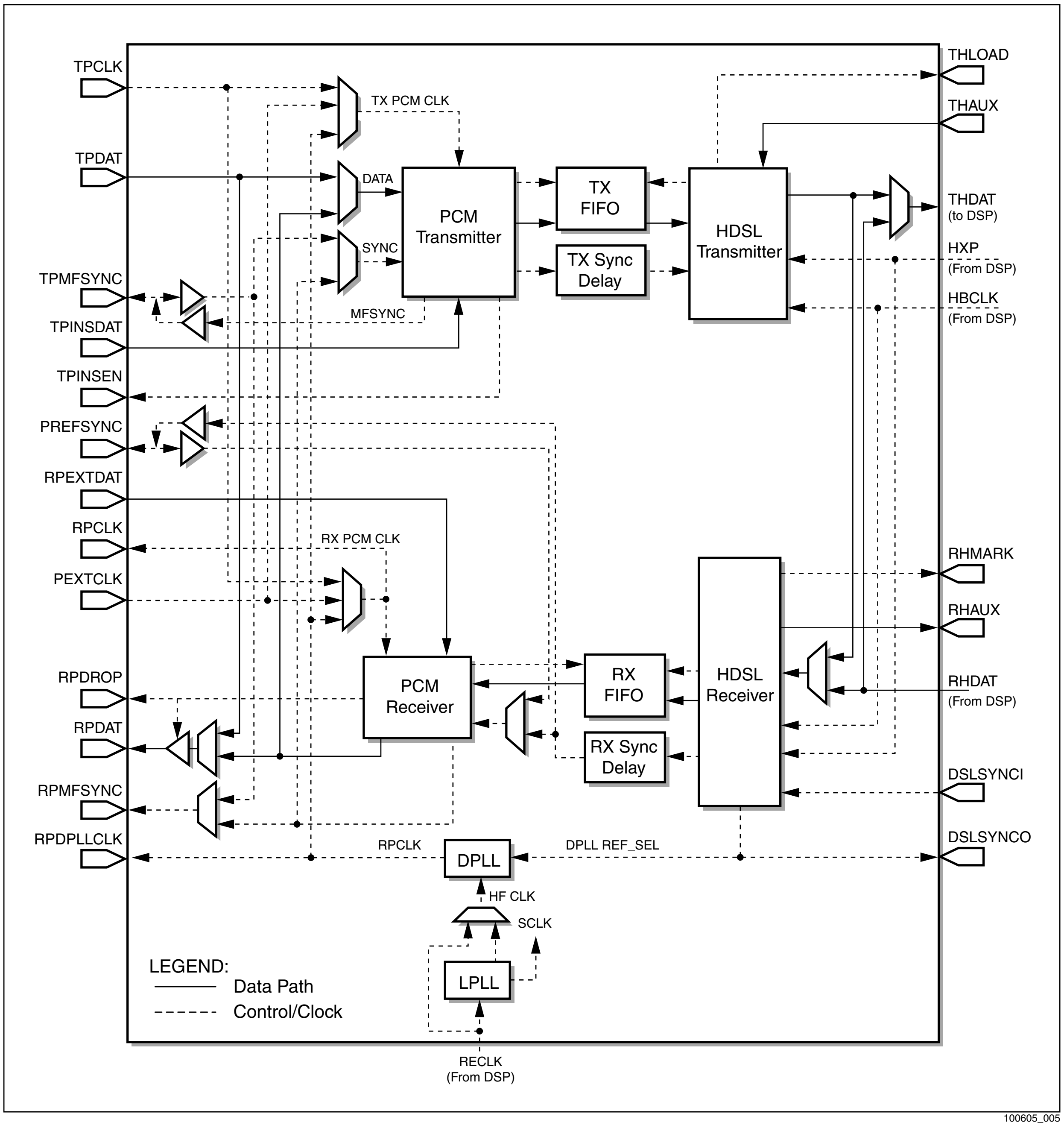


2.3 ZipWire2 DSL Framer Functional Summary

Figure 2-4 illustrates a detailed block diagram of the ZipWire2 DSL Framer section. The DSL Framer supports HDSL1, HDSL2, and custom frame structure applications. The DSL Framer provides clock, data, and frame format conversion from various PCM frame formats to various DSL applications. The DSL Framer supports Multi-Pair configuration such as T1 two loops, E1 two and three loops, or any Point-to-Multipoint (P2MP) application by cascading several DSL Framers. The DSL Framer provides full PCM termination capabilities, which include synchronization and management of E1 PRA and T1. The DSL rate can vary from 144 kbps up to 4,640 kbps (2 × E1 + Overhead), and the PCM rate can vary from 64 kbps up to 8,192 kbps (4 × E1) and any custom PCM rate and frame format within this range. The details of the DSL Framer section are described in Chapter 6.0.

The ZipWire2 DSL Framer can also be configured to provide T1 path termination capabilities and thereby eliminate the need for an external T1 Framer in some applications. In particular, the ZipWire2 DSL Framer is capable of generation and insertion of the T1 overhead in the transmit direction as well as alignment and checking of T1 overhead in the receive direction.

Figure 2-4. DSL Framer Detailed Block Diagram

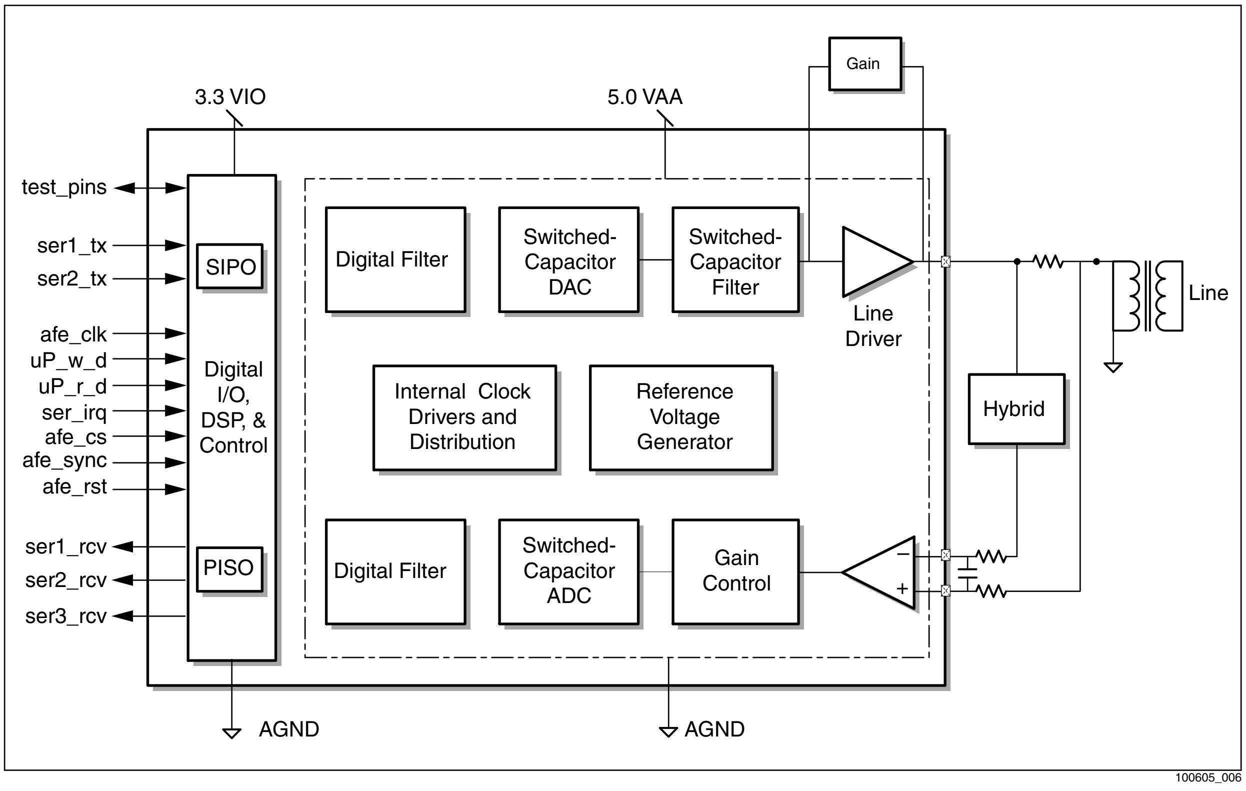


2.4 ZipWire2 AFE Functional Summary

Figure 2-5 illustrates a detailed block diagram of the ZipWire2 AFE. The ZipWire2 AFE performs the analog functions for transmission and reception of HDSL2 OPTIS or HDSL1 2B1Q line-code signals. ZipWire2 AFE includes the Digital-to-Analog (D/A) and Analog-to-Digital (A/D) conversion, data converter anti-aliasing and post filtering, gain control, and line driving.

The ZipWire2 AFE serial digital interface connects to the ZipWire2 Transceiver/Framer device. The serial interface protocol is proprietary. The AFE is controlled indirectly via the DSP transceiver. The analog interface consists of the line driver feedback resistors, impedance matching resistors, external hybrid, and transformer.

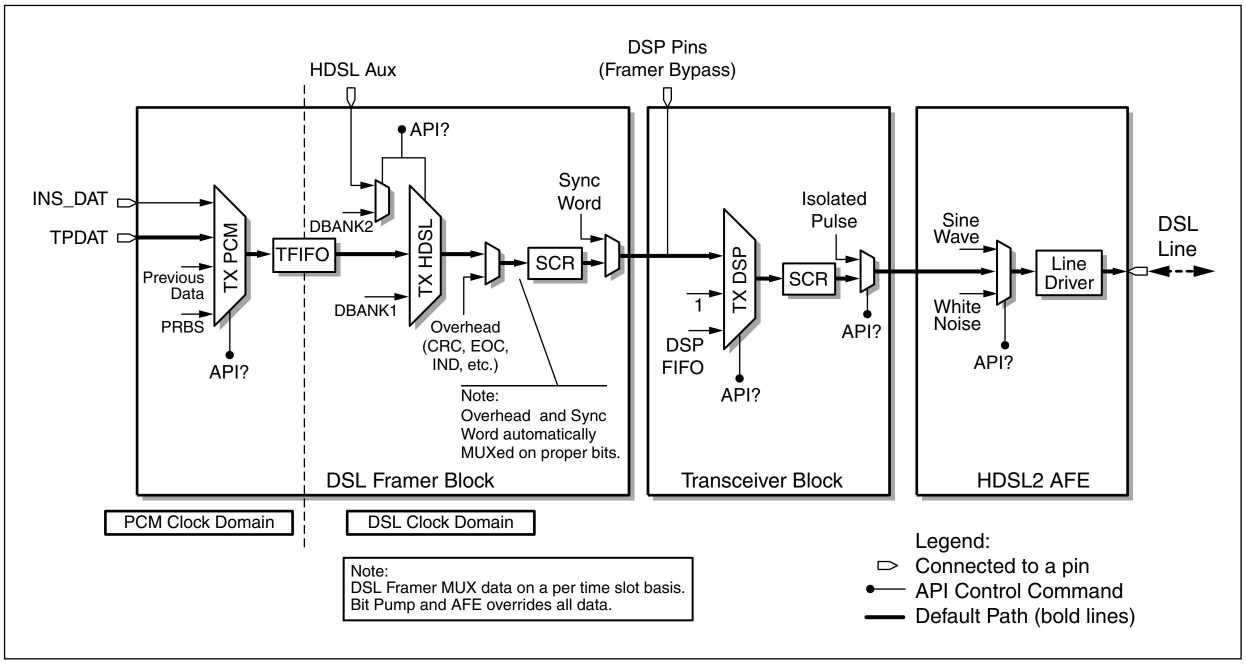
Figure 2-5. ZipWire2 AFE Block Diagram



2.5 ZipWire2 Transmit Path

Figure 2-6 illustrates the various input data sources that can be sent out the ZipWire2 link. This drawing includes all external inputs as well as internally generated data sources. This figure does not illustrate loopbacks.

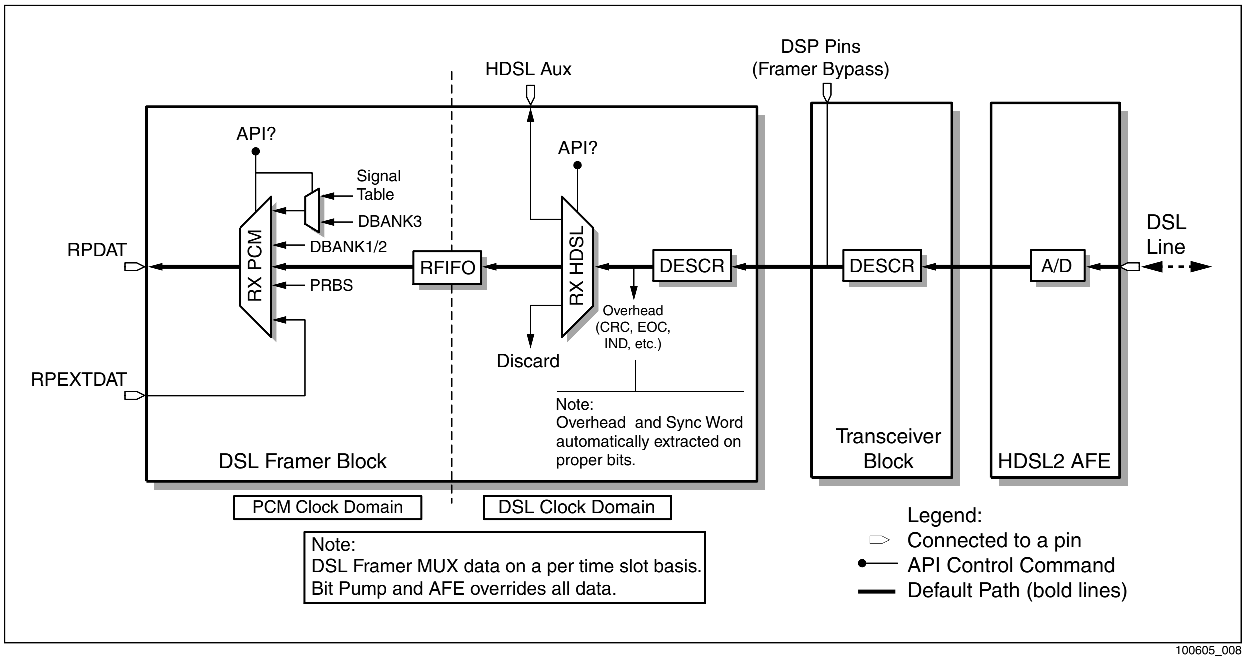
Figure 2-6. Detailed Transmit Data Path Block Diagram



2.6 ZipWire2 Receive Path

Figure 2-7 illustrates the various output destinations received from the ZipWire2 link. This drawing includes all external inputs, as well as internally generated data sources. This figure does not illustrate loopbacks.

Figure 2-7. Detailed Receive Data Path Block Diagram



3.0 Application Interfaces

This section illustrates various application configurations. Each figure will illustrate different interface configurations.

3.1 Using Internal 8051 Processor Only

Figure 3-1 illustrates the block diagram of a single device ZipWire2 system. In this application, the ZipWire2 8051 Processor can be used to control other external devices, i.e., T1/E1 Framer. There are no other microprocessors in the system.

For applications that need to modify the 8051 Processor code, refer to the (TBD) Document. The default Flash can support the Bt8370 T1/E1 Framer, DIP Switches, LEDs, and so on. The Bt8370 would be configured for a simple transparent mode. Refer to Chapter 9.0 for more detail.

Figure 3-1. Single Pair Hardware Configuration

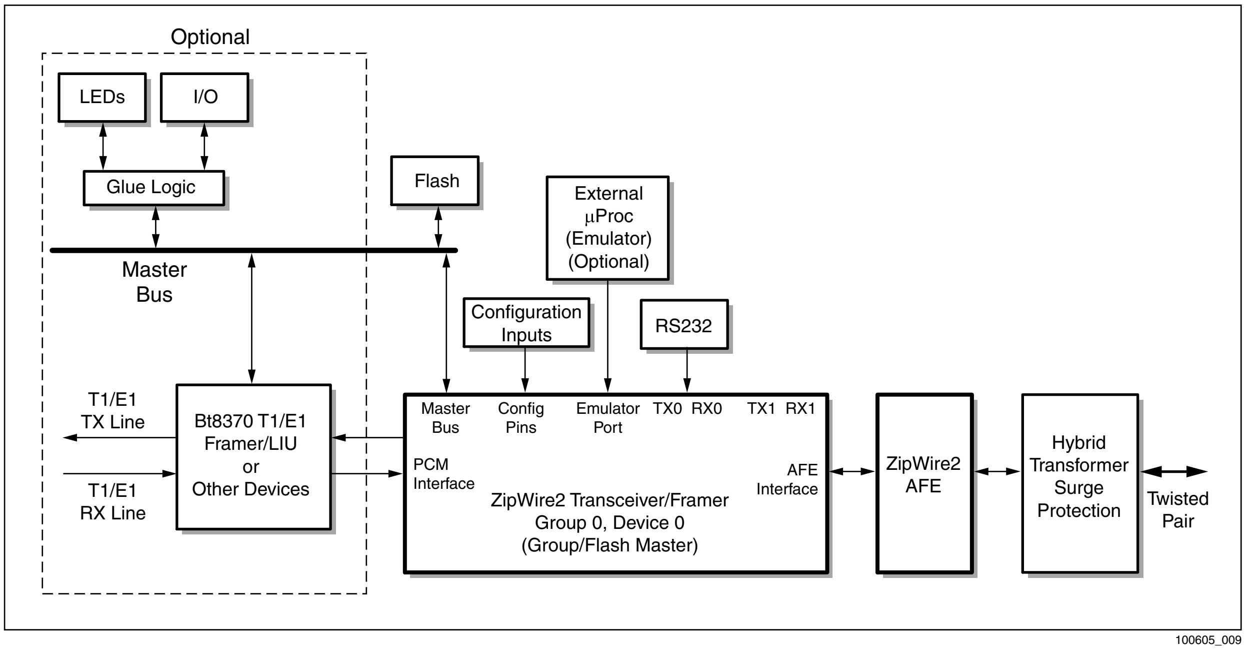
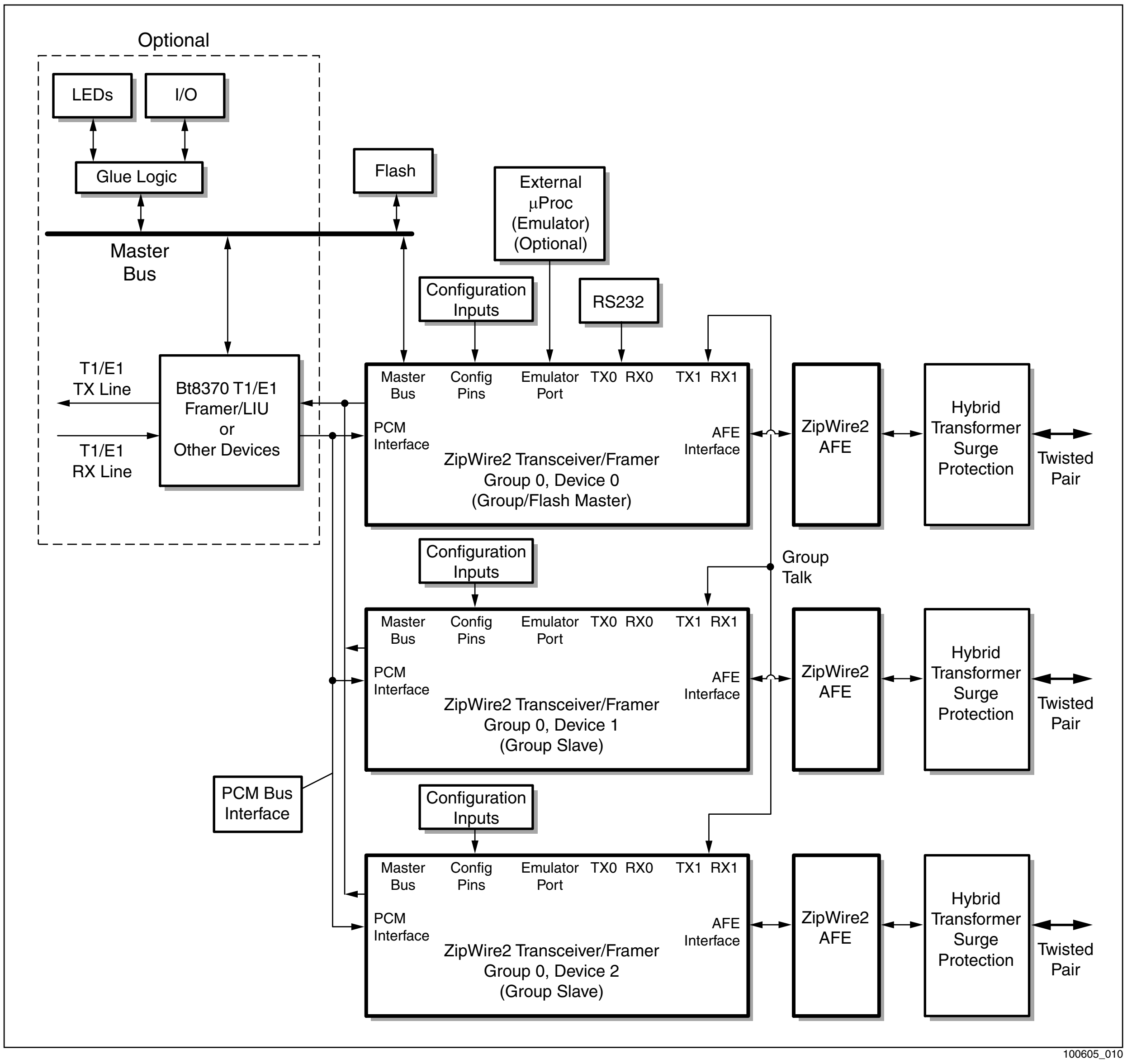


Figure 3-2 illustrates the hardware configuration for a standard HSDL1 2T1, 2E1, or 3E1 configuration. The PCM interface bus of the separate devices is connected to the T1/E1 Framer. The first device is designated as the Group and Flash Master that controls the external devices and could be connected to an external host processor via the RS232 port. The Group Master communicates with the other devices via the Group Talk Serial Link.

Figure 3-2. Multi-Pair Hardware Configuration


Optional
LEDs
I/O
Glue Logic
Flash
External µProc (Emulator) (Optional)
Master Bus
Configuration Inputs
RS232
T1/E1 TX Line
T1/E1 RX Line
Bt8370 T1/E1 Framer/LIU or Other Devices
Master Bus
Config Pins
Emulator Port
TX0 RX0
TX1 RX1
PCM Interface
AFE Interface
ZipWire2 Transceiver/Framer Group 0, Device 0 (Group/Flash Master)
ZipWire2 AFE
Hybrid Transformer Surge Protection
Twisted Pair
Group Talk
ZipWire2 Transceiver/Framer Group 0, Device 1 (Group Slave)
PCM Bus Interface
ZipWire2 Transceiver/Framer Group 0, Device 2 (Group Slave)
100605_010

3.2 Using an External Embedded Host Processor

Figure 3-3 illustrates a multidevice configuration with an external host processor present. The 8051 Processor can handle up to eight ZipWire2 devices within a group. One device is the Group Master and the other seven devices are Group Slaves. The 8051 processor does not limit the number of ZipWire2 groups.

The Group Talk Serial Link is used for downloading the program and communicating API commands from the Group Master to the Group Slaves.

NOTE: The PCM and AFE interfaces are not shown in these figures.

Figure 3-3. Group Master/Group Slave Multi-Pair Hardware Configuration

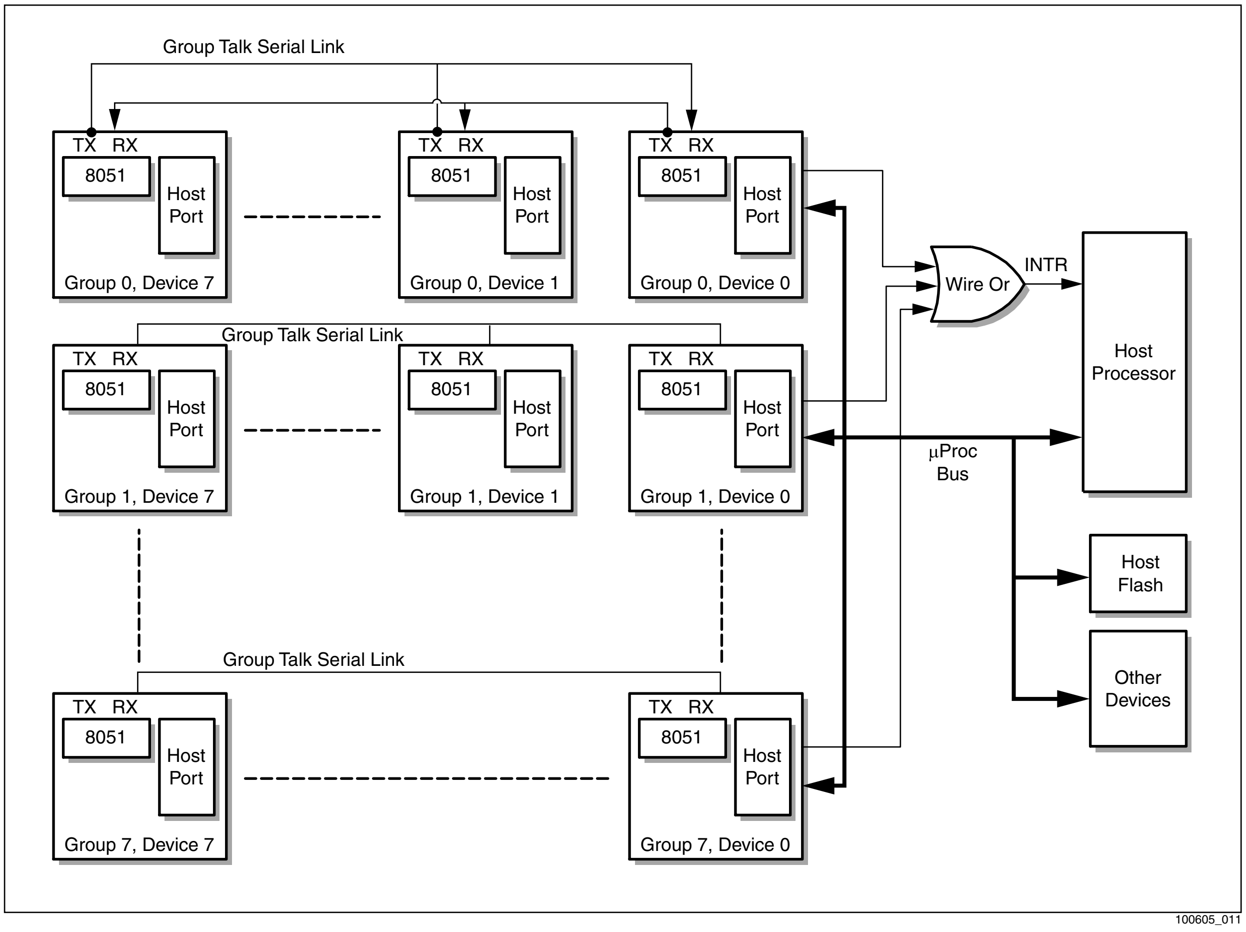


Figure 3-4 shows the hardware configuration when the host processor is connected directly to the Host Port of all of the ZipWire2. All of the ZipWire2 would be configured as Group Masters. This would allow the host processor to access all of the ZipWire2 without using the Group Talk interface.

The Group Talk interface is not used in this configuration.

Figure 3-4. Master Multi-Pair Hardware Configuration

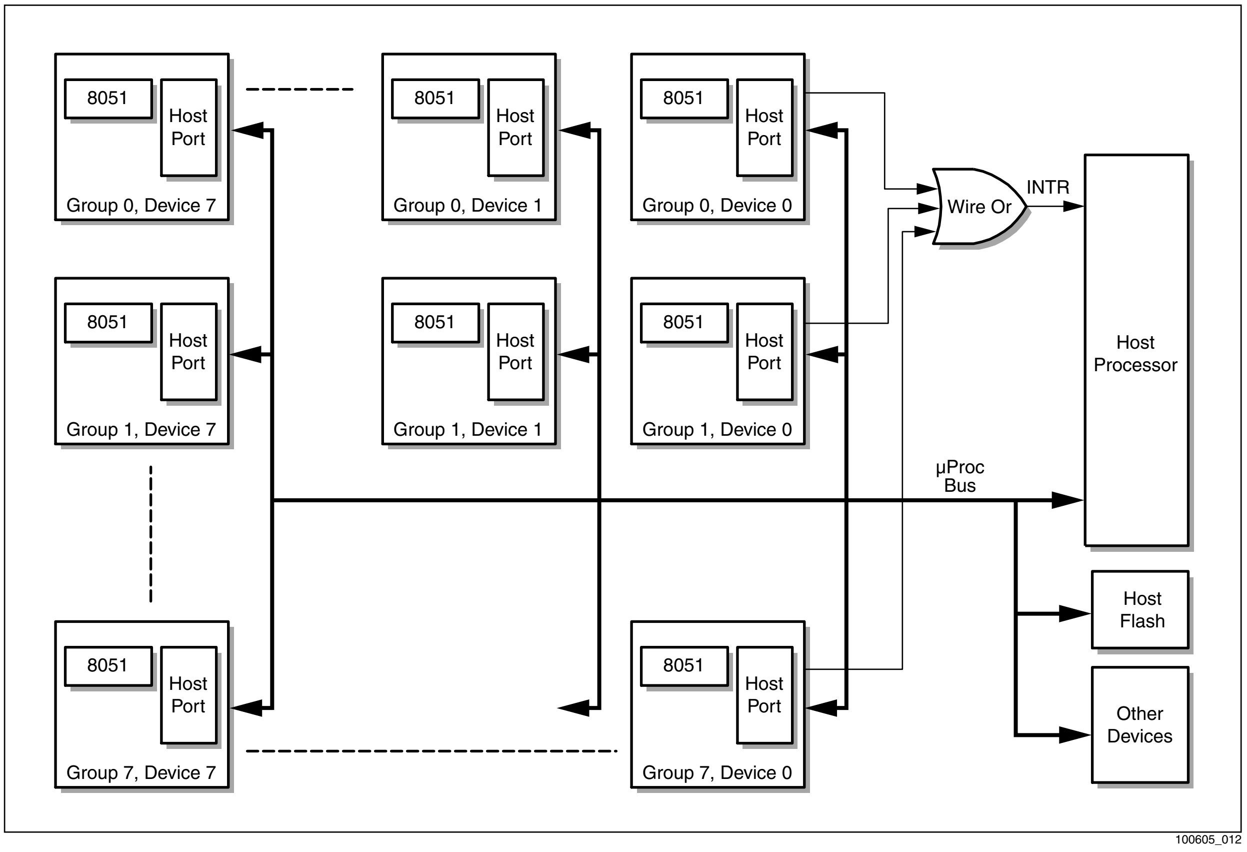
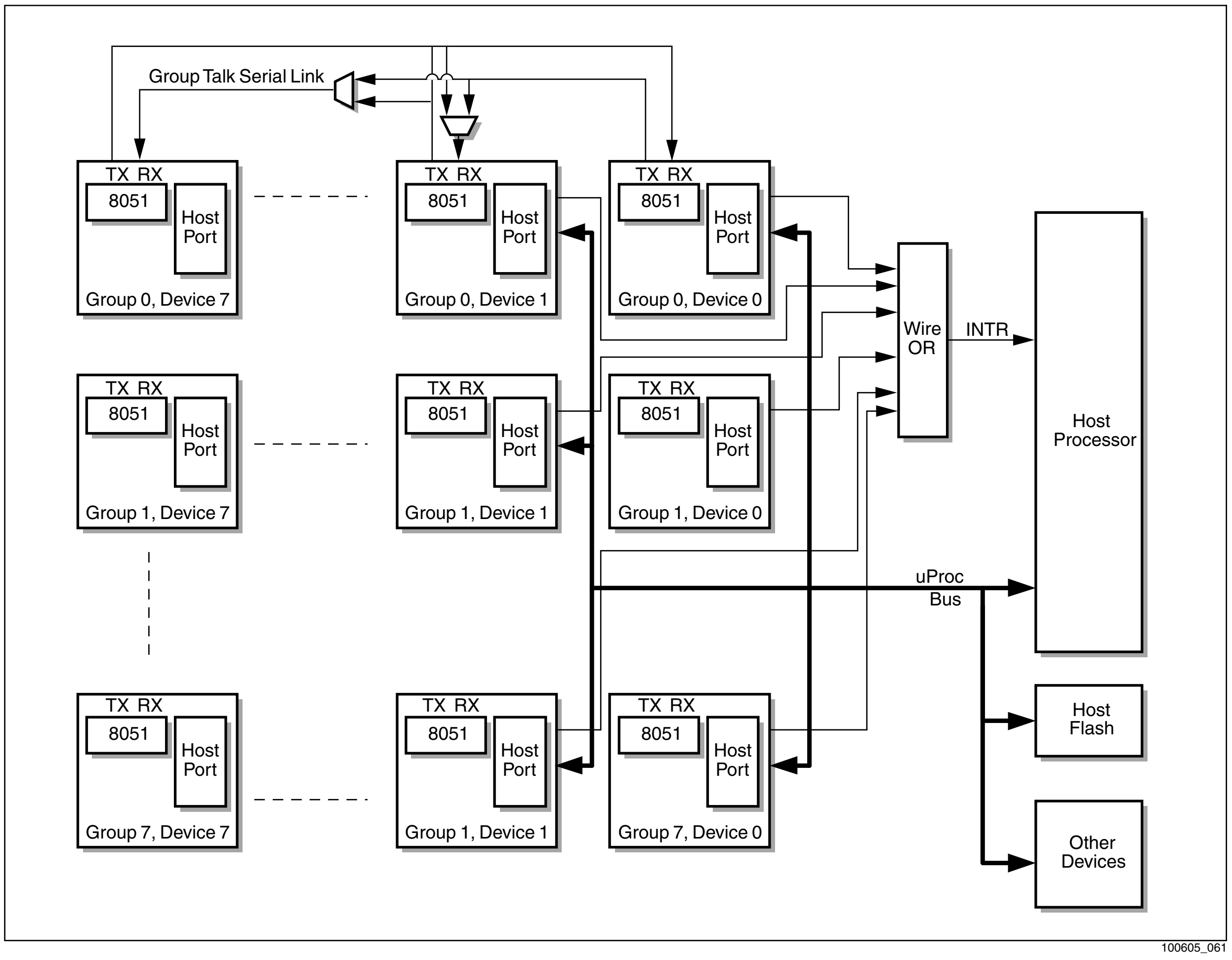


Figure 3-5 shows the hardware configuration when the host processor is connected directly to the Group Master and a redundant Group Master. This would allow the host processor to switch to the redundant Group Master in the unlikely event the Group Master ZipWire2 device has a physical defect that renders it unusable. By switching to the redundant Group Master, this will allow the other Group Slave devices to be serviced by the host processor.

The host processor would be responsible for switching an external MUX to change the source of the Group Master. Each ZipWire2 behaves identically when the API destination field received via the Host Port Interface matches the device ID. There is no special API command to switch the Group Master, the host processor just needs to communicate with the redundant Group Master. The device ID must remain fixed with the group, that is, the redundant Group Master would still have the device ID of 1.

The Group Talk interface is still provided to download the program to all of the ZipWire2 devices and to communicate API commands.

Figure 3-5. Redundant Group Master Multi-Pair Hardware Configuration



3.3 Multi-Pair DSL Framer Configuration (Cascade Mode)

The DSL Framer PCM bus can operate up to 8 MHz by cascading multiple ZipWire2 devices. Multi-Pair configuration is necessary to support several applications: Point-to-Multipoint, T1 transport over two HDSL wire pairs (Bellcore standard), or E1 transport using two or three HDSL wire pairs (ETSI standard). Several cascading DSL framers can support these applications and more.

The following pins are used in cascade mode: DSLSYNCI, DSLSYNCO, PEXTCLK, and RPEXTDAT.

Two options can be used to implement Multi-Pair configuration:

- PCM Bused (see Figure 3-6) enables the connection of an unlimited number of framers to receive the PCM highway interface in which each framer contributes/routes different time slots to/from the PCM highway. In this option, none of the framers carry a complete PCM frame; therefore, PCM framing termination (PRA) is not feasible.
- PCM Cascade (see Figure 3-7) enables transmit and receive PCM framing (PRA) by routing the receive PCM frame data from DSL framers 2 and 3 to DSL framer 1 (master) using the RPEXTDAT pin. This configuration is limited to three DSL framers.

On the transmit path, both configuration options behave the same and have the same capability.

The transmit PCM signal (clock, data, and sync) are connected to the transmit PCM pins of each DSL framer while each framer is programmed independently to route any incoming PCM data combination to the HDSL channel.

In the receive path, the master framer aligns the slaves' (DSL framer 2 and DSL framer 3) receive PCM time base using the PREFSYNC bidirectional pin. The recovered PCM clock is then provided to the slave framers using RPCLK and PEXTCLK pins. This capability enables the generation of a common receive PCM time base for all DSL framer receive channels and reliably reconstructs the PCM frame.

The active DPLL (typically located in the master framer) is able to select any DSL frame reference (for the DPLL phase detector) using DSLSYNCI and DSLSYNCO pins. This allows the master to switch DSL reference sources when the selected pair becomes inactive (HDSL loss-of-signal detection).

Figure 3-6. Multi-Pair Configuration—PCM Bused

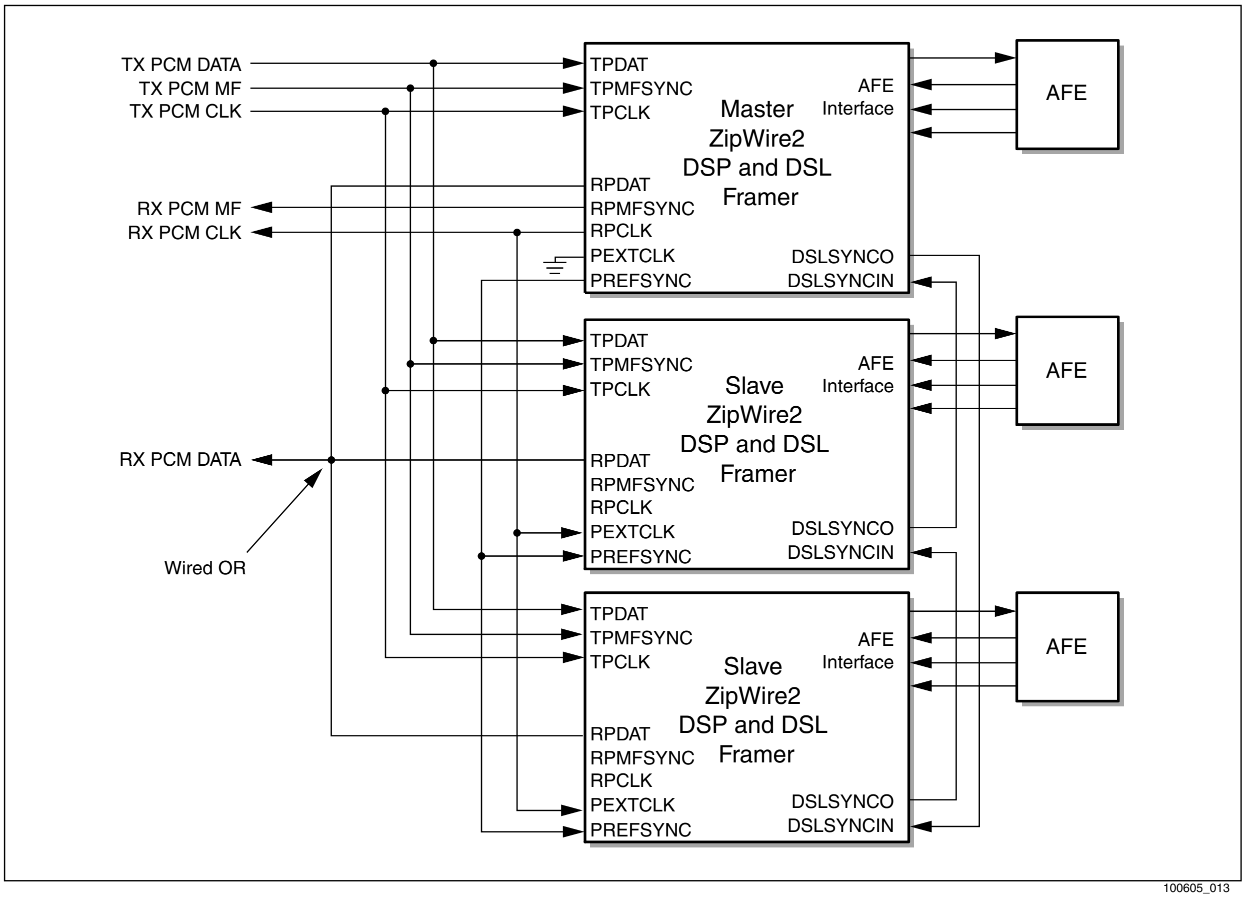
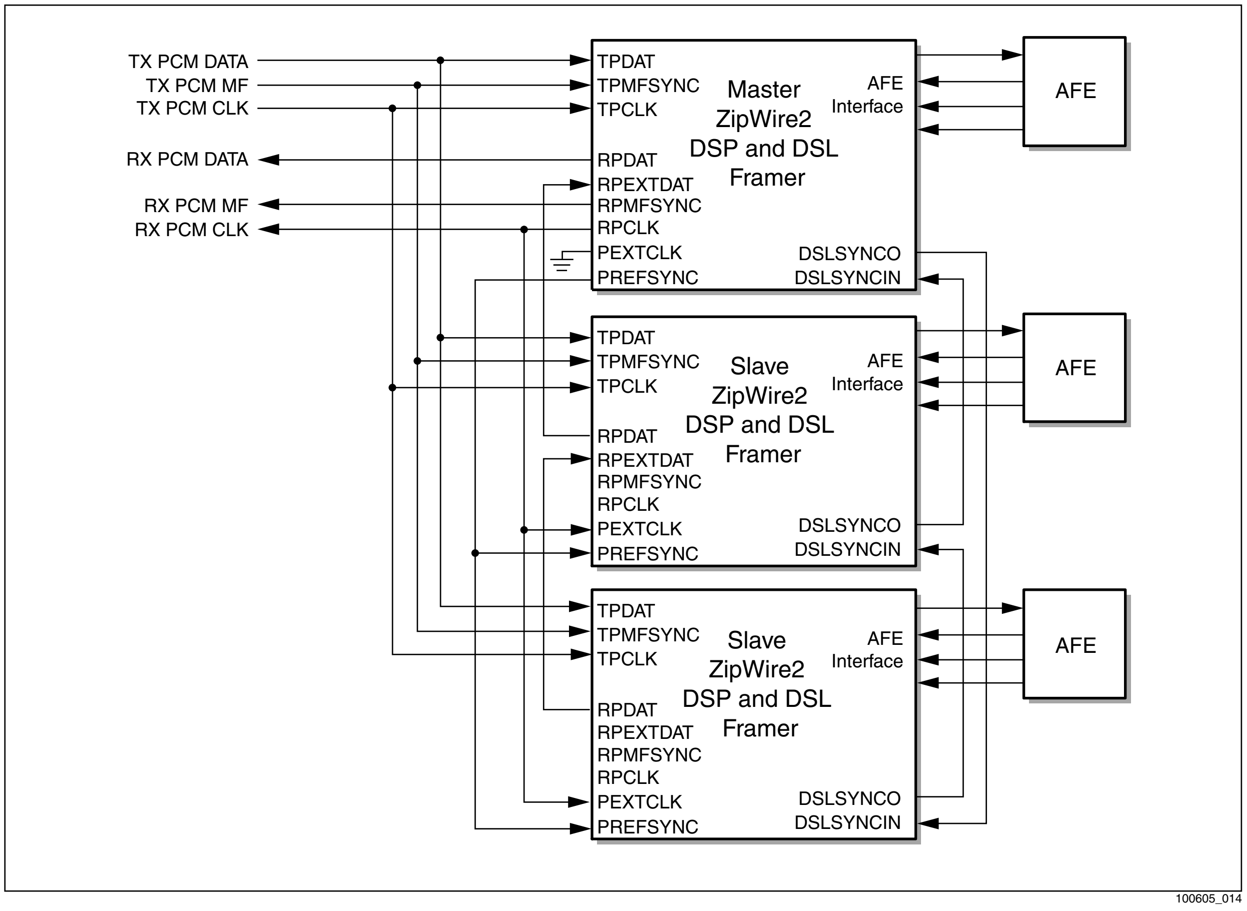


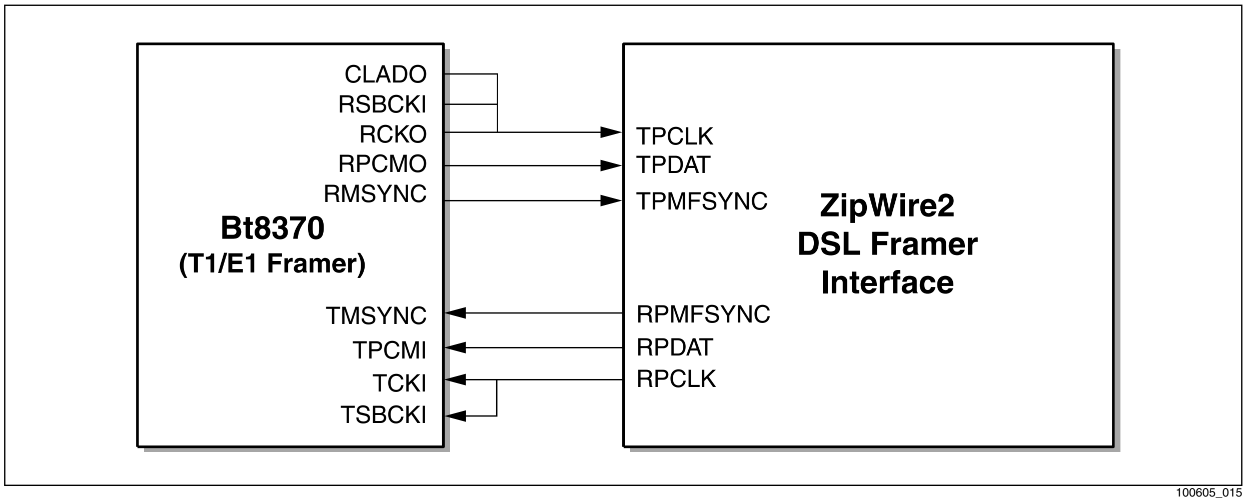
Figure 3-7. Multi-Pair Configuration—PCM Cascade



3.4 ZipWire2 Transceiver/Framer to Bt8370 T1/E1 Interface

Figure 3-8 illustrates one possible configuration for the ZipWire2 Transceiver/Framer to Bt8370 T1/E1 Interface.

Figure 3-8. ZipWire2 Transceiver/Framer to Bt8370 T1/E1 Interface

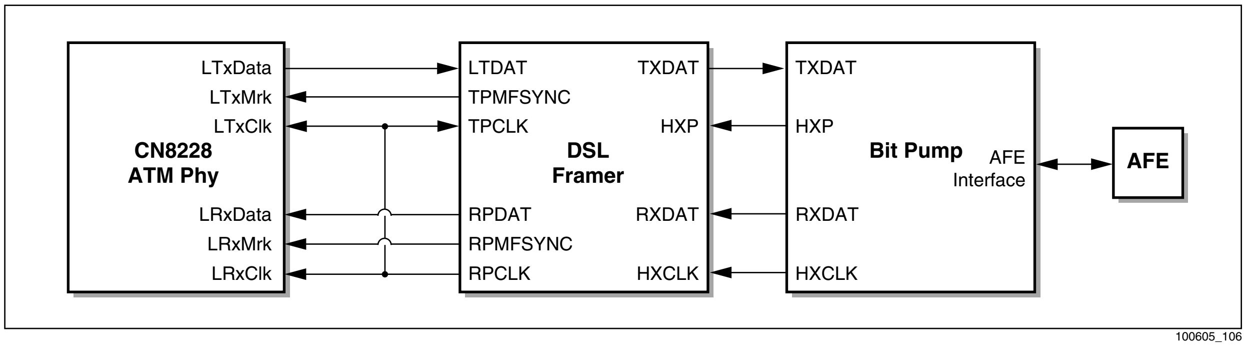


3.5 DSL Framer to CN8228 (ATM Phy) Interface

Figure 3-9 illustrates how to connect the ZipWire2 device to the CN8228 ATM Phy device when using the ZipWire2 DSL Framer block. In an HTU-C (central office) application, the DSL Framer DPLL would be programmed to open loop mode to provide the clock reference. In an HTU-R (remote terminal) application, the DSL Framer DPLL would be programmed to closed loop mode to recover the PCM clock reference from the HTU-C. The DSL Framer would generate the transmit and receive a multi-frame sync reference and feed it to the CN8228 device. The multi-frame sync signals would only be required in channelized applications where individual time slots are sourced from different devices.

Figure 3-9 illustrates only one port connection.

Figure 3-9. DSL Framer to CN8228 (ATM Phy) Interface Diagram

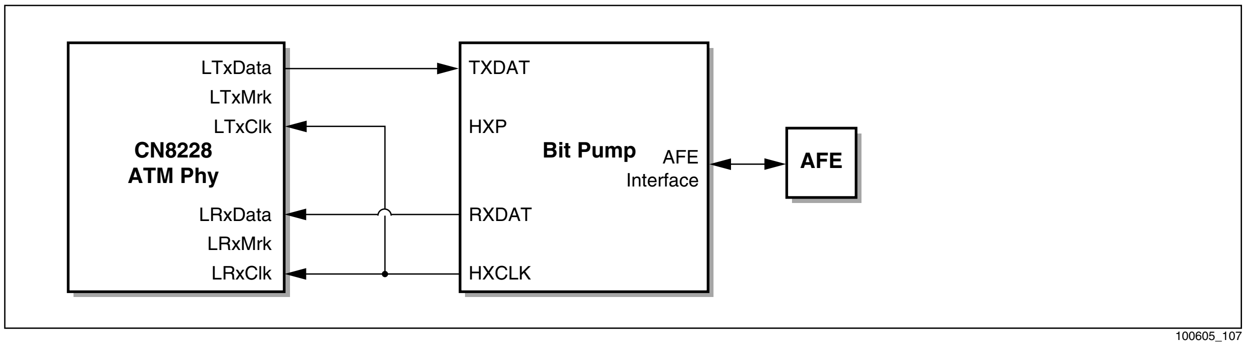


3.6 DSL Framer Bypass to CN8228 (ATM Phy) Interface

Figure 3-10 illustrates how to connect the ZipWire2 device to the CN8228 ATM Phy device when the ZipWire2 in operating in DSL Framer Bypass mode. In an HTU-C (central office) application, the bit pump transceiver would provide the clock reference. In an HTU-R (remote terminal) application, the bit pump transceiver would perform the timing recovery to recovery the clock reference from the HTU-C. The multi-frame sync references would not be connected.

Figure 3-10 illustrates only one port connection.

Figure 3-10. DSL Framer Bypass to CN8228 (ATM Phy) Interface Diagram



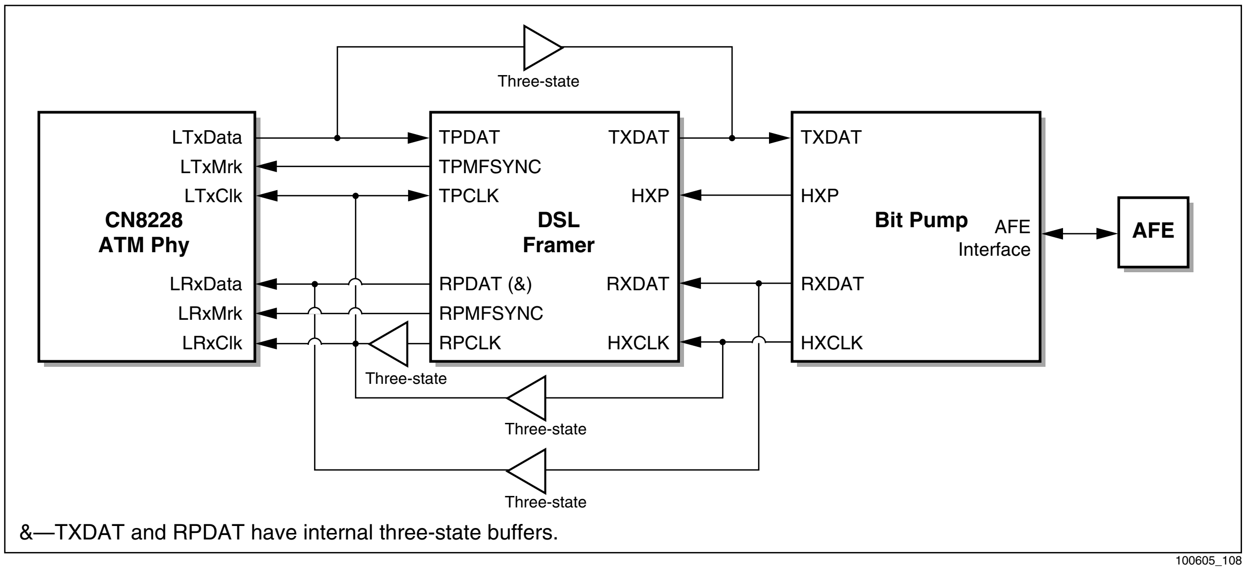
3.7 Dual Mode CN8228 (ATM Phy) Interface

Figure 3-11 illustrates how to connect the ZipWire2 device to the CN8228 ATM Phy device when both modes (see Figures 3-9 and 3-10) need to be supported. This interface connection would be required when the system needs to support interoperability with legacy HDSL1 applications that do not use the DSL Framer as well as HDSL2 applications that use the DSL Framer.

The DSL Framer has the ability to three-state the TXDAT and RPDAT signals. The other common signals must be externally three-stated to prevent contention. A multiplexer can be used instead of a three-state buffer.

Figure 3-11 illustrates only one port connection.

Figure 3-11. Dual Mode CN8228 (ATM Phy) Interface Diagram

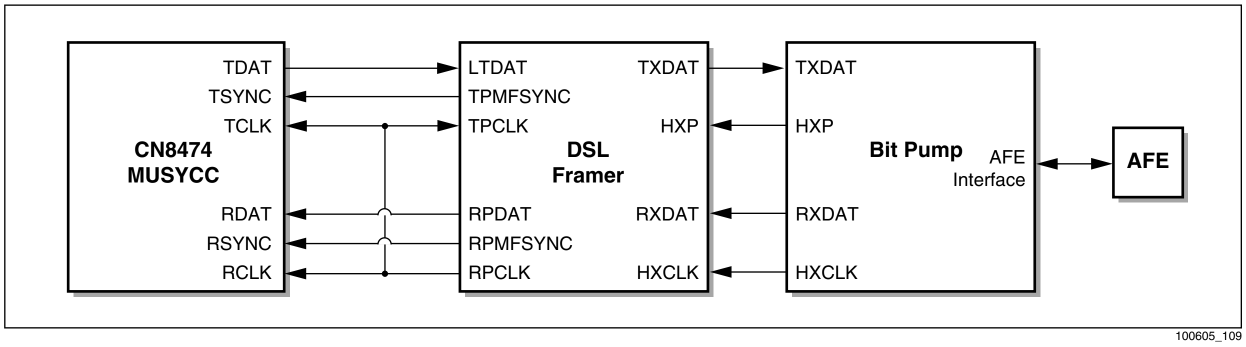


3.8 DSL Framer to MUSYCC Interface

Figure 3-12 illustrates how to connect the ZipWire2 device to the CN8474 Multichannel Synchronous Communications Controller (MUSYCC) device when using the ZipWire2 DSL Framer block. In an HTU-C (central office) application, the DSL Framer DPLL would be programmed to open loop mode to provide the clock reference. In an HTU-R (remote terminal) application, the DSL Framer DPLL would be programmed to closed loop mode to recover the PCM clock reference from the HTU-C. The DSL Framer would generate the transmit and receive multi-frame sync reference and feed it to the CN8474 device. The multi-frame sync signals would only be required in channelized applications where individual time slots are sourced from different devices.

Figure 3-12 illustrates only one port connection.

Figure 3-12. DSL Framer to MUSYCC Interface Diagram

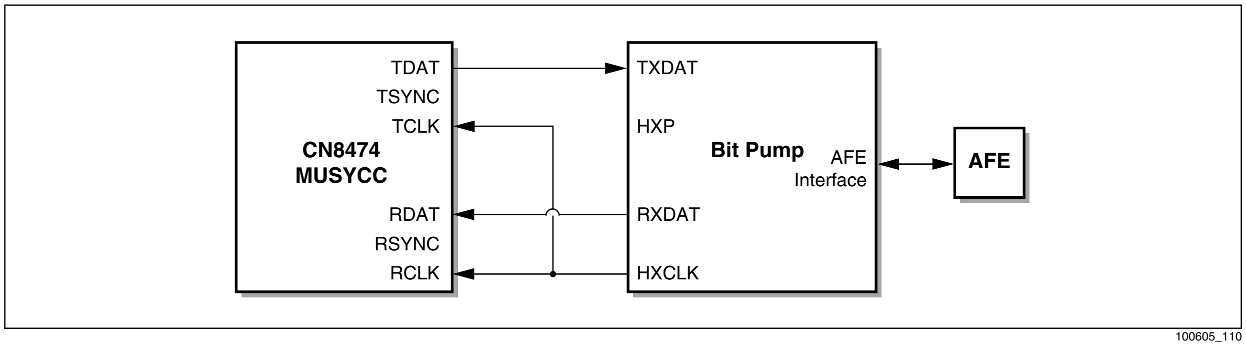


3.9 DSL Framer Bypass to MUSYCC Interface

Figure 3-13 illustrates how to connect the ZipWire2 device to the CN8474 Multichannel Synchronous Communications Controller (MUSYCC) device when the ZipWire2 in operating in DSL Framer Bypass mode. In an HTU-C (central office) application, the bit pump transceiver would provide the clock reference. In an HTU-R (remote terminal) application, the bit pump transceiver would perform the timing recovery to recovery the clock reference from the HTU-C. The multi-frame sync references would not be connected.

Figure 3-13 illustrates only one port connection.

Figure 3-13. DSL Framer Bypass to MUSYCC Interface Diagram



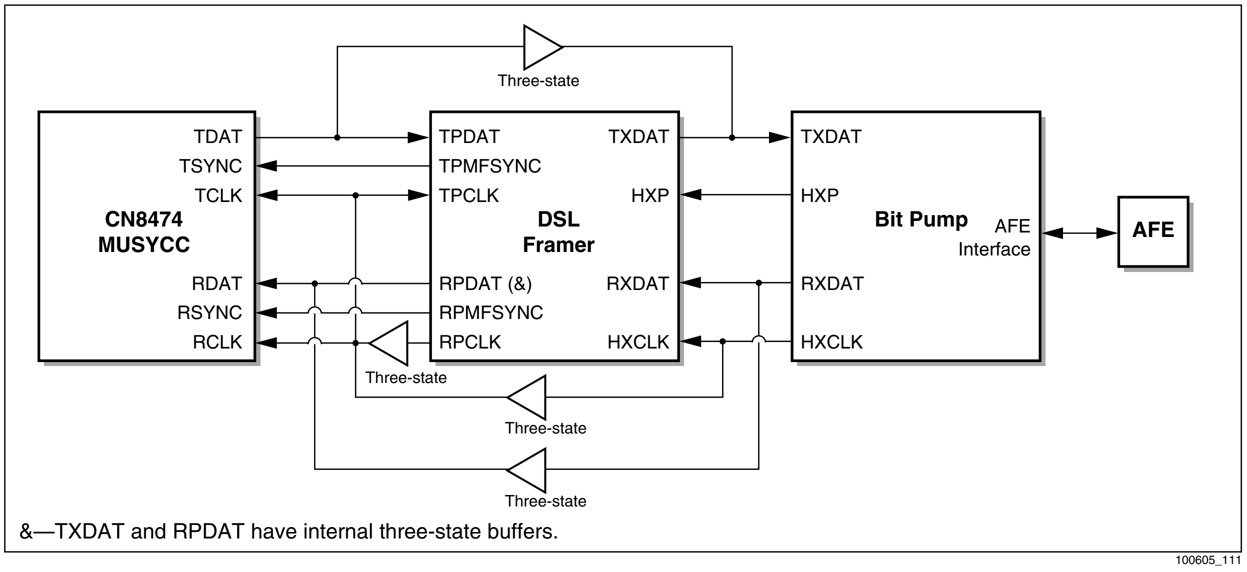
3.10 Dual Mode MUSYCC Interface

Figure 3-14 illustrates how to connect the ZipWire2 device to the CN8474 Multichannel Synchronous Communications Controller (MUSYCC) device when both modes (see Figures 3-12 and 3-13) need to be supported. This interface connection would be required when the system needs to support interoperability with legacy HDSL1 applications that do not use the DSL Framer as well as HDSL2 applications that use the DSL Framer.

The DSL Framer has the ability to three-state the TXDAT and RPDAT signals. The other common signals must be externally three-stated to prevent contention. A multiplexer can be used instead of a three-state buffer.

Figure 3-14 illustrates only one port connection.

Figure 3-14. Dual Mode MUSYCC Interface Diagram



3.11 Multiple ZipWire2 Devices to CN8228 or MUSYCC Interface

Figures 3-15 and 3-16 illustrate how to connect multiple ZipWire2 devices to the CN8228 ATM Phy or CN8474 Multi-channel Synchronous Communications Controller (MUSYCC) device. Figure 3-15 illustrates the configuration where there is one ZipWire2 device per one ATM or HDLC port. Figure 3-16 illustrates the configuration where there are multiple ZipWire2 devices bussed together per one ATM or HDLC port.

NOTE: This diagram does not show the dual mode option that supports either the DSL Framer or Framer Bypass mode.

Figure 3-15. Multiple ZipWire2 Devices to CN8228 or MUSYCC Interface Diagram

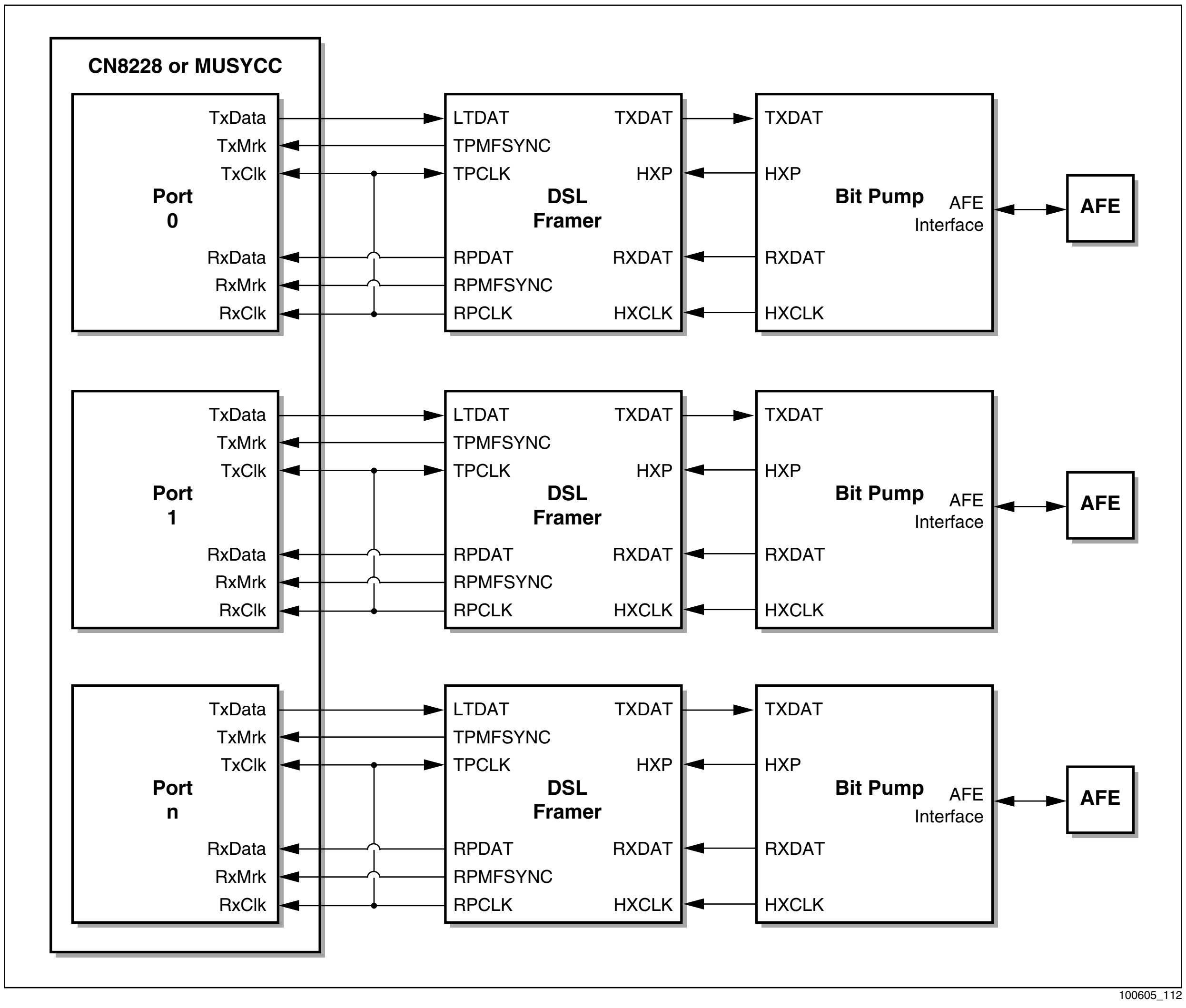
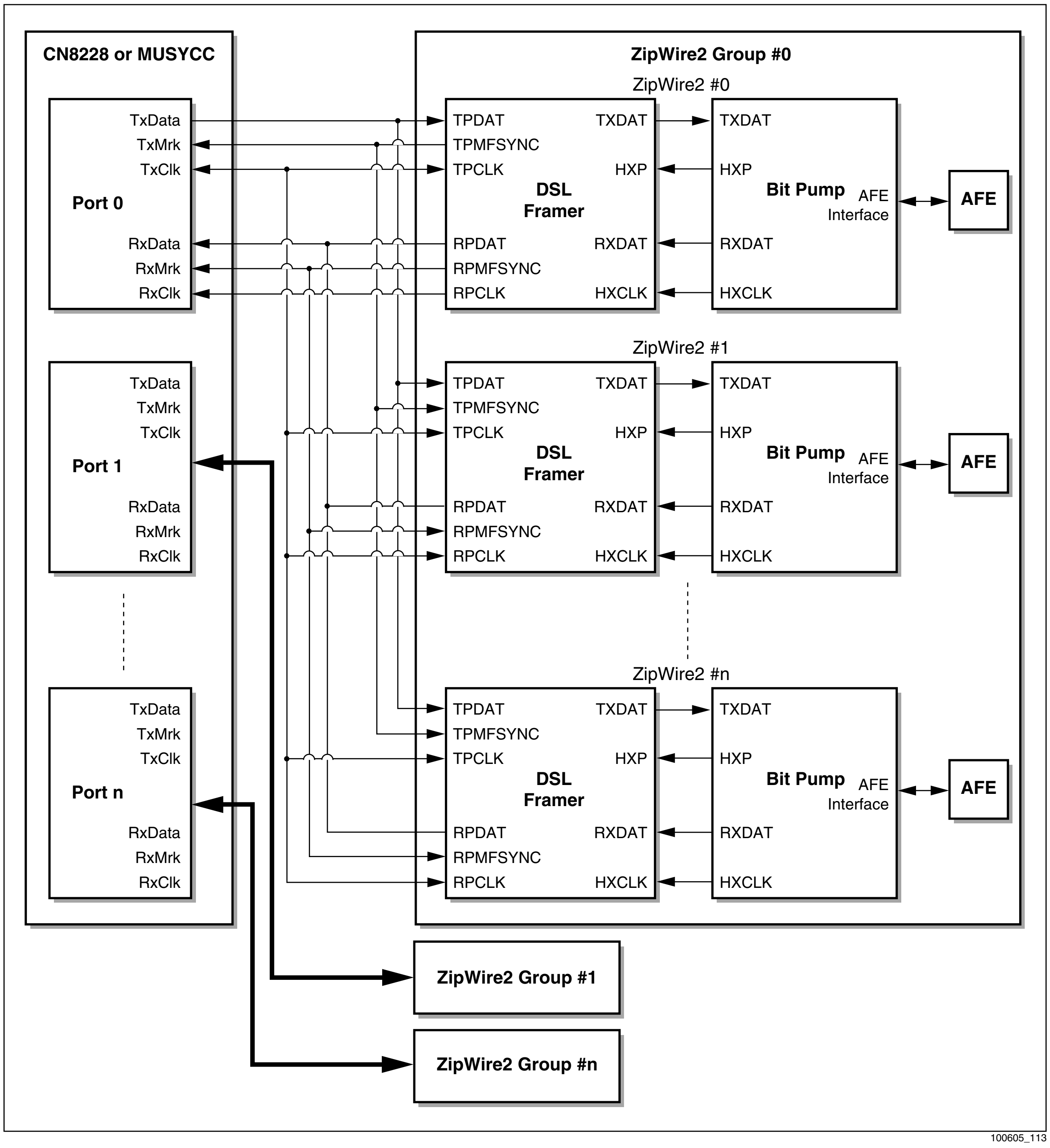


Figure 3-16. Multiple ZipWire2 Devices to CN8228 or MUSYCC Interface Diagram



3.12 Deliverables

The previous pages in this chapter provided an overview of several ZipWire2 applications. The following provides a list of the deliverables provided along with the Conexant chip set and a breakdown of the deliverables for the different ZipWire2 applications.

- Data Sheet
- Hybrid Component Values
- HEX File – For customers using a host processor and who do not need to modify the 8051 code.
- API.H – Contains API Opcodes, parameters, structures, etc.
- Source – For customers who need to modify the 8051 code. The bit pump code would be a .LIB (or .OBJ). The customers would require the DSL Manager, DSL Framer, T1/E1 Framer, and EVM source code to customize for their application. This would require the customer to purchase our recommended compiler tools.
- Application Code Examples

3.12.1 Customers who do not wish to modify 8051 code

- HEX File(s)
- Hybrid Component Values
- API.H
- Data Sheet

3.12.2 Customers who wish to modify 8051 code to control other devices

- HEX File(s)
- Hybrid Component Values
- API.H
- Data Sheet
- Source Code
- Application Code Examples
- Software Programmer’s Guide

3.12.3 Customers who wish to modify low-level DSL Framer code

- HEX File(s)
- Hybrid Component Values
- API.H
- Data Sheet
- Source Code
- Application Code Examples
- Software Programmer’s Guide
- DSL Framer Design Specification

4.0 Built-In 8051 Core Detailed Description

The ZipWire2 Transceiver/Framer has a built-in 8051 microprocessor core with the following features:

- Internal 256 bytes of direct and indirect RAM. Bytes from (0x20 to 0x2F) consist of bit-addressable bits. Bytes 0x00 to 0x1F consist of four banks for registers saving and usage during interrupts.
- Internal 2 kB non-programmable (masked) program ROM.
- Internal 8 kB data RAM.
- Dual port 1 kB Host Interface RAM.
- Two Asynchronous Serial (RS232) Interfaces.
- One Synchronous Serial Interface.
- Internal 64 kB Program RAM – lower 2 k and upper 1 k are inaccessible. Overlaid with ROM and host interface RAM during boot load sequence.
- Miscellaneous internal memory mapped peripherals.
- Programmable chip select decoder.
- Multiplex test bus interface.
- Thirteen interrupt lines with six of them being assigned to external pins.
- Three internal timers/counters.

The 8051 core executes in four oscillator cycles per instruction cycle and supports dual data pointers. The 8051 core will run at ~27 MHz but defaults to 7 MHz at power-on so that a slow external program ROM can be used.

4.1 Internal 8051 Data Space Memory Map

Table 4-1 illustrates the internal memory map of the internal 8051 data space.

Table 4-1. Internal 8051 Memory Map

Memory Range	Memory Size	Destination
0x0000 – 0x0FFF	4K	Transceiver Function Register
0x1000 – 0x2FFF	8K	DSP (Bit Pump)
0x3000 – 0x4FFF	8K	Data RAM
0x5000 – 0x6FFF	8K	DSL Framer
0x7000 – 0xAFFF	16K	Generic Chip Select #6
0xB000 – 0xEFFF	16K	Generic Chip Select #7
0xF000 – 0xFBFF	3K	Reserved
0xFC00 – 0xFFFF	1K	Host Port RAM

4.2 Internal 8051 Interrupt Mapping

All interrupt pins are configured as level triggered.

The Internal/External are describing which interrupt lines go to physical pins on the ZipWire2 device rather which interrupt lines are internal/external to the 8051 core. That is, that column is relative to the ZipWire2 Device and not the 8051 core.

Internal 8051—Internal to the 8051 core; does not come from other blocks of the ZipWire2 Device.

Internal Device—Internal to the ZipWire2 device, but external to the internal 8051.

External—External to the ZipWire2 device, connected to a pin.

In the description column, the external interrupt #n references that it is external to the internal 8051.

Table 4-2. Internal 8051 Interrupt Mapping

Interrupt #	Level	Internal External	Description
0	Active Low	Internal Dev	External Interrupt #0, DSP
1	Active Low	Internal 8051	Timer #0
2	Active Low	Internal Dev	External Interrupt #1, DSL Framer (ZipWire2 Framer)
3	Active Low	Internal 8051	Timer #1
4	Active Low	Internal 8051	Serial Port #0—RS232
5	Active Low	Internal 8051	Timer #2
6	Active Low	Internal 8051	Power Fail
7	Active Low	Internal 8051	Serial Port #1—Group Talk
8	Programmable	External	External Interrupt #2, User Definable
9	Active Low	Internal Dev	External Interrupt #3, Host Port RAM (*INTR_8051)
10	N/C	N/C	External Interrupt #4
11	N/C	N/C	External Interrupt #5
12	N/C	N/C	Watch-Dog, not implemented

4.3 ZipWire2 Transceiver Function Registers

The ZipWire2 Transceiver Function Registers (TFR) are memory-mapped in the data RAM space. These registers are used to control the memory map of all the chip selects, read input pins, control output pins, and control 8051 clock speed. The address range is fixed at 0x00–0xFF.

4.4 Configuration Pins

There are four separate sets of pins that can be used to control the software flow and configuration:

- START[5:0]—6-Bit Startup Command Input
- DEVADR[2:0]—3-Bit Device Address Input (see Section 4.6)
- BOOTOP[3:0]—4-Bit Boot Mode Input (see Section 4.6)
- DIP_SW[7:0]—8-Bit General Purpose Input (see Chapter 9.0 for details)

NOTE: In the ZipWire2 EVM, the 8051 microprocessor CS7 (generic chip select) is tied to a DIP switch which is then used to determine the system configuration, loopbacks, test modes, and so on. For customers who have a host microprocessor, it is recommended that this feature is implemented by using the Host Port API commands to generate various test modes, loopbacks, and so on. For customers who use the ZipWire2 without a host processor, the DIP Switch is dependent on whether they need to have the 8051 drive a different device.

Table 4-3. START Bit Definitions

START Bit	Description	Bit(s) Definition
0	DSL System State	0 = Disabled (Out-of-Service) 1 = Enabled
1	DIP Switch Present	0 = No 1 = Yes
2	E1/T1 Framer Present	0 = No 1 = Yes
4:3	Terminal Type	0 = HTU-C 1 = HTU-R 2 = Reg-C 3 = Reg-R
5	—	—

4.4.1 8051 Timer/Counter Description

Table 4-4. Internal 8051 Timers/Counters

Timer	Interval	Usage
Timer 0	50 ms	General Purpose Timer
Timer 1	115.2 K	RS232/Group Talk baud rate generator
Timer 2	—	Unused

4.5 Internal 8051 Communication Interfaces

Table 4-7 lists the interfaces to communicate with external devices.

Table 4-5. 8051 Communication Interfaces

Interface	Description
Master Bus Microprocessor Interface	Used by the Internal 8051 to control other external devices, i.e., T1/E1 Framer, LEDs, and so on.
Host Port RAM Interface	Used by an embedded host processor to send API commands to the ZipWire2 system. The Host Port RAM uses a mailbox protocol to pass the API parameters.
RS232 Serial Interface	Used by an external host processor (PC or Terminal) to send API commands to the ZipWire2 system.
Group Talk Serial Interface	Used by the Internal 8051 to send API commands to the ZipWire2 slave systems.

4.5.1 Master Bus Microprocessor Interface

The master bus microprocessor interface allows the internal 8051 to control other external devices, such as the T1/E1 Framer, LEDs, and so on. The master bus interface provides a glueless interface to multiplexed and non-multiplexed devices.

When connecting to a multiplexed address bus device, the MB_AD[7:0] contain the multiplexed address and data lines. The MB_ADDR[15:8] contain the upper 8-bits of the address while the MB_AD[7:0] contain the lower 8-bits. The MB_ADDR[7:0] are not connected. The falling edge of MB_ALE indicates a valid address. The *MB_OE behaves as an active-low read strobe. The *MB_WE behaves as an active-low write strobe.

When connecting to a non-multiplexed address bus device, the MB_ALE is not used. The MB_ADDR[15:0] contain the 16-bit address bus and the MB_AD[7:0] contain the 8-bit data bus. The *MB_OE behaves as an active-low output enable data strobe. The *MB_WE behaves as an active-low write strobe.

The Master Bus Interface timing is provided in Section 18.3.4.

4.5.2 Host Port RAM Interface

The Host Port RAM interface only needs to be connected when another embedded processor needs to communicate with the ZipWire2 device. When a ZipWire2 device is configured as a Group Master, the external Host Processor connects directly to the Host Port RAM. The Host Port RAM should be treated as a simple memory device.

The details of the Host Port RAM protocol is described in Section 15.4.

The Host Port timing is provided in Section 18.3.3.

4.5.3 RS232 Serial Interface

The Internal 8051 processor can communicate with an external host processor using a standard UART interface. The physical connection includes two lines: RXD0 (ball D19) and TXD0 (ball B20). The data is transferred in an asynchronous format:

```
115200 baud, 1 start bit, 8 data bits, 1 stop bit, no
parity.
```

4.5.4 Group Talk Interface

In multi-device configurations, the Group Talk Serial interface allows the Group Master to communicate to the Group Slaves via an UART interface. The Group Talk Serial interface is similar to the RS232 interface protocol except the Group Master will broadcast an API command to all the Slaves. Then only the targeted Slave (based on the destination field) will respond to (acknowledge) the Group Master. The physical connection includes two lines: RXD1 (ball B18) and TXD1 (ball A19). The data is transferred in an asynchronous format:

```
115200 baud, 1 start bit, 8 data bits, 1 stop bit, no
parity.
```

When a ZipWire2 device is not being serviced, the Group Slave TXD1 outputs will be three-stated to avoid bus contention.

The Group Master's TXD1 output is connected to all of the Group Slave's RXD1 inputs. All of the Group Slave's TXD1 outputs are connected to the Group Master RXD1 input (see Figure 3-3).

4.6 Program RAM Download

This section provides an overview of how to download a program into the internal ZipWire2 PRAM.

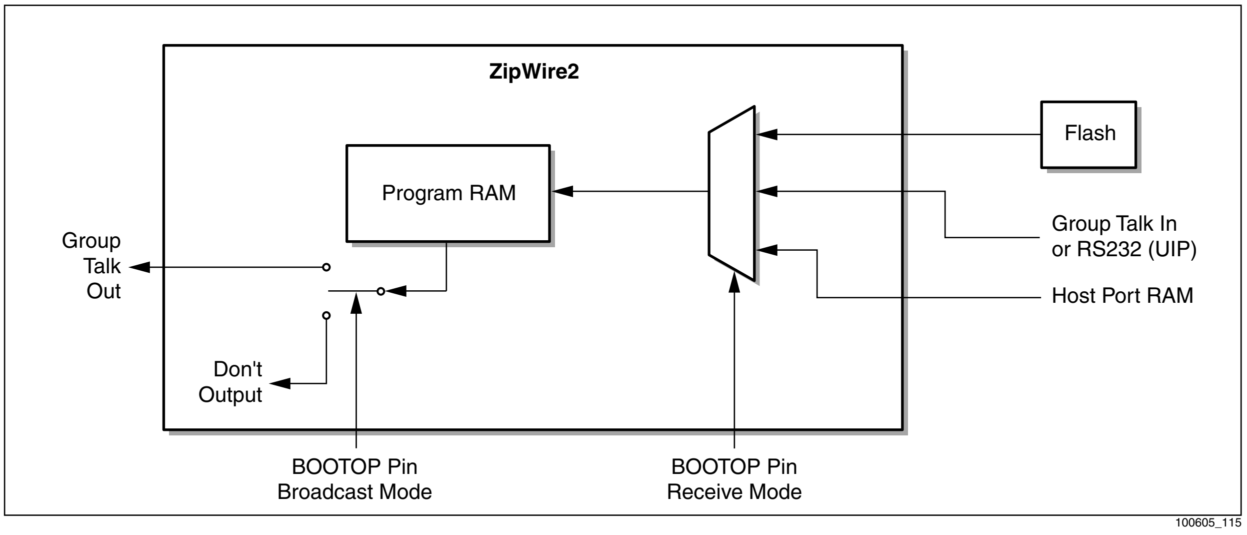
4.6.1 Summary

- Receive PRAM from an external flash (Group Master only).
  - Only one external flash required per board.
- Receive PRAM from an external host via the host port RAM interface (Group Master only).
- Receive PRAM from the Group Master via the Group Talk interface (Group Slave only).
- Receive PRAM from an external host via the RS232 Interface.
  - The User Interface Program (UIP) uses this option.
- The ZipWire2 device will be executing out of the internal boot ROM code when downloading the PRAM.
- The Group Master will be executing in operational mode when it broadcasts the PRAM contents to the slave. The slave will be executing out of the boot code.
- Be able to download to only one device at a time (allows software upgrade on a per-device basis).
- After boot load, operational code will validate download via API commands. This allows each device to perform a thorough self-test.
- Either the internal 8051 or external host processor can validate download procedures.
  - Internal 8051 can only validate one group.
  - External host processor can validate all groups.

4.6.2 Download Description

Figure 4-1 illustrates an overview of the PRAM download mechanism.

Figure 4-1. ZipWire2 PRAM Download Overview



4.6.3 Dip Switch #2—DEVADR and BOOTOP Pins

Dip Switch #2 is connected to the DEVADR and BOOTOP Pins on the ZipWire2 device. All DEVADR and BOOTOP pins are inputs.

Table 4-6. Dip Switch #2 Definition

Bit 7	Bit 6:4	Bit 3:0
Reserved	DEVADR[2:0] Pins	BOOTOP[3:0] Pins

Table 4-7. DEVADR Bit Definitions

DEVADR Bit	Master Description	Slave Description
2–0	If the host processor is responsible for validating the download, the DEVADR must be set to 0. If the internal 8051 is responsible for validating the download, then the DEVADR specifies how many devices are connected within the group. This information is used after the download is complete to validate each slave device successfully completed the download procedure. The number of devices is equal to one more than the DEVADR value—a value of 2 implies 3 devices.	Group Device ID. Must be in sequential order. If 4 devices are in a group (including the Group Master), the DEVADR bits for the 3 slave devices must be labeled 1–3.

Table 4-8. BOOTOP Bit Definitions

Bit #	Name	Description
1–0	Receive Mode	0 = Slave—Download the program RAM from the Group Talk. 1 = Reserved 2 = Master—Download the program RAM from the external Flash. 3 = Master—Download the program RAM from the Host Port.
2	Broadcast and Validate Mode	0 = Host Processor is responsible for triggering the slave download process and validating the download procedure. 1 = Internal 8051 is responsible for triggering the slave download process and validating the download procedure. The master will broadcast its program RAM contents. The DEVADR pins are used to determine the number of devices expected. This option is only intended for a single processor configuration
3	Reserved	Must set low. Used during production test.

4.6.4 Download Protocol Overview

When downloading from an external flash, the program contents are simply copied 1 byte at time from the flash to the internal 8051 PRAM.

When downloading from a host processor or Group Talk, API commands are used to download blocks of the PRAM contents. The maximum block size is 75 bytes. The host processor uses the host port RAM to transfer the data.

In a group environment, the Group Master uses the Group Talk interface to transfer the data to the slave devices. The protocol supports a broadcast mode so multiple slave devices can be downloaded simultaneously. The Group Slaves do not acknowledge any of the download API commands.

The start download API command is used to trigger the download process. This initial packet will have a data parameter of two bytes, which is the actual size of the RAM code to be downloaded. All subsequent packets will have a data parameter length of 75 bytes until the last one, which will contain only the length of byte necessary to complete the download. Upon completion of the download, the Group Master will query the status of each slave to assure that it properly received the download and is operating properly. A new download is started for slave devices not in proper operation and notification to the host is provided if this second attempt fails.

4.6.5 Download Times

4.6.5.1 Host Port RAM Interface

The host port RAM interface will take TBD seconds to complete. The exact time is a function of the internal 8051 and host processor's clock speeds and loading.

4.6.5.2 Group Talk Interface

The Group Talk interface is an asynchronous serial stream operating at 115200 baud. Each block of 75 bytes of PRAM data include 6 additional bytes of overhead. Thus a 65535-byte program would ideally take 6.1 seconds to download. However, the Group Master has to service itself and other slave devices (if they are in operational mode), so the download process will take ~10 seconds.

The exact download time is a function of the program size and the Group Master's microprocessor load.

4.6.6 Download and Device Validation

There are two aspects of validation: download and device. The download validation consists of a simple checksum that validates that the transmitted program data successfully reached the slave device. The download validation is performed during the boot code mode. The checksum byte is the last byte sent.

The device validation is a thorough self-test that qualifies the integrity of the device. The device validation is performed during the operational code. After the slave performs its self-test, the Group Master or host processor will query the self-test results to determine if the device is functional.

In a single processor environment, the Group Master, which is attached to an external flash, uses the DEVADR pins to determine how many devices are present in the system. In a dual processor environment, the host processor uses a profile registry to determine how many devices are present in the system.

4.6.7 Download Group Master Device

Each Group Master can receive the program code from either an external flash or from the host processor via the host port interface.

When receiving from an external flash, the Group Master will read the entire 60 kbyte from the flash into internal Program RAM. When receiving from the host processor, the Group Master uses the API handshake protocol to download the program contents into internal PRAM. The size of the operational code is variable and is specified in the protocol. After the Group Master has received its Program RAM contents, the device switches to operational code.

The Download and Broadcast Mode pin (BOOTOP #2) determines the Group Master flow after it switches to operational mode. When the BOOTOP #2 pin is low, the host processor is responsible for triggering the Group Master to broadcast the Program RAM to the slave devices. The Group Master can either transmit its current Program RAM contents or transmit a separate Program RAM content. The host processor is then responsible for validating each slave device downloaded successfully. The Host Processor uses the _DSL_VERSIONS (0x8A) API command to determine if the slave download was successful.

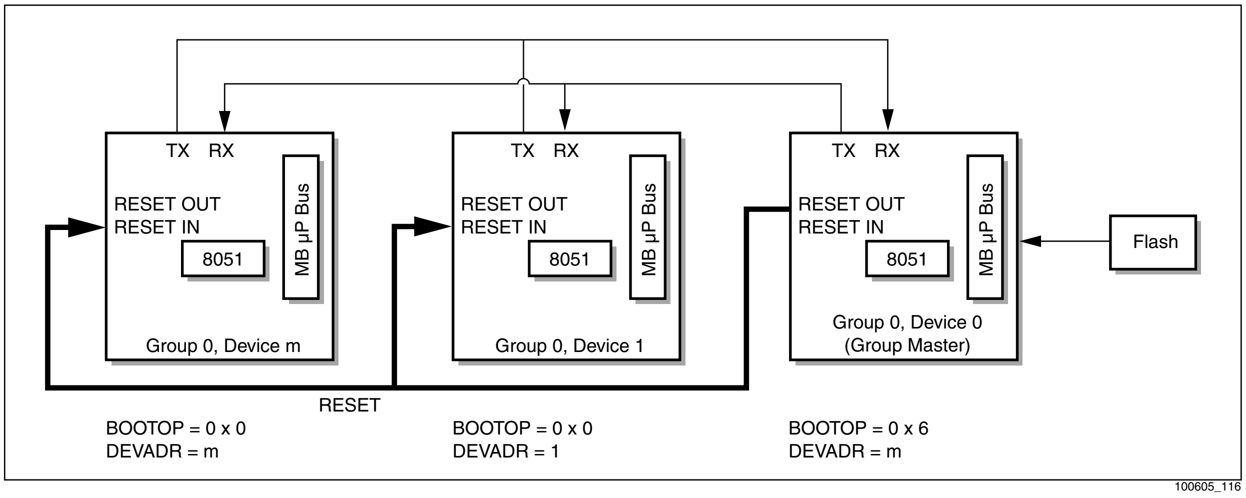
4.6.8 Download Group Slave Device

Each Group Slave can only receive the program code from the Group Talk interface. The Group Slave waits for the Download Start (Length) API to indicate that the download has started. Each block of data is transferred using the Download Data API command. The Group Slave maintains a running checksum of the data. When the Download Complete (Checksum) command is received, the Group Slave compares the received checksum to its computed checksum to determine the validity of the data download. If the checksum fails, the Group Slave performs a software reset and waits for another download. If the checksum passes, the Group Slave switches to operational code where it can respond to additional API commands.

4.6.9 Download Single Processor Configuration

Figure 4-2 illustrates the single processor configuration with the appropriate DEVADR/BOOTOP pin settings. The external flash is connected to the Group Master. In this application, all devices have the same program.

Figure 4-2. Single Processor Configuration



After power-on (or reset), the Group Master and Slave execute from the internal ZipWire2 boot code. The Group Master will read the contents from the flash into internal program RAM while the slaves await the download start API command. After the Group Master finishes downloading the code from external flash, it switches to operational mode. While in operation mode, the Group Master polls the DEVADR and BOOTOP pins to determine if it needs to download the code to a slave device.

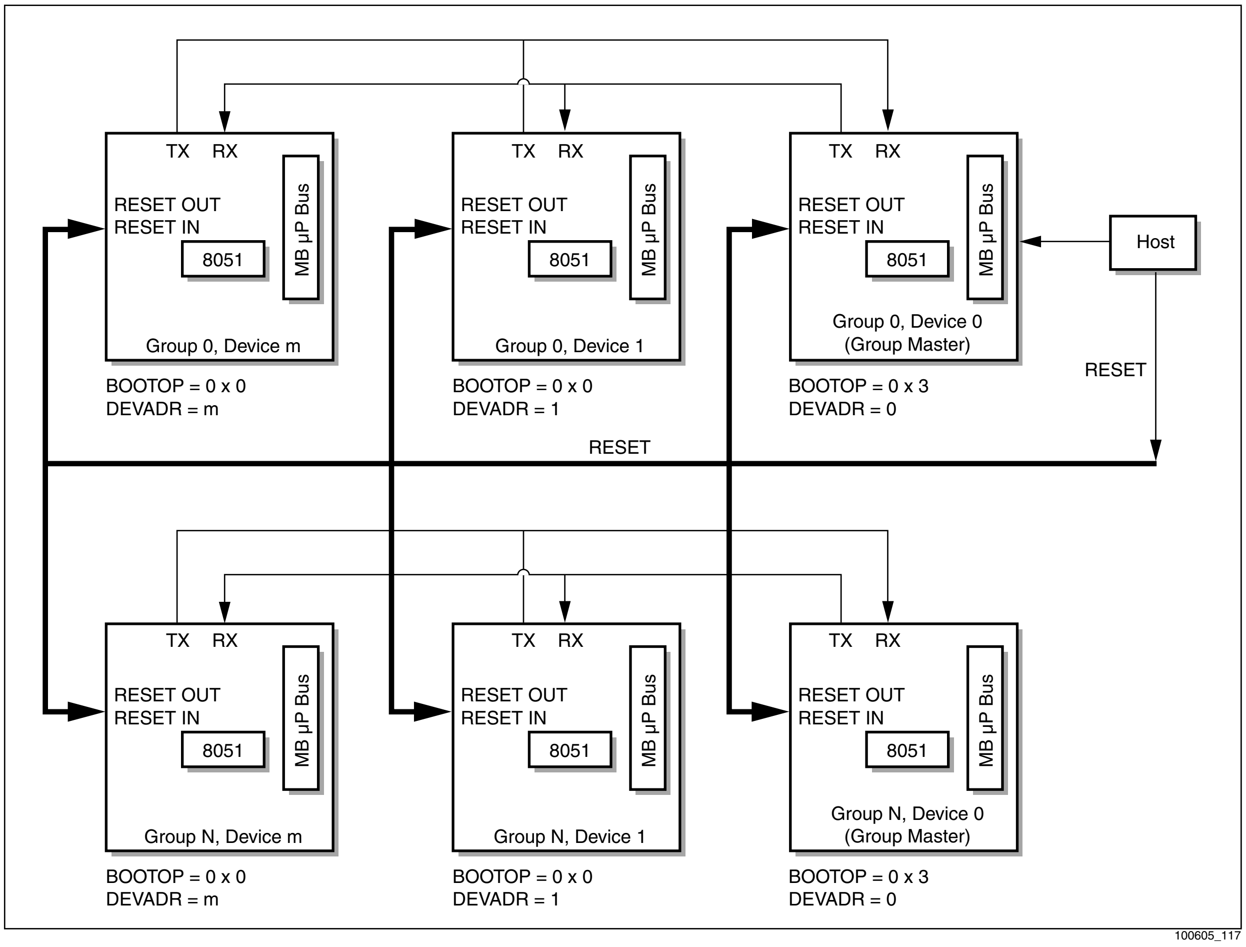
If the DEVADR is 0 (one device only) or the BOOTOP Broadcast and Validate Mode bit is not set, the master will not broadcast its program RAM contents to the slave devices; otherwise the download is performed using the Group Talk interface. After the slave download is completed, the slave devices perform a download validation. If successful, the slave devices switch to operational mode, otherwise the failed slave device performs a software reset and waits for another download.

When in operational mode, the slave devices perform a self-test. The master queries each slave device using the Group Talk protocol to confirm if the device is functional. The master only queries the number of slave devices based on the DEVADR pins. If the master detects that a slave did not successfully download, then the master will issue an *RST_OUT to force the slaves to reattempt the download for all slave devices. The master detects a failed slave device if a time-out occurs during the Group Talk status request.

4.6.10 Download Dual Processor Configuration

Figure 4-3 illustrates a dual processor configuration with the appropriate DEVADR/BOOTOP pin settings. The host processor is connected to each Group Master. In this application, each device can have a unique program. The Group Master only downloads to devices within its group.   There are no intergroup dependencies.

Figure 4-3. Host Processor Configuration



After power-on (or reset), the Group Master and Slave execute from the internal ZipWire2 boot code. The Group Masters will receive the internal program RAM contents from the host processor while the slaves await the start download API command. After each Group Master finishes downloading the code from the host processor, they switch to operational mode. While in operational mode, the host processor will send an API command to one of the Group Masters to initiate the download to the slaves. The host processor can either tell the Group Master to download its current program RAM or to broadcast a separate program. The host processor can select the master to download the code to all devices (broadcast) or can select one slave. All slave devices would receive the same program code.

When the Group Master is broadcasting its current program RAM, the host processor issues the _DSL_VERSIONS API command to determine when the download is complete. When downloading a separate program, the host processor has to manually feed the program contents so it already knows when the download is completed.

After the slave download is completed, the slave devices perform a download validation. If successful, the slave devices switch to operational mode, otherwise the failed slave device performs a software reset and waits for another download. When in operational mode, the slave devices perform a self-test. The host processor queries each slave device using an API command to confirm if the device is functional. The host processor only queries the number of slave devices based on its profile registry. If the host processor detects a slave did not successfully download, then the host processor can reissue the download command. Only the failed slave devices will reperform the download. The successful slave devices will ignore any download API commands because they are already in operational mode.

4.6.10.1 Redundant Download Group Master

In the event one Group Master incurs a physical defect, one of the slave devices can be redefined as the Group Master. The board must have the host processor microprocessor bus connected to the second slave device. In addition, the Group Talk transmit and receive signals must be swapped for the new Group Master.

4.6.10.2 Device Uniqueness

There are two possible configurations to allow each ZipWire2 device to have a unique program.

Group Master/Slave Configuration

This configuration requires the master and slave devices to be in operational mode. When programming a device with a new code, the host processor would issue the hardware reset API command to have that device reperform the boot procedure. The master can now download the new code. Because the other devices are in operational mode, they will ignore the download API commands.

All Group Masters Configuration

This configuration requires each device to be connected to the host processor via the host port interface. The host can program each ZipWire2 device independently. The Group Talk is not used in this configuration.

4.6.10.3 Group Uniqueness

Each group of ZipWire2 devices can have a unique PRAM. Each device within the group has the same PRAM. The host processor can download a unique PRAM to each master, which is then broadcast to each of its slaves.

4.6.11 Download API Commands

This section provides a summary of the download API commands. Refer to Chapter 17.0 for details on each command.

4.6.11.1 Boot Code API Commands

The following API commands are supported while the ZipWire2 device is in the boot code. During operational mode, the Group Slave devices will ignore all of the download API commands. The Group Master can forward these commands to slave devices; however, the Group Master will ignore these commands if the destination is for itself.

Table 4-9. Boot Code API Commands

Command	Opcode	Data Length	Description
Reset	0x00	1	This command is used to reset the ZipWire device.
Download Start (Length)	0x53	2	This command is used to begin a new download. The size of program code (length) is used to validate the download procedure. The low byte is sent first.
Download Data	0x54	Up to 75	This command transfers the next block of the program data.
Download End (Checksum)	0x55	1	This command is used to indicate the end of the download. The download data checksum is passed in so the 8051 can validate the download contents. See Section 4.6.11.3.

4.6.11.2 Operational API Commands

During operational mode, the Group Master supports one additional API command that allows the Group Master to download its current PRAM contents to the slave devices. For certain applications, this offloads the host processor from having to download the slave devices.

Table 4-10. Operational API Commands

Command	Opcode	Data Length	Description
Download Slave	0x56	1	This command is used to trigger the Group Master to download its PRAM contents to the slave devices.

4.6.11.3 Program RAM Checksum

The PRAM checksum uses the following formula to determine the checksum.

~ (Byte 1 + Byte 2 + … + Byte L) + 1: Sum all of the bytes and take the 2’s complement.

4.6.12 Download Examples

In this example, the application has 8 devices in a group (labeled device 1–8). The Group Master (device 1) is connected to the host processor using the host port RAM interface. The 7 other devices are connected on the Group Talk interface. Devices 1–4 require the HDSL2 program while devices 5–8 require the HDSL1 program.

The following steps list the sequence to perform the download to the ZipWire2 devices. Minimal error checking is provided in this example.

1. Power-on (or reset) all devices.
2. Host processor downloads HDSL2 program to Group Master.
3. After download is complete, host waits for Group Master to reach operational mode and validates download.
4. Host issues the Download Slave API command to the Group Master. The Group Master then downloads the HDSL2 (current PRAM) code to all of the slave devices.
5. Host waits for download to complete and validates download.
6. After download is complete, host issues a reset to devices 5–8.
7. Host issues the Download Start, Download Data, and Download End API commands with the broadcast destination (0x0F) to download the HDSL1 program to devices 5–8. Devices 1–4 will ignore these API commands.
8. Host validates download.

5.0 ZipWire2 DSP Detailed Description

This section provides a detailed description of the DSP block of the ZipWire2 chip set.

5.1 ZipWire2 Clocks

The ZipWire2 DSP block is responsible for generating the DSP, Microprocessor, AFE, and Framer reference clocks. The ZipWire2 Framer has a DPLL to generate the PCM receive clock.

Figure 5-1. ZipWire2 Clocks

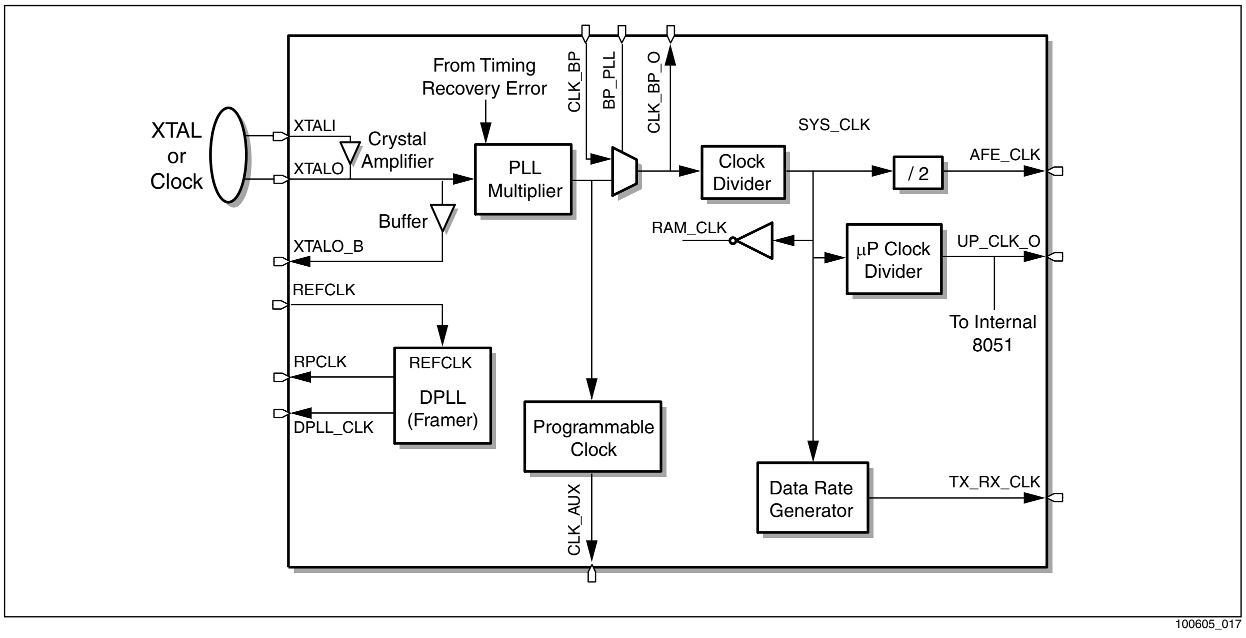
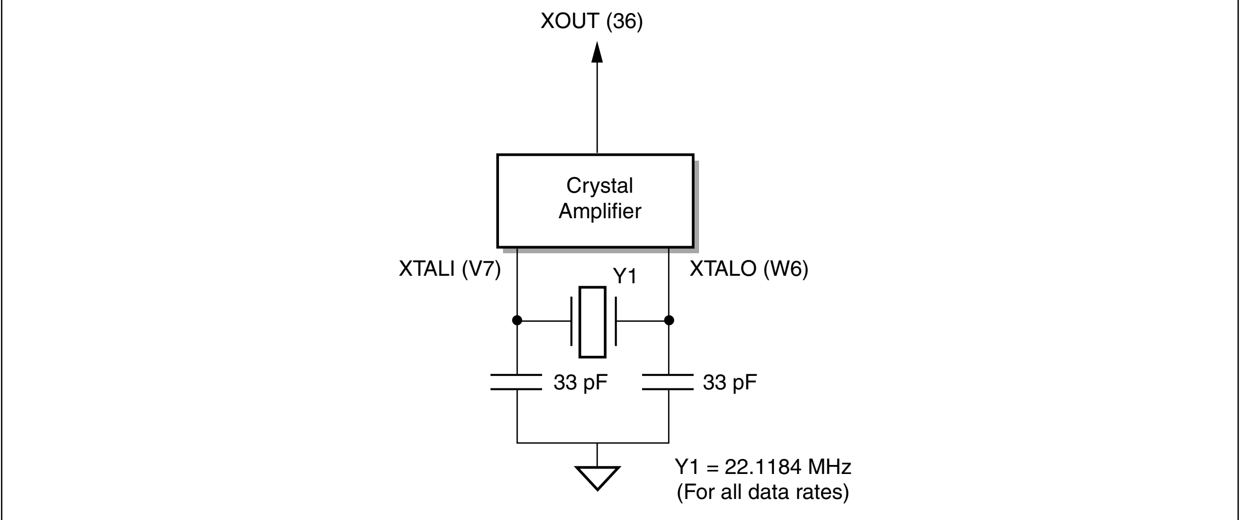


Figure 5-2. Crystal Interface



NOTE(S):
The value of the capacitors may vary, depending on the PCB capacitance.

100605_057

Table 5-1. ZipWire2 Clocks

Clock or Node	Frequency	Description
XTAL	22.1184 MHz	External crystal or clock input
XTALI	22.1184 MHz	Crystal input
XTALO	22.1184 MHz	Crystal output
XTALO_B	22.1184 MHz	Buffered crystal output
CLK_BP	54 MHz	External high speed reference clock input
CLK_BP_O	54 MHz	High speed reference clock output
BP_PLL	Control Pin	Selects either internal PLL clock (0) or CLK_BP pin (1)
SYS_CLK	54 MHz	Internal system clock
RAM_CLK	54 MHz	Inverted system clock, controls RAM timing
AFE_CLK	27 MHz	AFE clock reference
8051_CLK	6–27 MHz	Internal µP clock, SYS_CLK divide by 2, 4, 6, or 8
CLK_UP_O	6–27 MHz	External µP clock output, SYS_CLK divide by 2, 4, 6, or 8
TX_RX_CLK	144–2320 kbps	Data rate clock output
CLK_AUX	TBD	Programmable clock

6.0 ZipWire2 Framer Detailed Description

Figure 2-2 provides the detailed block diagram of the DSL Framer block. This section provides a detailed description of the various modules of the DSL Framer block.

6.1 Distinguishing Features

- Supports all legacy features of Bt8953A and RS8953B
- Compliant with *ETSI RTS/TM-06008* [1]
- 1, 2 or 3 pair T1/E1 ETSI and Bellcore standard application
- PCM Interface up to 8 MHz
- ISDN Primary Rate Access (PRA)
- Custom N x 64 over 1, 2 or 3 pairs
- Asymmetric PCM rate and frame format capability
- Various rates PCM clock recovery (64 kHz up to 8 MHz)
- Low jitter (wander) stuffing generator
- Flexible Stuff Bit ID (SBID) mapping, including majority vote decision (HDSL2 applications)
- Three programmable PCM and DSL sync detectors (supports grouped and spread sync word patterns)
- Two programmable PRBS/BER meters to both PCM and HDSL sides
- 12 programmable performance monitoring counters (can be used for CRC, BPV, or FEBE error counters)
- 3 programmable CRC generators
- Programmable scrambler/descrambler
- Supports variable time slot size (8, 4, 2, or 1) and therefore variable PCM custom frequency (N x 64, N x 32, N x 16, and N x 8, respectively)

6.2 Common Functions

6.2.1 DATA FIFO

The DSL Framer contains two DATA FIFOs: TX_FIFO and RX_FIFO. These FIFOs are used to provide rate buffering between the PCM side data rate and the DSL side data rate. Each FIFO is capable of storing 512 bits (two E1 frames).

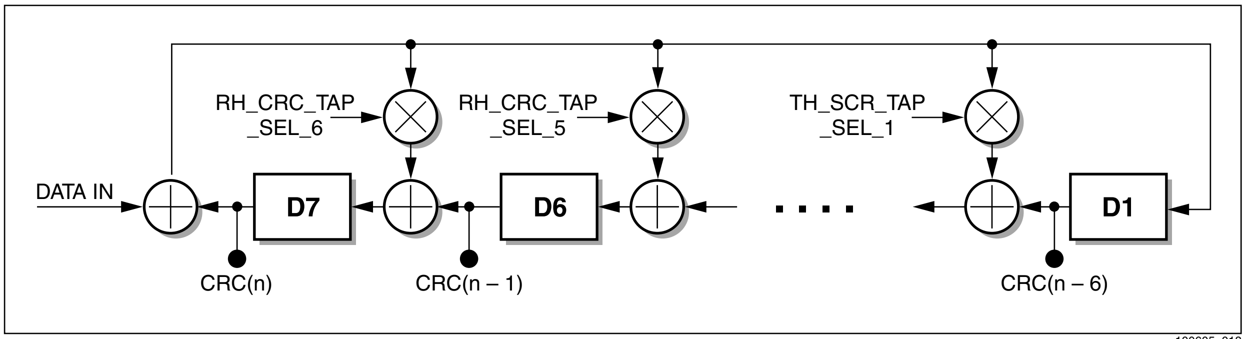
6.3 HDSL Section

6.3.1 General HDSL Functions

6.3.1.1 CRC Generator

A generic CRC generator with selectable taps (up to 7th order) is implemented.

Figure 6-1. Generic CRC Generator

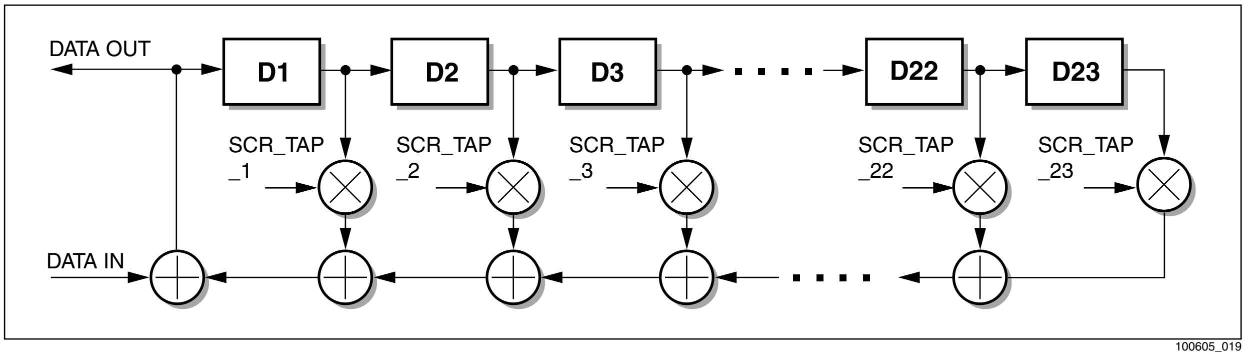


CRC calculation can be corrupted for debugging purposes. This mode simply inverts the CRC calculation.

6.3.1.2 Scrambler/ Descrambler

The Scrambler/Descrambler operation can be bypassed for debugging.

Figure 6-2. Generic Scrambler Generator

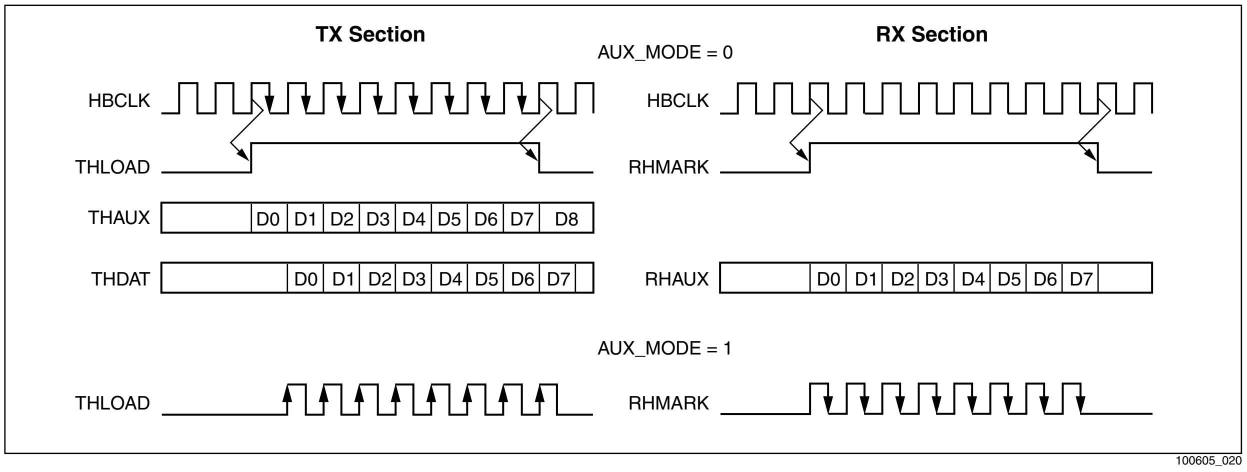


6.3.1.3 Auxiliary Channel

The HDSL Auxiliary Channel (THAUX, RHAUX) provides an alternate source of HDSL payload. This channel can support any payload size and optionally can function as an alternate source for the Z-bits or any other selected overhead. Figure 6-3 illustrates the HDSL Auxiliary Channel Timing.

The Auxiliary Channel interface has two operational modes. In the first mode, THLOAD and RHMARK signals simply mark high during auxiliary input mode. The second mode generates gated clock in pins THLOAD and RHMARK during Auxiliary mode to clock the serial device directly. This mode prevents additional glue logic in the interface between DSL Framer and the serial device.

Figure 6-3. HDSL Auxiliary Channel Timing



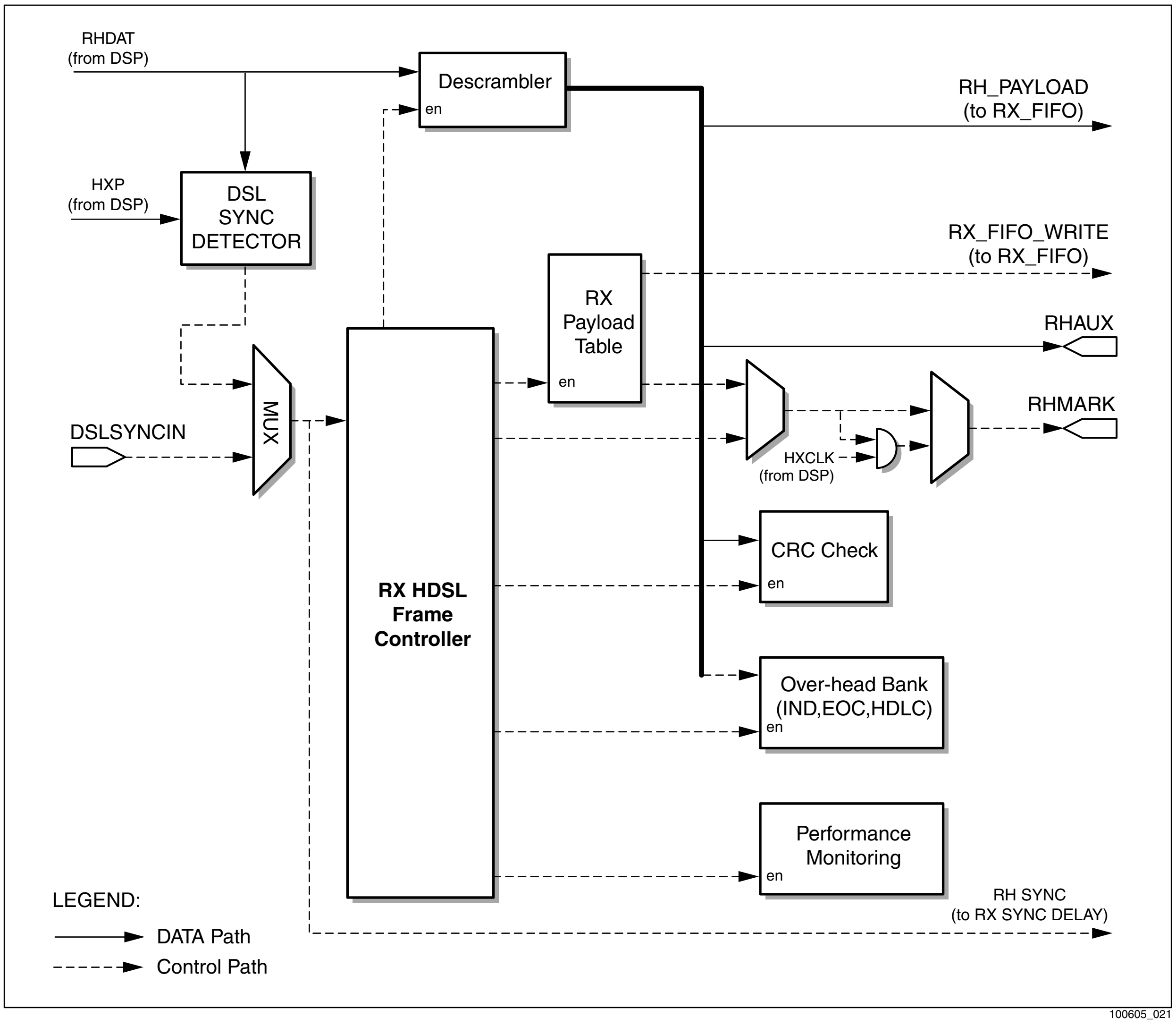
6.3.1.4 RX DSL Reference Phase Measurement

While working in Multi-Pair configuration, the DSL Framer is capable of measuring the receiving DSL phase difference between two pairs. This phase is mainly used to determine the delta delay between two HDSL channels in point to multi-point application (can be also used for debugging or link delay measurement).

6.3.2 HDSL Receiver Functionality

Figure 6-4 illustrates a detailed block diagram of the HDSL Receiver Section.

Figure 6-4. HDSL Receive Section Block Diagram



6.3.2.1 DSL Sync Detector (DSD)

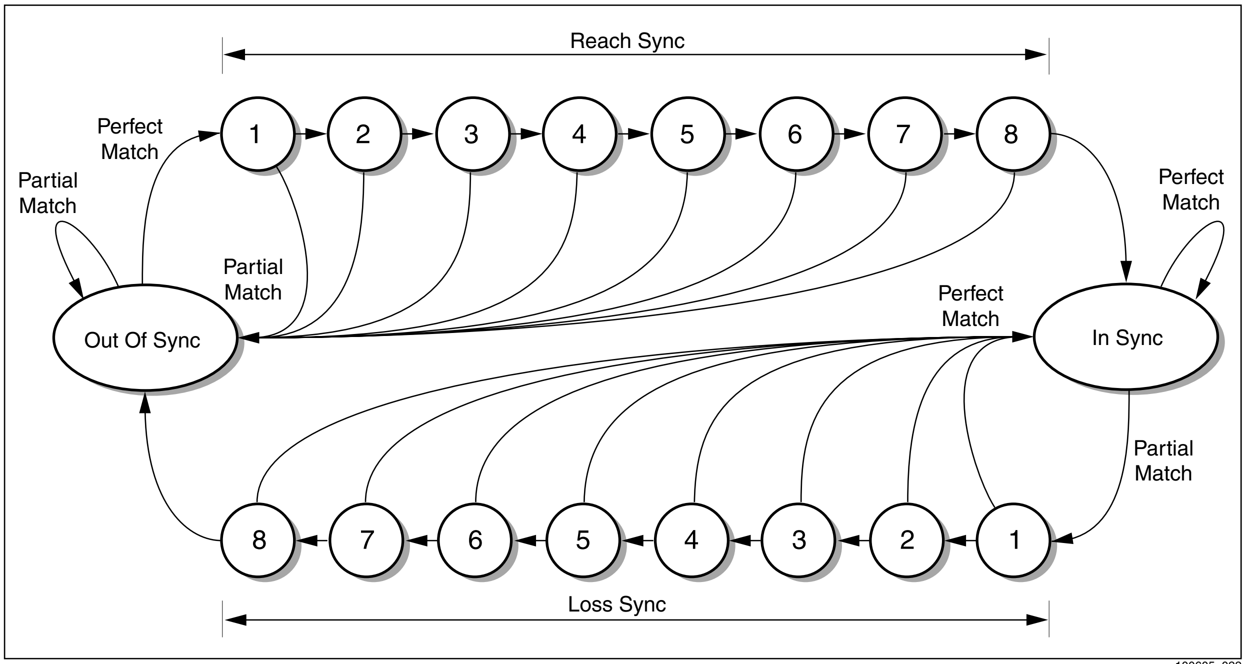
The DSL Sync Detector (DSD) acquires and maintains synchronization of the HDSL.

To support the wide variety of frame formats, the DSD is designed in a flexible way which provides the following capabilities:

- Synchronized to any grouped bit sync pattern up to 16 bits long.
- HDSL frame size (Nominal) can be up to 2^{16} (65536) bits long.
- Stuff size can be 2, 4, 6, or 8 bits. For application without the necessity for stuff bits (HDLC applications), the DSD can search for the sync word without searching in variable frame length, but search for fix location instead.

NOTE: The default configuration is adequate for most standard applications. For certain custom applications, this option requires modifying the low-level DSL Framer code.

Figure 6-5. DSD Synchronization State Machine



6.3.2.2 Tip/Ring Reversal Detection

During 2B1Q mode, Tip/Ring reversal is automatically detected and corrected by the DSD.

In HDSL2 applications, the Tip/Ring reversal cannot be detected by DSL Framer due to the non-symbol alignment nature of the DSP operation. In this case, the DSP is responsible for detecting and correcting Tip Ring Reversal.

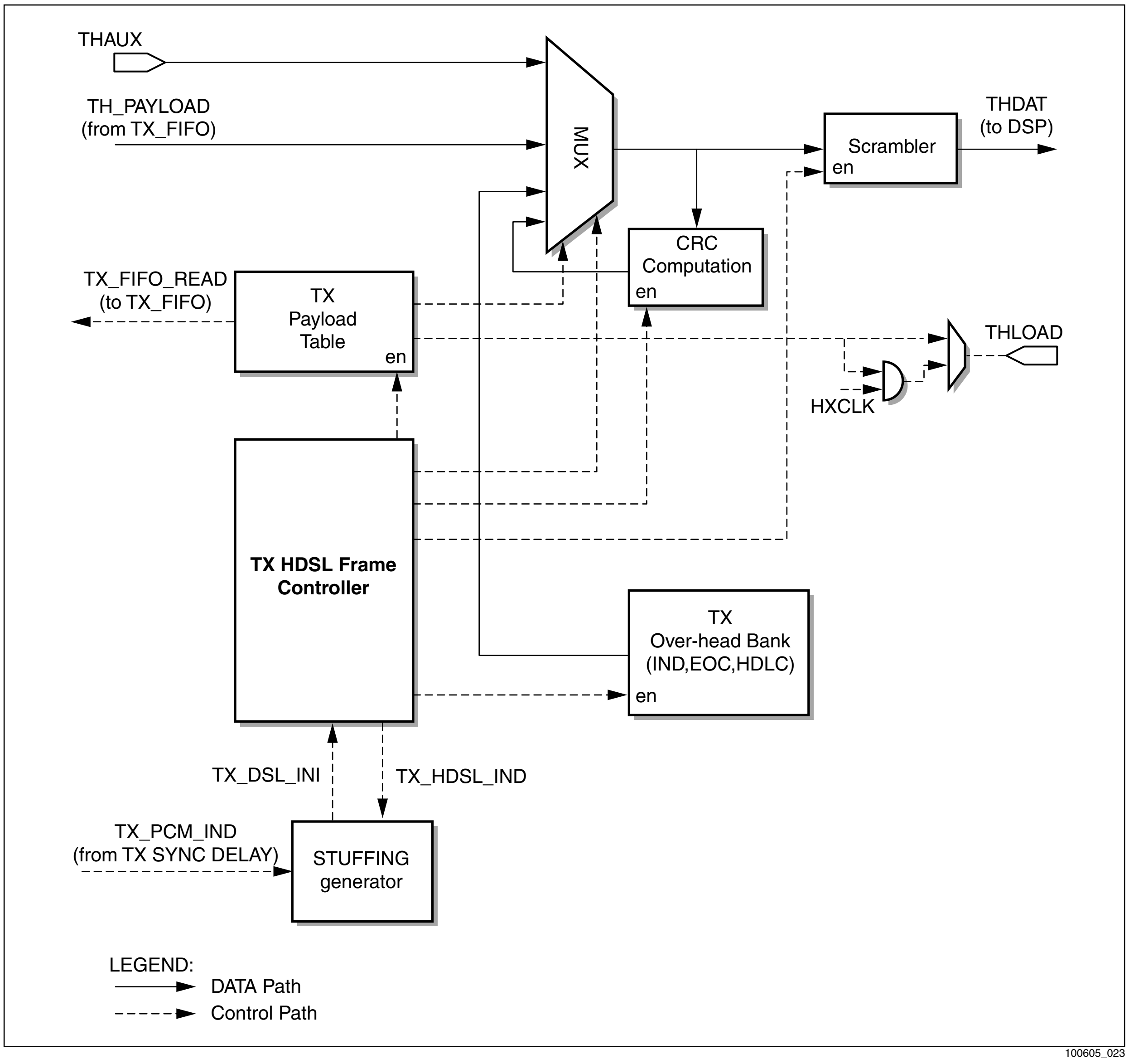
6.3.2.3 RX HDSL Payload Table

TBD

6.3.3 HDSL Transmitter Functionality

Figure 6-6 illustrates a detailed block diagram of the DSL Transmitter section.

Figure 6-6. HDSL TX Section Block Diagram



6.3.3.1 Stuffing Generator

The stuffing generator synchronizes the DSL frame period to the PCM Frame period by adding 0, 2, 4, 6, or 8 STUFF bits (0,4 in HDSL1 application) to the DSL frame period.

The stuffing generator can be bypassed for nonvariable frame length application and also as an additional debugging tool.

6.4 PCM Section

The PCM section (receiver and transmitter) is composed of two major blocks, the PCM mapper and the Layer 3 Framer.

The PCM mapper functions as a formatter which maps/extracts PCM payload data into/from the HDSL channel through the FIFO. In addition, the mapper can override the data with data bank or generate a PRBS sequence.

The Layer 3 Framer synchronizes to PCM Frame or Multi-Frame, extracts/inserts Overhead data from/into the PCM Frame, and checks for any block errors (such as CRC).

6.4.1 PCM Interface

On the PCM interface, the DSL Framer can interface to standard E1, T1, or any PCM custom N × 64 kHz frame format.

6.4.2 General PCM Functions

6.4.2.1 CRC Generator

The PCM section contains two generic CRC generators which are functionally identical to the one located on the DSL side (see section Section 6.3.1.1 for more details).

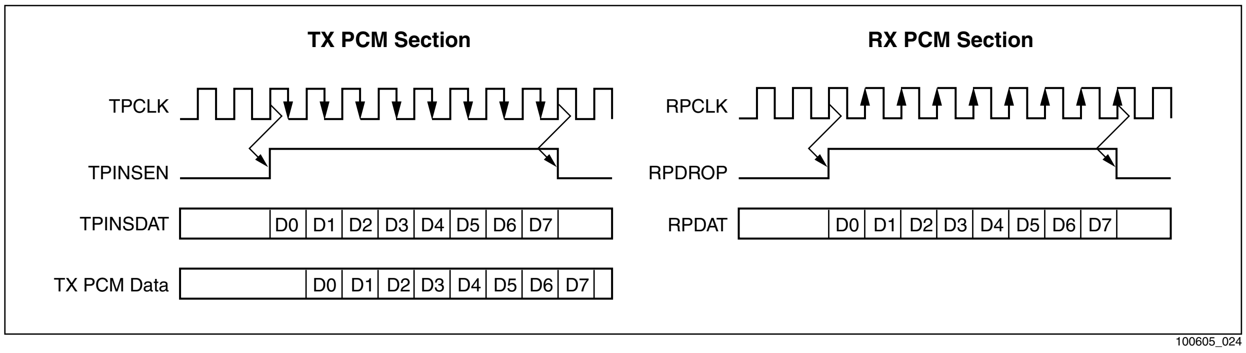
The PCM CRC calculation can be corrupted for debugging purposes. This mode simply inverts the CRC calculation. In addition, the CRC generator can be bypassed or recalculate.

Any CRC computation format can be generated. CRC computation can be disabled/enabled. There is the capability to replace any bit in the frame with either 0 or 1 for CRC computation purposes. This capability allows supporting any CRC operation method.

6.4.2.2 Insert/Drop

An alternate PCM source is fed into the PCM formatter, using TPINSEN, TPINSDAT, and RPDROP pins (see Figure 6-7).

Figure 6-7. Insert/Drop Timing Diagram



6.4.2.3 Overhead Handling

The PCM transmitter and PCM receiver can handle up to 24 OverHead (OH) bytes. These OHs can function as Sa bits, E bits, and A bits for E1 applications. They can be used to generate any in-band management.

NOTE: This option requires modifying the low-level DSL Framer code.

6.4.2.4 E1 Grooming

To support the E1 P2MP application, it is necessary to groom Channel Associated Signaling (CAS) from different sites. Each remote site has a different PCM Frame sync that needs aligning in the central site.

6.4.2.5 MF Phase Measurement

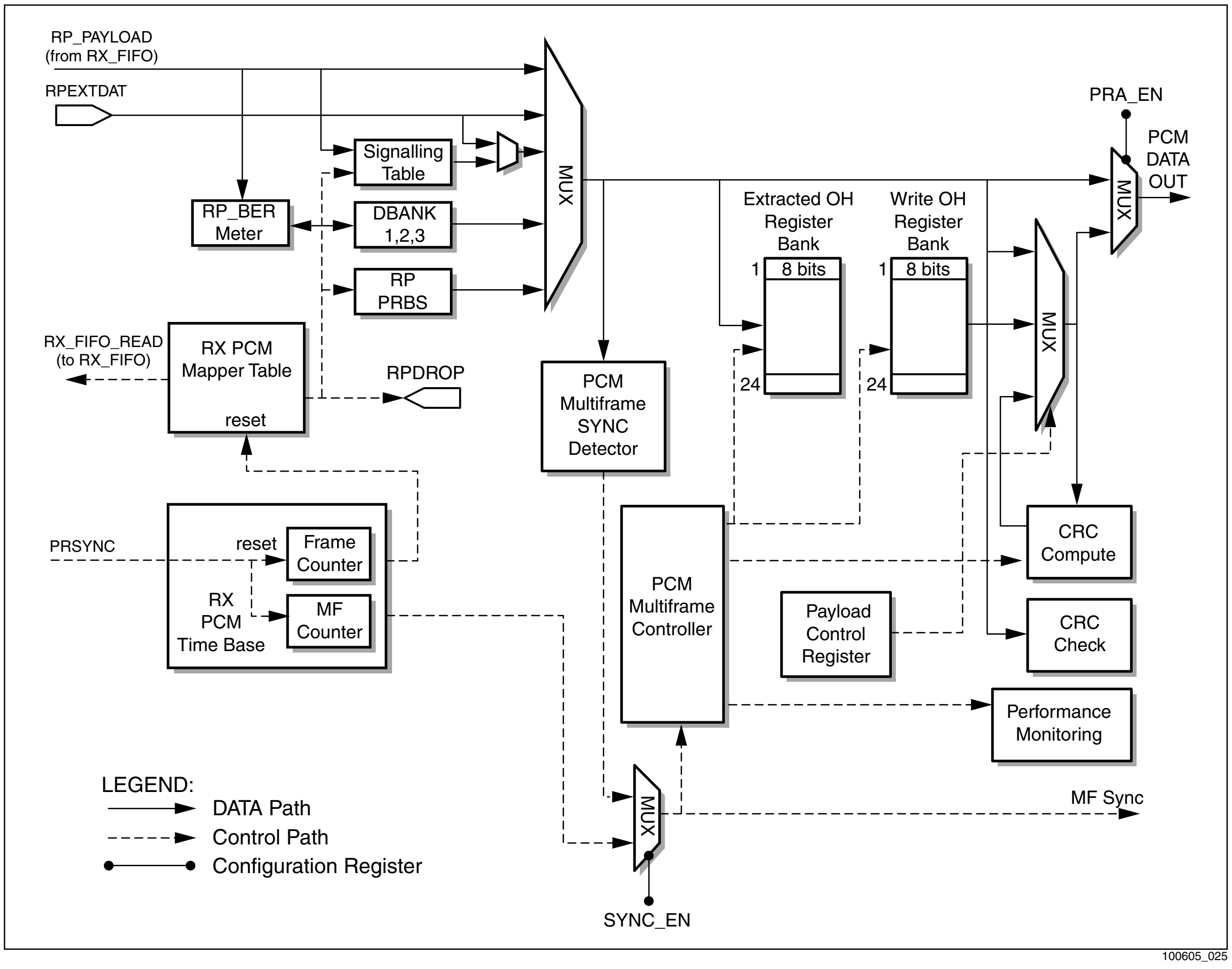
During E1 Point-to-Multi-Point application, PCM Multi-Frame phase measurement between TPMFSYNC and RPMFSYNC pins with respect to internal transmit MF Sync (MFSYNC) is necessary at the remote site to compensate for misalignment between different remote sites. This phase then can be used to internally align the HDSL transmits frame to the PCM Frame boundary in each site. It can provide the value of each site to the central office and to align the receive channel signaling to the receive E1 MF.

6.4.3 PCM Receiver

Figure 6-8 illustrates a detailed block diagram of the PCM Receiver section. The major tasks of the PCM Receiver are:

1. Generates RX PCM time base aligned with the HDSL reference (WL delay apart).
2. Assembles ongoing PCM Frame using flexible RX PCM Mapper Table. Major tasks of this table are:
   a. Assembles RX PCM Frame from selectable sources: HDSL payload, PRBS sequence, DATA BANK 1, 2 or 3, Signaling table (Grooming Mode), and RPEXTDAT input pin.
   b. Enables BER meter per time slots basis.
   c. Asserts RPDROP pin to signify specific TS's in RPDAT output.
   d. Inserts external PCM data (in RPEXTDAT input) to the receive PCM Frame. Used in Multi-Pair configuration.
3. Generates receive user interface SYNC signals such as RPMFSYNC and PREFSYNC (Multi-Pair configuration PCM time-base sync).
4. Synchronizes to any Layer 3 Frame/MF.
5. Checks CRC (Selectable) and computes CRC on the final ongoing frame.
6. Extracts overhead bits.
7. Provides capability to override each overhead bit by the MPU.

Figure 6-8. PCM Receiver Block Diagram

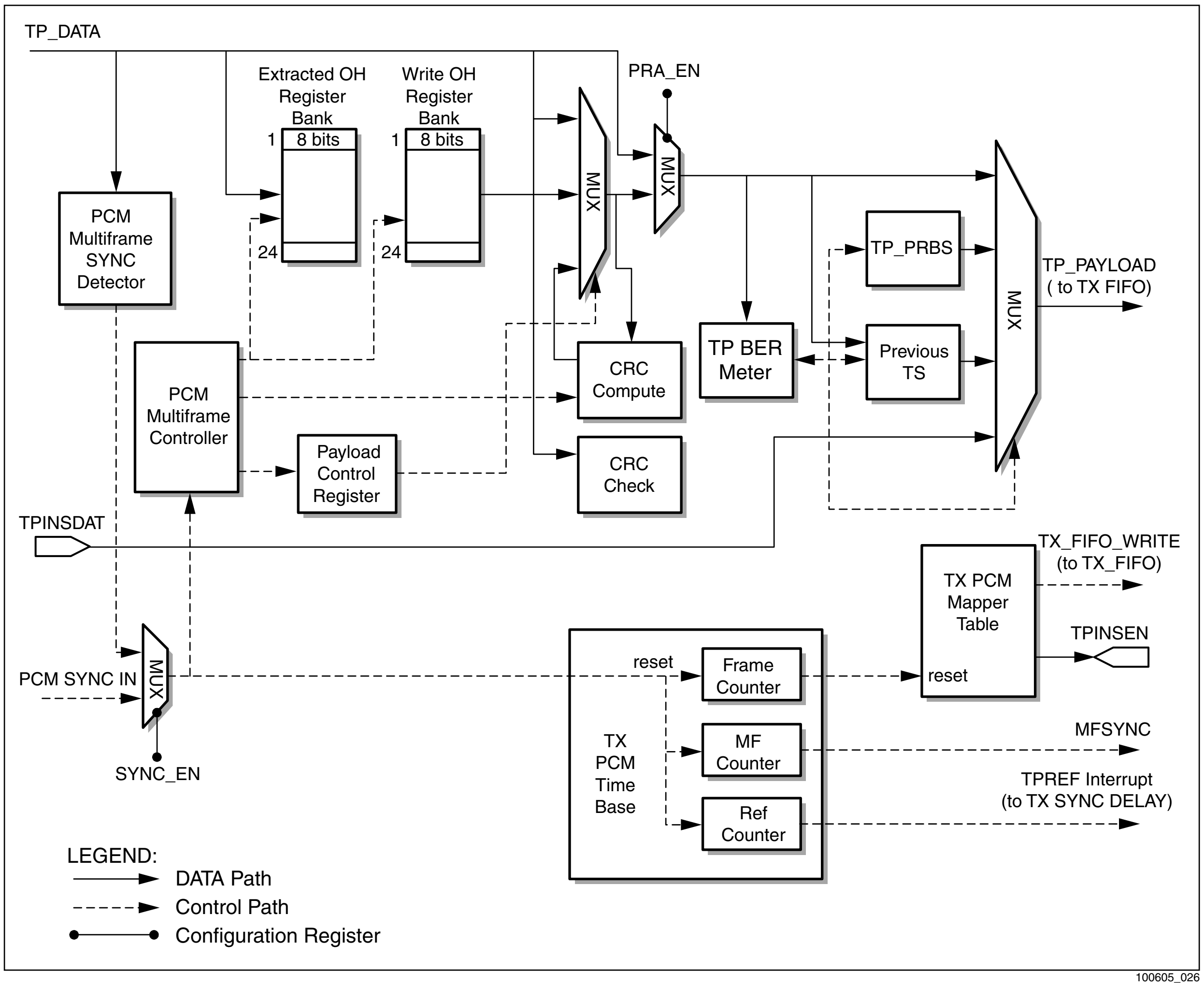


6.4.4 PCM Transmitter

Figure 6-9 illustrates a detailed block diagram of the PCM Transmitter section. The major tasks of the PCM Transmitter are:

1. Generates Tx PCM time base aligned with the incoming Frame/Multi-Frame sync.
2. Maps PCM Frame to the TX_FIFO using TX PCM Mapper Table.
3. Enables BER meter per time slots basis.
4. Inserts alternate PCM channel, using TPINSDAT, TPINSEN pins.
5. Synchronizes to any Layer 3 Frame/MF.
6. Checks CRC (Selectable) and computes CRC on the final ongoing frame.
7. Extracts overhead bits.
8. Provides capability to override each overhead bit by the MPU.

Figure 6-9. PCM Transmitter Block Diagram



6.4.4.1 PCM Sync Detector

The Sync Detector is able to synchronize to any sync pattern, grouped or spread, up to 16 bits long. This capability allows the DSL Framer to synchronize to E1, T1, or any other frame. This requires modifying the low-level DSL Framer code.

6.5 Test and Diagnostics

6.5.1 Performance Monitoring

The DSL Framer supports up to 12 performance monitoring counters, divided equally to three sections.

Each performance-monitoring counter can function as a CRC error counter for the Sever Error Second (SES) indicator, Far End Block Error (FEBE) counter, Bipolar Violation (BPV) error counter, or any other necessary performance indicator counter.

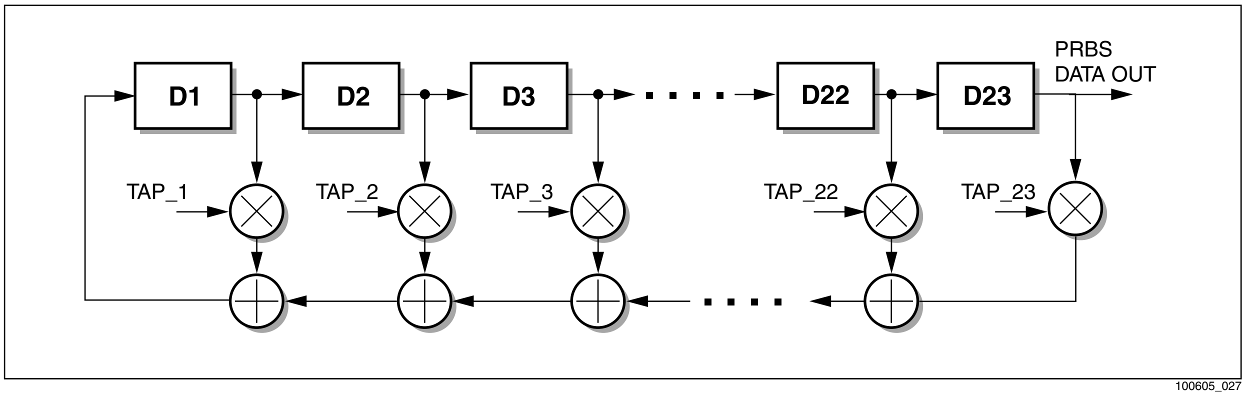
- Receive HDSL: Supports up to four performance monitoring counters.
- Receive PCM: Supports up to four performance monitoring counters.
- Transmit PCM: Supports up to four performance monitoring counters.

6.5.2 PRBS and BER Meter

The DSL Framer has two PRBS/BER meter modules supporting BER measurement towards both the HDSL and PCM side. TP_PRBS and RP_BER function as a BER meter towards the HDSL side, and RP_PRBS and TP_BER function as a BER meter towards the PCM side.

The PRBS sequence can override TPDAT and RPDAT per time slot basis, and achieve any framed or unframed test pattern examination. The PRBS pattern is programmable and selected for both RP_BER and TP_BER by PRBS_TAP_[2:0] registers, indicate up to 23rd-order PRBS (Tap [23:0]):

Figure 6-10. Generic PRBS Generator



Example:

For a PRBS pattern of $2^{15} - 1$, the polynomial is $x^{15} + x^{14} + 1$.

For a PRBS pattern of $2^{23} - 1$, the polynomial is $x^{23} + x^{18} + 1$.

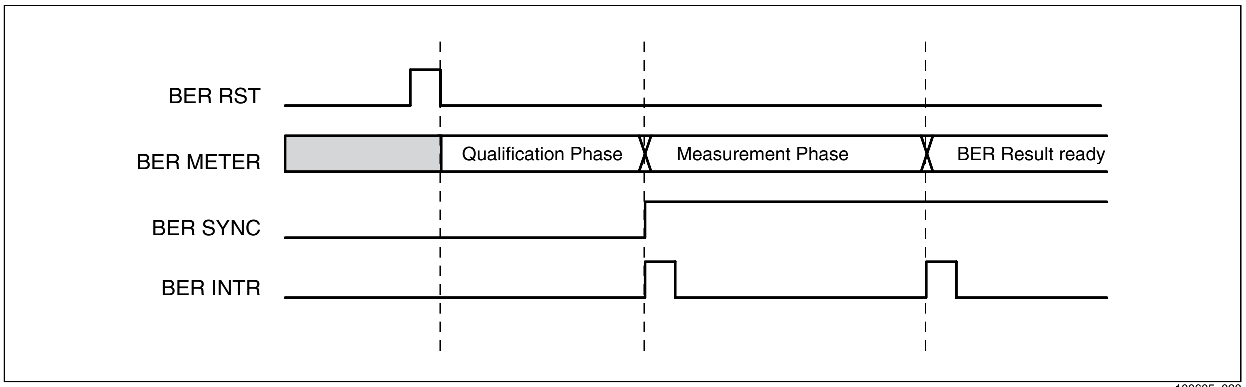
For a QRSS $2^{20} - 1$ pattern (polynomial – $x^{20} + x^{17} + 1$), 14-bit 0 suppression is implemented.

The TP_BER and RP_BER sequence can be inverted.

The constant value per time slot basis can override TSER and RSER instead of PRBS. The MPU configures *BER_SCALE* to specify the test measurement interval from a range of 2^{21}–2^{31} bit length.

The BER Measurement Timing is shown in Figure 6-11.

Figure 6-11. PRBS and BER Meter Timing



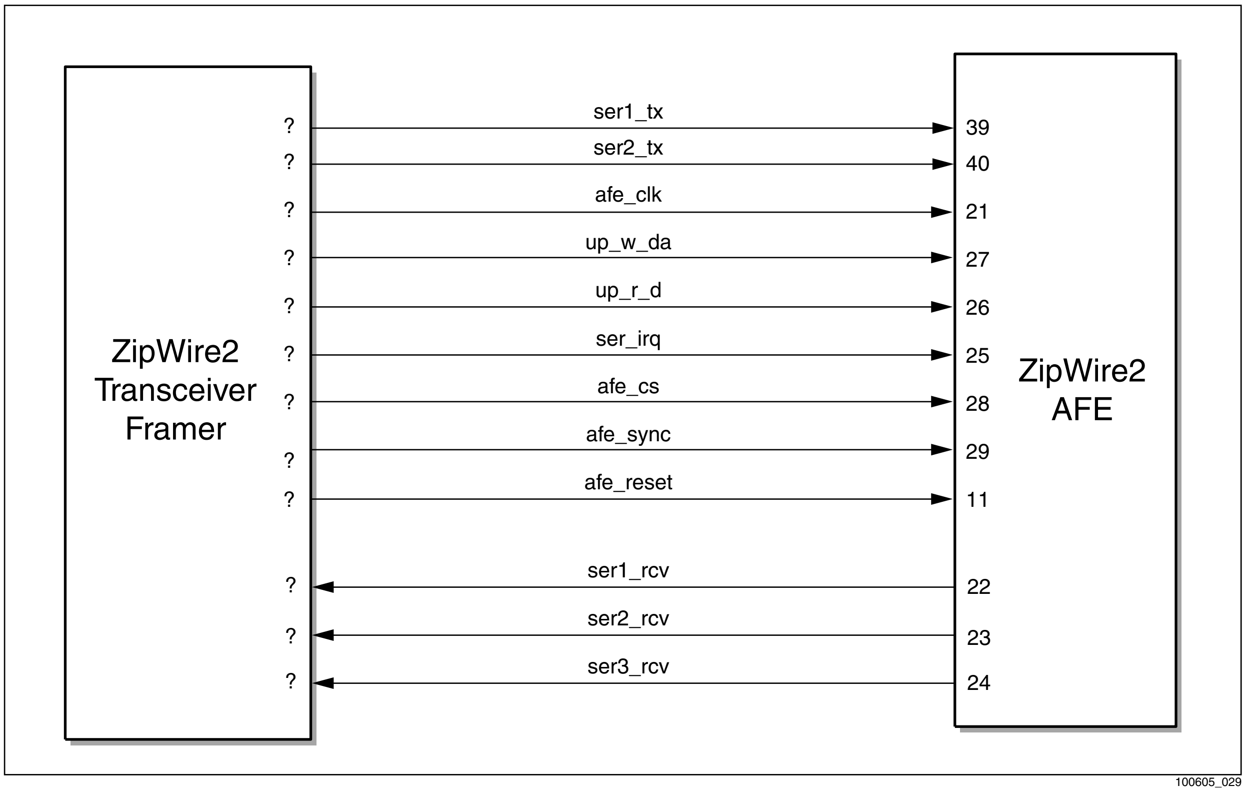
100605_028

7.0 Hardware Interfaces

7.1 ZipWire2 Transceiver/Framer to AFE Interface

Figure 7-1 illustrates the ZipWire2 Transceiver/Framer to AFE interface. The two devices must be connected as shown.

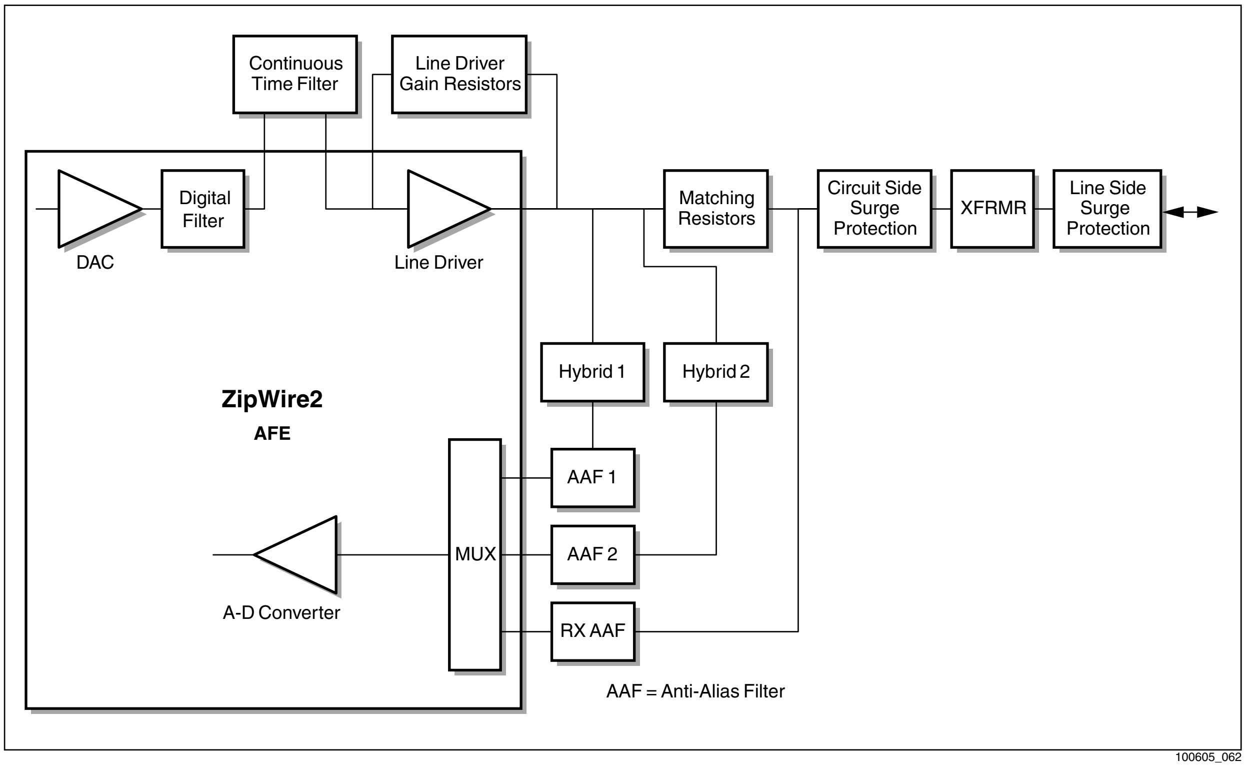
Figure 7-1. ZipWire2 Transceiver/Framer to AFE Interface



7.2 Transmission Line Interface

Figure 7-2 illustrates a block diagram of the DSL transmission line interface. The DSL interface consists of the continuous time filter, line drive feedback resistors, impedance matching resistors, compromise hybrid, transformer, and surge protection. All signals are differential pairs. Only NPO-type capacitors should be used in the DSL transmission line interface except for the surge protection blocks. The NPO capacitors are selected because of their high linearity characteristics. All capacitors should be 5% tolerant while the resistors should be 1% tolerant.

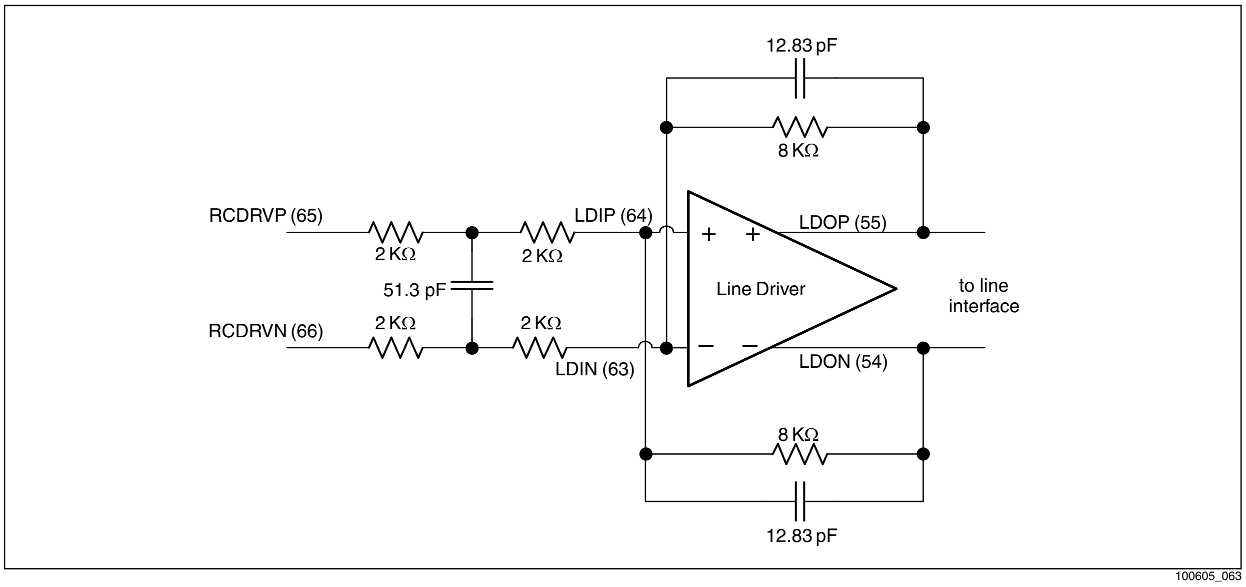
Figure 7-2. DSL Transmission Line Interface



7.2.1 Continuous Time Filter and Line Driver Control

Figure 7-3 illustrates the external Continuous Time Filter and Line Driver Control connections. The external Continuous Timer Filter filters out clock images created from the switched-capacitor filters. The external line driver gain control resistors set the line driver gain.

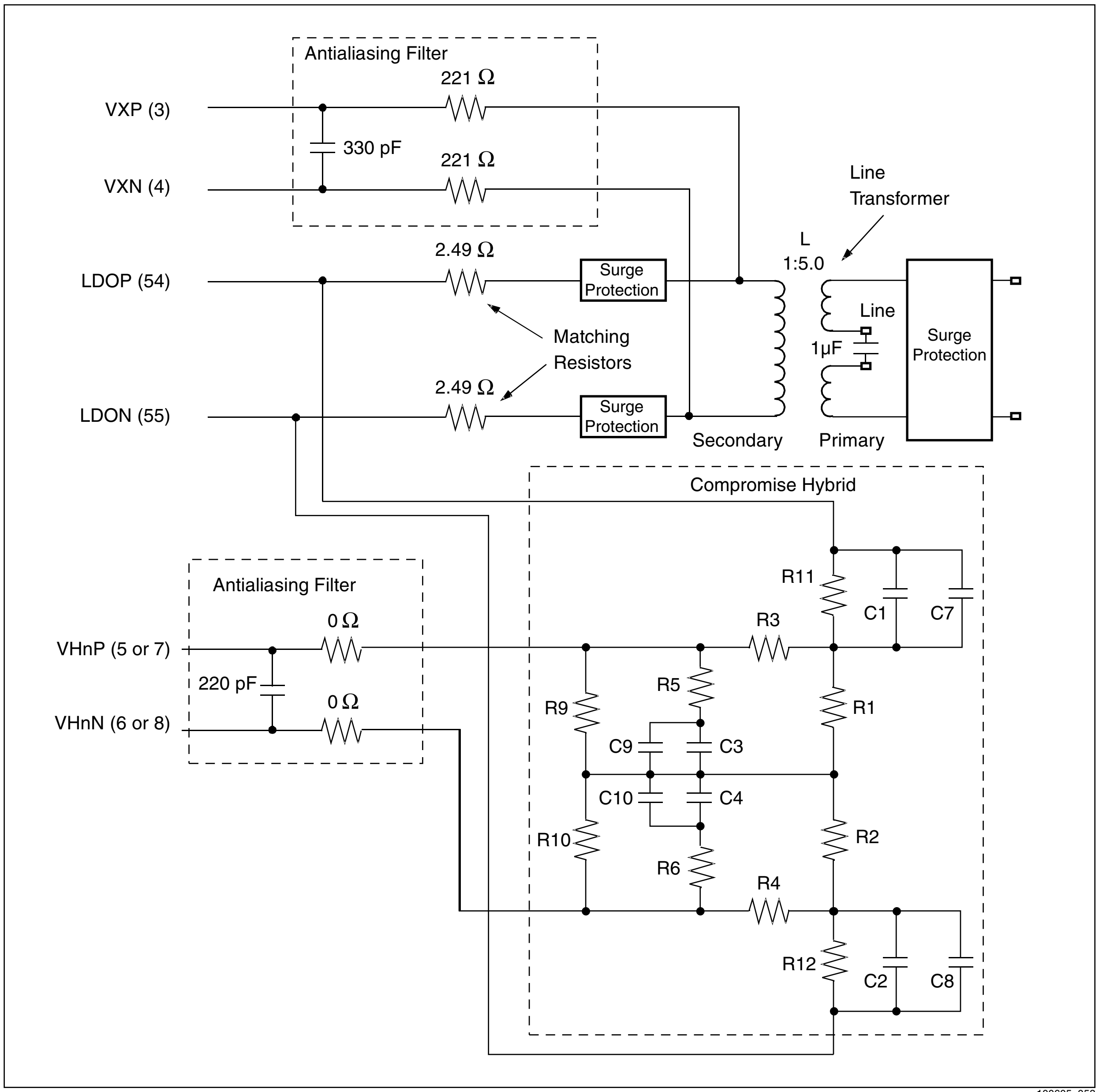
Figure 7-3. Continuous Time Filter and Line Driver Control



7.2.2 Compromise Hybrid, Matching Resistors, and Transformer

Figure 7-4 illustrates the hybrid topology.

Figure 7-4. Hybrid Topology



7.2.2.1 Compromise Hybrid

The purpose of the compromise hybrid is to model the impedance of the transmission line. This model generates an approximation of the transmitted signal's echo. The echo replica is then subtracted from the signal on the line transformer to generate a first-order approximation of the received signal. Although the CN8980 contains a digital Echo Canceller (EC), the hybrid is needed to reduce the signal-level input to the Analog-to-Digital Converter (ADC). This eliminates ADC overflow on short loops and increases the resolution of the digitized received signal for better digital signal processing performance.

The CN8980 includes two hybrid inputs to accommodate a wider range of loop characteristics and data rates. The CN8980 only uses one hybrid topology but the hybrid component values change to match the application requirements. The hybrid topology only requires passive components.

In a single data rate application, hybrid 1 is designed for flat loops while hybrid 2 is designed for bridge-tapped loops. In a multi-rate application, hybrid 1 is designed for higher data rates while hybrid 2 is designed for lower data rates. During the bit pump startup, the software will examine both hybrids to determine which hybrid provides the best echo cancellation. The CN8980 can also operate with only hybrid 1 present; however, this may limit performance on certain loops. The Analog Front End API (Section 17.4.5) command sets the number of hybrids present in the system.

In the hybrid section, two capacitors are placed in parallel to achieve nonstandard capacitor values.

7.2.2.2 Impedance Matching Resistors

Impedance matching resistors (2.49 Ω) are placed in the transmit path so that the output impedance of the line interface more closely matches the impedance of the transmission line and load. This maximizes the power transferred to the receiver on the other end of the line. The load is assumed to be 135 Ω.

7.2.2.3 Transformer

The line transformer provides DC isolation from the transmission line by creating a high-pass filter. The winding ratio of the transformer must be 5.0:1 (line side:circuit side) to generate the appropriate voltage level on the line. The primary inductance (L) of the transformer (line side) is a very critical parameter. If the inductance is too high, the cutoff frequency of the filter will be too low and the CN8980 Echo Canceller and Equalizer will not be able to cancel out the low frequency components of the echo and Inter-Symbol Interference (ISI). If L is too low, part of the information in the signal will be filtered out, thereby decreasing the Signal-to-Noise (SNR) ratio. In addition, the line transformer must meet certain return loss requirements to maximize system performance.

7.2.2.4 Anti-Alias Filters

Anti-aliasing filters are needed to filter out high frequencies that would be aliased back into the passband as noise. These filters are made of all passive components. The cutoff frequency (f_c) is designed to be as low as possible to achieve maximum attenuation of aliasing frequencies without filtering out the desired signal.

7.2.3 Surge Protection

TBD

7.3 Voltage Reference and Compensation Circuitry

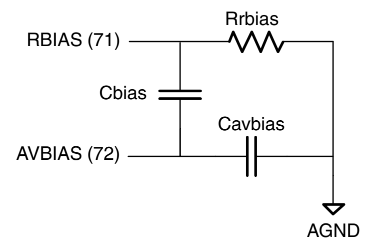
Compensation capacitors must be connected between all of the CN8980 voltage reference pins and analog ground. The voltage reference signals, their associated pin numbers, and the recommended compensation capacitor values are listed in Table 7-1.

Table 7-1. ZipWire2 AFE Compensation Capacitor Values

Signal Name	Pin Number	Value
VRNTX	69	0.1 µF
VRPTX	70	0.1 µF
VCMI	73	0.1 µF
VCMO	74	0.1 µF
VBGN	77	0.1 µF
VBGP	78	0.1 µF
VRNRX	79	0.1 µF
VRPRX	80	0.1 µF

In addition to the compensation capacitors, external passive components are needed to set the bias current used in the CN8980. This network is shown in Figure 7-5. The recommended value of the resistor is given in Table 7-2.

Figure 7-5. ZipWire2 AFE Bias Current Network



100605_064

Table 7-2. ZipWire2 AFE Bias Current Network Values

Signal Name	Value
Rrbias	10.0 kΩ
Cbias	0.1 µF
Cavbias	0.1 µF

7.4 Framer Bypass Interface (ZipWire2 Transceiver DSL Interface)

Figure 7-6 illustrates the ZipWire2 transceiver interface timing diagrams. There are four signals in the ZipWire2 Transceiver DSL interface: HXCLK, HXP, TXDAT, and RXDAT.

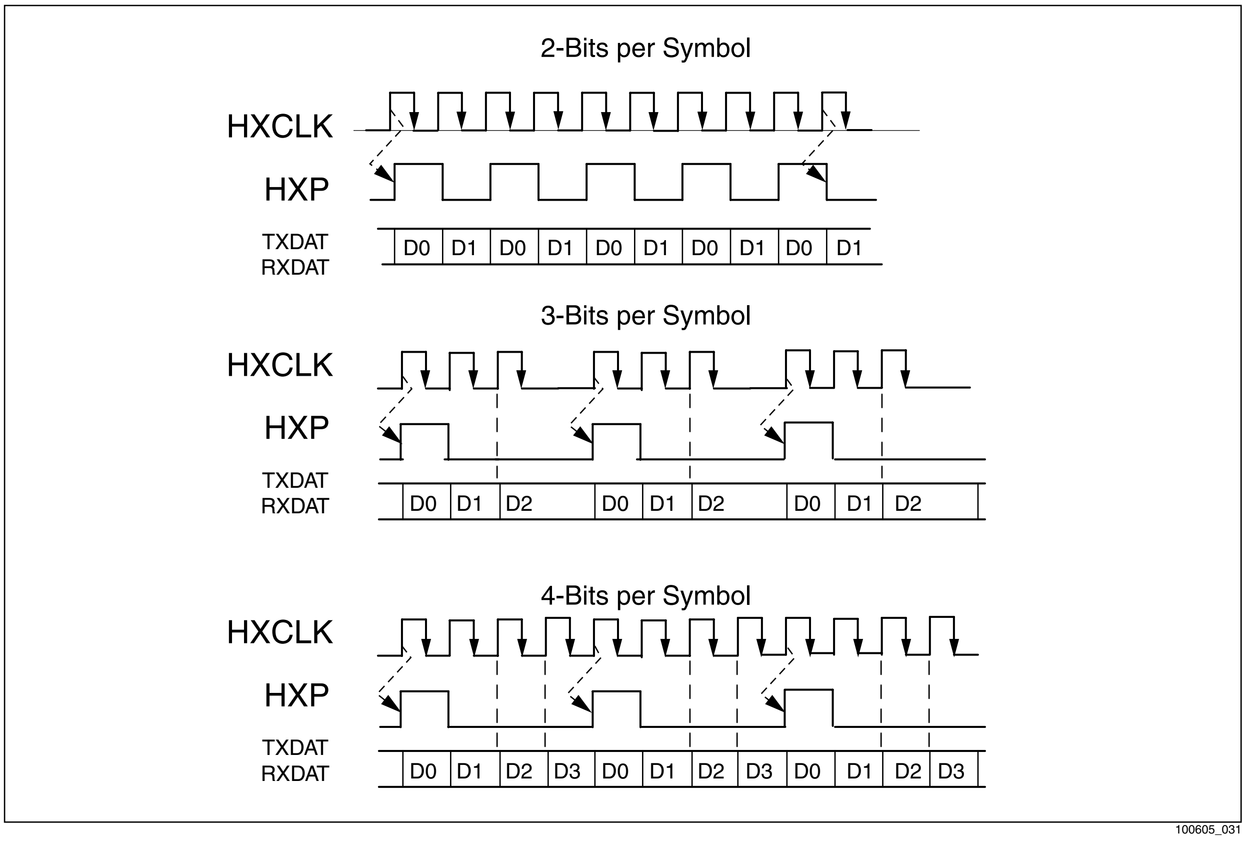
The ZipWire2 transceiver only operates as an interface master. The ZipWire2 TxDAT and RxDAT cannot be slaved from an external clock. The HXCLK and HXP are outputs. The HXCLK operates at the desired data rate while the HXP operates at the effective symbol rate.

The HXP provides the phase alignment to mark the least significant bit of the serial data stream. The HXP is equivalent to the QCLK symbol in the previous ZipWire1 products.

The TXDAT (input) and RXDAT (output) correspond to the serial data. These figures illustrate positive edge transitions; negative edge timing can selected using the API command (*command to be determined*).

NOTE: The 3-bits per symbol suppresses the number of clock pulses.

Figure 7-6. ZipWire2 Transceiver DSL Interface



The interface can operate in the following modes as controlled by the API command *DSL_Frame_Structure* as explained in Section 17.3.8.

Mode	Data Bits	Line Code
1-Bit Uncoded	1	2 PAM (startup only)
1-Bit Trellis Coded	1	4 PAM (not supported)
2-Bit Uncoded	2	4 PAM (2B1Q)
2-Bit Trellis Coded	2	8 PAM
3-Bit Uncoded	3	8 PAM
3-Bit Trellis Coded	3	16 PAM
4-Bit Uncoded	4	16 PAM

Regardless of being in Trellis Coded or Uncoded mode, the DSL Interface remains the same, i.e., 3-bit Trellis Coded and 3-Bit Uncoded have the same timing to the external interface. The Trellis Coded modes will internally generate a coded symbol requiring the one larger PAM line code.

7.5 Test and Diagnostic Interface (JTAG)

The Test and Diagnostic Interface comprises a test access port and two Serial Test Ports (STP). The test access port conforms to *IEEE Std. 1149.1-1990 (IEEE Standard Test Access Port and Boundary Scan Architecture)*. Also referred to as Joint Test Action Group (JTAG), this interface provides direct serial access to each of the transceiver's I/O pins. This capability can be used during an in-circuit board test to increase the testability and reduce the cost of the in-circuit test process.

The serial test ports can be viewed as a real-time virtual probe for looking at the transceiver's internal signals. A majority of the receiver's signal path is accessible through these outputs.

8.0 Pin Descriptions

The ZipWire2 solution is available in a two- or three-device chip set. The two-device chip set consists of a ZipWire2 Transceiver/Framer and a ZipWire2 AFE/Line Driver. The three-device chip set consists of a ZipWire2 Transceiver, a ZipWire2 Framer, and a ZipWire2 AFE/Line Driver.

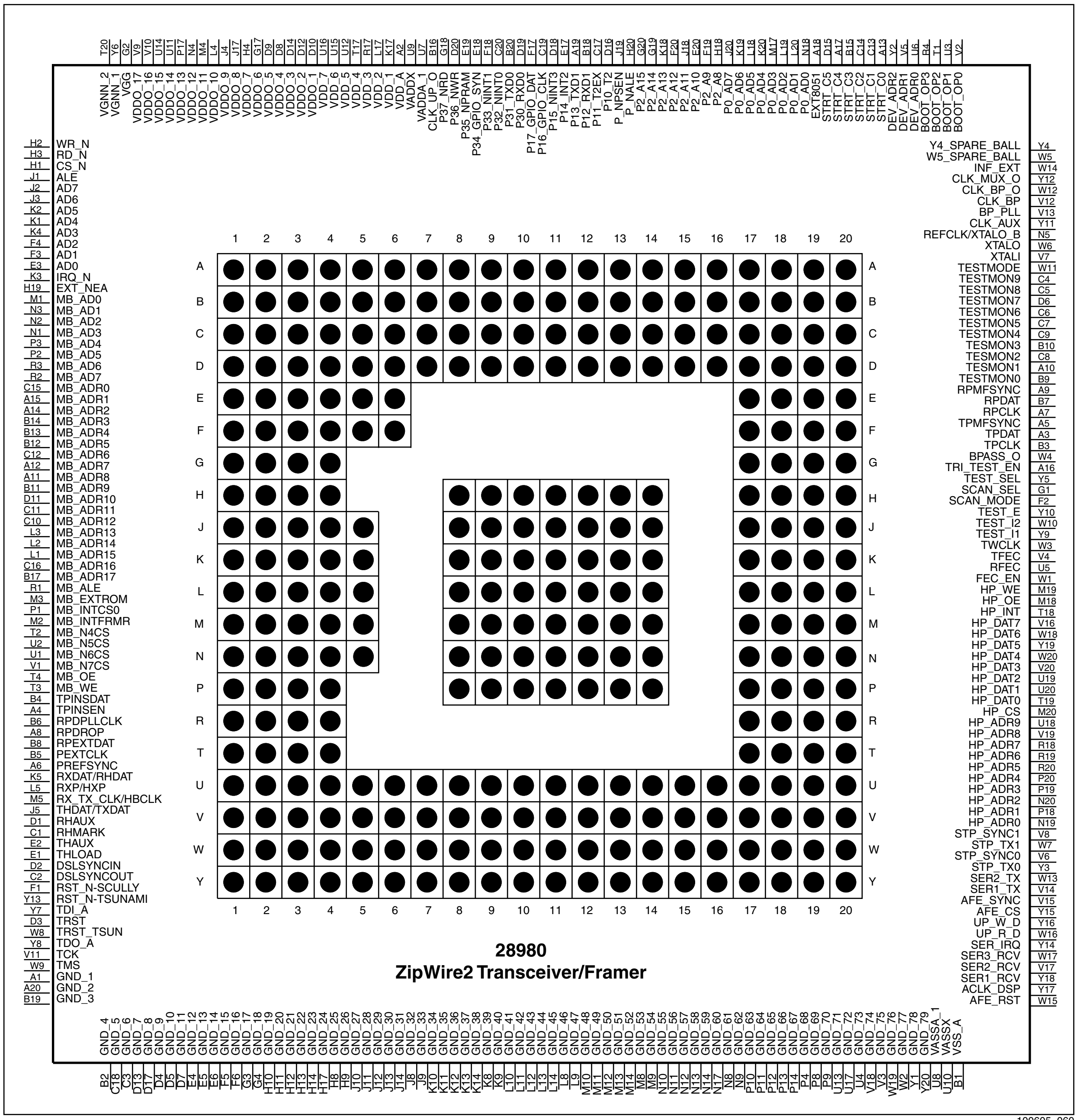
8.1 ZipWire2 Pin Assignments

This section provides the pin assignments for the ZipWire2 devices.

8.1.1 ZipWire2 Transceiver/Framer Pin Assignments

The ZipWire2 Transceiver/Framer is packaged in a 27 × 27 mm Ball Grid Array (BGA) and contains 314 balls (see Figure 8-1).

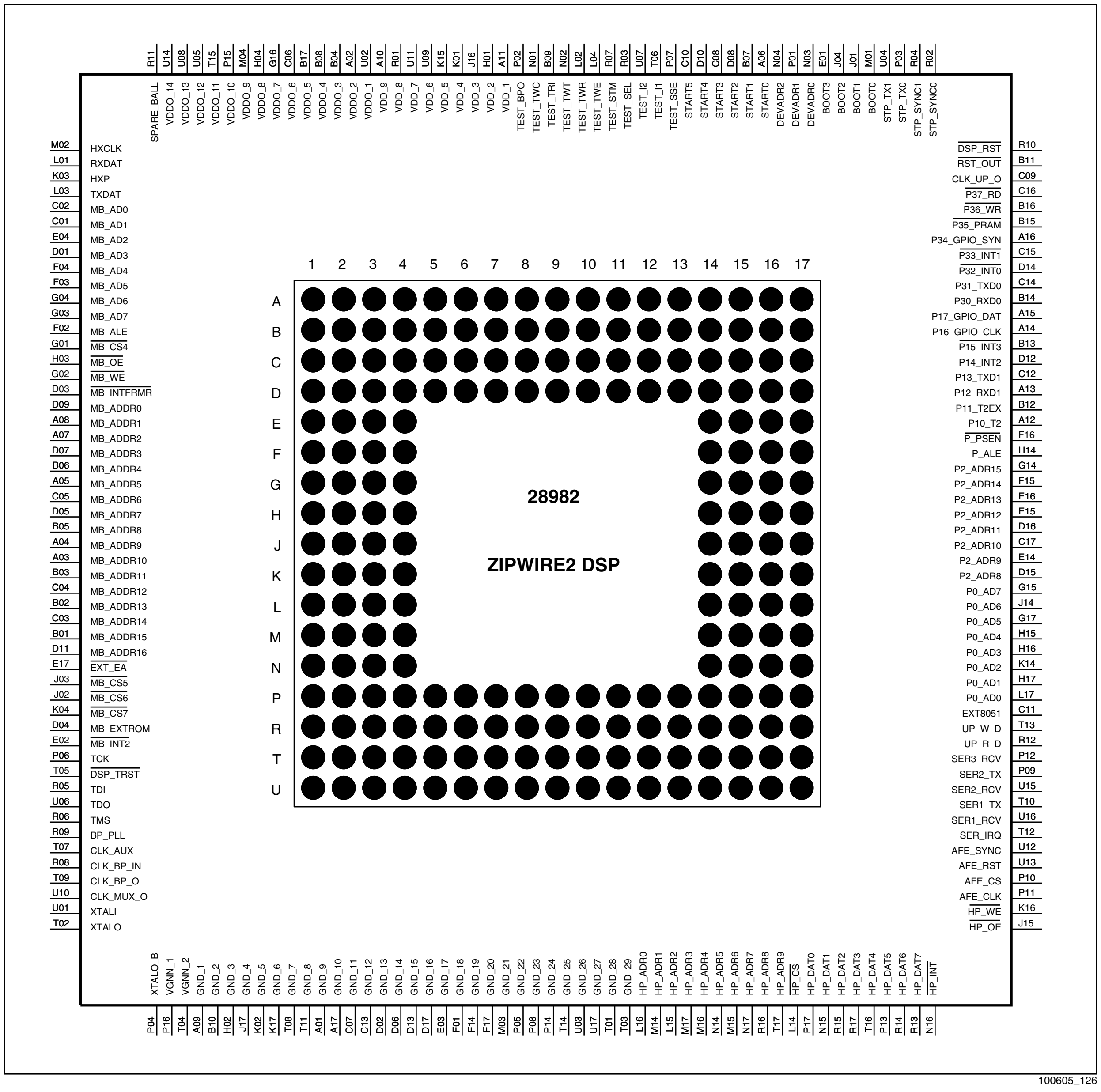
Figure 8-1. ZipWire2 Transceiver/Framer Pin Assignments



8.1.2 ZipWire2 Transceiver Pin Assignments

The ZipWire2 Transceiver is packaged in a 15 x 15 mm Chip-Array Ball Grid Array (CABGA) (see Figure 8-2).

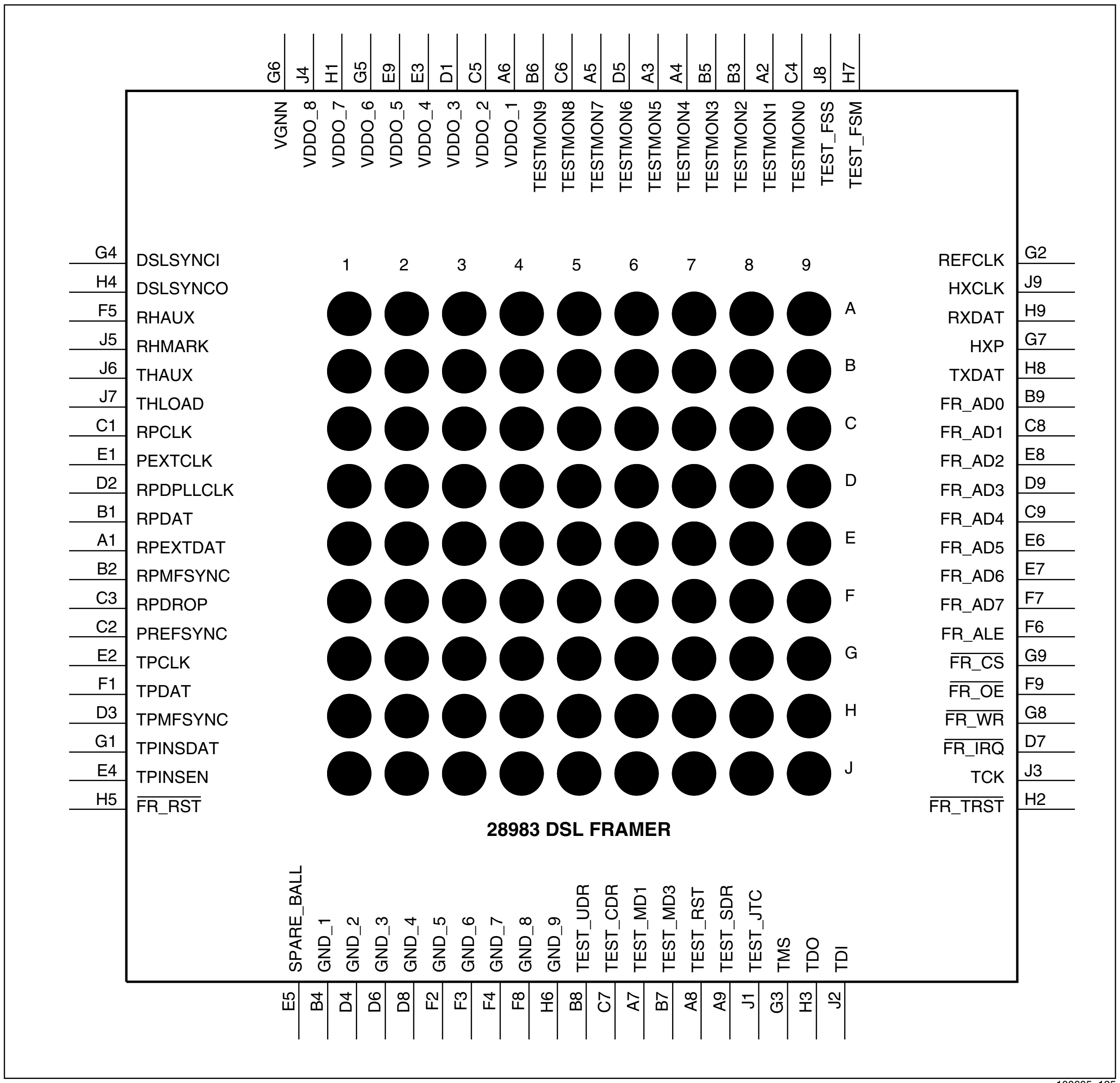
Figure 8-2. ZipWire2 Transceiver Pin Assignments



8.1.3 ZipWire2 Framer Pin Assignments

The ZipWire2 Framer is packaged in a 9 x 9 mm CABGA (see Figure 8-3).

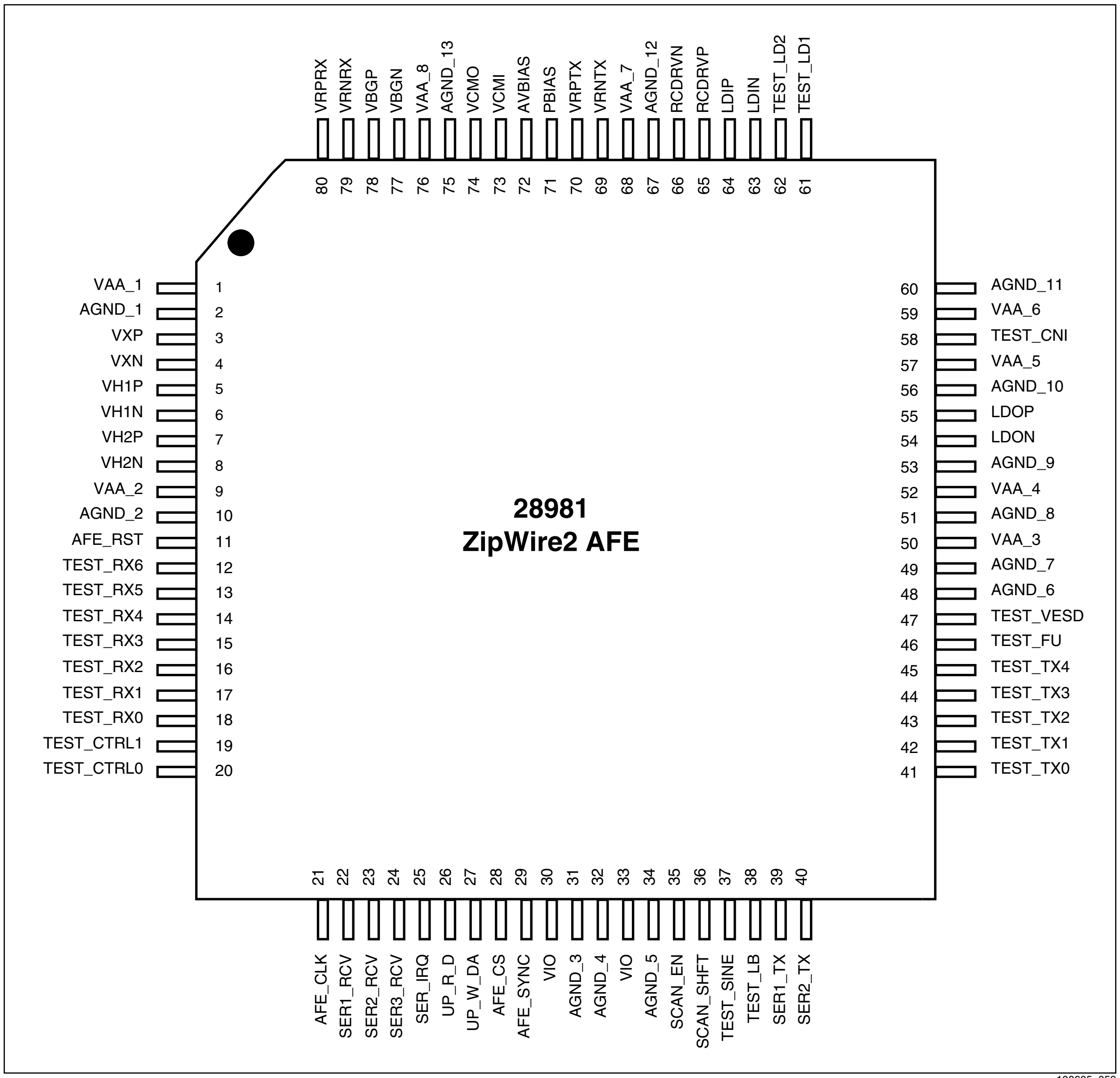
Figure 8-3. ZipWire2 Framer Pin Assignments



8.1.4 ZipWire2 AFE Pin Assignments

The ZipWire2 AFE/Line Driver is packaged in an 80-pin Thin Quad Flat Pack (TQFP). In addition, the bottom of the device contains a GND pad that should be soldered to the board.

Figure 8-4. ZipWire2 AFE Pin Diagram



8.2 ZipWire2 Signal Descriptions

This section provides the signal descriptions for both the ZipWire2 Transceiver/Framer and ZipWire2 AFE/Line Drive devices.

8.2.1 ZipWire2 Transceiver/Framer Signal Descriptions

Table 8-1 defines the ZipWire2 Transceiver/Framer Signal (pin) descriptions.

Table 8-1. ZipWire2 Transceiver/Framer Signal Definitions (1 of 9)

Signal	Name	I/O	PU/PD[(5)]	Description
Power and Ground				
VDD (VCORE)	DSP Core Voltage	—	—	Dedicated supply pins powering the DSP core. Must be connected to +2.5 V.
VDDO (VIO)	I/O Voltage	—	—	Dedicated supply pins powering the I/O. Must be connected to +3.3 V.
VGNN (VESD)	5 V ESD Protection	—	—	Dedicated supply pins used to bias input protection diodes. If interfacing to other 5 V powered devices, connect VGNN to +5 V. Otherwise connect VGNN to +3.3 V.
GND	Ground	—	—	Common ground for ZipWire2 device.
Clocks				
XTALI	Crystal Input	I	—	Crystal = 22.1184 MHz.
XTALO	Crystal Output	O	—	Connection point for the crystal.
XTALO_B	Crystal Clock Out	O	—	Buffered-crystal oscillator output, 22.1184 MHz.
REFCLK	Framer Reference Clock Input	I	—	DSL Framer reference clock input. For most applications, connect REFCLK to XTALO_B (22.1184 MHz). REFCLK can be connected to another system clock for applications that require the DSL Framer DPLL recovered clock (RPCLK) to be phase locked to the system clock.
CLK_AUX	Auxiliary Clock Output	O	—	Programmable Auxiliary clock output.
CLK_MUX_O	Muxed Clock Outputs	O	—	Output Clock MUX. Can select from many of the internal clock sources. Internal use only, should be floating, No-Connect.
BP_PLL	Bypass PLL	I	—	Set the bit pump clock source. When set to low, the ZipWire2 will use its internal PLL. When set high, the ZipWire2 will bypass the internal PLL and use the CLK_BP_IN as its reference clock. Provided for internal test purposes only. This pin should be connected low.
CLK_BP_O	Bit Pump Clock Output	O	—	Bit pump output clock. Provided for internal test purposes only. This pin should be left a No-Connect.

Table 8-1. ZipWire2 Transceiver/Framer Signal Definitions (2 of 9)

Signal	Name	I/O	PU/PD(5)	Description
CLK_BP_IN	Bit Pump Clock Input	I	—	Optional bit pump input clock. Provided for internal test purposes only. This pin should be left a No-Connect.
PCM Interface				
RPCLK	Receive PCM Clock	O	—	Clocks the PCM receive outputs: RPDAT, RPMSYNC, and RPDROP. Normally derived by the internal clock recovery (DPLL). Can be derived by PEXTCLK or TPCLK. Rising edge or falling edge output transition are selectable.
PEXTCLK	PCM External Clock	I	—	Optionally sources the RPCLK or TPCLK or both RPCLK and TPCLK.
RPDPLLCLK	RX PCM DPLL Clock	O	—	DPLL clock output. Optionally, this pin can be used externally when the DPLL doesn't function as clock recovery but as clock generator.
RPDAT	Receive PCM Data	O	—	During specified time slots, data is clocked out by RPCLK. This pin can be optionally three-stated during inactive time slots or when DSL Framer is bypassed.
RPEXTDAT	Receive PCM External Data	I	—	Used in Multi-Pair configuration, when all the receive PCM data (from all channels) need to be routed through the master framer for PCM layer framing and overhead handling, like E1 PRA, CRC calculation, and so on.
RPMFSYNC	Receive PCM Multi-Frame Sync	O	—	Active high output from the receive time base. Optionally, programmed to mark either Frame or Multi-Frame boundaries during framed application.
RPDROP	Drop Indicator	O	—	Active high output indicates when specific receive PCM time slots are present on RPDAT. Time slot size can be 1,2,4, or 8 bit long. This pin also controls the three-state output enable of RPDAT in Multi-Pair configuration.
PREFSYNC	Receive PCM Reference Sync	I/O	PD	Used in Multi-Pair configuration. When configured as a master, this output is the internal receive PCM time base reset which aligns the Receive PCM time base of each slave to the master. When slave mode is selected this signal is an RX PCM reset input (Default).
TPCLK	Transmit PCM Clock	I	—	Normally samples the PCM transmit inputs: TPDAT, TPMSYNC, and TPINSDAT on the falling edge and clocks out TPINSEN on the rising edge. The edge transition is selectable.
TPDAT	Transmit PCM Data	I	—	During specified time slots, data is sampled in by a selected clock source (TPCLK, PEXTCLK or DPLL Recovery clock).
TPMFSYNC	Transmit PCM Multi-Frame Sync	I/O	PD	This input resets the Transmit PCM time base during framed application and is ignored in unframed mode. This signal internally delayed by a programmable bit and frame offset to coincide with TPDAT bit 0, frame 0. Optionally programmed to mark either frame or multi-frame boundaries. Has internal pull-downs.
TPINSDAT	Transmit PCM Insert Data	I	—	Alternate source of PCM transmits serial data. TPINSDAT replaces TPDAT when TPINSEN is active.
TPINSEN	Transmit PCM Insert Enable	O	—	Active high output indicates when specific TPINSDAT time slots are sampled.

Table 8-1. ZipWire2 Transceiver/Framer Signal Definitions (3 of 9)

Signal	Name	I/O	PU/PD[5]	Description
				HDSL Interface
RHAUX	Receive Auxiliary Data	O	—	RHDAT after descrambling is provided as an auxiliary channel and clocked out by HXCLK.
RHMARK	Receive Auxiliary Data Mark	O	—	Active high output indicates when specific HDSL time slots are present on RHAUX. Optionally, this output provides gated HXCLK instead.
THAUX	Transmit Auxiliary Data	I	—	Alternate source of HDSL transmits data. TAUX is mapped into selected HDSL time slots when THLOAD is active.
THLOAD	Transmit Auxiliary Data Load	O	—	Active high output indicates when specific HDSL time slots shall be replaced by THAUX. Optionally, this pin provides gated HXCLK instead.
DSLSYNCI	DSL Reference Sync In	I	—	Used in Multi-Pair configuration to select the internal RDSL_REF_SEL for DPLL phase reference and RH_BSP reset.
DSLSYNCO	DSL Reference Sync Out	O	—	The selected DSL sync for DPLL reference and RH_BSP reset is output in DSLSYNCO to allow cascading the framers in Multi-Pair configuration.
				Framer Bypass Interface[1]
RXDAT	DSP Receive Data	O	—	Data is clocked out on the rising edge of HXCLK.
HXP	DSP Receive Symbol Alignment	O	—	Symbol clock that provides symbol boundary. Rising edge marks the least significant bit.
TXDAT	DSP Transmit Data	I	—	Data is sampled on the falling edge of HXCLK. THDAT shall be aligned to the symbol boundary on HXP.
HXCLK	DSP Data Rate Clock	O	—	This clock signal operates at the DSL data rate and controls the HDSL interface data signals—HXP, RXDAT, RHAUX, RHMARK, TXDAT, THAUX, and THLOAD. Output data is clock out on the rising edge of HXCLK while input data is sampled on the falling edge of HXCLK.
				Master Bus Interface
$\overline{\text{EXT\_EA}}$	Boot ROM Select	I	—	Selects source of boot ROM code: Low = Execute boot ROM from external ROM. This option is only required when executing from an emulator. High = Execute boot ROM from internal ROM. This is the normal operating mode. The *EXT_EA and EXT8051 pins need to be adjusted when switching between the internal 8051 and an external processor.
MB_EXTROM	Program Code Select	I	—	Selects source of Program code. Low = Execute program from internal Program RAM (PRAM), the contents of the PRAM are downloaded via an external flash, Serial Boot Link, or the Host Port RAM interface (normal operation). High = Execute program from external ROM (emulation mode). When the *EXT_EA pin is low (external mode), the program code also uses the external ROM. This pin should be tied low for customer applications. This pin is provided for internal testing.

Table 8-1. ZipWire2 Transceiver/Framer Signal Definitions (4 of 9)

Signal	Name	I/O	PU/PD(5)	Description
MB_ADDR[0-16]	Master Bus Address	O	—	Master Bus Address bits 0–16. Bits 0–15 contain the 16-address bits. Bit 16 is a unused general purpose latched output. This may be used in the future to program the external flash or facilitate paging. Customers should leave MB_ADDR16 as a No-Connect.
MB_AD[0-7]	Master Bus Address/Data	I/O	PU	Master Bus Address/Data bits 0–7. Address Bits [0–7] only used in MUXed Mode based on ALE.
MB_ALE	Address Latch Enable	O	—	For multiplexed devices, the falling edge indicates the MB_AD[7:0] signals containing a valid address. For non-multiplexed devices, MB_ALE can be ignored.
$\overline{\text{MB\_OE}}$	Output Enable	O	—	The $\overline{\text{MB\_OE}}$ output enabled behaves as a read strobe. Falling edge of *MB_OE indicates the external device to present the data on the bus. The internal 8051 will sample the data on the rising edge of $\overline{\text{MB\_OE}}$.
$\overline{\text{MB\_WE}}$	Write Enable	O	—	The $\overline{\text{MB\_WE}}$ output enabled behaves as a write strobe. The internal 8051 will present its data on the bus on the falling edge of *MB_WR. The external device should sample the data on the rising edge of $\overline{\text{MB\_WE}}$.
$\overline{\text{MB\_INTFRMR}}$	DSL Framer Interrupt Request	I	—	Active-low ZipWire2 framer interrupt pin. Internally connected to 8051 Interrupt 3 and to P15NINT3 (Emulation mode). When using the internal DSL Framer, this pin should be floating (No-Connect). In Framer Bypass mode, this interrupt pin can be connected to an external device. However, the embedded 8051 code would need modification.
$\overline{\text{MB\_INT2}}$	Master Bus User Interrupt #2	I	—	Active-low interrupt pin connected to the 8051 Interrupt 2. User-definable interrupt pin that can be connected from any external device. Requires internal software to be modified to use this interrupt input.
$\overline{\text{MB\_CS4}}$	Framer Chip Select	O	—	Active-low ZipWire2 framer chip select. In Framer Bypass mode, this interrupt pin can be connected to an external device. However, the embedded 8051 code would need modification.
$\overline{\text{MB\_CS5}}$	External ROM Chip Select	O	—	Active-low external ROM chip select. When Flash Master, connect directly to Flash's Chip-select pin. Otherwise, a No-Connect is used.
$\overline{\text{MB\_CS6}}$	User Chip Select #6	O	—	Active-low user-definable chip select. On EVM, connected to external RAM.
$\overline{\text{MB\_CS7}}$	User Chip Select #7	O	—	Active-low user definable chip select. On EVM, connected to 3–8 decoder to decode T1/E1 Framer, LEDs, DIP Switches, and so on.

Table 8-1. ZipWire2 Transceiver/Framer Signal Definitions (5 of 9)

Signal	Name	I/O	PU/PD(5)	Description
DSL Framer Microprocessor Interface(2)				
FR_AD[0-7]	DSL Framer Address/Data	I/O	PD	DSL Framer multiplexed address/data lines. When using the internal 8051, these must be connected to the MB_AD[0–7] pins.
$\overline{\text{FR\_CS}}$	DSL Framer Chip Select	I	—	DSL Framer chip select. When using the internal 8051, these must be connected to the $\overline{\text{MB\_CS4}}$ pin.
FR_ALE	DSL Framer Address Latch Enable	I	—	DSL Framer address latch enable. When using the internal 8051, these must be connected to the $\overline{\text{MB\_ALE}}$ pin.
$\overline{\text{FR\_OE}}$	DSL Framer Output Enable	I	—	DSL Framer output enable. When using the internal 8051, these must be connected to the $\overline{\text{MB\_OE}}$ pin.
$\overline{\text{FR\_WE}}$	DSL Framer Write Enable	I	—	DSL Framer write enable. When using the internal 8051, these must be connected to the $\overline{\text{MB\_WE}}$ pin.
$\overline{\text{FR\_IRQ}}$	DSL Framer Interrupt Request	O	—	DSL Framer interrupt request. When using the internal 8051, these must be connected to the $\overline{\text{MB\_INTFRMR}}$ pin.
Configuration Pins				
BOOT[3-0]	BOOT Configuration Pins	I	—	See Section 4.4.
START[5-0]	START Configuration Pins	I	—	See Section 4.4.
DEVADR[2-0]	Device Address Configuration Pins	I	—	See Section 4.4.
Reset				
$\overline{\text{DSP\_RST}}$	DSP Reset	I	—	Asynchronous active-low input that places the device in an inactive state. This resets the DSP and internal 8051 blocks. This pin should be connected to the host processor's $\overline{\text{RST\_OUT}}$ (multi-processor configuration) or an external hardware reset button (single-processor configuration).
$\overline{\text{FR\_RST}}$	DSL Framer Reset	I	—	Asynchronous active-low input that places the device in an inactive state. This pin should be connected to the ZipWire2's $\overline{\text{RST\_OUT}}$ pin. This allows the internal 8051 software to reset the DSL Framer block.
$\overline{\text{RST\_OUT}}$	Reset Out	O	—	Active-low output that allows the ZipWire2 device to reset external devices.
AFE Interface				
SER1_TX	Serial Transmit Data to AFE	O	—	Connect directly to ZipWire2 AFE SER1_TX, pin 39. Serial interface stream that allows the bit pump to control the AFE. In addition, provides the desired transmitted symbol data.
SER2_TX	Serial Transmit Data to AFE	O	—	Connect directly to ZipWire2 AFE SER2_TX, pin 40.
SER_IRQ	AFE, μP Control Signal	O	—	Connect directly to ZipWire2 AFE SER_IRQ, pin 25.

Table 8-1. ZipWire2 Transceiver/Framer Signal Definitions (6 of 9)

Signal	Name	I/O	PU/PD(5)	Description
AFE_SYNC	AFE, µP Control Signal	O	—	Connect directly to ZipWire2 AFE AFE_SYNC, pin 29.
AFE_CS	AFE, µP Control Signal	O	—	Connect directly to ZipWire2 AFE AFE_CS, pin 28.
AFE_RST	AFE, µP Control Signal	O	—	Connect directly to ZipWire2 AFE AFE_RST, pin 11.
UP_W_D	AFE, µP Control Signal	O	—	Connect directly to ZipWire2 AFE UP_W_D, pin 27.
UP_R_D	AFE, µP Control Signal	O	—	Connect directly to ZipWire2 AFE UP_R_D, pin 26.
AFE_CLK	AFE Master Clock	O	—	Connect directly to ZipWire2 AFE AFE_CLK, pin 21, 27 MHz clock.
SER1_RCV	Receive Transmit Data from AFE	I	—	Connect directly to ZipWire2 AFE SER1_RCV, pin 22. Serial interface stream from AFE to bit pump. In addition, provides the incoming received symbol data.
SER2_RCV	Receive Transmit Data from AFE	I	—	Connect directly to ZipWire2 AFE SER2_RCV, pin 23.
SER3_RCV	Receive Transmit Data from AFE	I	—	Connect directly to ZipWire2 AFE SER3_RCV, pin 24.
Host Port RAM Interface				
HP_ADR[0-9]	Host Port Address	I	—	Host Port RAM Address bits 0–9.
HP_DAT[0-7]	Host Port Data	I/O	PU	Host Port RAM Data bits 0–7.
$\overline{\text{HP\_CS}}$	Host Port Chip Select	I	—	Host Port RAM Chip-Select. Active-low.
$\overline{\text{HP\_WE}}$	Host Port Write Enable	I	—	Host Port RAM Write Enable. Active-low.
$\overline{\text{HP\_OE}}$	Host Port Output Enable	I	—	Host Port RAM Output Enable. Active-low.
$\overline{\text{HP\_INT}}$	Host Port External Interrupt	O	—	Active-low interrupt that signifies the API protocol is complete (see Section 15.4).
Emulation Port(3)				
EXT8051	External 8051 Mode	I	—	Selects source of the microprocessor: Low = Use internal 8051. This is the normal operating mode. High = Use an external 8051 or emulator. Note: Both the $\overline{\text{EXT\_EA}}$ and EXT8051 pins need to be adjusted when switching between the internal 8051 and an external processor.
CLK_UP_O	External 8051 Clock	O	—	Connect to external 8051 XTALI, ~27 MHz.
P0_AD[7-0]	Emulation Port #0 Address/Data [7–0]	I/O	PU	Connect to external 8051 P0 pins. Becomes output test pins when EXT_8051 is disabled.

Table 8-1. ZipWire2 Transceiver/Framer Signal Definitions (7 of 9)

Signal	Name	I/O	PU/PD[(5)]	Description
P_ALE	Emulation ALE	I	PU	Connect to external 8051 ALE.
$\overline{\text{P\_PSEN}}$	Emulation PSEN	I	PU	Connect to external 8051 PSEN.
P2_ADR[15-8]	Emulation Port #0 Address [15–8]	I/O	PU	Connect to external 8051 P2 pins. Becomes output test pins when EXT_8051 is disabled.
P10_T2	Emulation Timer #0	I	—	Connect to external 8051 Port 1, Bit 0.
P11_T2EX	Emulation Timer #2	I	—	Connect to external 8051 Port 1, Bit 1.
P12_RXD1	Group Talk Rx Data	I	—	Connect to external 8051 Port 1, Bit 2. See Section 4.5.4 for Group Talk connections.
P13_TXD1	Group Talk Tx Data	O	—	Connect to external 8051 Port 1, Bit 3. When Group Slave, signal will be three-stated to avoid contention. See Section 4.5.4 for Group Talk connections.
P14_INT2	User Interrupt #2 Control	O	—	Connect to external 8051 Port 1, Bit 4.
$\overline{\text{P15\_INT3}}$	DSL Framer Interrupt Control.	O	—	Connect to external 8051 Port 1, Bit 5.
P16_GPIO_CLK	Spare I/O	I/O	PU	No connection required.
P17_GPIO_DAT	Spare I/O	I/O	PU	No connection required.
P30_RXD0	RS232 Rx Data	I	—	Connect to external 8051 Port 3, Bit 0. In addition, need to connect to RS232 Receive Data.
P31_TXD0	RS232 Tx Data	O	—	Connect to external 8051 Port 3, Bit 1. In addition, need to connect to RS232 Transmit Data.
$\overline{\text{P32\_INT0}}$	DSP Interrupt Control	O	—	Connect to external 8051 Port 3, Bit 2.
$\overline{\text{P33\_INT1}}$	Host Port Interrupt Control	O	—	Connect to external 8051 Port 3, Bit 3.
P34_GPIO_SYN	Spare I/O	I/O	PU	No connection required.
$\overline{\text{P35\_PRAM}}$	P3B5	I/O	PU	µP emulation program Read Select.
$\overline{\text{P36\_WR}}$	Emulation WR	I/O	PU	Connect to external 8051 *RD.
$\overline{\text{P37\_RD}}$	Emulation RD	I/O	PU	Connect to external 8051 *WR.
JTAG Interface				
$\overline{\text{DSP\_TRST}}$	DSP Test Port Reset	I	PU	Active-low resets the TAP controller. This pin should be connected high for normal operation.
$\overline{\text{FR\_TRST}}$	DSL Framer Test Port Reset	I	PU	Active-low resets the TAP controller. This pin should be connected high for normal operation.
TDI	Test Data In	I	PU	JTAG test data input per ***IEEE Std. 1149.1-1990.*** Used for loading all serial instructions and data into internal test logic. Sampled on the rising edge of TCK. TDI can be left unconnected when not being used because it is internally pulled high.
TDO	Test Data Out	O	—	JTAG test data input per ***IEEE Std. 1149.1-1990***. Three-state output used for reading all serial configuration and test data from internal test logic. Updated on the falling edge of TCK.

Table 8-1. ZipWire2 Transceiver/Framer Signal Definitions (8 of 9)

Signal	Name	I/O	PU/PD(5)	Description
TCK	Test Clock	I	—	JTAG test data input per ***IEEE Std. 1149.1-1990***. Used for all test interface and internal test logic operations. If unused, TCK should be pulled low.
TMS	Test Mode Select	I	PU	JTAG test data input per ***IEEE Std. 1149.1-1990***. Input signal used to control the test-logic state machine. Sampled on the rising edge of TCK. TMS can be left unconnected when not being used because it is internally pulled high.
Serial Test Port(4)				
STP_TX0	Serial Test Port #0 Data	O	—	DSP Serial Test Port 0 Data.
STP_SYNC0	Serial Test Port #0 Sync	O	—	DSP Serial Test Port 0 Sync.
STP_TX1	Serial Test Port #1 Data	O	—	DSP Serial Test Port 1 Data.
STP_SYNC1	Serial Test Port #1 Sync	O	—	DSP Serial Test Port 1 Sync.
Miscellaneous and Test Modes				
TEST_TWE	—	I	—	Leave floating. No-Connect.
TEST_TWR	—	O	—	Leave floating. No-Connect.
TEST_TWT	—	I	—	Leave floating. No-Connect.
TEST_TWC	—	O	—	Leave floating. No-Connect.
TEST_I1	—	I	—	Must be tied low (to GND).
TEST_I2	—	I	—	Must be tied low (to GND).
TEST_SSE	—	I	—	Must be tied low (to GND).
TEST_STM	—	I	—	Must be tied low (to GND).
TEST_UDR	—	I	—	Must be tied high (to 3.3 V).
TEST_CDR	—	I	—	Must be tied high (to 3.3 V).
TEST_MD1	—	I	—	Must be tied high (to 3.3 V).
TEST_MD3	—	I	—	Must be tied high (to 3.3 V).
TEST_RST	—	I	—	Must be tied high (to 3.3 V).
TEST_SDR	—	I	—	Must be tied high (to 3.3 V).
TEST_JTC	—	I	—	Must be tied high (to 3.3 V).
TEST_FSM	—	I	PU	Leave floating. No-Connect.
TEST_FSS	—	I	PU	Leave floating. No-Connect.
TEST_SEL	—	I	—	Must be tied low (to GND).
TEST_TRI	—	I	—	Must be tied low (to GND).
TEST_BPO	—	O	—	Leave floating. No-Connect.

Table 8-1. ZipWire2 Transceiver/Framer Signal Definitions (9 of 9)

Signal	Name	I/O	PU/PD(5)	Description
TESTMON[9-0]	—	O	—	Leave floating. No-Connect.

NOTE(S):
(1) When using the internal DSL Framer, these pins should be floating (No-Connect).
(2) DSL Framer microprocessor pins are brought out to separate pins for internal production testing.
(3) Only when external 8051 or Emulator is present, otherwise No-Connect.
(4) Internal test ports, these pins should be floating (No-Connect).
(5) Internally Pulled-Up (PU) or Pulled-Down (PD) with a 50–200 kΩ resistor.

8.2.2 ZipWire2 AFE Signal Descriptions

Table 8-2 defines the ZipWire2 AFE Signal (pin) descriptions.

Table 8-2. ZipWire2 AFE Signal Descriptions (1 of 3)

Signal Name	Pin Number	I/O	Description
Power and Ground			
VAA	1, 9, 50, 52, 57, 59, 68, 76	—	+5 V Analog Supply
AGND	2, 10, 31, 32, 34, 48, 49, 51, 53, 56, 60, 67, 75	—	Analog Ground
VIO	30, 33	—	+3.3 V Digital I/O Supply
Transmit Section			
LDON	54	O	Transmit, Negative (–) Line Driver Output
LDOP	55	O	Transmit, Positive (+) Line Driver Output
LDIN	63	I	Transmit, Negative (–) Line Driver Input
LDIP	64	I	Transmit, Positive (+) Line Driver Input
RCDRVP	65	O	Transmit, Positive (+) RC Driver Output
RCDRVN	66	O	Transmit, Negative (–) RC Driver Output
Transmit References			
VRNTX	69	REF	Transmit, Negative (–) Voltage Reference
VRPTX	70	REF	Transmit, Positive (+) Voltage Reference
Receive Section			
VXP	3	I	Receive, Positive (+) Transformer Line Input
VXN	4	I	Receive, Negative (–) Transformer Line Input
VH1P	5	I	Receive, Positive (+) Hybrid 1 Analog Input
VH1N	6	I	Receive, Negative (–) Hybrid 1 Analog Input
VH2P	7	I	Receive, Positive (+) Hybrid 2 Analog Input
VH2N	8	I	Receive, Negative (–) Hybrid 2 Analog Input
Receive References			
RBIAS	71	REF	Reference, Current Reference Resistor
AVBIAS	72	REF	Reference, Compensation Capacitor
VCMI	73	REF	Reference, Input Common Mode Voltage
VCMO	74	REF	Reference, Output Common Mode Voltage
VBGN	77	REF	Reference, Negative (–) Band-gap Reference, Decouple
VBGP	78	REF	Reference, Positive (+) Band-gap Reference, Decouple

Table 8-2. ZipWire2 AFE Signal Descriptions (2 of 3)

Signal Name	Pin Number	I/O	Description
VRNRX	79	REF	Receive, Negative (–) Voltage Reference
VRPRX	80	REF	Receive, Positive (+) Voltage Reference
DSP Interface			
AFE_RST	11	I	AFE Microprocessor Reset
AFE_CLK	21	I	AFE Master Clock (27 MHz)
SER1_RCV	22	O	Rx, Serial Output Data, Line 1
SER2_RCV	23	O	Rx, Serial Output Data, Line 2
SER3_RCV	24	O	Rx, Serial Output Data, Line 3
SER_IRQ	25	O	AFE, Microprocessor Control Signal
UP_R_D	26	O	AFE, Microprocessor Control Signal
UP_W_D	27	I	AFE, Microprocessor Control Signal
AFE_CS	28	I	AFE, Microprocessor Chip Select
AFE_SYNC	29	I	AFE, Microprocessor Sync
SER1_TX	39	I	Tx, Serial Input Data, Line 1
SER2_TX	40	I	Tx, Serial Input Data, Line 2
Miscellaneous and Test			
TEST_RX6	12	I/O	Rx, DSP Test I/O Pin(1)
TEST_RX5	13	I/O	Rx, DSP Test I/O Pin(1)
TEST_RX4	14	I/O	Rx, DSP Test I/O Pin(1)
TEST_RX3	15	I/O	Rx, DSP Test I/O Pin(1)
TEST_RX2	16	I/O	Rx, DSP Test I/O Pin(1)
TEST_RX1	17	I/O	Rx, DSP Test I/O Pin(1)
TEST_RX0	18	I/O	Rx, DSP Test I/O Pin(1)
TEST_CTRL1	19	I	Rx/Tx, DSP Test Control, Pin 1(1)
TEST_CTRL0	20	I	Rx/Tx, DSP Test Control, Pin 0(1)
SCAN_EN	35	I	ASIC Scan Enable Input(1)
SCAN_SHFT	36	I	ASIC Scan Shift Input(1)
TEST_SINE	37	I	Test Mode, Internal Sine Wave Enable(1)
TEST_LB	38	I	Test Mode, Analog-in to Analog-out Loopback Control(1)
TEST_TX0	41	I/O	Tx, DSP Test I/O Pin(1)
TEST_TX1	42	I/O	Tx, DSP Test I/O Pin(1)
TEST_TX2	43	I/O	Tx, DSP Test I/O Pin(1)
TEST_TX3	44	I/O	Tx, DSP Test I/O Pin(1)

Table 8-2. ZipWire2 AFE Signal Descriptions (3 of 3)

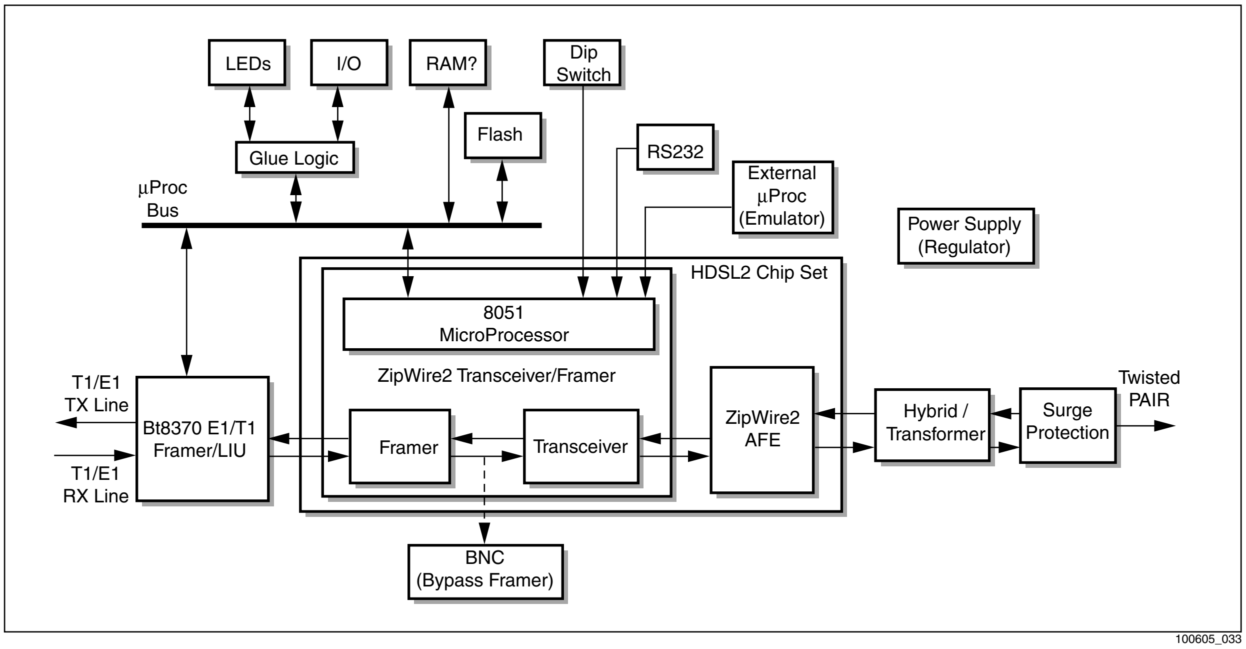
Signal Name	Pin Number	I/O	Description
TEST_TX4	45	I/O	Tx, DSP Test I/O Pin(1)
TEST_FU	46	I	Test Mode(1)
TEST_CNI	58	I	Test Mode(1)
TEST_VESD	47	I	ESD Protection(3)
TEST_LD1	61	I	Transmit, LD Spare(2)
TEST_LD2	62	I	Transmit, LD Spare(2)

NOTE(S):
(1) These pins must be tied to AGND.
(2) These pins must be left unconnected.
(3) These pins must be tied to +5 V.

9.0 EVM Specific

This section describes any interaction of the software that is specific to the ZipWire2 EVMs. Figure 9-1 shows the block diagram of the ZipWire2 Evaluation Module (EVM). This section is provided since the ZipWire2 EVM uses the same software that is sent to customers.

Figure 9-1. EVM Block Diagram



The Generic Chip Select (CS7) is used to address the external devices on the EVM.

Table 9-1. Generic Chip Select CS7 Memory Map

Memory Range	A13	A12	A11	Device
0xB000 – 0xB7FF	0	0	0	Bt8370 T1/E1 (input/output)
0xB800 – 0xBFFF	0	0	1	LEDs #1 (output) / DIP Switch #3 (input)
0xC000 – 0xC7FF	0	1	0	LEDs #2 (output) / DIP Switch #4 (input)
0xC800 – 0xCFFF	0	1	1	Reserved—Internal Use Only
0xD000 – 0xD7FF	1	0	0	Reserved
0xD800 – 0xDFFF	1	0	1	Reserved
0xE000 – 0xE7FF	1	1	0	Reserved
0xE800 – 0xEFFF	1	1	1	Reserved

9.1 Bt8370 E1/T1 Framer

A Bt8370 E1/T1 Framer/LIU is connected to the ZipWire2 framer PCM interface. The Bt8370 can then be connected to an external BER meter for bit error measurements.

9.2 EVM LEDs and Miscellaneous Output

The LED Registers are 8-bit write only and are used to display status information via LEDs about the CN8980 system. An LED is lit when a 1 is present in the corresponding bit of the register. Table 9-2 and Table 9-3 defines the bits of the LED Registers.

Table 9-2. DSL Status LED #1 Register—Write Only

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DPLL ERR (red)	RFIFO ERR (red)	TFIFO ERR (red)	LOSW (red)	LOS (red)	Bad NMR (red)	Fatal Error (red)	System IN SYNC (green)

If the Fatal Error LED is set, then LEDs 2–7 provide an error code (TBD).

Table 9-3. T1/E1 Framer Output / LED #2 Register—Write Only

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
T1/E1 Error (red)	T1/E1 LOF (red)	T1/E1 Sync (green)	Test Mode (orange)	Multi-EVM Reset	XOE Control	T1/E1 Term 75	T1/E1 Term 100

9.3 EVM DIP Switches

The DIP switches are used to determine the program configuration and flow (see Section 4.4). Table 9-4 defines the DIP switch bit definitions.

NOTE: If the DIP switch is used in customer applications, the DIP switch bits would most likely be connected to either +5 V (high) or GND (low) to hard-code their application.

Table 9-4. DIP Switch 3 Bit Definitions

DIP Switch #	Description	Bit(s) Definition
1–0	Frame Structure	00 = Bypass 01 = HDSL1 10 = HDSL2 11 = Reserved
4–2	Configuration	000 = 2T1 001 = 2E1 010 = 3E1 011 = Reserved 100 = 1T1 101 = 1E1 110 = Reserved 111 = Reserved
7–5	Test Modes and Loopbacks	000 = Normal Operation 001 = Isolated +3 010 = 4-Level Scr 1's 011 = AFE Hybrid Analog Loopback 100 = BP Digital Near Loopback 101 = CU PRA on PRA Loopback 110 = FR LIU Line Loopback(1) 111 = Reserved

NOTE(S):
(1) Requires T1/E1 Framer.

Table 9-5. DIP Switch 4 Bit Definitions

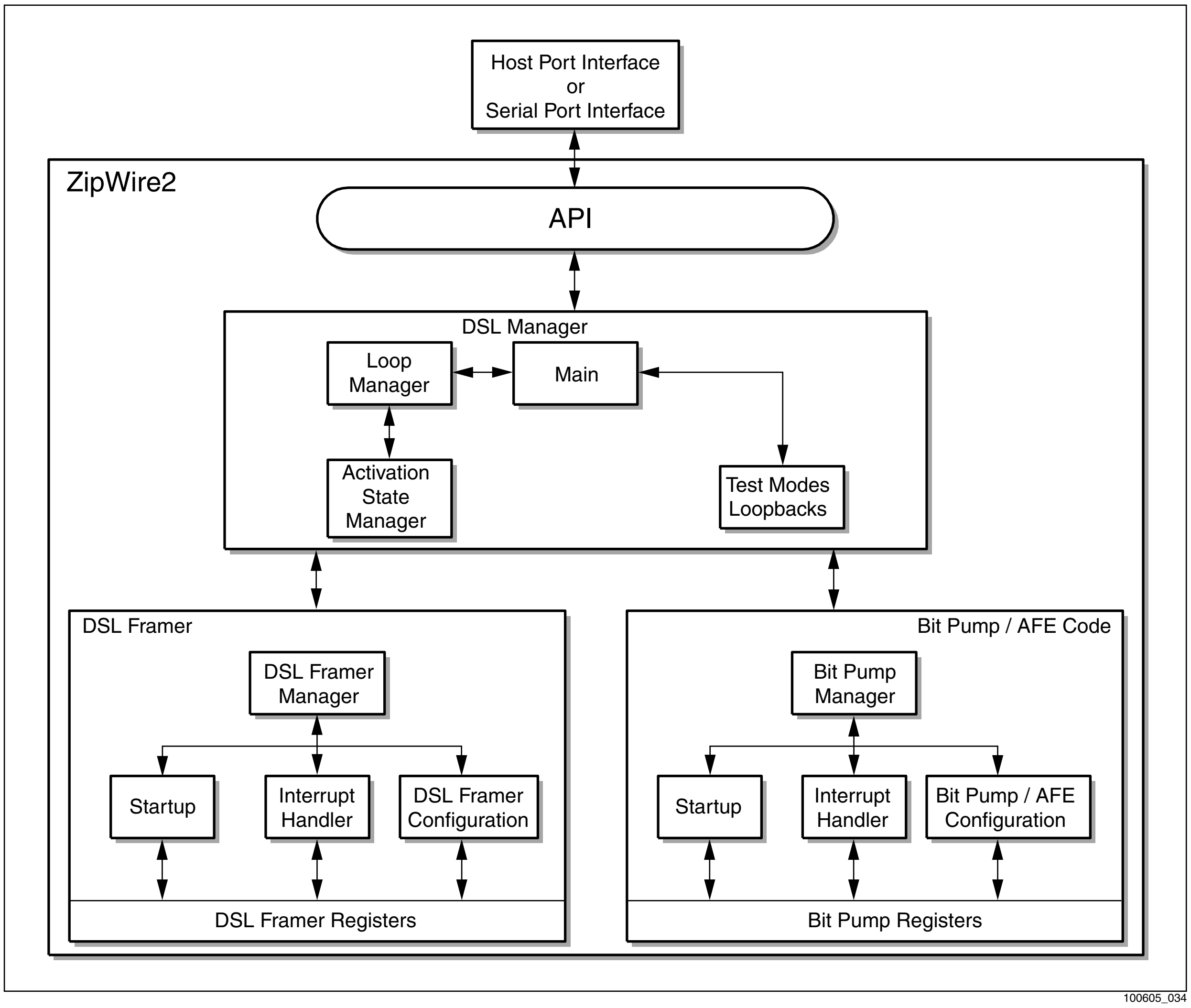
DIP Switch #	Description	Bit(s) Definition
7–0	Undefined	—

10.0 Software Overview

Figure 10-1 shows the software overview of the HDSL chip sets. The Bit Pump code is responsible for the transceiver functionality (DSP + AFE) while the DSL Framer code is responsible for the framing and mapping functionality. The DSL Manager calls the Bit Pump and DSL Framer code to activate and maintain the system. The Host Processor (or Serial) interface provides a way to control the system from a host processor.

NOTE: Even though the AFE is a separate chip, the AFE is controlled by the DSP.

Figure 10-1. Software Overview



10.1 Software Features

The following provides a list of software features.

- Standalone mode
- OPTIS (HDSL2) support
- HDSL2 applications: 1T1, 1E1, Single Pair, and Custom
- API
- Loopbacks
- Test modes
- Ability to control other devices, i.e., T1/E1 Framer
- BER meter(s)
- Performance Monitoring
- EOC/HDLC
- One-second timer
- 2B1Q (HDSL1) support
- HDSL1 Applications: 2T1, 2E1, 3E1, 1T1, 1E1, Single Pair, and Custom
- Loop reversal – only applies to 2T1, 2E1, and 3E1
- Switch master loop – only applies to 2T1, 2E1, and 3E1
- PRA
- Autobaud

11.0 Embedded 8051 Software Features

This section describes the major software blocks (features) of the ZipWire2 chip set.

11.1 Activating the ZipWire2 Modem

The activation period is the time from when the activate (go) command is issued until the ZipWire2 is passing payload data. The ZipWire2 solution (DSP and DSL Framer) is extremely flexible and can support several options. See Table 11-1 for the major phases of activation.

Table 11-1. Activation Phases

Phase	Description
Pre-Activation	Pre-activation allows the two modems to communicate messages to determine the other aspects of training. The pre-activation protocol is typically independent of the line coding.
DSP Training	DSP training is the handshake that allows the two modems to train the DSP engine (adapt filters, perform timing recovery, and so on).
DSL Line Coding	DSL line coding sets the final line coding configuration, such as 4PAM, 8PAM, Trellis On/Off, and so on.
Frame Format	Frame format determines how the payload data is encapsulated into the DSL frame. The frame typically consists of a sync word, EOC, CRC, and so on.

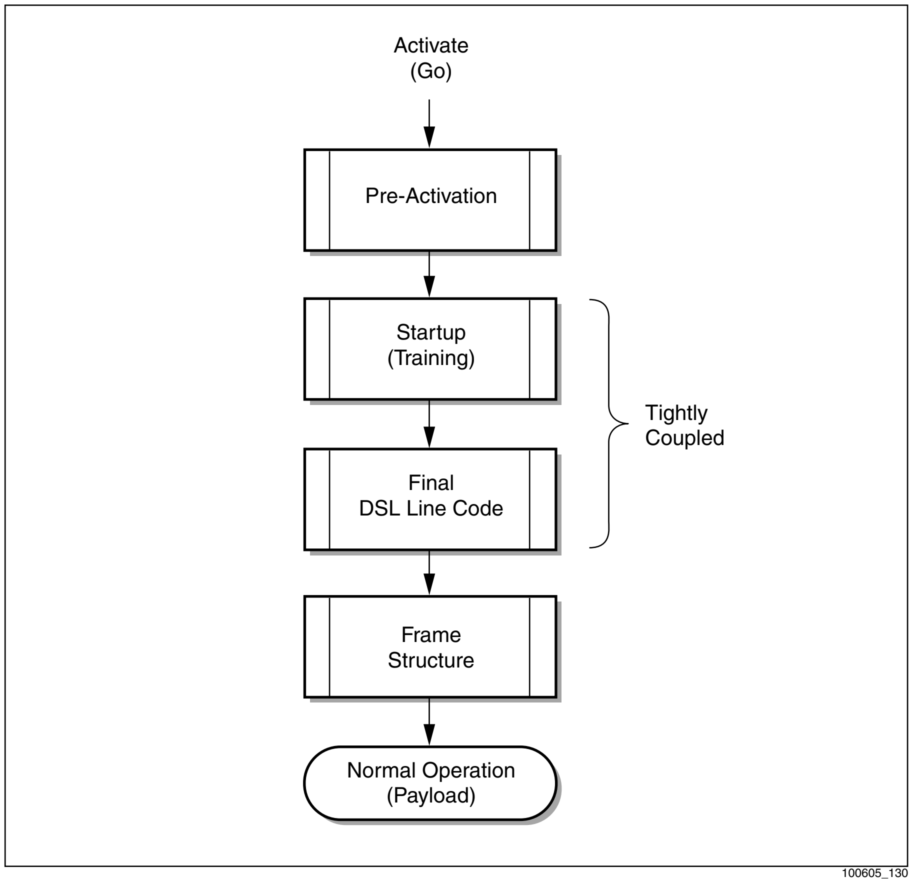
11.1.1 Activation State Manager (ASM)

The Activation State Manager (ASM) sits on top of these activation phases to determine when the link is good, and when to bring down and retrain the link. The ASM performs flow control and error handling.

In certain standards, the different activation phases overlap while other standards are completely independent tasks. For example, in HDSL1 (2B1Q), the frame format passes the sync word and indicator bits before the DSP finishes its training. In HDSL2 (OPTIS), the DSP must complete its training before the DSL Framer begins transmitting and looking for the sync word.

Figure 11-1 illustrates an overview of the activation. See Section 12.11 for more details on the ASM.

Figure 11-1. Activation State Manager (ASM) Overview



11.1.2 Pre-Activation

Pre-activation can be as simple as determining if a far-end unit is present or as sophisticated as remotely configuring the far-end unit and determining the optimal data rate. The ZipWire2 supports the following pre-activation modes:

- OPTIS (HDSL2 1T1) Pre-Activation
- AutoBaud
- G.hs (future revision of silicon)

11.1.2.1 OPTIS (HDSL2 1T1) Pre-Activation

OPTIS pre-activation provides a simple ping to determine that the far-end is present and also communicates with the transmit power back-off control. The OPTIS pre-activation only supports the HDSL2 (1T1) mode.

11.1.2.2 AutoBaud

AutoBaud is a Conexant solution that pings to determine that the far-end is present, determines line quality, and remotely configures the optimal data rate and frame format (layer 2 information).

11.1.2.3 G.hs

G.hs is specified in the *ITU G.shdsl* standards. G.hs is also included in the ADSL standards.

G.hs is currently not supported by the ZipWire2 silicon.

11.1.3 DSP Training

The DSP (bit pump) Training handshake allows the two modems to train and pass data. This includes the 2-, 4-, and 16-level timelines, precoder tap exchange, and so on. The ZipWire2 supports the following training modes:

- 2B1Q—always uncoded
- OPTIS—includes precoder tap exchange, can support both coded and uncoded modes
- G.shdsl—includes precoder tap exchange, can support both coded and uncoded modes

11.1.4 DSL Line Coding

The DSL line coding is typically defined by standards such as uncoded 4-PAM for 2B1Q or coded 16-PAM for OPTIS and G.shdsl. The DSL line coding is tightly coupled with the training mode.

In non-standard applications, there could be some advantages to switching to a different DSL line code. For example, the ZipWire2 modems could use the 2B1Q training to train the DSP then switch to uncoded 16-PAM line code; this could allow the modems to achieve twice the data rate at the expense of loop length. Another example would be using the G.shdsl to train (which includes the precoder tap exchange), then switch to a coded 4-PAM line code; this would allow the modems to achieve an extended reach at the expense of data rate.

11.1.5 Frame Format

The ZipWire2 supports the following frame formats:

- Framer Bypass—used to support legacy RS8973 applications.
- HDSL1
- HDSL2 (OPTIS)
- G.shdsl

The frame format is typically defined by the standards. However, as with the DSL line coding, there is flexibility in how the frame format can be configured. For example, the modems could train using the 2B1Q mode then switch to uncoded 16-PAM and use the G.shdsl frame format. Another example could train using the OPTIS coded 16-PAM then use the G.shdsl frame format.

There is minimal dependency from the frame format to the DSP training. The main requirement is that the DSP data rate (regardless of training mode and DSL line coding) matches the expected frame format.

11.2 Loopbacks

There are several loopbacks in the ZipWire2 chip set, as illustrated in Figure 11-2. The loopbacks are controlled by the API Command loopbacks. For additional information, see Section 17.4.11.

NOTE: The T1/E1 Framer loopbacks are supported only when the T1/E1 is enabled. See Section 17.7.6 for a description of this command.

Figure 11-2. ZipWire2 Loopbacks

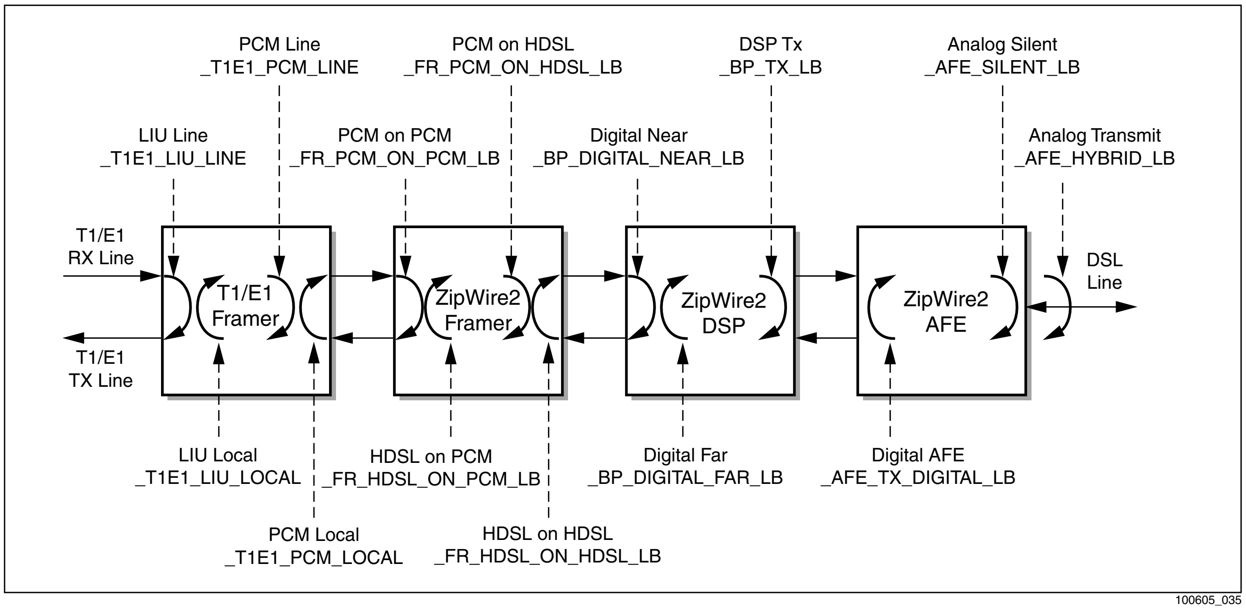
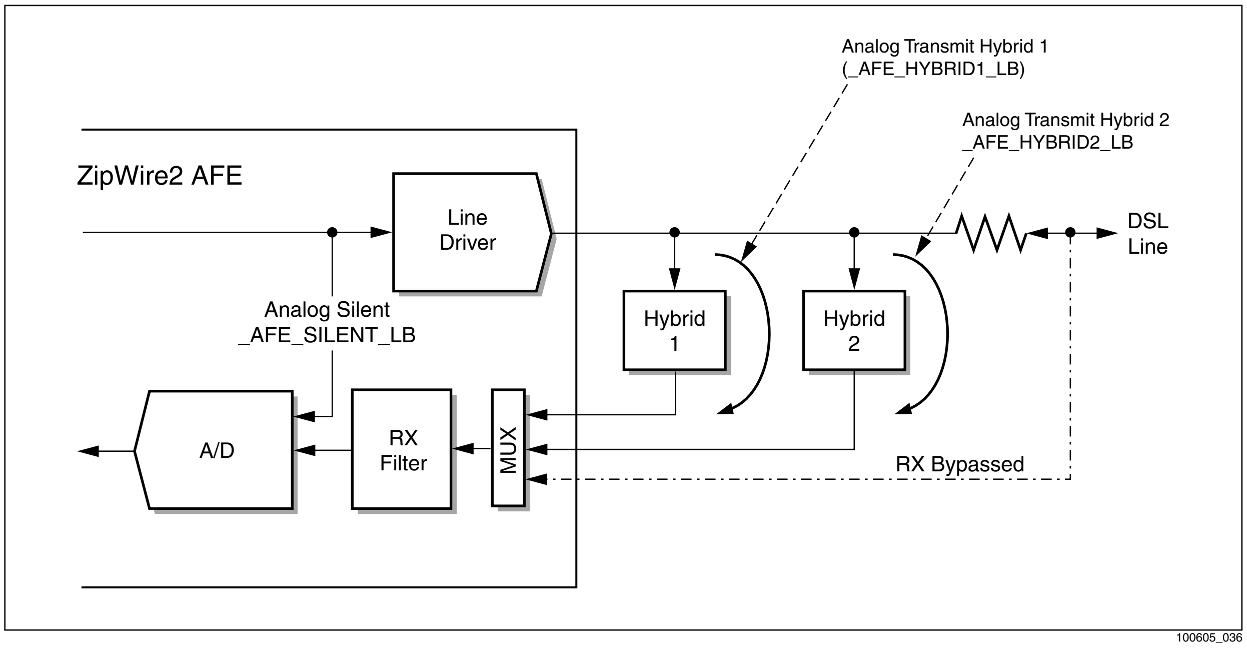


Figure 11-3. Detailed AFE Loopbacks



11.3 BER Meters

The ZipWire2 device has three independent BER meters. The internal BER meters can be used to test the HDSL link or PCM interface data path without an external BER tester.

- DSL Framer Tx PCM block—overrides transmitted PCM data on selected time slots; OH unaffected.
- DSL Framer Rx PCM block—overrides received PCM data on selected time slots; OH unaffected.
- Bit Pump Only—overrides all transmitted DSL data with a scrambled 1s pattern. DSL Framer errors should be ignored.

11.3.1 DSL Framer Tx PCM BER Meter

PCM time slots from TSER or RSER can be examined for test patterns on a per time slot basis, or the entire frame unframed PCM channel from TSER can be examined. The user can select from several Pseudo-Random Bit Sequence (PRBS) patterns (see Section 17.4.22) or an 8-bit fixed pattern (see Section 17.4.23).

11.3.2 DSL Framer Rx PCM BER Meter

TBD

11.3.3 Bit-Pump-Only BER Meter

The Bit-Pump-Only BER meter uses its internal scrambled 1s generator and descrambler to detect bit errors. For the BER meter to function properly, both the HTU-C and HTU-R must issue the _BP_BER_METER_STATE API command. Because the DSP uses its own internal scrambled 1s generator, the BER meter cannot be used while transporting external payload data. The Bit-Pump-Only BER meter is only valid when the bit pump has successfully completed startup. Refer to Section 17.4.12 and Section 17.4.13 for details.

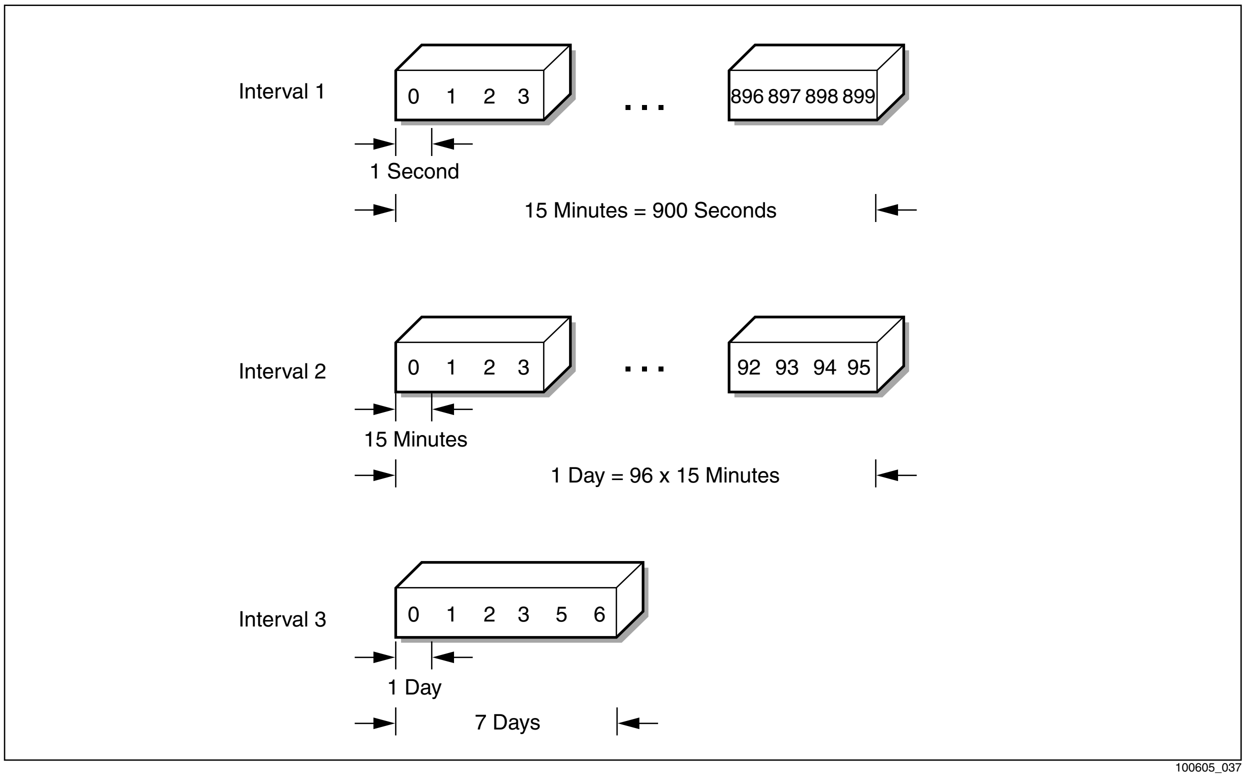
11.4 Performance Monitoring (Error History)

Performance monitoring maintains a history of CRC and FEBE errors at different time intervals. The performance monitoring feature is useful to determine when a link deployed in the field begins to experience problems. This information is typically only requested when a serious problem is reported. Figure 11-4 shows the structure of the records.

In addition, the ZipWire2 Framer Error Counters maintain a running accumulation of CRC and FEBE errors from the time the link reached normal operation or when the Clear ZipWire2 Framer Error Counters API command was last issued. These error counters are each 16-bits (2-bytes) wide. The CRC and FEBE error accumulation is useful during development to verify the quality of the link since it reports all errors and is updated every 6 ms HDSL frame.

The performance monitoring API commands are described in Section 17.4.34 to Section 17.4.41.

Figure 11-4. CRC and FEBE Error Records at Three Time Intervals



Interval 1 records the number of CRC and FEBE errors occurring in 1-second intervals for 15-minute time periods, and is updated every second.

Each 1-second entry is 1-byte wide and can record up to 255 errors. The error counters will stop incrementing when they reach the maximum of 255 errors. Record entry 0 corresponds to the previous second, while entry 899 corresponds to 900 seconds previous.

Interval 2 records the number of CRC and FEBE errors occurring in each 15-minute for a 24-hour time period, and is updated every 15 minutes. Each 15-minute entry is 2-bytes wide and can record up to 65,535 errors. The error counter will stop incrementing when it reaches the maximum of 65,535 errors. Record entry 0 corresponds to the previous 15-minute interval, while entry 95 corresponds to 96, 15-minute intervals previous.

Interval 3 records the number of CRC and FEBE errors occurring in each 24-hour interval for a 7-day period and is updated daily. Each one-day entry is 2-bytes wide and can record up to 65,535 errors. The error counters will stop incrementing when they reach the maximum of 65,535 errors. Record entry 0 corresponds to the previous 1-day interval while entry 6 corresponds to 7, 1-day intervals previous.

11.5 DSL Framer Interrupt Handler

The DSL Framer interrupt handler processes 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 as well as DPLL errors are handled when they occur; EOC request is processed and pair ID is validated during the startup stage.

11.5.1 Sync Status

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

11.5.2 Error Status Reporting

The Tx/Rx interrupt functions check the status registers or indication bits for errors. Whenever an error occurs, the corresponding error counter will increment. Use the API commands listed in Section 17.4.34 to query the error counter results.

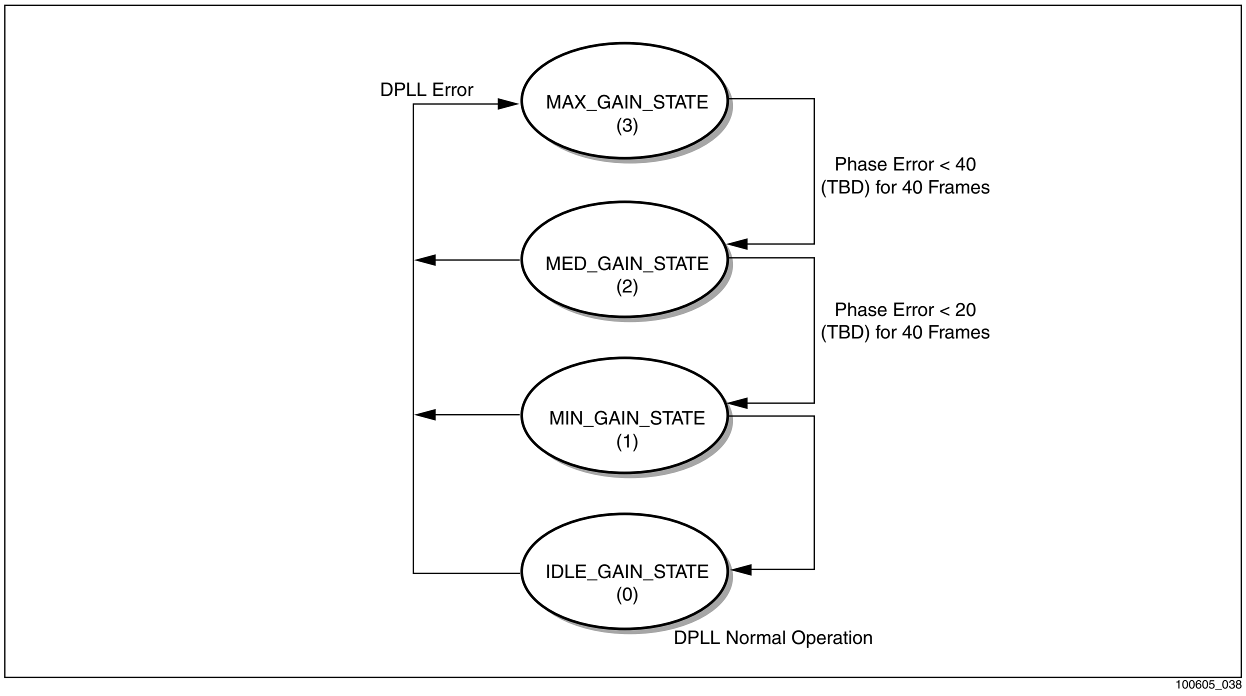
11.5.3 Tx/Rx FIFO Error Handling

In the Tx/Rx Interrupt functions, whenever a FIFO error occurs, the corresponding FIFO gets reset. Resetting the FIFO causes three HDSL Frames to be lost (corrupted), which would cause another Tx/Rx FIFO error, so the first three passes into the Tx/Rx Interrupt function are ignored after resetting the Tx/Rx FIFO. If the master loop RFIFO reports an error, then a RX_RST is also issued.

11.5.4 DPLL Error Handling

The DPLL State Machine function is responsible for processing any DPLL errors. Figure 11-5 illustrates the DPLL State Machine. The DPLL Error Interrupt is triggered whenever the DPLL Phase Error exceeds the threshold (*threshold to be determined*). 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 11-5. DPLL State Diagram



The basic concept of the DPLL State Machine is to increase (open up) the DPLL bandwidth whenever an error occurs; this allows 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.

MAX_GAIN_STATE:	Whenever a DPLL error occurs, the DPLL state is set to MAX_GAIN_STATE. The DPLL bandwidth is set to TBD (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 MED_DPLL_GAIN_VALUE. When the DPLL Phase Error reads less than 20 for 40 consecutive frames, the DPLL State is set to MIN_GAIN_STATE.
MIN_GAIN_STATE:	The DPLL bandwidth is set to MIN_DPLL_GAIN_VALUE. A PCM and FIFO reset is issued which resynchronizes the PCM time base 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 does nothing.

NOTE: The DPLL may take up to 10 seconds to stabilize after a DPLL error is detected.

11.5.5 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 will be called. A loop's pair ID is validated only after a valid, unique pair ID occurs consecutively for six frames.

The pair ID only applies to HDSL1 operation.

11.5.6 Indicator Bit Termination

If CRC error is detected, the FEBE indication bit is set; if E1/T1 LOS and OOF are detected, the LOSD indication bit is set. The updated indication bits will be written to the TIND register in the next HDSL transmit interrupt handler.

11.6 Dynamic Master Loop

The switching master loop is only supported in HDSL1 2E1, 2T1, and 3E1 modes.

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. Whenever a master loop failure is detected, the system switches the master loop to the next available loop in normal operation.

11.7 Tip/Ring Reversal

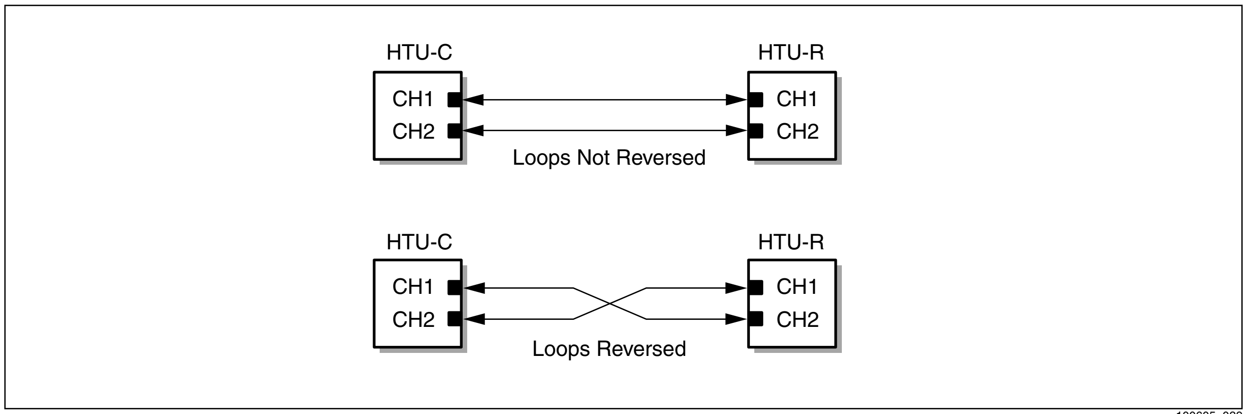
Tip/ring reversal is defined as the reversal of a twisted pair of wires. The ZipWire2 device and software automatically handle any tip/ring reversal.

11.8 Loop Reversal

The loop reversal is only supported in HDSL1 2E1, 2T1, and 3E1 modes. The loop reversal is not applicable in single pair systems (1T1 and 1E1).

Loop reversal is defined as the loop pairs are reversed as shown in Figure 11-6. The HTU-R monitors the Sync Word (T1) or loop's pair ID (E1) and configures the HTU-R PCM and HDSL Map tables accordingly.

Figure 11-6. Loop Reversal Definition



11.9 Embedded Operation Channel (EOC) Operation

This section provides an overview of the Embedded Operations Channel (EOC).

11.9.1 Feature Overview

- Conforms to ANSI Standards.
- Internal 8051 handles the low-level processing:
  – HDLC-like messaging: flag insertion, data transparency, and frame error checking.
  – Provides flow control: handles errors and time-out.
- Internal 8051 processes the commands as specified in the standards:
  – Performs discovery probe, inventory, and configuration.
  – Performs full status once-a-second.
  – Separate shadowed buffers for each command so host can query results at its leisure.
- Additional Features:
  – Uses 2 of the proprietary commands for User Defined Messages and API over EOC commands.
- Host Responsibilities:
  – Update certain databases.
  – Can initiate request or response messages.

11.9.2 Does Not Support

The software currently doesn’t support the following features. The silicon supports these features but the software has not been implemented:

- Regenerators
- Virtual Terminal Commands

11.9.3 EOC General Overview

The EOC provides a communication channel between the ZipWire2 terminal units. This allows the units to communicate configuration and status messages. The HTU-C is the master and initiates the EOC messages. The HTU-R can optionally support initiating a message. Therefore, both the HTU-C and HTU-R must respond to message requests.

In the ZipWire2 solution, the HTU-R only the initiates the Discovery Probe request which is used as a final qualifier to determine the link is stable. The HTU-R supports all other EOC messages; however, the host processor must trigger the messages.

There are two types of messages: requests and responses.

Table 11-2. EOC Message Types

Msg Type	Msg ID Range	Description
Request	0x00–0x7F	Used to configure the far-end, query the far-end for status, or command the far-end to perform some task.
Response	0x80–0xFF	Used to acknowledge a request and to provide status (or other information).

A complete transaction consists of a request by the near-end followed by a response from the far-end.

11.9.4 EOC Frame Format

The EOC channel shall carry messages in an HDLC-like format as defined in *ITU-T G.997.1,* paragraph 6.2. The channel shall be treated as a stream of octets; all messages shall be an integral number of octets.

The frame format uses a compressed form of the HDLC header and is illustrated in Table 11-3. The destination address field shall be the least significant 4 bits of octet 1; the source address field shall occupy the most significant 4 bits of the same octet (the address field.) There is no control field. One or more sync octets (0x7E) shall be present between each frame. Interframe fill shall be accomplished by inserting sync octets as needed. The Information Field contains exactly one Message as defined below. The maximum length of a frame shall be 75 octets, not including the sync pattern or any octets inserted for data transparency.

Table 11-3. Frame Format for HDSL2 EOC

Octet #	MSB Contents	LSB	
	Sync pattern 0x7E		
1	Source address bits 7...4	Destination address bits 3...0	
2	Message ID per Table 11-5		Information
	Message Content – Octet 1		Field
	...		...
	Message Content – Octet L		...
L + 3	FCS Octet 1		
L + 4	FCS Octet 2		
	Sync pattern 0x7E		

11.9.5 EOC Unit Addresses

Each unit uses one source and destination address when communicating with upstream units, and a separate independent source and destination address when communicating with downstream units. Each address shall have a value between 0x0 and 0xF as defined in Table 11-4.

NOTE: The standards have yet to completely define the regenerator specifications; therefore, the ZipWire2 code does not support any Regenerator destination. In addition the Adjacent Device option is not supported.

Table 11-4. EOC Device Address

Value (C Constant)	Device
0x00 (_EOC_ADJACENT)	Adjacent Device
0x01 (_EOC_H2TUC)	HTU-C
0x02 (_EOC_H2TUR)	HTU-R
0x03 (_EOC_H2RU1)	Regenerator 1
0x04 (_EOC_H2RU2)	Regenerator 2
0x05 (_EOC_H2RU3)	Regenerator 3
0x06 (_EOC_H2RU4)	Regenerator 4
0x07 (_EOC_H2RU5)	Regenerator 5
0x08 (_EOC_H2RU6)	Regenerator 6
0x09 (_EOC_H2RU7)	Regenerator 7
0x0A (_EOC_H2RU8)	Regenerator 8
0x0B–0x0E	Reserved
0x0F (_EOC_H2BCAST)	Broadcast

11.9.6 EOC Message IDs

Table 11-5 summarizes the request and response message IDs. Messages 0-127 represent request messages. Messages 128-255 represent messages that are sent in response to request messages. Each request message is acknowledged with the corresponding response. Request/Response Message IDs usually differ by an offset of 128.

Table 11-5. Summary of EOC Request Message IDs (1 of 2)

Message ID (decimal)	Message ID (hex)	Message Type	Unit Which Initiates
Request Messages			
0	0x00	Reserved	—
1	0x01	Discovery Probe	H2TU-C, H2TU-R(1), H2RU
2	0x02	Inventory Request	H2TU-C, H2TU-R(1)
3	0x03	Configuration Request—HDSL2	H2TU-C
4	0x04	Configuration Request—DS1	H2TU-C
5	0x05	Configuration Request—Loopback Time-out	H2TU-C, H2TU-R(1)
6	0x06	Virtual Terminal Connect Req.	H2TU-R(1), H2RU(1)
7	0x07	Virtual Terminal Disc. Req.	H2TU-R(1), H2RU(1)
8	0x08	Keyboard Data Message	H2TU-R(1), H2RU(1)
9	0x09	Maintenance Request—System Loopback	H2TU-C, H2TU-R(1)
10	0x0A	Maintenance Request—Element Loopback	H2TU-C, H2TU-R(1)
11	0x0B	Status Request	H2TU-C, H2TU-R(1)
12	0x0C	Full Status Request	H2TU-C, H2TU-R(1)
13–14	0x0D–0x0E	Reserved	—
15	0x0F	Soft Restart/Power Back-Off Disable Request	H2TU-C
16–63	0x10–0x3F	Reserved (Future)	—
64–88	0x40–0x58	Reserved for Line Management Request	—
89–111	0x59–0x6F	Reserved	—
112	0x70	Proprietary Message (User-Defined Message)	—
113	0x71	Proprietary Message (API over EOC)	—
114–119	0x72–0x78	Proprietary Message	—
120	0x79	External Message	—
121–124	0x7A–0xC	Reserved	—
125–127	0x7D–0x7F	Excluded	—
Response Messages			

Table 11-5. Summary of EOC Request Message IDs (2 of 2)

Message ID (decimal)	Message ID (hex)	Message Type	Unit Which Initiates
128	0x80	Reserved	—
129	0x81	Discovery Response	—
130	0x82	Inventory Response	—
131	0x83	Configuration Response—HDSL2	—
132	0x84	Configuration Response—DS1	—
133	0x85	Configuration Response—Loopback Time-out	—
134	0x86	Virtual Terminal Connect Response	—
135	0x87	Reserved	—
136	0x88	Screen Data Message	—
137	0x89	Maintenance Status	—
138	0x8A	Reserved	—
139	0x8B	Status/SNR	—
140	0x8C	Performance Status HDSL2 Network Side	—
141	0x8D	Performance Status HDSL2 Customer Side	—
142	0x8E	Performance Status DS1	—
143–191	0x8F–0xBF	Reserved (Future)	—
192–216	0xC0–0xD8	Segment Management Response (Reserved)	—
217–239	0xD9–0xEF	Reserved (Future)	—
240	0xF0	Proprietary Message (User-Defined Message)	—
241	0xF1	Proprietary Message (API over EOC)	—
242–247	0xF2–0xF7	Proprietary Message Response	—
248–252	0xF8–0xFC	Reserved	—
253–255	0xFD–0xFF	Excluded	—

NOTE(S):
(1) Optional support. A unit may initiate this message.

11.10 EOC Implementation Details

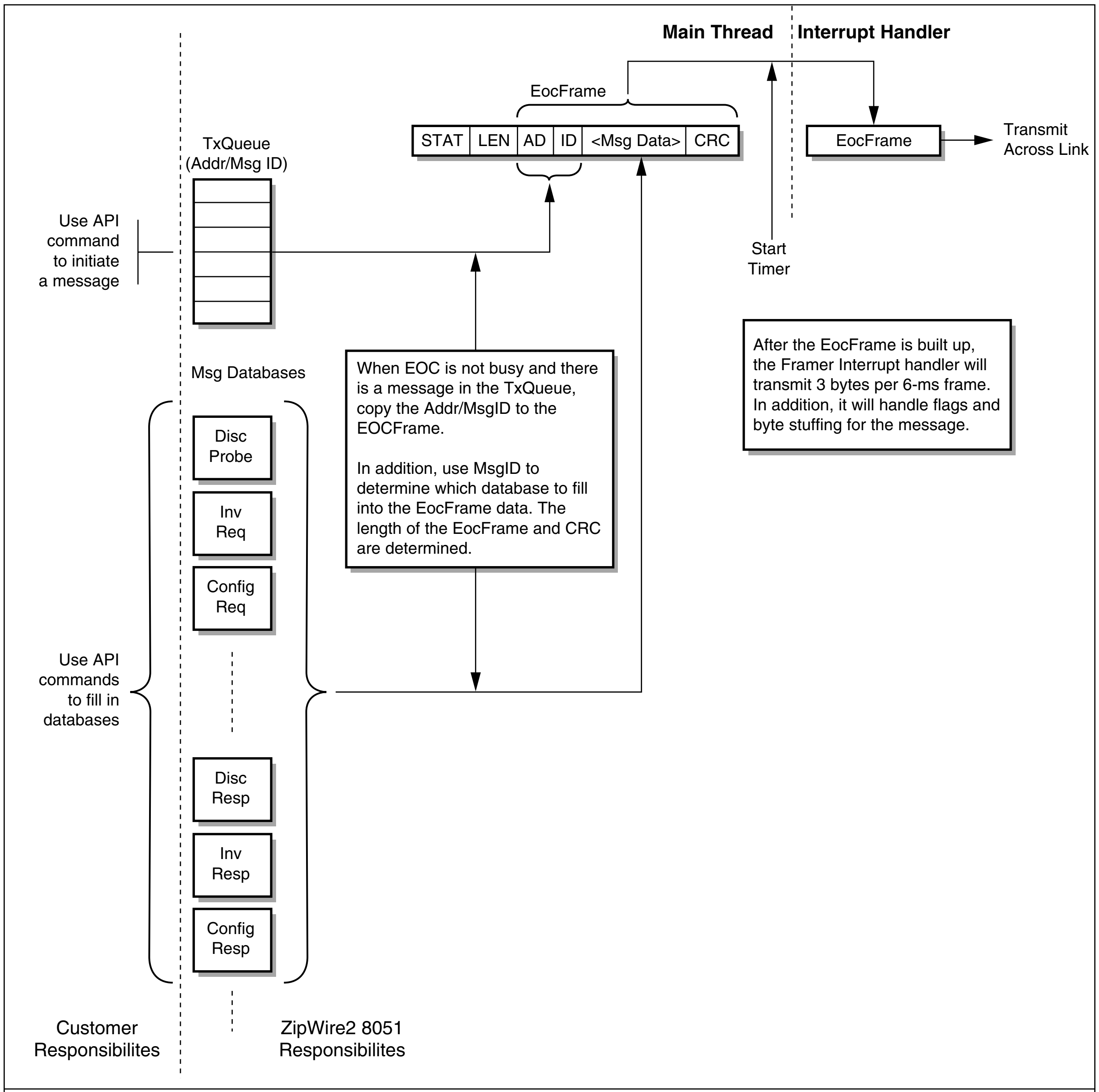
This section provides some details on how the EOC is implemented. Understanding what the 8051 is doing will assist customers in designing their host processor code.

11.10.1 EOC Transmit

Figure 11-7 illustrates the EOC transmit implementation. The request and response message databases are either filled in by the host processor using API commands or from the internal 8051 processor. After the database contains the appropriate information, the host processor or internal 8051 initiates the message. The message ID and destination address are placed into the TxQueue to wait for any current messages to complete.

Once the EOC channel is ready, the internal 8051 will build up the EOC frame into the XmtEocData[]. This consists of using the destination address and message ID from the TxQueue, copying the associated message database into the XmtEocData[], and generating the CRC value. After the EOC frame is complete, the DSL Framer interrupt handler will transmit 3 bytes per 6-ms frame. The interrupt handler will insert the appropriate data transparency bytes and sync octets (flags).

Figure 11-7. EOC Implementation Details—Transmit


Main Thread
Interrupt Handler
EocFrame
STAT
LEN
AD
ID
<Msg Data>
CRC
EocFrame
Transmit Across Link
TxQueue (Addr/Msg ID)
Use API command to initiate a message
Start Timer
After the EocFrame is built up, the Framer Interrupt handler will transmit 3 bytes per 6-ms frame. In addition, it will handle flags and byte stuffing for the message.
Msg Databases
When EOC is not busy and there is a message in the TxQueue, copy the Addr/MsgID to the EOCFrame.
In addition, use MsgID to determine which database to fill into the EocFrame data. The length of the EocFrame and CRC are determined.
Disc Probe
Inv Req
Config Req
Use API commands to fill in databases
Disc Resp
Inv Resp
Config Resp
Customer Responsibilites
ZipWire2 8051 Responsibilites

NOTE(S): The 8051 can also fill in the databases and initiate messages.

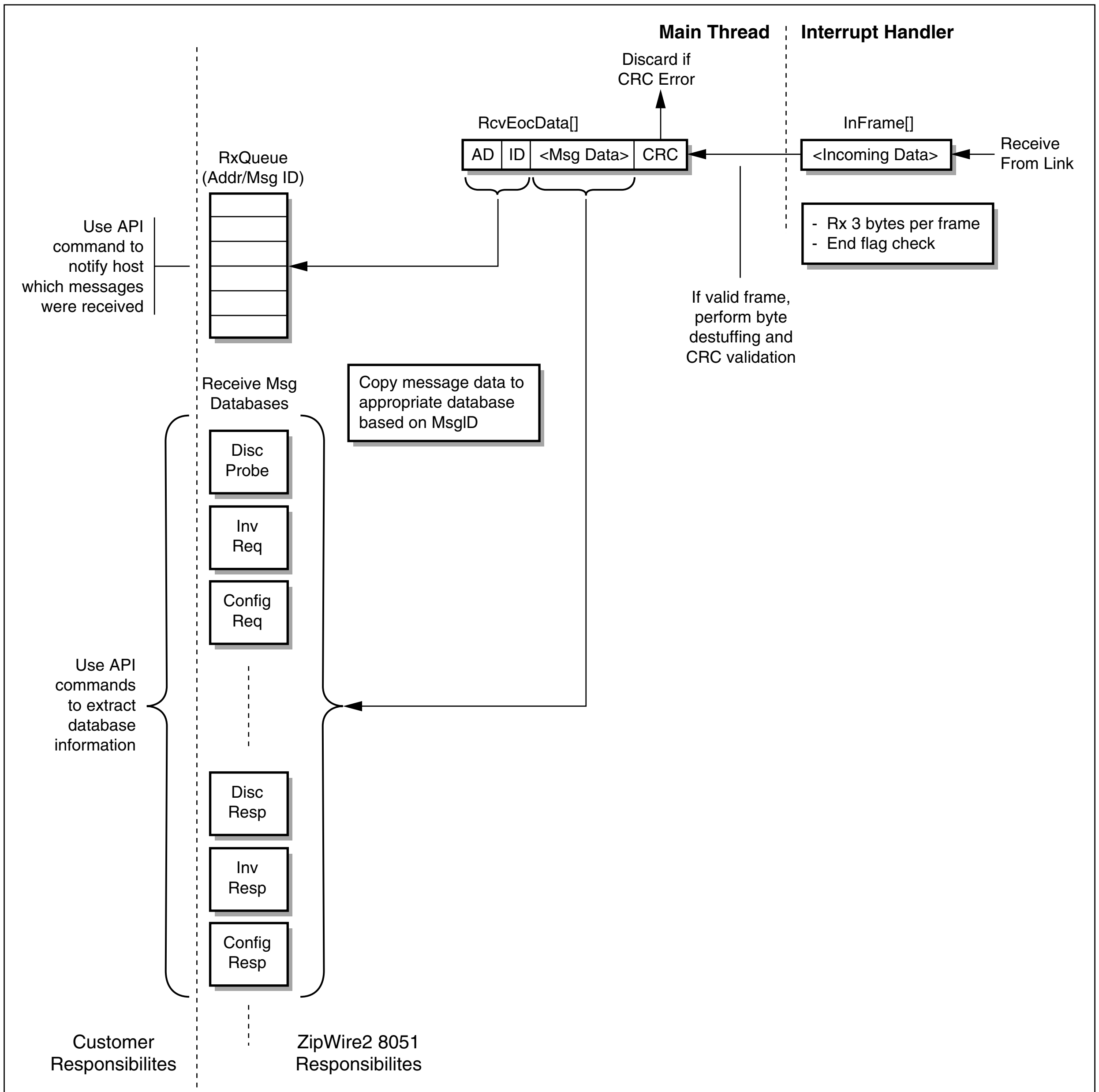
100605_103

11.10.2 EOC Receive

Figure 11-8 illustrates the EOC receive implementation. The DSL Framer will receive 3 bytes per 6-ms frame. The interrupt handler will examine the incoming EOC data for a non-flag character to mark the beginning of the EOC frame. The next flag character will mark the end of the EOC frame. If the data is a valid frame, the internal 8051 will then undo the data transparency and validate the CRC. If the CRC is valid and the destination address matches the terminal unit, the message data is copied into the appropriate receive message database based on the message ID.

The actions taken next are dependant on how the specific message control was configured and the message type (request or response). If a request message is received, the internal 8051 can either generate the response message using the current response database or the internal 8051 can forward the received message to the host processor using the RxQueue buffer. If a response message is received, the internal 8051 can either process the response information or forward the information to the host processor using the RxQueue buffer. See Section 11.10.7 for more details on the message control and RxQueue buffer.

Figure 11-8. EOC Implementation Details—Receive



NOTE(S): The 8051 can also fill in the databases and initiate messages.

100605_104

11.10.3 EOC Transaction Time

The worst-case time for a transaction would be 600 ms, assuming a 75-byte message.

$75 \times 2 = 150$ bytes	Times 2 if every byte required data transparency
150 / 3 = 50 frames	3 bytes transmitted per frame
50×6 ms = 300 ms	6 ms per frame
300 ms $\times 2$ = 600 ms	Times 2 for request and response message

11.10.4 EOC Transaction Time-Out

According to the standards, the only time restriction is that a regenerator must forward a message with 300 ms and that it is up to the initiating terminal unit to handle dropped packets or a no-response. The ZipWire2 solution sets the transaction time-out to 1 second to allow for queuing.

The ZipWire2 solution does not create a time-out for when the message is in the TxQueue buffer. The host processor should create a watch-dog time-out of 20 seconds to safeguard against any issues with the message queuing.

11.10.5 EOC Message Control

The EOC message control determines how each request and response message is handled by the internal 8051. In general, the internal 8051 can either process the message or forward the message to the host for processing.

Message Control Summary:

- Auto-response to request message
- Auto-trigger a request message
- Notify host that a request or response message was received
- Notify host if an error is detected

11.10.5.1 Auto-Respons e To Request Message

This option determines if the internal 8051 or the host is responsible for processing the received request message.

0—Host Responds. The internal 8051 will notify the host when the message is received. The host then extracts the EOC message database. After the host performs any necessary tasks, the host fills in the response data and triggers the response message. The host could also use the current response database and trigger the response message.

1—Internal 8051 Responds. The internal 8051 performs any necessary tasks (if possible) and sends out the response message using the current response database. The host can overwrite the response database.

11.10.5.2 Auto-Trigger a Request Message

This option determines if the internal 8051 will send out the request message on a regular internal. This probably only applies to status messages that need to be updated once-a-second.

0—Don’t send out command

1—Internal 8051 will send this command out once-a-second

11.10.5.3 Notify Host That a Message Was Received

This option determines if the internal 8051 will notify the host when a message is received.

0—Don’t notify
1—Notify

11.10.5.4 Notify Host When Error Is Detected

This option determines if the internal 8051 will notify the host when an error is detected. This only applies to a transmitted request message since the request messages activate the time-out timer to determine if a valid response message is received (acknowledged). The response messages do not include an acknowledge.

Since the EOC protocol specifies invalid frames are dropped (discarded), there is no way to associate the frame error with a specific message.

0—Don’t notify
1—Notify

11.10.6 EOC Transmit Queue

The EOC TxQueue provides a queuing mechanism for 10 messages. The TxQueue is necessary since the internal 8051 (for both request and response messages) and host processor can initiate a message at any time. The TxQueue structure provides the following information.

```
typedef strucut
{
        BP_U8BIT dest_addr;
        BP_U8BIT msg_id;
        BP_U8BIT msg_control;
        BP_U8BIT status;
        BP_U8BIT start_time;
} EOC_TX_QUEUE_t;
```

dest_addr	Destination address, see Section 11.9.5.
msg_id	Request or Response message ID, see Section 11.9.6.
msg_control	Message control, see Section 11.10.5.
status	Message status, see Table 11-6.
start_time	Only used by request messages. Sets the time the message began transmission across the EOC channel. Does not include time in the TxQueue. See Section 11.10.4.

Table 11-6. EOC TxQueue Status Bits

Status Bits	Description	Bit(s) Definition
2–0	Message state	0—IDLE. 1—Message successfully complete. 2—Message error, refer to bits 4 and 5. 3—Message in progress. Message is being transmitted or awaiting for response. 4—Message in TxQueue.
5–4	Error condition (only if Message State = 2)	0—Timed-out. 1—Undefined. 2—Undefined. 3—Undefined.
7–6	Reserved	—

11.10.7 EOC Receive Queue

TBD

11.10.8 EOC Proprietary Messages

The ZipWire2 solution uses four messages (2 request and 2 response) to implement the API over EOC and the User-Defined Messages features.

11.10.8.1 User-Defined Message Request – Message ID 112

The User-Defined Message Request message is assigned Message ID 112. This allows terminal units to send proprietary message to the far-end using the EOC channel.

Table 11-7. User-Defined Message Request Information Field

Octet #	Contents	Data Type	Reference
1	112	Message ID	—
2	Length of Message (L)	Unsigned char	—
3	Message data byte # 1	Unsigned char	—
4	Message data byte # 2	Unsigned char	—
3 + L	Message data byte # L	Unsigned char	—

11.10.8.2 User-Defined Message Response—Message ID 240

The User-Defined Message Response message is assigned Message ID 240. This message will be sent in response to the User-Defined Message Request message. This message just provides an acknowledge.

NOTE: The second byte is set to 0 to ensure future compatibility.

Table 11-8. User-Defined Message Response Information Field

Octet #	Contents	Data Type	Reference
1	240	Message ID	—
2	0	Unsigned char	—

11.10.8.3 API Over EOC Request—Message ID 113

The API Over EOC Request message is assigned Message ID 113. This allows the terminal units to issue API commands to the far-end using the EOC channel.

NOTE: The Length byte must not exceed 64 bytes (value of 63).

Table 11-9. API Over EOC Request Information Field

Octet #	Contents	Data Type	Reference
1	113	Message ID	—
2	API destination	Unsigned char	—
3	API opcode	Unsigned char	—
4	Reserved	Unsigned char	—
5	API Length (L), a 0 is 1 byte	Unsigned char	—
6	Reserved	Unsigned char	—
7	API data byte # 1	Unsigned char	—
8	API data byte # 2	Unsigned char	—
9 + L	API data byte # L	Unsigned char	—

11.10.8.4 API Over EOC Response—Message ID 241

The API Over EOC Response message is assigned Message ID 241. This message will be sent in response to the API Over EOC Request message. This message provides the API status acknowledge (both control and status commands) and any return data (status commands only).

NOTE: For a control command, the API length is set to 0 to ensure future compatibility.

Table 11-10. API Over EOC Response Information Field

Octet #	Contents	Data Type	Reference
1	241	Message ID	—
2	API destination	Unsigned char	—
3	API opcode	Unsigned char	—
4	API acknowledge status	Unsigned char	—
5	API Length (L), a 0 is 1 byte	Unsigned char	—
6	Reserved	Unsigned char	—
7	API result byte # 1	Unsigned char	—
8	API result byte # 2	Unsigned char	—
9 + L	API result byte # L	Unsigned char	—

11.10.8.5 Redefining Proprietary Messages

For the other proprietary messages, the host can define the message format. This allows the host to conform to other solutions proprietary messages.

In addition, the API over EOC and User-Defined messages can be redefined.

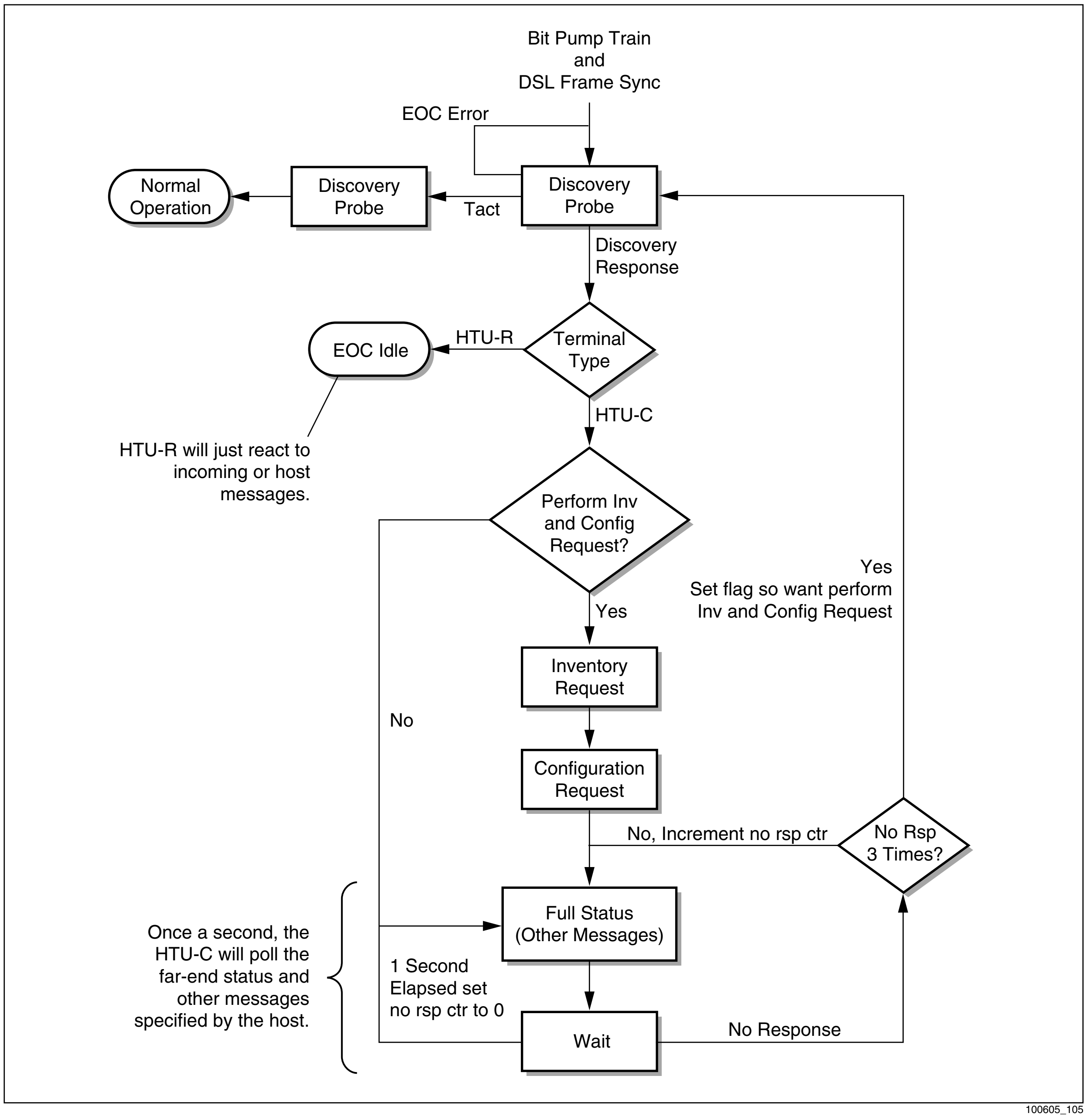
NOTE: The current software does not support this feature.

11.10.9 EOC Application State Machine

Figure 11-9 illustrates the EOC Application State machine. The Discovery Probe message is used as a final qualifier to determine when the system has successfully started up. The Discovery Probe message is sent once a second until either the Discovery Response message is received or the activation time-out (Tact) expires. If the activation time-out expires, the system goes to the normal operation state but the EOC channel is not functional. This will allow the ZipWire2 to be compatible with systems that do not support the EOC.

After the Discovery Probe, this state machine then sends the Inventory Request and Configuration Request commands to build-up the initial databases. Then once a second, the state machine will query the far-end's Full Status to update the databases.

Figure 11-9. EOC Application State Machine


Bit Pump Train
and
DSL Frame Sync
EOC Error
Normal Operation
Discovery Probe
Tact
Discovery Probe
Discovery Response
EOC Idle
HTU-R
Terminal Type
HTU-R will just react to incoming or host messages.
HTU-C
Perform Inv and Config Request?
Yes
Set flag so want perform Inv and Config Request
Yes
Inventory Request
No
Configuration Request
No, Increment no rsp ctr
No Rsp 3 Times?
Once a second, the HTU-C will poll the far-end status and other messages specified by the host.
Full Status (Other Messages)
1 Second Elapsed set no rsp ctr to 0
Wait
No Response
100605_105

11.10.10 EOC API Commands

Table 11-11 lists a summary of the EOC API commands. Refer to Chapter 17.0 for complete details.

Table 11-11. EOC API Commands Summary

Command	Control Status	Opcode	Description
EOC Send Command	S	0xB0	Initiate a request or response message. The index into the TxQueue is returned so the host can query for status.
EOC Get Message Status	S	0xB1	Get the status for the specified message.
EOC Get Database Data	S	0xB2	Extract the database information.
EOC Set Database Data	C	0x60	Fill in the database information.
EOC Set Message Control	C	0x61	Set the message control to determine how each message is processed.
EOC Read Receive Queue	S	0xB3	Read the RxQueue to determine which messages were received.
EOC Set Proprietary Length	C	0c62	Set the length of the proprietary message.

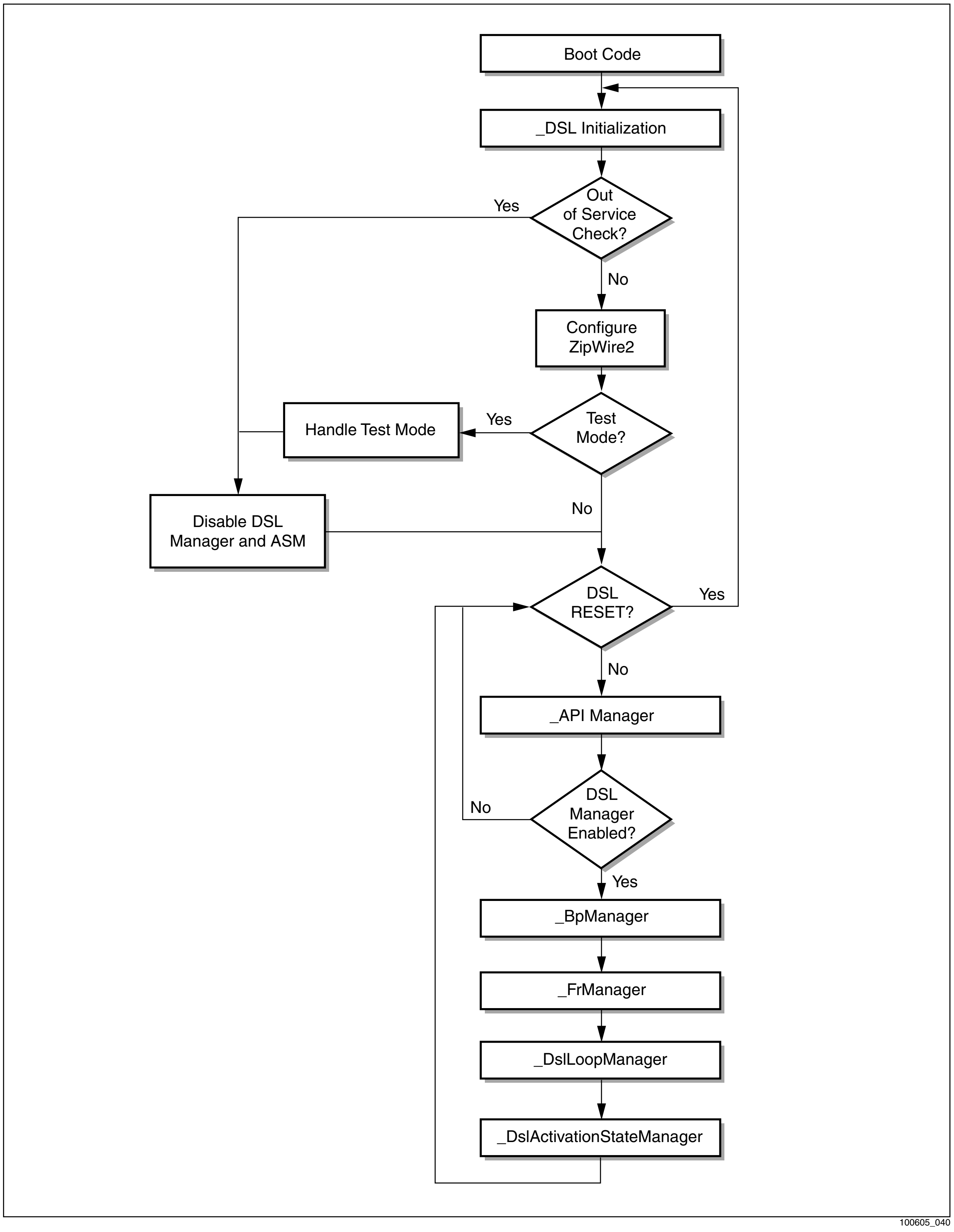
11.11 T1/E1 Framer and LIU Support

The code supports a T1/E1 Framer and LIU (Line Interface Unit). The framer and LIU code contain the minimum drivers necessary to configure the framer/LIU for a transparent mode. In addition, a minimal set of API commands is supported to configure the framer/LIU from a host processor.

12.0 Embedded 8051 Code

This section provides an overview of the ZipWire2 code. Figure 12-1 shows a detailed block diagram of the main program flow.

Figure 12-1. Main Program Flow



12.1 Boot Code State

This state downloads the external Flash into internal Program Memory (PRAM). If multiple devices are connected, the Flash contents are serially copied from the master device to the slave device(s).

Refer to Section 4.6 for details.

12.2 DSL Initialization State

In the initialization state, the software initializes the devices to a default configuration. After initialization, the activation state is changed to the Out-of-Service Check state.

12.3 Out-Of-Service Check

The Out-of-Service state queries the STARTUP pins (see Section 4.4) to determine whether the program should default into an IDLE (Out-of-Service) state. If the OOS is enabled, the program bypasses the ZipWire2 configuration states and disables the DSL loop manager and Activation State Manager. The PCM Rx clock is set to equal the PCM Tx clock to prevent an invalid clock driving the PCM bus. The host processor would then be responsible to enable and configure the system through API commands.

12.4 Configure ZipWire2 State

The Configure ZipWire2 state configures the ZipWire2 system based on the STARTUP, BOOTOP, and DIP Switches.

12.5 Handle Test Mode States

The Handle Test Mode state queries the STARTUP pins (see Section 4.4 and Table 9-4) to determine whether any test modes should be executed. If a test mode is selected, the program executes the specified test mode, then disables the DSL Loop Manager and Activation State Manager. The host processor could then issue an API command to enable the DSL Loop Manager and Activation State Manager.

12.6 DSL Reset Check

When the program is continuously executing the DSL loop manager and Activation State Manager, the host processor can issue the _SYSTEM_RESET API command, as described in Section 17.3.6, to perform a software reset. When this API is issued, the activation state is changed to the DSL Initialization state.

12.7 API Manager

The API manager is responsible for processing API commands from the Host Port/RS232 interface, as well as the Group Talk Serial Link.

The external host processor only communicates with the Group Master. The Group Master will forward any slave API commands through the Group Talk Serial Link Protocol. In addition to the host processor requests, the Group Master will periodically poll the slaves for status information. These status results are stored in the Group Master Host Port RAM so the host processor can efficiently query the slave status without the latency of the API and Group Talk protocols.

The following assumptions are defined by the API protocol (refer to Chapter 15.0):

- The host processor will not send any new message until the previous message is acknowledged.
- The Group Master will not send any new message to a slave until the previous message is acknowledged.
- The Group Master can process and acknowledge to the host processor a command targeted for the Group Master while the Group Master is waiting for the slave ACK (waiting for slave to process a local request initiated by the Group Master).
- Every 100 ms, the Group Master will request status information from a slave assuming that no incoming API commands are received. The Group Master uses a simple round robin state machine to query multiple slave devices. This should ensure that the Group Master can update the Host Port RAM Status registers approximately once each second per device.

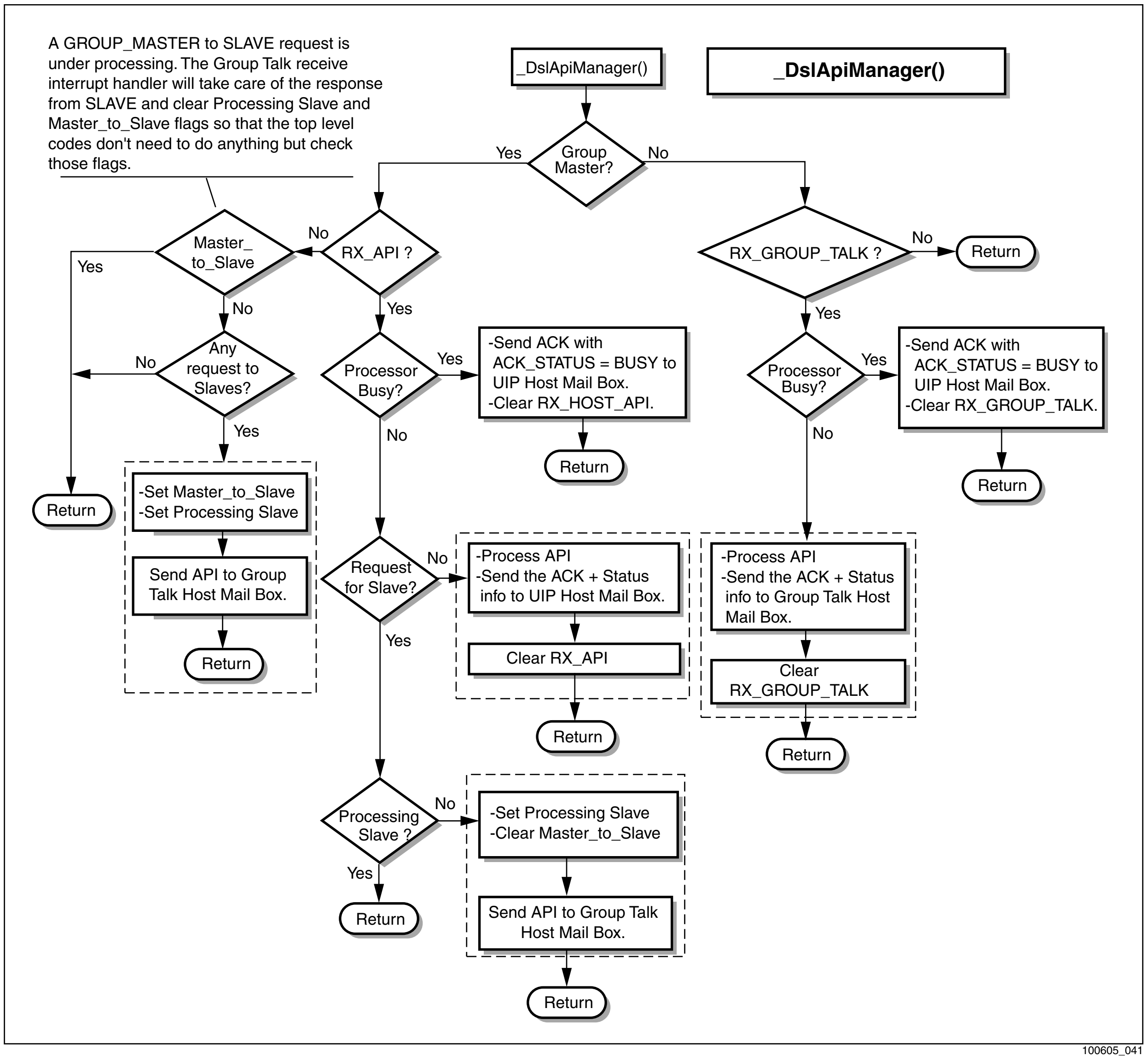
- If the Group Master receives an API command targeted for a slave while the Group Master is processing a local slave request, the Host API command will only get processed after the local slave request is completed. Further round-robin slave request commands will be held off until the host processor command is completed.

Table 12-1 lists API Manager flags and their descriptions. Figure 12-2 illustrates the details of the API Manager Flow.

Table 12-1. API Manager Flag Description

Flag	Description
Group Master Flag	This flag determines whether the devices is a Group Master or Group Slave.
RX API Message Flag	This flag is set whenever an API Command is received from the Host Port/RS232 Interface (Group Master) or from the Group Talk Interface (Slave). This flag is cleared when the API command is processed and the acknowledge is sent back to the host processor (Group Master) or to the Group Master (Slave).
Request For Slave Flag	This flag is set when the Group Master needs to send an API message to a slave. This flag can either be set when the host processor sends an API message where the slave is the destination or when the Group Master is not processing any other API commands. The Group Master uses a simple round-robin state machine to query multiple slave devices. This flag is cleared when the Group Master sends the API command to the slave.
Processing Slave Flag	This flag is set when the Group Master is currently processing a Group Slave API Command (Group Master is waiting for Rx Slave ACK). This flag is cleared when the Group Master receives the Slave Acknowledge and processes the results (by either forwarding the results to the host processor or processing the results locally).
Master To Slave Flag	This flag is set when the Group Master initiated a message for the Group Slave. This flag is low when the host processor requests a message for the Group Slave.
Rx Slave ACK	This flag is set when the Group Master receives the slave acknowledge from the Group Talk port. This flag is cleared when the Group Master processes the slave results.

Figure 12-2. API Manager Flow


A GROUP_MASTER to SLAVE request is under processing. The Group Talk receive interrupt handler will take care of the response from SLAVE and clear Processing Slave and Master_to_Slave flags so that the top level codes don't need to do anything but check those flags.
_DslApiManager()
_DslApiManager()
Group Master?
Yes
No
RX_API ?
No
Master_ to_Slave
Yes
No
Any request to Slaves?
No
Yes
-Set Master_to_Slave
-Set Processing Slave
Send API to Group Talk Host Mail Box.
Return
Return
Yes
Processor Busy?
Yes
-Send ACK with ACK_STATUS = BUSY to UIP Host Mail Box.
-Clear RX_HOST_API.
Return
No
Request for Slave?
No
-Process API
-Send the ACK + Status info to UIP Host Mail Box.
Clear RX_API
Return
Yes
Processing Slave ?
No
-Set Processing Slave
-Clear Master_to_Slave
Send API to Group Talk Host Mail Box.
Return
Yes
Return
RX_GROUP_TALK ?
No
Return
Yes
Processor Busy?
Yes
-Send ACK with ACK_STATUS = BUSY to UIP Host Mail Box.
-Clear RX_GROUP_TALK.
Return
No
-Process API
-Send the ACK + Status info to Group Talk Host Mail Box.
Clear RX_GROUP_TALK
Return
100605_041

12.8 Bit Pump Manager

The Bit Pump Manager is responsible for maintaining the bit pump portion of the Activation State Manager. This includes the startup training process, as well as adapting to temperature and environment changes during normal operation.

12.9 DSL Framer Manager

The DSL Framer Manager is responsible for maintaining the DSL Framer portion of the activation state manager.

- Framing
- Overhead bits
- Sync word
- Indicator bits
- HDLC/EOC
- Performance monitoring

12.10 DSL Loop Manager

The DSL Loop Manager is responsible for loop reversal and switching the master loop.

- The master loop is the first channel to reach normal operation.
- When subsequent channels reach normal operation, the pair ID/sync word is used to determine loop reversal.
- When the master loop goes down, the master loop is set to the next available channel.

NOTE: The DSL Loop Manager is only supported by the HDSL1 standard 2T1, 2E1, and 3E1 configurations.

12.11 HDSL2 Activation State Manager

Figure 12-3 illustrates the state diagram for the HTU-C activation. Figure 12-4 illustrates the state diagram for the HTU-R activation.

Figure 12-3. HDSL2 HTU-C Activation State Diagram

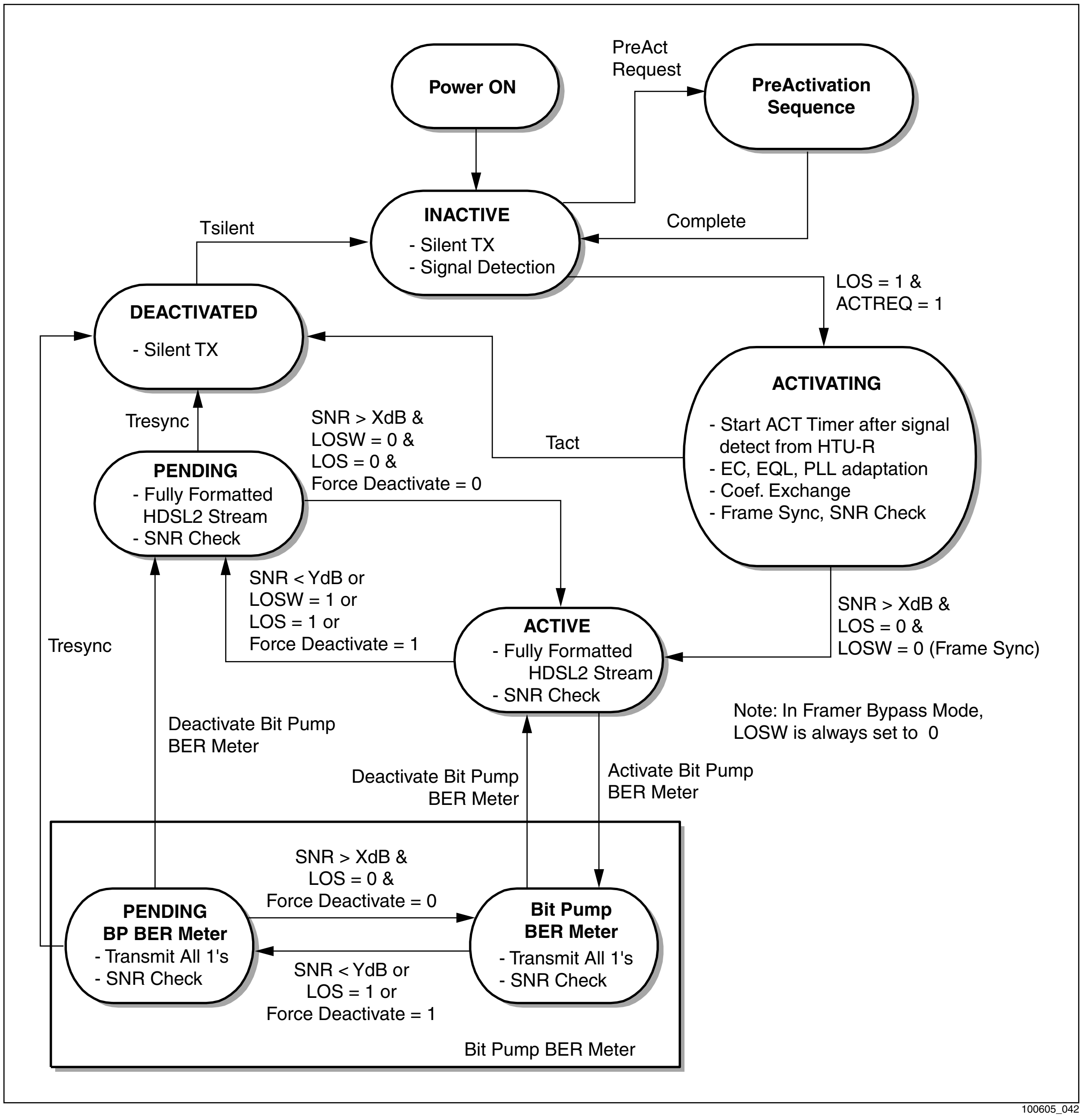
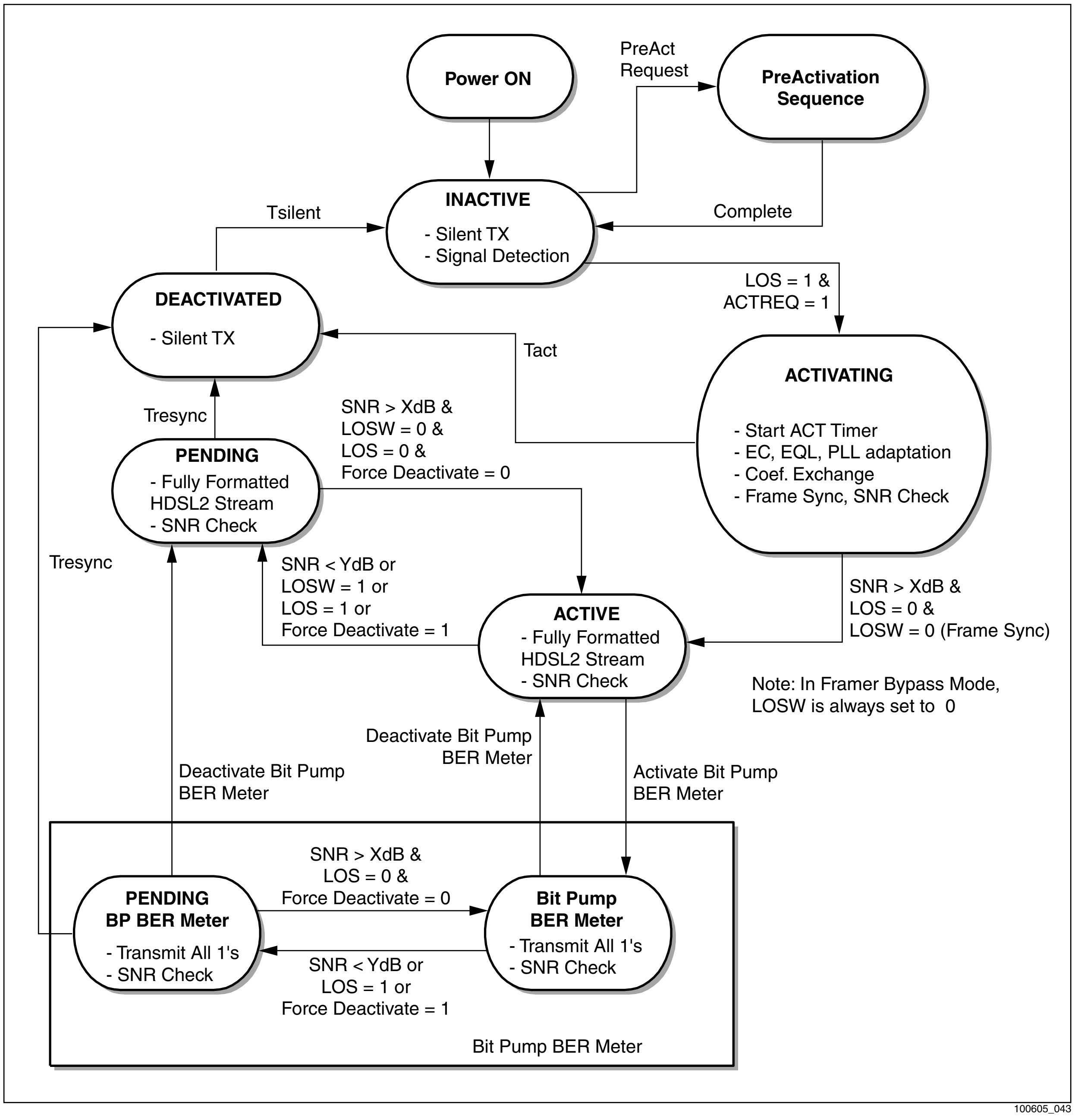

Power ON
PreAct Request
PreActivation Sequence
INACTIVE
- Silent TX
- Signal Detection
Complete
Tsilent
LOS = 1 & ACTREQ = 1
DEACTIVATED
- Silent TX
ACTIVATING
- Start ACT Timer after signal detect from HTU-R
- EC, EQL, PLL adaptation
- Coef. Exchange
- Frame Sync, SNR Check
Tact
Tresync
SNR > XdB & LOSW = 0 & LOS = 0 & Force Deactivate = 0
PENDING
- Fully Formatted HDSL2 Stream
- SNR Check
SNR < YdB or LOSW = 1 or LOS = 1 or Force Deactivate = 1
Tresync
ACTIVE
- Fully Formatted HDSL2 Stream
- SNR Check
SNR > XdB & LOS = 0 & LOSW = 0 (Frame Sync)
Note: In Framer Bypass Mode, LOSW is always set to 0
Deactivate Bit Pump BER Meter
Deactivate Bit Pump BER Meter
Activate Bit Pump BER Meter
SNR > XdB & LOS = 0 & Force Deactivate = 0
PENDING BP BER Meter
- Transmit All 1's
- SNR Check
Bit Pump BER Meter
- Transmit All 1's
- SNR Check
SNR < YdB or LOS = 1 or Force Deactivate = 1
Bit Pump BER Meter
100605_042

Figure 12-4. HDSL2 HTU-R Activation State Diagram


Power ON
PreAct Request
PreActivation Sequence
INACTIVE
- Silent TX
- Signal Detection
Tsilent
Complete
LOS = 1 & ACTREQ = 1
DEACTIVATED
- Silent TX
Tact
ACTIVATING
- Start ACT Timer
- EC, EQL, PLL adaptation
- Coef. Exchange
- Frame Sync, SNR Check
Tresync
SNR > XdB & LOSW = 0 & LOS = 0 & Force Deactivate = 0
PENDING
- Fully Formatted HDSL2 Stream
- SNR Check
Tresync
SNR < YdB or LOSW = 1 or LOS = 1 or Force Deactivate = 1
SNR > XdB & LOS = 0 & LOSW = 0 (Frame Sync)
ACTIVE
- Fully Formatted HDSL2 Stream
- SNR Check
Note: In Framer Bypass Mode, LOSW is always set to 0
Deactivate Bit Pump BER Meter
Deactivate Bit Pump BER Meter
Activate Bit Pump BER Meter
SNR > XdB & LOS = 0 & Force Deactivate = 0
PENDING BP BER Meter
- Transmit All 1's
- SNR Check
Bit Pump BER Meter
- Transmit All 1's
- SNR Check
SNR < YdB or LOS = 1 or Force Deactivate = 1
Bit Pump BER Meter
100605_043

12.12 HDSL1 Activation State Manager

This section describes how the HDSL1 Activation State Manager is implemented using the bit pump and DSL Framer code. The Activation State Manager is based on the *ETSI ETR-152* and *ANSI T1E1.4 HDSL* specifications.

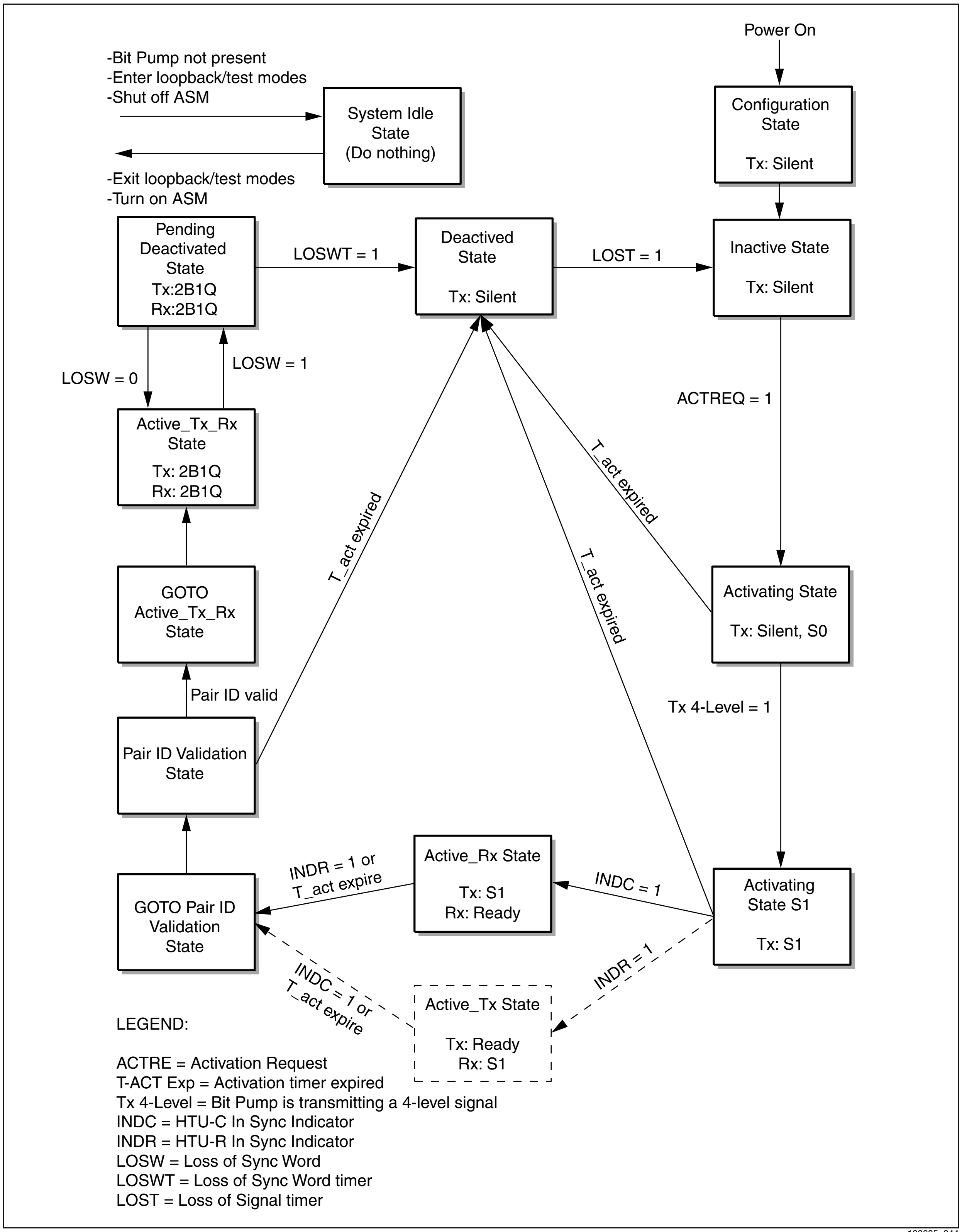
Figure 12-5 and Figure 12-6 illustrates the state diagrams for activation of HTU-C and HTU-R, respectively. Upon an activation request, the HTU-C side transmits a two-level signal to the far end. The HTU-R side, upon an activation request, waits for the HTU-C signal. Once the HTU-C two-level signal is detected, the HTU-R performs frequency lock, line characterization, and echo cancellation coefficient calculation. Upon completion, HTU-R will transmit a two-level signal back to HTU-C and wait for a four-level signal. The HTU-C will then perform characterization based on the HTU-R two-level signal. When HTU-C completes its characterization, it will send a four-level 2B1Q signal. At this stage, normal operation is reached with transmission of 2B1Q signal across the link.

12.12.1 HTU-C Activation

Figure 12-5 illustrates the state diagram for activation at HTU-C side.

NOTE: Several states are not explicitly stated in the ETR-152 HDSL standard. These extra states were added in order to ease the implementation of the HDSL standard for the bit pump and DSL Framer devices.

Figure 12-5. HDSL1 Activation State Machine at HTU-C



LEGEND:

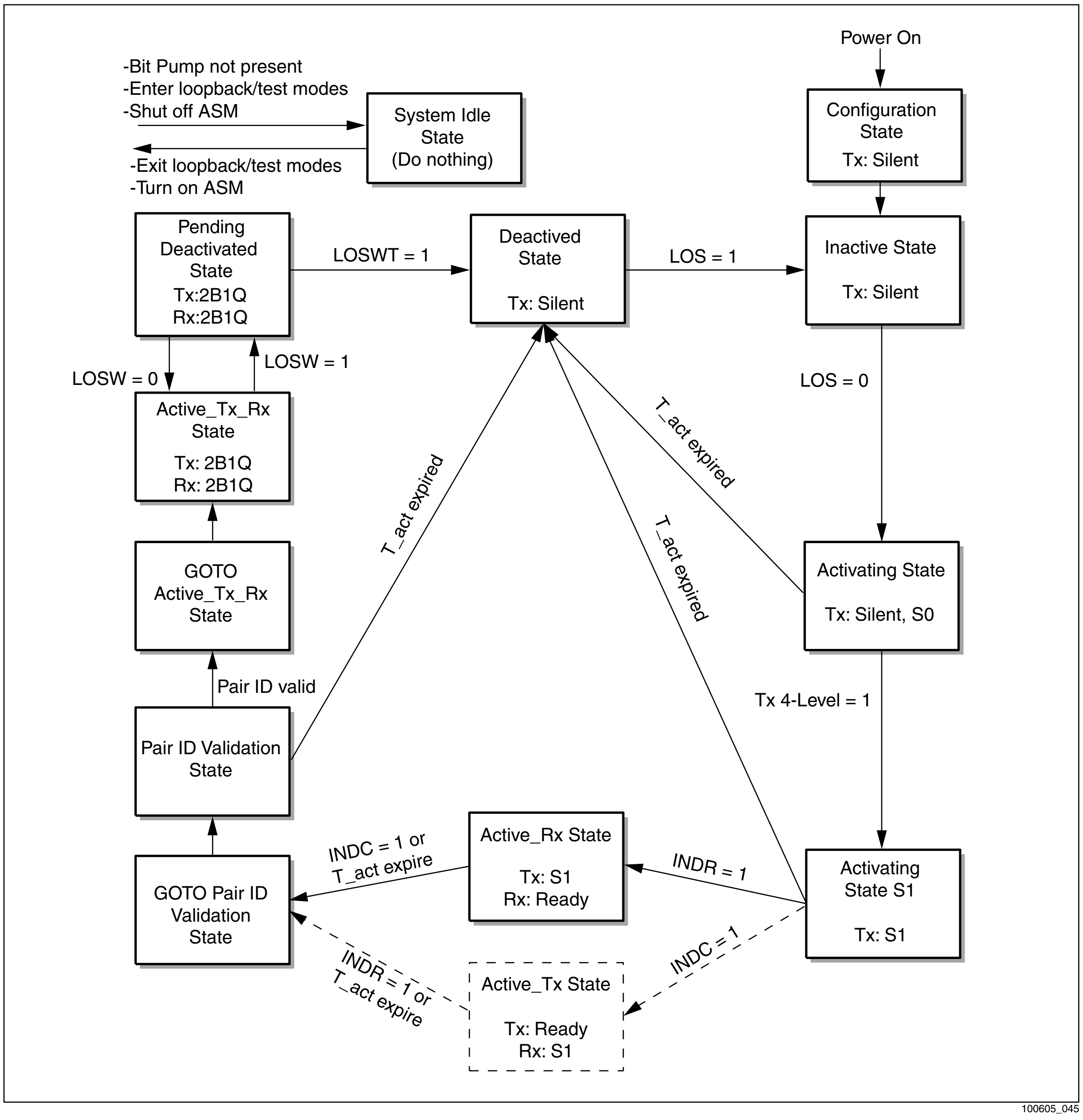
ACTRE = Activation Request
T-ACT Exp = Activation timer expired
Tx 4-Level = Bit Pump is transmitting a 4-level signal
INDC = HTU-C In Sync Indicator
INDR = HTU-R In Sync Indicator
LOSW = Loss of Sync Word
LOSWT = Loss of Sync Word timer
LOST = Loss of Signal timer

100605_044

12.12.2 HTU-R Activation

Figure 12-6 illustrates the state diagram for activation at the HTU-C side.

Figure 12-6. HDSL1 Activation State Machine at HTU-R


Power On
-Bit Pump not present
-Enter loopback/test modes
-Shut off ASM
System Idle State (Do nothing)
-Exit loopback/test modes
-Turn on ASM
Configuration State Tx: Silent
Pending Deactivated State Tx:2B1Q Rx:2B1Q
LOSWT = 1
Deactived State Tx: Silent
LOS = 1
Inactive State Tx: Silent
LOSW = 0
LOSW = 1
Active_Tx_Rx State Tx: 2B1Q Rx: 2B1Q
LOS = 0
T_act expired
T_act expired
T_act expired
GOTO Active_Tx_Rx State
Activating State Tx: Silent, S0
Pair ID valid
Tx 4-Level = 1
Pair ID Validation State
Active_Rx State Tx: S1 Rx: Ready
INDC = 1 or T_act expire
INDR = 1
Activating State S1 Tx: S1
GOTO Pair ID Validation State
INDR = 1 or T_act expire
Active_Tx State Tx: Ready Rx: S1
INDC = 1
100605_045

13.0 HDSL2 Standards Compliance

This section highlights how the ZipWire2 device complies with certain aspects of the HDSL2 standards:

- Pulse templates
- PSD
- Transmit power
- Tomlinson coefficients—exchange of DFE coefficients into Tx Precoder
- Encoder coefficients
- Scrambler
- CRC
- Activation sequence
- EOC
- Frame structure
- Activation sequence/timeline

13.1 Bit-Level Mapping

The bit-level mapping converts the digital data into the appropriate voltage level. Bit 0 is the Least-Significant Bit (LSB) and is sent first.

13.1.1 Four-Level 2B1Q Mapping (HDSL1)

Table 13-1. 2B1Q PAM4 Bit-to-Level Mapping

Bit 1 (Magnitude)	Bit 0 (Sign)	Level
0	0	–3
1	0	–1
1	1	+1
0	1	+3

13.1.2 Sixteen-Level Optis Mapping (HDSL2)

Table 13-2. Optis PAM16 Bit-to-Level Mapping

Bit 3	Bit 2	Bit 1	Bit 0	Level
0	0	0	0	–15
0	0	0	1	–13
0	0	1	0	–11
0	0	1	1	–9
0	1	0	0	–7
0	1	0	1	–5
0	1	1	0	–3
0	1	1	1	–1
1	1	0	0	+1
1	1	0	1	+3
1	1	1	0	+5
1	1	1	1	+7
1	0	0	0	+9
1	0	0	1	+11
1	0	1	0	+13
1	0	1	1	+15

14.0 DSL Frame Structure

The ZipWire2 chip set supports both the HDSL1 and HDSL2 Frame Structures. In addition, the DSL Framer can be bypassed.

14.1 Bypass DSL Frame Structure

In Framer Bypass Mode, the application would connect to the ZipWire2 Transceiver Clock and Data Interface. The digital clock and data are operating at the DSL data rate. Refer to Section 7.4 for more details.

14.2 HDSL2 Configurations

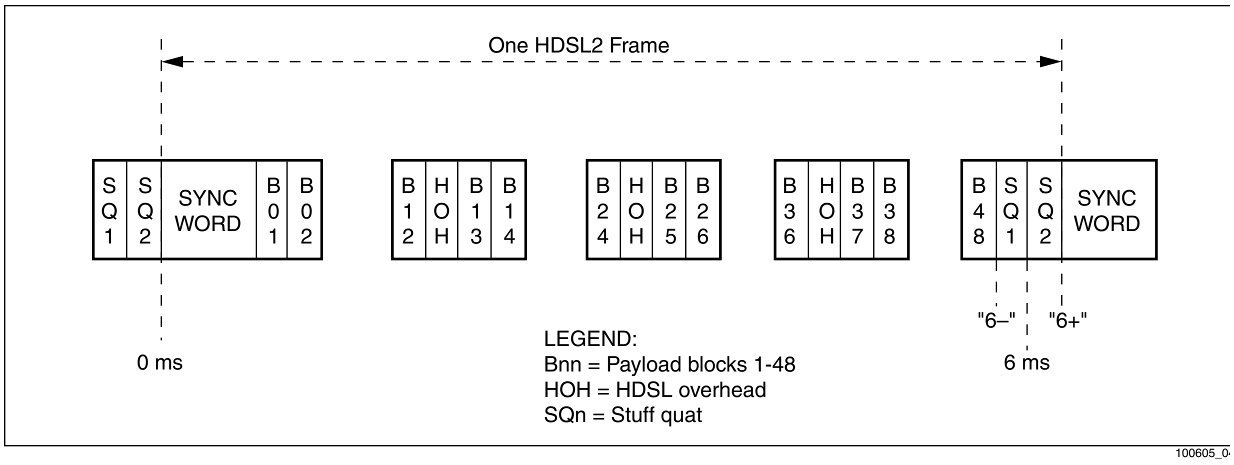
The system supports several kinds of configurations for T1/E1 transmission using HDSL2 technology. The basic structure of an HDSL2 frame is shown in Table 14-1, where each frame is nominally 6 ms in length and consists of 48 payload blocks. Each payload block contains a single F- or Z-bit, plus an application-specific number of payload bytes. Group of 12 payload blocks are concatenated and separated by an ordered set of HDSL2 overhead bits, where a 10-bit SYNC word pattern identifies the starting location of the HDSL2 frame. Fifty overhead bits are defined in one HDSL2 frame, but the last 4 STUFF bits are nominally present in alternate frames. Therefore, one frame contains an average of 48 overhead bits. Figure 14-1 shows the frame structure. The payload block structure for different applications is shown in the following subsections.

Table 14-1. HDSL2 Frame Structure and Overhead Bit Allocation

Frame Bit #	HOH Bit #	Symbol	Full Name	HOH Register Bit
1–10	1–10	SW1–SW10	SYNC Word	—
11–2326	—	B01–B12	Payload Blocks 1–12	—
2327–2328	11–12	CRC1–CRC2	Cyclic Redundancy Check	—
2329	13	SBID1	Stuff Bit ID Copy 1	SBID[0]
2330	14	LOSD	DS1 Loss of Signal Detect	TBD
2331–2338	1522	EOC01–EOC08	EOC Bit 1–8	TBD
2339–4654	—	B13–B24	Payload Blocks 13–24	—
4655–4656	23–24	CRC3–CRC4	Cyclic Redundancy Check	—
4657	25	UIB	Unspecified Indicator Bit	TBD
4658	26	SEGA	Segment Anomaly (same as FEBE)	TBD
4659–4666	27–34	EOC09–EOC16	EOC Bit 9–16	TBD
4667–6982	—	B25–B36	Payload Blocks 25–36	—
6983–6984	35–36	CRC5–CRC6	Cyclic Redundancy Check	—
6985	37	SBID2	Stuff Bit ID Copy 2	SBID[1]
6986	38	SEGD	Segment Detect	TBD
6987–6994	39–46	EOC17–EOC24	EOC Bit 17–24	TBD
6995–9310	—	B37–B48	Payload Blocks 37–48	—
9311	47	SB1	Stuff Bit 1	STUFF[0]
9312	48	SB2	Stuff Bit 2	STUFF[1]
9313	49	SB3	Stuff Bit 3	STUFF[2]
9314	50	SB4	Stuff Bit 4	STUFF[3]

NOTE: The Frame Bit # field is based on the 1T1 (1552 kbps) application. Other data rates will have a different number of frame bits.

Figure 14-1. HDSL2 Frame Structure



14.2.1 HDSL2 _1T1

HDSL2_1T1 runs the standard 1-loop T1 mapping at 1552 kbps with 1 loop carrying all the payloads from T1. Each payload block contains 1 F-bit followed by 24 payload bytes (Figure 14-2). The relation between the payload bytes and PCM time slot is shown in Table 14-2.

Figure 14-2. Payload Block Structure for 1T1 Application

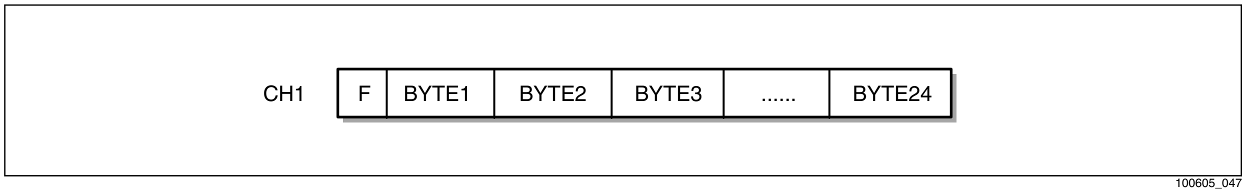


Table 14-2. 1T1 Framing

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

14.2.2 HDSL2 _1E1

HDSL2_1E1 runs the standard 1-loop E1 mapping at 2320 kbps with 1 loop carrying all the payloads from E1. Each payload block contains 1 Z-bit followed by 36 payload bytes (Figure 14-3). The relation between the payload bytes and PCM time slot is shown in Table 14-3.

Figure 14-3. Payload Block Structure for 1E1 Application

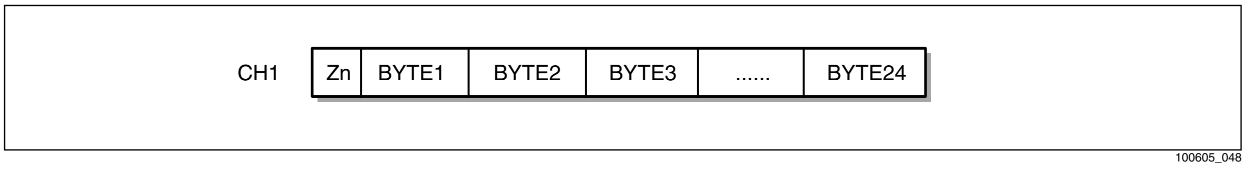


Table 14-3. 1E1 Framing

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																		

14.2.3 HDSL2—Single Pair

TBD

14.3 HDSL1 Configurations

The system supports several kinds of configurations for T1/E1 transmission using HDSL technology. The basic structure of an HDSL frame is shown in Table 14-4, in which 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 14-4 shows the frame structure. The payload block structure for different applications are shown in the following subsections.

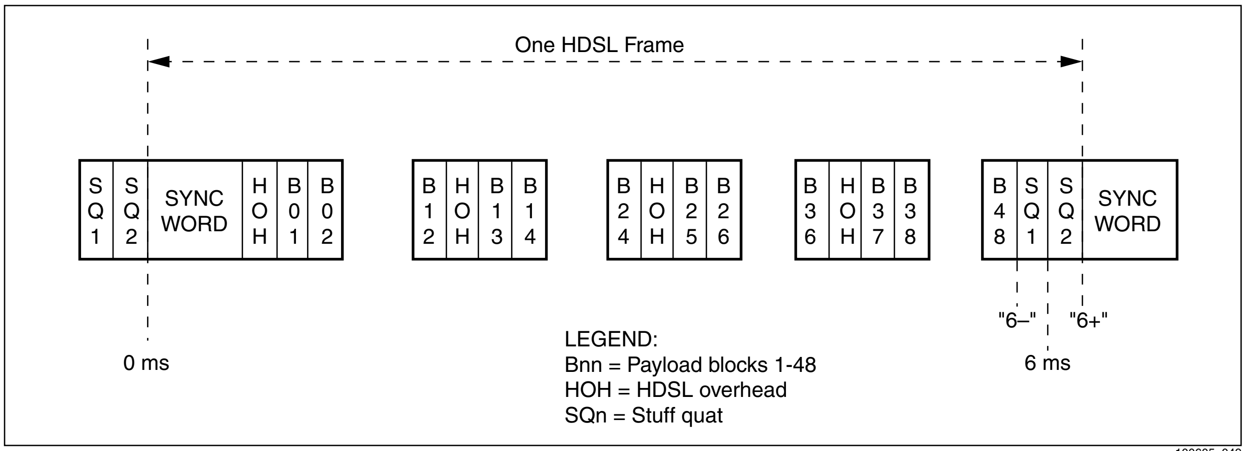
Table 14-4. HDSL1 Frame Structure and Overhead Bit Allocation (1 of 2)

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]
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]

Table 14-4. HDSL1 Frame Structure and Overhead Bit Allocation (2 of 2)

HOH Bit	Symbol	Bit Name	HOH Register Bit
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 14-4. HDSL1 Frame Structure



14.3.1 HDSL1 _2T1

This runs the standard 2-loop T1 mapping at 784 kbps, with each loop carrying one-half the payloads from T1. Each payload block contains 1 F-bit, followed by 12 payload bytes (Figure 14-5). The relation between the payload bytes and PCM time slot is shown in Table 14-5.

Figure 14-5. Payload Block Structure for 2T1 Application

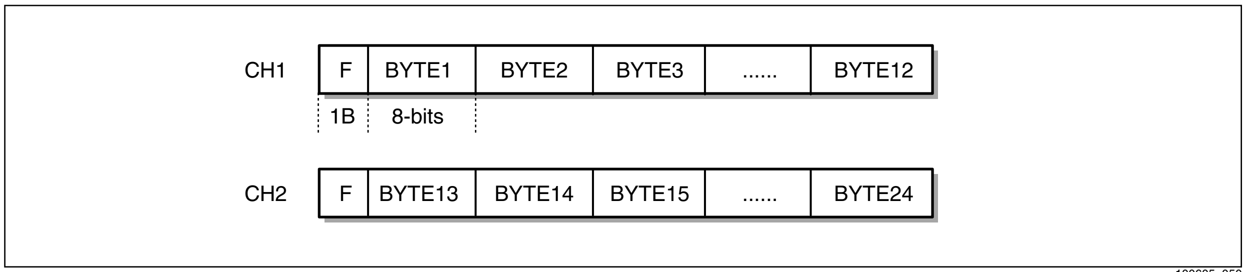


Table 14-5. 2T1 Framing

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

14.3.2 HDSL1_2E1

This runs the standard 2-loop E1 mapping at 1168 kbps, with each loop carrying one-half the payloads from E1. Each payload block contains 1 Z-bit, followed by 18 payload bytes (Figure 14-6). The relation between the payload bytes and PCM time slot is shown in Table 14-6.

Figure 14-6. Payload Block Structure for 2E1 Application

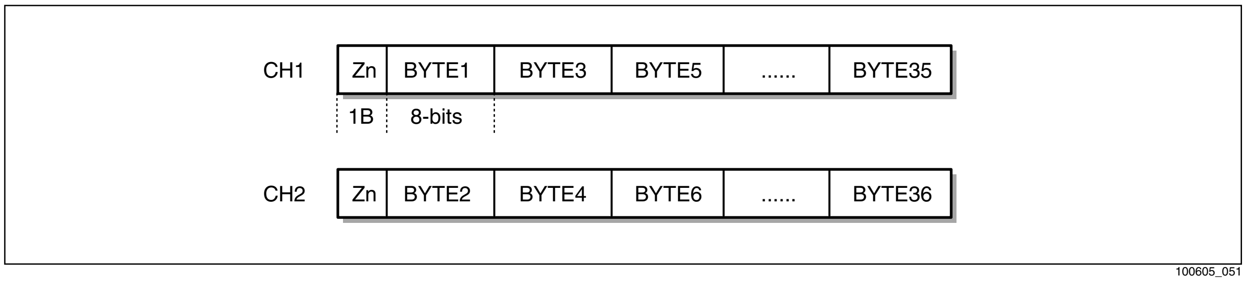


Table 14-6. 2E1 Framing

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

14.3.3 HDSL1_3E1

This runs the standard 3-loop E1 mapping at 784 kbps, with each loop carrying one-third the payloads from E1. Each payload block contains 1 Z-bit, followed by 12 payload bytes (Figure 14-7). The relation between the payload bytes and PCM time slot is shown in Table 14-7.

Figure 14-7. Payload Block Structure for 3E1 Application

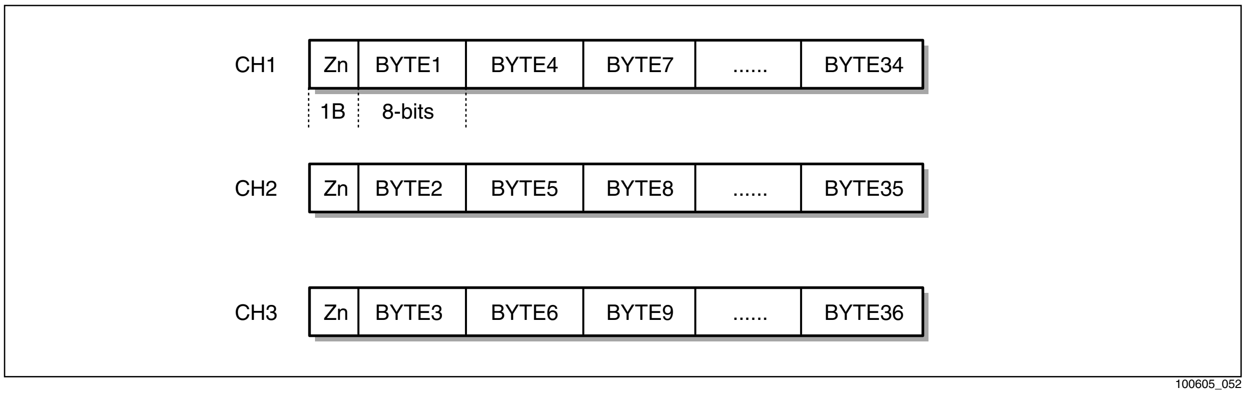


Table 14-7. 3E1 Framing

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	21	25	28	31

14.3.4 HDSL1_1T1

This runs the standard 1-loop T1 mapping at 1552 kbps, with 1 loop carrying all the payloads from T1. Each payload block contains 1 F-bit, followed by 24 payload bytes (Figure 14-8). The relation between the payload bytes and PCM time slot is shown in Table 14-8.

Figure 14-8. Payload Block Structure for 1T1 Application

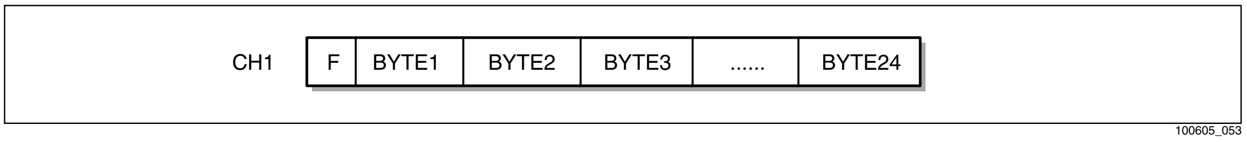


Table 14-8. 1T1 Framing

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

14.3.5 HDSL1_1E1

This runs the standard 1-loop E1 mapping at 2320 kbps, with 1 loop carrying all the payloads from E1. Each payload block contains 1 Z-bit, followed by 36 payload bytes (Figure 14-9). The relation between the payload bytes and PCM time slot is shown in Table 14-9.

Figure 14-9. Payload Block Structure for 1E1 Application

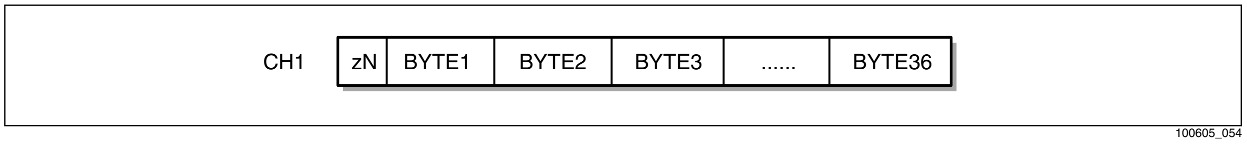


Table 14-9. 1E1 Framing

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																		

14.3.6 HDSL1_DSL_CUSTOM

To customize the DSL Framer code for applications other than standard 1T1, 2T1, 1E1, 2E1, and 3E1, the DSL Framer code needs to be modified.

15.0 API: Microprocessor Communicator Channel Protocol

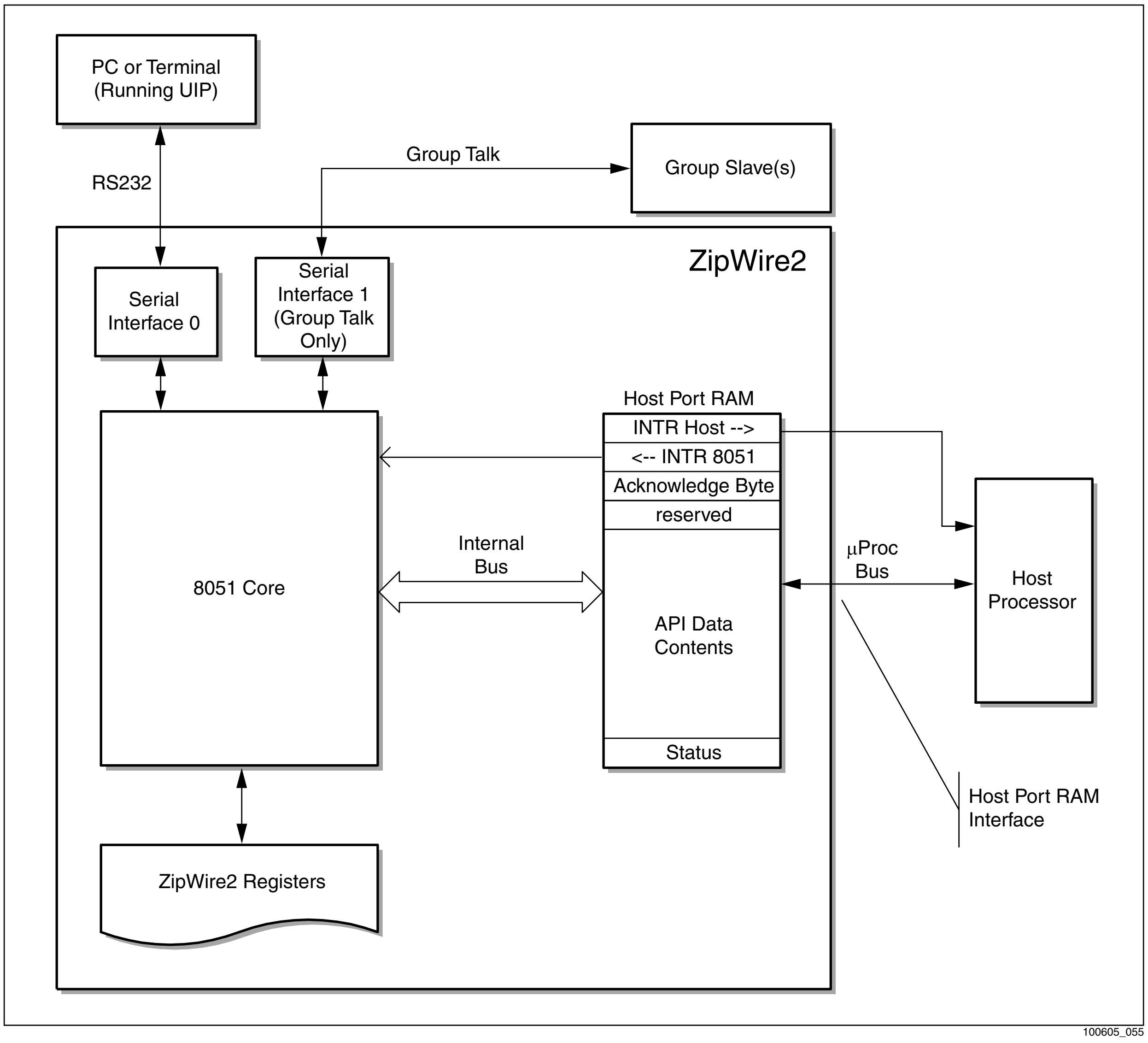
The API (Application Programmer Interface) commands are passed from an external processor to the ZipWire2 8051 Microprocessor via 3 possible interfaces (see Figure 15-1).

- Host port RAM interface
- RS232 serial interface
- Group Talk serial interface (Group Slave only)

In applications that have another external embedded microprocessor in the system, the external processor writes to the host port RAM. In applications that only use the internal ZipWire2 microprocessor, an external terminal (PC) could communicate to the system via the RS232 interface.

Only the host port RAM interface or RS232 serial interface protocol should be implemented in a system because the RS232 interface uses the host port RAM for local storage. In a multiple device configuration using the Group Talk protocol, the Group Master would be connected to the host processor (via either the host port or RS232 interface). The Group Slave would be connected to the Group Master via the Group Talk interface.

Figure 15-1. Communication Channel Protocols



15.1 API Message Time-Out

The 8051 processor is guaranteed to locally process and complete an API command within 1 ms. When using the serial RS232 or Group Talk interface, each byte takes ~100 µs when running at 115 k (87 µs to transfer the data plus 10 µs for the microprocessor to process). Therefore, the serial RS232 and Group Talk interfaces can each add up to ~7 ms to complete certain API commands such as performance history dumps.

15.2 API Master and Slave Implementation

The API Communication Protocols are Master and Slave implementations where the host processor is the master and the ZipWire2 8051 processor is the slave. This implies that only the host processor can initiate any messages through the communication channel. The 8051 processor cannot initiate any messages to the host processor but will only react to incoming messages. However, in Group Talk mode, the Group Master will initiate messages to the Group Slave to query for status information.

15.2.1 No Peer-to-Peer Protocol

A Peer-to-Peer protocol (which allows either the host or 8051 to initiate a message) is not implemented. One major advantage with the Peer-to-Peer protocol would be that the 8051 processor could generate an asynchronous interrupt if a fatal error was detected. However, it was assumed the host processor would have to poll for the ZipWire2 chip sets for line quality, CRC errors, and so on, at some regular interval, so a Fatal Error indicator is included in the status registers. More importantly, when using the multidevice drop Group Talk protocol, there are hardware limitations that prevent a slave from soliciting the master.

15.3 API Message Structure

The API message structure is defined to allow a consistency between the host port RAM interface, the RS232 serial interface, the Group Talk interface, and the EOC channel. The message consists of a fixed-length header packet and the data parameters, up to 75 bytes.

15.3.1 Incoming Message Structure

Table 15-1 lists the incoming message structure from the host processor to the 8051 processor. The message structure contains a variable-byte structure that consists of two sections (the header and the data) with the following fields:

1. Header: destination, opcode, a reserved byte, and length
2. Data: data parameters

This message structure is the same for control and status request commands.

Table 15-1. Incoming Messages from the Host Processor

Header Section				Data Section
Destination	Opcode	Reserved	Length	Data Parameters
1 Byte	1 Byte	1 Byte	1 Byte	(Length + 1) Bytes

15.3.2 Outgoing Message Structure

Table 15-2 lists the outgoing messages from the 8051 processor to the host processor. The message structure includes a status acknowledge message and the status results (when a status command is requested).

For control commands, only the acknowledge response section is sent from the 8051 processor to the host processor. For status request commands, both the acknowledge response and data sections are sent from the 8051 processor to the host processor.

Table 15-2. Outgoing Messages from the 8051 Processor

Acknowledge Response Section (Header)				Data Section
Destination	Opcode	ACK Status	Length	Data Parameters
1 Byte	1 Byte	1 Byte	1 Byte	(Length + 1) Bytes

Control Command Response				
Status Command Response				

15.3.3 Header Section–Destination Field

The destination field selects the device to which the command is targeted. Only the 4 lower bits are used to decode the destination. The upper 4 bits are reserved for the header byte in the RS232 serial interface protocol. When using the host port RAM interface, the upper 4 bits should be set to 0.

Table 15-3. Destination Field Specification

Destination	Device
0	_ZIP_WIRE0
1	_ZIP_WIRE1
2	_ZIP_WIRE2
3	_ZIP_WIRE3
4	_ZIP_WIRE4
5	_ZIP_WIRE5
6	_ZIP_WIRE6
7	_ZIP_WIRE7
8	_EVM
9	_T1_E1_FRAMER
10–14	Reserved
15	_ZIP_WIRE_BROADCAST

Destination values 8 and above are only intended when the 8051 is controlling other external devices (i.e., T1/E1 Framer). When a second embedded controller is available, the second processor should control the other external devices.

The broadcast destination value (15) is only used during the program download. This allows multiple slave devices to program simultaneously.

15.3.3.1 Header Section–Opcode Field

The opcode field selects the specific control command or status request to be executed. The most significant bit (bit 7) is used to determine if the opcode is a control or status request as shown in Table 15-4. See Chapter 17.0 for opcode values and descriptions.

Table 15-4. API Opcode Type

Bit # 7	Opcode Range (HEX)	Opcode Range (DEC)	API Type
0	0x00–0x7F	0–127	Control Command
1	0x80–0xFF	128–255	Status Request

15.3.4 Header Section—Reserved Byte or ACK Status

For incoming messages from the host processor, the third byte of the header is reserved for future expansion and to be compatible with the outgoing message field that contains the acknowledgement status (ACK Status) byte. For incoming messages, the reserved byte must be set to 0.

15.3.4.1 Acknowledge Status Byte

For each control and status request message received and processed from the host processor, the 8051 processor will generate an acknowledge status byte which should be used for error checking and to determine the host processor behavior. Table 15-5 lists the codes that are generated.

Table 15-5. Acknowledge Status Codes (1 of 2)

Status	Description	Value	#define (as defined in DSL_API.H)
Not Completed	Current API command has not been completed or no API command was sent to this device. This is used when multiple INTR_HOST interrupts are ORed together to the host processor.	0x00	_ACK_NOT_COMPLETE
API Successful	No Error, message was successfully completed. Any status data is valid.	0x01	_ACK_PASS
Busy	8051 Busy, unable to process command. The host should re-send command after some 500 ms.	0x02	_ACK_BUSY
Not Applicable	The API command is not applicable to the current H/W or S/W configuration.	0x03	_ACK_NOT_APPLICABLE
Invalid Dest	An invalid Destination was specified.	0x04	_ACK_INVALID_DEST
Invalid Opcode	An invalid Opcode was specified.	0x05	_ACK_INVALID_OPCODE
Invalid Length	An invalid Length was specified.	0x06	_ACK_INVALID_LENGTH
Invalid Data	An invalid data parameter(s) was detected.	0x07	_ACK_INVALID_DATA
Invalid Checksum	An invalid Checksum parameter(s) was specified.	0x08	_ACK_INVALID_CHKSUM
No Result	The 8051 was unable to complete the specified status request. No results are returned.	0x09	_ACK_NO_RESULT
Not Available	The ZipWire2 is currently out-of-service.	0x0A	_ACK_NOT_AVAILABLE
Invalid Slave Checksum	The Group Master detected an invalid checksum from the Group Slave. The data is not sent to the host processor.	0x0B	_ACK_CHECKSUM_WRONG
Slave Timed-Out	The Group Master timed-out during the Group Slave request from the host processor. Any status data will not be sent to the host processor.	0x0C	_ACK_TIME_OUT

Table 15-5. Acknowledge Status Codes (2 of 2)

Status	Description	Value	#define (as defined in DSL_API.H)
Boot ROM Wake-up	The 8051 Boot ROM has successfully initialized and is awaiting the host download. This code is only set when the PRAM download is selected from host port RAM or the Group Talk.	0x0D	_ACK_BOOT_WAKE_UP
Operational Code Wake-up	The 8051 has successfully reached the operational code.	0x0E	_ACK_OPER_WAKE_UP
Reserved	These codes are reserved for future expansion of the 8051 code.	0x0F–0x17	—
Customer Defined	These codes are reserved for use by the customer application.	0x18–0x1F	—

15.3.5 Header Section–Message Length Field

The message length field is a value from 0–255 (0x00–0xFF) to provide the number of bytes in the data parameters field (0–74). The number of bytes in the data field will always be at least 1, so a 0 length specifies that 1 byte will be in the data field; thus the length is equal to the number of bytes in the data parameter plus 1. The maximum length is then 75 bytes.

15.3.6 Data Section–Data Parameter Field

For control commands, the data parameter field provides additional data (or parameters) for the given API command. In commands where there is no need for additional data, 0s should be placed as the data byte to ensure future compatibility.

NOTE: For invalid parameters, the software will typically use the default value in addition to returning the Invalid Data Acknowledge Status byte, see Section 15.3.4.1.

For status request commands, the data parameter field provides the results for the given API command.

15.4 Host Port RAM Interface Protocol

The host port RAM is a dual port RAM so both processors can read and write to the RAM. Registers 0x00 (INTR_HOST) and 0x01 (INTR_8051) are special registers that behave uniquely. Table 15-6 lists the mapping for the host port RAM and provides a brief description of the register function.

Table 15-6. Host Port RAM Mapping (1 of 2)

Register	Label	Description
0x000	INTR_HOST	Writing to this register generates an interrupt to the host processor. Reading this register clears the interrupt. The 8051 should only write to this register while the host processor should only read this register.
0x001	INTR_8051	Writing to this register generates an interrupt to the 8051. Reading this register clears the interrupt. The host processor should only write to this register while the 8051 should only read this register.
0x002	Host Port Version	Host Port Version ID. The host software would this ID to determine the host port RAM address mapping and protocols.
Incoming API Message Structure		
0x003	API In Destination	Incoming API Destination Field.
0x004	API In Opcode	Incoming API Opcode Field.
0x005	API In Reserved	Reserved for future expansion, set to 0.
0x006	API In Length	Incoming API Data Length. Specifies the number of API Data bytes for incoming API commands. A 0 represents 1 byte; the maximum is 75 bytes (value of 74).
0x007	API In Reserved	Reserved for RS232 protocol. Value ignored when connected to host port RAM.
0x008–0x052	API In Data	Incoming API Data—up to 75 bytes.
0x053	Reserved	Reserved for RS232 protocol. Value ignored when connected to host port RAM.
Outgoing API Message Structure		
0x054	API Out Destination	Outgoing API Destination Field.
0x055	API Out Opcode	Outgoing API Opcode Field.
0x056	API Out Status Acknowledge	Acknowledge Byte. When the INTR_HOST is detected, the host processor queries this byte to determine what further action needs to take place.
0x057	API Out Length	Outgoing API Data Length. Specifies the number of API Data bytes for outgoing API commands. A 0 represents 1 byte; the maximum is 75 bytes (value of 74). For control commands and non-successful status acknowledges, this value will be set to 0.

Table 15-6. Host Port RAM Mapping (2 of 2)

Register	Label	Description
0x058	Reserved	Reserved for RS232 protocol. Value ignored when connected to host port RAM.
0x059–0x0A3	API Out Data	Outgoing API Data—up to 75 bytes.
0x0A4	Reserved	Reserved for RS232 protocol. Value ignored when connected to host port RAM.
Reserved for Group Talk and Internal Use		
0x0A5–0x3BF	Reserved	
Status Information		
0x3C0–0x3C7	Status_0	Status for Device #0[(1)]
0x3C8–0x3CF	Status_1	Status for Device #1[(1)]
0x3D0–0x3D7	Status_2	Status for Device #2[(1)]
0x3D8–0x3DF	Status_3	Status for Device #3[(1)]
0x3E0–0x3E7	Status_4	Status for Device #4[(1)]
0x3E8–0x3EF	Status_5	Status for Device #5[(1)]
0x3F0–0x3F7	Status_6	Status for Device #6[(1)]
0x3F8–0x3FF	Status_7	Status for Device #7[(1)]

NOTE(S):
(1) See Section 17.3.13.

15.4.1 INTR_HOST and INTR_8051 Registers

Whenever register 0x00 or 0x01 is written to, an interrupt is generated. Whenever register 0x00 or 0x01 is read, the interrupt is cleared. The hardware interrupts are active-low. The INTR_HOST (addr 0x00) and INTR_8051 (addr. 0x01) are used to signal to the other processor that data is available in the host port RAM.

15.4.2 Host Port Acknowledge Register

Register 0x058 (ACKNOWLEDGE) is used as the API acknowledge status byte. Whenever the host processor detects the INTR_HOST interrupt, the host must query the acknowledge byte to determine the course of action. The acknowledge status byte can be viewed as an Interrupt Service Register (ISR).

The all-0s (Not Completed) value is used in a multiple-group master environment where the INTR_HOST interrupt lines are ORed together to the host processor. When the INTR_HOST is detected, the host processor polls each device's acknowledge byte to determine where the interrupt was generated.

The host processor is responsible for clearing the acknowledge register before exiting its INTR_HOST interrupt handler. This ensures the acknowledge byte is 0 (Not Completed) in the event that another device generates a separate INTR_HOST request.

Table 15-7. Acknowledge Status Register (Interrupt Source Register)

Bit 7	Bit 6	Bit 5	Bit 4:0
Unsolicited Interrupt	Reserved	API Response	Acknowledge Status

Unsolicited Interrupt Read STATUS_8 to determine the source of the unsolicited interrupt.

API Response This bit sets when the 8051 processor responds to a host API command. In addition, this bit sets when the 8051 processor triggers a boot ROM or operational code wake-up interrupt. The Acknowledge Status byte provides the status code to determine the host program's flow.

NOTE: The API response and wake-up are mutually exclusive events and therefore can share the same bit-field.

Acknowledge Status The Acknowledge Status byte provides the status code of the current API command (see Table 15-5).

15.4.3 Host Port Status Registers

Registers 0x3C0–0x3FF contain the status for each of the devices within that particular group. This eliminates the inefficiency of the API message protocol when the host processor needs to query for common status during normal operation. Each device has up to 8 bytes of status.

NOTE: The host processor should never write to the Status_n registers since the 8051 processor uses this RAM as the storage location for the status information. Writing to these registers could corrupt the status information and cause unpredictable behavior. Reading the Status_n register will return the same result as the STATUS_n API command.

Table 15-8 lists the details of the host port RAM status mapping.

Table 15-8. Host Port RAM Status Mapping

Register Offset	Label	Description
0x00	Status Byte 0	See API Command STATUS_0
0x01	Status Byte 1	See API Command STATUS_1
0x02	Status Byte 2	See API Command STATUS_2
0x03	Status Byte 3	See API Command STATUS_3
0x04	Status Byte 4	See API Command STATUS_4
0x05	Status Byte 5	See API Command STATUS_5
0x06	Status Byte 6	See API Command STATUS_6
0x07	Status Byte 7	See API Command STATUS_7

15.5 Host Port RAM Interface Sequence of Events

The host processor writes to the host port RAM based on the desired API command, then writes 0xFE to the INTR_8051 to trigger to the 8051 that an API command is ready. The 8051 interrupt handler will set a flag to process the API command. The API command is then processed in the main thread when the 8051 processor has available time. After the 8051 processor processes the API command and writes the results into the host port RAM, the 8051 processor writes to the INTR_HOST register to acknowledge to the host that the API command is complete. The host processor can now read the acknowledge status register and any valid status information. The host processor then needs to write a 0 into the acknowledge status register and the INTR_HOST register. The host processor then reads from INTR_HOST to clear the interrupt. This completes the API command.

From the time the host processor writes to the INTR_8051 and until the INTR_HOST is detected, the host processor must not write to the host port RAM. Writing to the host port RAM could corrupt the current API command. In addition, the host processor should only read the host port RAM after the INTR_HOST is detected. Reading the host port RAM before the INTR_HOST is detected will not corrupt any data, but the contents of the data would be invalid. The host processor should write to the INTR_HOST to clear the interrupt only after the API status results are read and the acknowledge byte is written to 0.

The 8051 processor is guaranteed to only read/write to or from the host port RAM after the INTR_8051 is detected and until the INTR_HOST is generated.

Table 15-9 illustrates the host port RAM sequence of events. The <Processor>-Main indicates this task is accomplished in the main thread while the <Processor>-ISR indicates this task is accomplished in the interrupt service handler.

NOTE: Host implementation is application- and processor-specific and may differ depending on the host processor environment.

Table 15-9. Host Port RAM Message Protocol Events (1 of 2)

Processor	Action
Host-Main	Check if group is available (host semaphore)
Host-Main	Lock group (host semaphore)
Host-Main	Write API content registers
Host-Main	Write to INTR_8051 register to initiate message (addr 0x01) to 0xFE
8051	Detect INTR_8051
8051-ISR	Read API content registers
8051-ISR	Write INTR_8051 to 0x00
8051-ISR	Clear INTR_8051 interrupt by reading from the INTR_8051 register

Table 15-9. Host Port RAM Message Protocol Events (2 of 2)

Processor	Action
8051-Main	Perform API task
8051-Main	If status request, write results into host port RAM
8051-Main	Write INTR_HOST register to initiate acknowledge (addr 0x00) to 0xFE
8051-Main	Write acknowledge status byte
Host	Detect INTR_HOST
Host-ISR	Read acknowledge to determine which device generated interrupt
Host-ISR	Write the acknowledge status byte register to 0
Host-ISR	Write INTR_HOST to 0x00
Host-ISR	Clear INTR_HOST interrupt by reading from the INTR_HOST register
Host-Main	If status request, read API data length and data registers to processes results
Host-Main	Clear group (host semaphore)
Host-Main	Host is clear to send another message

NOTE: In applications configured as Figure 3-3, the communication group channel must be blocked when sending to any device in that group. The host processor cannot send another message to that group until the API command is completed (or until time-out).

In multitasking environments, the host semaphore is used to prevent the host from sending any additional commands until the current command is completed. If the communication channel is unavailable to send another command, the host processor could put the task to sleep and switch to process other tasks. When the INTR_HOST is detected and the data is processed, the task can be awakened and processed.

15.5.0.1 Host Processor Polling Method

In non-multitasking environments, simply poll and wait until INTR_HOST can be implemented. This is accomplished by polling the INTR_HOST register and looking for a 0xFE.

15.5.1 Multi Device System

In a multi-ZipWire2 device system (refer to Section 3.2), the INTR_HOST interrupts lines would be wire-ORed together. The host processor must be able to handle when multiple ZipWire2 devices send back an acknowledge at the same time.

The Not Completed acknowledge status code (see Table 15-5) is critical in multi-ZipWire2 device systems. When the host processor detects the INTR_HOST interrupt, the host processor polls the acknowledge status request to determine which device generated the interrupt.

15.6 RS232 Serial Interface Protocol

The RS232 serial interface protocol is an asynchronous serial channel that allows an external PC (or terminal) to communicate with the ZipWire2 chip set via an RS232 connector (see Section 4.5.3 for hardware details).

15.6.1 Host Processor to 8051 Processor Message Structure

All RS232 messages sent from the host processor to the 8051 processor are contained in a variable byte structure that must be at least 7-bytes long, as shown in Table 15-10. The message structure contains a variable byte structure that consists of two sections (the header and the data) with the following fields:

1. Header: destination, opcode, a reserved byte, length, and checksum
2. Data: data parameters and checksum.

The header and data sections contain a checksum to guarantee the message packet transfer. The header checksum is required in the Group Talk multidrop protocol. Each slave will monitor the serial link to determine if the destination field matches their device ID. For messages destined for another slave device, the other slave devices must now how many bytes (data parameters) to ignore before they begin looking for the header section. See Section 15.7 for the details regarding the checksum calculation.

NOTE: The upper 4-bits of the destination field are set to all 1s, which implies the destination field has the following format: 0xF<destination>. This is used to indicate the start of a message.

Table 15-10. Host Processor to 8051 Processor RS232 Message Structure

Header Section					**Data Section**	
Destination	Opcode	Reserved	Length	CS	Data Parameter	CS
1 Byte	1 Byte	1 Byte	1 Byte	1 Byte	(Length + 1) Bytes	1 Byte

15.6.2 RS232 Acknowledge Message Structure

The 8051 processor will acknowledge all valid RS232 messages (control and status requests) with a 5-byte long Acknowledge message, as shown in Table 15-11. The acknowledge status byte is described in Section 15.3.4.1.

Table 15-11. RS232 Acknowledge Response Message Structure

Acknowledge Response Packet				
Destination	Opcode	ACK Status	Length	CS
1 Byte	1 Byte	1 Byte	1 Byte	1 Byte

The checksum value is calculated according to the formula described in Section 15.7.

15.6.3 ZipWire2 8051 Processor to Host Processor Status Message Structure

For status request commands, the 8051 processor will send the variable length results after the acknowledge response. The data results are only sent if the acknowledge status byte was successful. Table 15-12 illustrates the message structure format.

The destination and opcode fields match what the 8051 received. The number of bytes is equal to the length +1, i.e., a length of 0 equals 1 byte and a length of 74 equals 75 bytes. The length does not include the checksum byte.

Table 15-12. 8051 Processor to Host Processor RS232 Message Structure

Acknowledge Response Section (Header)				**Data Section**	
Destination	Opcode	ACK Status	CS	Data Parameter(s)	CS
1 Byte	1 Byte	1 Byte	1 Byte	(Length + 1) Bytes	1 Byte

Control Command Response or invalid message
Status Command Response

The host processor first reads the API data length to determine the number of the status bytes. The number of status bytes is then read. The final byte (checksum) is then read to complete the message.

15.6.4 RS232 Message Transfer Protocol

The application sends a command to any of the devices in the system by transmitting a message over the serial communication channel. Every command that is correctly received and decoded by the 8051 processor is acknowledged by sending a special acknowledge message back to the application. In response to a status request command, the 8051 processor also sends a status response message that contains the information requested.

The 8051 processor is guaranteed to acknowledge a received message within specified time (see Section 15.1). The host processor will usually retransmit a message that was not acknowledged within this time limit. The host processor should send no new messages before the previous one was acknowledged unless the time limit has been exceeded. When the 8051 processor receives a status request command, it responds (after acknowledging the command) by sending a status message to the host processor.

The following examples illustrate the 8051 processor sequence of events. In the examples, all commands are sent to device number 0 (destination = 0).

Example 1: Set the _DSL_SYSTEM_CONFIG (0x01) to 0x48 (1T1, In Service, HTU-C).

Table 15-13. Example 1—Incoming RS232 Message

Header Section					Data Section	
Destination	Opcode	Reserved	Length	CS	Data 0	CS
0xF0	0x01	0x00	0x00	0x5B	0x48	0xE2

Table 15-14. Example 1—Outgoing RS232 Message

Header Section				
Destination	Opcode	ACK	Length	CS
0xF0	0x01	0xFF	0x00	0xA4

Example 2: Query the _DSL_STATUS (0x85); assume all return bytes are 0.

Table 15-15. Example 2—Incoming RS232 Message

Header Section					Data Section	
Destination	Opcode	Reserved	Length	CS	Data 0	CS
0xF0	0x85	0x00	0x00	0xDF	0x00	0xAA

Table 15-16. Example 2—Outgoing RS232 Message

Header Section					Data Section			
Destination	Opcode	ACK	Length	CS	Data 0	...	Data 6	CS
0xF0	0x85	0xFF	0x07	0x27	0x00	...	0x00	0xAA

Example 3: Processor Busy during query the _DSL_STATUS (0x85). No data results will be returned.

Table 15-17. Example 3—Incoming RS232 Message

Header Section					Data Section	
Destination	Opcode	Reserved	Length	CS	Data 0	CS
0xF0	0x85	0x00	0x00	0xDF	0x00	0xAA

Table 15-18. Example 3—Outgoing RS232 Message

Header Section				
Destination	Opcode	ACK	Length	CS
0xF0	0x85	0x01	0x00	0xDE

15.7 RS232 Checksum Function

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

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

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

The same rule is used by the 8051 processor to calculate the checksum byte of the status message sent to the host processor.

15.7.1 RS232 Multi-Device System

In a multi-ZipWire2 device system (refer to Section 3.1), the RS232 interface can only be connected to one Group Master device; therefore, the RS232 serial protocol can only support up to 8 devices (master plus 7 slaves).

15.7.2 Group Talk Serial Interface Protocol

The Group Talk serial interface is very similar to the RS232 interface protocol. The main difference is that the Group Master will broadcast an API command to all the slaves. Only the targeted slave (based on the destination field) will respond (acknowledge) to the Group Master.

NOTE: The customer should not have to be concerned with the Group Talk serial protocol. The customer only must make sure the hardware signals are connected correctly.

15.7.3 Boot RAM Software Download

The program download from the Group Master to the Group Slave(s) is done through the Group Talk serial interface using API commands. When a slave device is first powered on (or reset), the internal boot ROM of the ZipWire2 device supports a handful of API commands to download the program RAM. When a slave device is in operational mode, these download API commands will be ignored. This allows slave devices to be programmed without affecting the other slave devices operating mode.

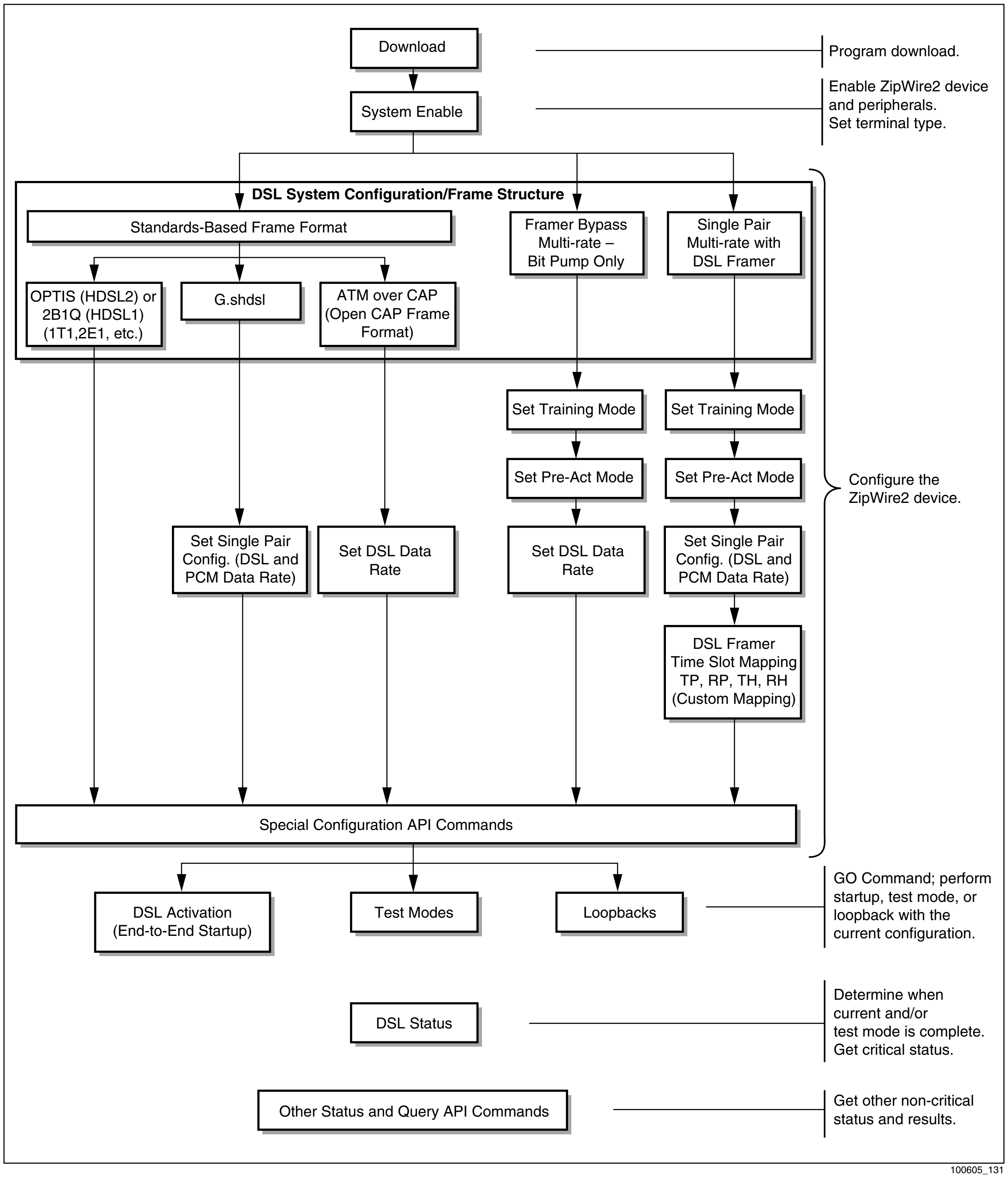
16.0 ZipWire2 API Configuration

The ZipWire2 chip set is extremely flexible and provides an extensive set of API commands. This section describes tries to simplify the API command by addressing which API commands are used in different applications.

16.1 API Command Sequencing

Figure 16-1 illustrates the API command flow to configure the ZipWire2 device for different applications. See Section 11.1.

Figure 16-1. API Command Sequencing


Download
Program download.
System Enable
Enable ZipWire2 device and peripherals.
Set terminal type.
DSL System Configuration/Frame Structure
Standards-Based Frame Format
Framer Bypass Multi-rate – Bit Pump Only
Single Pair Multi-rate with DSL Framer
OPTIS (HDSL2) or 2B1Q (HDSL1) (1T1,2E1, etc.)
G.shdsl
ATM over CAP (Open CAP Frame Format)
Set Training Mode
Set Training Mode
Set Pre-Act Mode
Set Pre-Act Mode
Configure the ZipWire2 device.
Set Single Pair Config. (DSL and PCM Data Rate)
Set DSL Data Rate
Set DSL Data Rate
Set Single Pair Config. (DSL and PCM Data Rate)
DSL Framer Time Slot Mapping TP, RP, TH, RH (Custom Mapping)
Special Configuration API Commands
DSL Activation (End-to-End Startup)
Test Modes
Loopbacks
GO Command; perform startup, test mode, or loopback with the current configuration.
DSL Status
Determine when current and/or test mode is complete. Get critical status.
Other Status and Query API Commands
Get other non-critical status and results.
100605_131

16.2 Indirect Configuration

The following features of the device are configured indirectly based on other API commands.

16.2.1 Scrambler/Descrambler Taps

The DSL scrambler and descrambler taps are based on the HDSL1 and HDSL2 standards, as shown in Table 16-1. The terminal type and frame structure API commands are used to determine the proper tap selection. The transmit scrambler will match the far end's receive descrambler.

Table 16-1. Scrambler/Descrambler Taps

Standard	HTU-C to HTU-R	HTU-R to HTU-C
HDSL1	$x^{23} + x^{5} + 1$	$x^{23} + x^{18} + 1$
HDSL2	$x^{23} + x^{5} + 1$	$x^{23} + x^{18} + 1$

16.2.2 CRC Tap

The DSL CRC Tap is based on the HDSL1 and HDSL2 standards, as shown in Table 16-2. The frame structure API command is used to determine the proper tap selection. The CRC polynomial is the same for both the HTU-C to HTU-R direction and the HTU-R to HTU-C direction.

Table 16-2. CRC Tap

Standard	CRC Tap
HDSL1	CRC6: $x^{6} + x + 1$
HDSL2	CRC6: $x^{6} + x + 1$

16.2.3 Sync Word

The sync word is based on the HDSL1 and HDSL2 standards, as shown in Tables 16-3 and 16-4. The frame structure and DSL configuration API commands, as well as the physical channel number 2T1 are used to determine the proper sync word. The sync word is the same for both the HTU-C to HTU-R direction and HTU-R to HTU-C direction.

The least significant bit is transmitted first.

The HDSL2 standards define a 10-bit sync word.

Table 16-3. Sync Word—HDSL2

HDSL2	Loop #1
G.shdsl	TBD
T1 OPTIS	TBD

The HDSL1 standards define a 14-bit sync word.

Table 16-4. Sync Word—HDSL1

HDSL1	Loop #1	Loop #2	Loop #3
E1	0x72	0x72	0x72
T1	0x72	0x27	N/A
Single Pair	0x72	N/A	N/A

16.2.4 Pair ID (Z-Bits)

The pair ID is based on the ETSI HDSL1 standards. The pair ID is the first three bits of the Z-bit field, as shown in Table 16-5. The frame structure and DSL configuration API commands are used to determine the proper Pair ID. The HTU-C defines the loop number pair ID, while the HTU-R matches the HTU-C.

Table 16-5. Pair ID of the Z-bit Field

	Z-Bit 2	Z-Bit 1	Z-Bit 0
Loop 1	0	0	1
Loop 2	0	1	0
Loop 3	1	0	0

16.3 Single Pair Configuration

The following points are assumed:

- Each device (DSL channel) can be independently configured.
- Multiple devices can share PCM.
- There can be subgroups (multiple PCM buses) within a ZipWire2 group.
- There is no loop reversal or switching master loop.
- RCLK equals TCLK. DPLL is bypassed so switching the DPLL source is unnecessary.

The single-pair-configuration API commands allow the application to customize the DSL and PCM data rates. The single pair configuration has no interaction with other devices, except that the devices share a common PCM bus. More specifically, the ZipWire2 devices will not handle loop reversal or switching of the master loop. The RCLK (PCM receive clock) must be sourced from TCLK (PCM transmit clock). Single-pair-configuration API commands are described in Table 16-6.

Table 16-6. Single Pair Configuration API Commands

API Command	Description
_SP_TOTAL_PCM_TSLOTS	Indicates the total number of PCM time slots available. X = 1...64
_SP_TOTAL_DSL_TSLOTS	Indicates the total number of DSL time slots available. X = 1...36
_SP_USED_TSLOTS	Indicates the number of time slots used by the PCM and DSL. X = 1...MIN (TOTAL PCM, TOTAL DSL)
_SP_PCM_FBIT	0 = No F-Bit, E1, or Nx64 1 = F-Bit present, T1
_SP_STARTING_TIME SLOT	Specifies starting time-slot location in PCM data stream.

From these API commands, the ZipWire2 software can determine the following information:

- Data rate: (Total number of DSL time slots x 64 k) + 16 k, where 16 k is the fixed overhead data
- PCM and HDSL Mapping

17.0 ZipWire2 API Commands

A modular API-based command set allows a seamless initialization of the ZipWire2 chip set. Once in operation, a comprehensive set of diagnostic and testing commands assist in performance monitoring, and fault detection and isolation tasks. The commands are divided into 2 categories:

- Control Commands (0x00 to 0x7F)—control the operation modes of the devices and set various parameters.
- Status Request Commands (0x80 to 0xFF)—Inquire for status and monitoring information from the device.

The Control and Status commands are then further divided:

- Level 1 API commands
  - Primary Control and Status commands for mainstream (standard) applications.
- Level 2 API commands
  - Secondary Control and Status commands for custom applications (used in conjunction with Level 1 commands). Also provides diagnostic and debugging commands for use during development and in-service diagnostics.
- Level 3 API commands
  - Low-Level API commands that are primarily used during in-house development and characterization. Customers will typically not need to use these commands.
- Read/Write API commands
  - The Read/Write Register commands allow the user to access any register located in XDATA space. The bit pump and framer registers are located in the 8051 XDATA space. These commands are primarily used in a development or debugging environment.
- T1/E1 Framer API commands
  - Bt8370 T1/E1 Framer-specific API commands.
- EVM-Specific API commands
  - EVM-Specific API commands, LEDs, DIP Switches, etc.

The C constant definitions are included in *DSL_API.H*.

17.1 API Commands: Quick Reference

Table 17-1 lists a summary of the API commands.

Table 17-1. API Command Summary (1 of 4)

HEX	Decimal	Command (C Constant)	In Length	Out Length	Page Ref
		Control Commands			
0	0	_DSL_RESET_SYSTEM	1	N/A	17-18
1	1	_DSL_SYSTEM_ENABLE	1	N/A	17-7
2	2	_DSL_AFE_CONFIG	1	N/A	17-42
3	3	_DSL_TRAINING_MODE	1	N/A	17-14
4	4	_DSL_SCR_DESCR_CONFIG	1	N/A	17-85
5	5	_DSL_PCM_MF_LEN	1	N/A	17-20
6	6	_DSL_SYSTEM_CONFIG	1	N/A	17-9
7	7	_DSL_STARTUP_MODE	1	N/A	17-16
8	8	_DSL_LOST_TIME_PERIOD	1	N/A	17-17
9	9	_DSL_LOOPBACK	1	N/A	17-47
A	10	_DSL_TRANSMIT_EXT_DATA	1	N/A	17-43
B	11	_DSL_ACTIVATION	1	N/A	17-11
C	12	_DSL_FORCE_DEACTIVATE	1	N/A	17-43
D	13	_DSL_TEST_MODE	1	N/A	17-44
E	14	_DSL_DATA_RATE	2	N/A	17-21
F	15	_BP_PREACTIVATION_MODE	1	N/A	17-15
10	16	_DSL_FR_PCM_CONFIG	1	N/A	17-19
11	17	_DSL_FR_HDSL_CONFIG	1	N/A	17-91
12	18	_DSL_PCM_CLK_CONF	1	N/A	17-54
13	19	_AFE_TX_GAIN	1	N/A	17-87
14	20	_BP_REVERSE_TIP_RING	1	N/A	17-85
15	21	_BP_BER_METER_STATE	1	N/A	17-49
16	22	_DSL_TX_ISO_PULSE	1	N/A	17-45
17	23	_DSL_TX_FIXED_PATT	1	N/A	17-46
18	24	_BP_ERLE_TEST_MODE	1	N/A	17-52
1B	27	_DSL_SINGLE_PAIR_CONFIG	4	N/A	17-20
21	33	_DSL_AUX_CLK_SELECT	1	N/A	17-54

Table 17-1. API Command Summary (2 of 4)

HEX	Decimal	Command (C Constant)	In Length	Out Length	Page Ref
23	35	_DSL_TP_BER_STATE	1	N/A	17-58
24	36	_DSL_RP_BER_STATE	1	N/A	17-58
25	37	_DSL_PRBS_CONFIGURE	1	N/A	17-59
26	38	_DSL_CONST_FILL	1	N/A	17-60
27	39	_DSL_DBANK	3	N/A	17-61
2B	43	_DSL_AUTO_WATER_LEVEL	1	N/A	17-93
2C	44	_DSL_TX_WATER_LEVEL	2	N/A	17-93
2D	45	_DSL_RX_WATER_LEVEL	2	N/A	17-94
30	48	_DSL_TP_MAPPER_VALUE	65	N/A	17-62
31	49	_DSL_TP_MAPPER_WRITE	1	N/A	17-63
32	50	_DSL_RP_MAPPER_VALUE	65	N/A	17-63
33	51	_DSL_RP_MAPPER_WRITE	1	N/A	17-64
34	52	_DSL_TH_MAPPER_VALUE	65	N/A	17-65
35	53	_DSL_TH_MAPPER_WRITE	1	N/A	17-66
36	54	_DSL_RH_MAPPER_VALUE	65	N/A	17-66
37	55	_DSL_RH_MAPPER_WRITE	1	N/A	17-67
40	64	_DSL_CLEAR_ERROR_CTRS	1	N/A	17-68
41	65	_DSL_INJECT_CRC_ERROR	1	N/A	17-74
42	66	_DSL_CRC_FEBE_ERR_STATE	1	N/A	17-75
4E	78	_DSL_FR_TX_RESET	1	N/A	17-89
4F	79	_DSL_FR_RX_RESET	1	N/A	17-90
52	82	_DSL_MASK_INTR_HOST	1	N/A	17-92
53	83	_DSL_DOWNLOAD_START	2	N/A	17-35
54	84	_DSL_DOWNLOAD_DATA	1–75	N/A	17-36
55	85	_DSL_DOWNLOAD_END	1	N/A	17-36
56	86	_DSL_DOWNLOAD_SLAVE	1	N/A	17-37
58	88	_DSL_DPLL_CLOCK_GEN	5	N/A	17-95
60	96	_EOC_SET_DATABASE	1	N/A	17-32
61	97	_EOC_SET_MSG_CONTROL	2	N/A	17-33
62	98	_EOC_SET_PROPRIETARY_LEN	2	N/A	17-35
75	117	_DSL_WRITE_REG	3	N/A	17-97

Table 17-1. API Command Summary (3 of 4)

HEX	Decimal	Command (C Constant)	In Length	Out Length	Page Ref
76	118	_DSL_WRITE_AFE	2	N/A	17-98
7F	127	_TDEBUG_MASK	1	N/A	—
Status Commands					
80	128	_DSL_READ_CONTROL	1	—	17-38
82	130	_DSL_FAR_END_ATTEN	1	1	17-29
83	131	_DSL_NOISE_MARGIN	1	1	17-30
84	132	_DSL_TIMING_RECOVERY	1	2	17-84
85	133	_DSL_STATUS	1	8	17-22
86	134	_DSL_STATUS_STATIC	1	—	17-25
8A	138	_DSL_VERSIONS	1	11	17-26
8B	139	_DSL_CONFIG_PINS	1	3	17-40
8C	140	_DSL_TP_BER_RESULTS	1	5	17-55
8D	141	_DSL_RP_BER_RESULTS	1	5	17-57
8E	142	_DSL_SLM	1	1	17-83
8F	143	_DSL_STAGE_NUMBER	1	5	17-39
90	144	_DSL_AAGC_SETTING	1	1	17-41
91	145	_AFE_READ_TX	1	2	17-88
92	146	_BP_BER_RESULTS	1	10	17-50
93	147	_BP_ERLE_RESULTS	1	16	17-53
95	149	_DSL_CRC_FEBE_IN_PROGRESS	1	10	17-76
96	150	_DSL_CRC_ERR_INTERVAL 1	1	50	17-77
97	151	_DSL_CRC_ERR_INTERVAL 2	1	48	17-78
98	152	_DSL_CRC_ERR_INTERVAL 3	1	14	17-79
99	153	_DSL_FEBE_ERR_INTERVAL 1	1	50	17-80
9A	154	_DSL_FEBE_ERR_INTERVAL 2	1	48	17-81
9B	155	_DSL_FEBE_ERR_INTERVAL 3	1	14	17-82
9C	156	_DSL_OPER_ERR_CTRS	1	10	17-69
9D	157	_DSL_TIME	1	12	17-73
9E	158	_DSL_HDSL_PERF_ERR_CTRS	1	10	17-70
9F	159	_DSL_PCM_PERF_ERR_CTRS	1	8	17-71
A0	160	_DSL_READ_REG	3	—	17-98
A1	161	_DSL_READ_AFE	2	—	17-98
A2	162	_DSL_SYSTEM_PERF_ERR_CTRS	1	6	17-72

Table 17-1. API Command Summary (4 of 4)

HEX	Decimal	Command (C Constant)	In Length	Out Length	Page Ref
B0	176	_EOC_SEND_COMMAND	2	1	17-31
B1	177	_EOC_GET_MSG_STATUS	1	1	17-31
B2	178	_EOC_GET_DATABASE	1	—	17-32
B3	179	_EOC_READ_RX_QUEUE	1	2	17-34

17.2 API Command Set Documentation Convention

The following conventions are used to document each of the API commands.

17.2.1 API Command Names

A description of the overall command is provided first.

C constant	Opcode #define as defined in DSL_API.H
Opcode	Opcode value as defined in DSL_API.H
Type	Opcode type: Control or Status
Incoming Bytes	The number of data parameters for the incoming message. The documentation uses a value of 1 to imply 1 byte. The maximum is 75 bytes.
Outgoing Bytes	The number of data parameters for the outgoing message. The documentation uses a value of 1 to imply 1 byte. The maximum is 75 bytes. For control commands, this value will be 'None' because there is no data sent back to the host.

Incoming Data Parameter Description

Byte #	Content	Description
1	Byte1 Name	Incoming byte 1 description
2	Byte2 Name	Incoming byte 2 description
N	ByteN Name	Incoming byte N description

Outgoing Data Parameter Description

Byte #	Content	Description
1	Byte1 Name	Outgoing byte 1 description
2	Byte2 Name	Outgoing byte 2 description
N	ByteN Name	Outgoing byte N description

NOTE: The Outgoing Data Parameter description is only provided for Status commands.

If required, addition descriptions for the incoming and outgoing data parameters are provided at the end of the command.

17.3 Level 1 API Commands

17.3.1 DSL System Enable

This command enables or disables the ZipWire2 System and provides information about the supported peripherals. This command matches the START configuration pins.

This command will the cause the ZipWire2 device to deactivate any training or test modes and transition to the Configure ZipWire2 state (see Figure 12-1). The ASM and DSL Loop Manager will be disabled.

C constant	_DSL_SYSTEM_ENABLE
Opcode	0x01
Type	Control
Incoming Bytes	1
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	System State	See the bit-field description below. Default value = START pins.

Bit 7:6	Bit 5	Bit 4:3	Bit 2	Bit 1	Bit 0
Unused	Reserved	Terminal Type	E1/T1 Framer Present	EVM Dip Switch and LEDs Present	DSL System State

Terminal Type Sets the DSL terminal type. In any operational ZipWire2 system, one side of one ZipWire2 device must be configured as an HTU-C and the other side of the ZipWire2 device must be configured as an HTU-R. The terminal type determines the activation procedure, timing recovery, scrambler and descrambler taps, and more.

In a multi-pair system, each bit pump on a board may be individually set as an HTU-C or HTU-R. However, most applications configure all bit pumps on a given board to the same terminal type.

Value	Option	Description	C Constant
0x00	HTU-C	Central Office Terminal	_DSL_HTUC
0x01	HTU-R	Remote Terminal	_DSL_HTUR
0x02	REG-C	Regenerator HTU-C	_DSL_REGC
0x03	REG-R	Regenerator HTU-R	_DSL_REGR

NOTE: If the current ROM-build does not support the desired terminal type setting, the ZipWire2 device will be forced to use whatever terminal type the ROM-build does support. For example, assume a ROM-build that only contains the HTU-C code and the user selects the terminal type to be HTU-R. The software will force the terminal type to be HTU-C.

E1/T1 Framer Present

Specifies whether the internal 8051 controls the Bt8370 E1/T1 Framer. This bit is provided for the Evaluation Module (EVM) operation. When enabled, the E1/T1 Framer is configured to basic transparent mode.

Customers may add the Bt8370 E1/T1 Framer (or equivalent) to their system. Customers could also customize the Bt8370 E1/T1 Framer operation to match their specific requirements, but this would require the customer to modify the internal 8051 code.

0 = E1/T1 Framer not present
1 = E1/T1 Framer present

EVM Dip Switch and LED Present

Specifies whether the internal 8051 controls the EVM dip switches and LEDs. This bit is provided for the EVM operation. Customers may add this circuitry to their system. Customers may also customize this circuitry to match their specific requirements, but this requires the customer to modify the internal 8051 code.

0 = EVM Dip Switch and LED not present
1 = EVM Dip Switch and LED present

DSL System State (Out-of-Service)

This command enables or disables the ZipWire2 Transceiver.

Setting the ZipWire2 State to Off (Out-of-Service) puts the chip in a power-down mode. Any further control and status commands issued to this bit pump (other than Bit Pump On/Off) returns the Not Available Acknowledge Status (see Table 15-5).

Setting the ZipWire2 State to On (In-Service) puts the system in a default configuration. Other API commands must then be issued to properly configure the device.

17.3.2 DSL System Configuration

This command configures the ZipWire2 basic mode of operation. This command will preconfigure other API commands, such as the DSL and PCM data rate, DSL line code, training mode, and so on. See Section 16.1 for descriptions of how the ZipWire2 is configured for different applications.

This command causes the ZipWire2 device to deactivate any training and test modes, and transition to the Configure ZipWire2 state (see Figure 12-1). The device is configured based on the settings. The ASM and DSL Loop Manager will be disabled.

C constant	_DSL_SYSTEM_CONFIG
Opcode	0x06
Type	Control
Incoming Bytes	1
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	DSL Configuration	See the bit field description below. The configuration pins determine the default value.

Bit 7:4	Bit 3:0
DSL Configuration	Frame Structure

DSL Configuration

The DSL Configuration configures the ZipWire2 device.

For standard modes such as 1T1or 2E1, the software automatically determines the DSL data rate, HDSL and PCM Mapping, and so on. See Section 16.0 which describes how the ZipWire2 is configured.

The Single Pair Configuration mode will configure the system based on the Single Pair API commands. The Single Pair configuration is used in multi-rate with DSL Framer applications. See the Single Pair API commands.

The Framer Bypass mode configures the ZipWire2 for bit pump-only operation; the DSL Framer is bypassed. The DSL Data Rate command is then used to set the desired data rate.

Value	Option	Description	C Constant
0x00	2T1	Configure for standard 2T1 mode.	_DSL_2T1
0x01	2E1	Configure for standard 2E1 mode.	_DSL_2E1
0x02	3E1	Configure for standard 3E1 mode.	_DSL_3E1
0x03	Framer Bypass	Configure for DSL Framer Bypass mode.	_DSL_FRAMER_BYPASS
0x04	1T1	Configure for standard 1T1 mode.	_DSL_1T1
0x05	1E1	Configure for standard 1E1 mode.	_DSL_1E1
0x06	Single Pair	Configure for Single Pair Configuration mode.	_DSL_SINGLE_PAIR
0x07–0x0F	Reserved	—	—

Frame Structure (Frame Format)

The frame structure determines the relative position of the payload and overhead bits (sync word, EOC, Indicator, etc.) within the DSL frame. See Chapter 14.0 for more details of the various frame structures.

Value	Option	Description	C Constant
0x00	Framer Bypass	The internal ZipWire2 Framer is disabled. The DSP serial transmit and receive data are derived from the TXDAT, RXDAT, and TX_RX_CLK pins.	_FRAMER_BYPASS_FORMAT
0x01	HDSL1 Frame	Uses the HDSL1 frame structure.	_HDSL1_FRAME_FORMAT
0x02	HDSL2 Frame	Uses the HDSL2 (OPTIS) frame structure.	_HDSL2_FRAME_FORMAT
0x03	G.shdsl Frame	Uses the G.shdsl frame structure.	_GSHDSL_FRAME_FORMAT
0x05–0x0F	Reserved	—	Reserved

17.3.3 DSL Activation

This command configures the DSL Activation.

C constant	_DSL_ACTIVATION
Opcode	0x0B
Type	Control
Incoming Bytes	1
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	Activation Configuration	See bit-field description below

Bit 7	Bit 6	Bit 5:4	Bit 3:2	Bit 1	Bit 0
Reserved	Auto Tx Ext Data	Activation Time-Out Setting	Startup Sequence Source	Activation Request State	DSL Loop Manager State

Auto Transmit External Data

The Startup Sequence Source option (below) determines the source of the data during startup. When the ZipWire2 is configured to transmit internal scrambled 1s, the ZipWire2 device can then be configured to automatically transmit external payload, once the ASM reaches normal operation. Setting this bit will enable the autotransmit external data feature.

Clearing this bit will disable the autotransmit external data feature. In certain applications, the host processor may need to configure some other device before the external payload is transmitted. In this scenario, the host would issue the Transmit External Data command (see Section 17.3.2) once the ZipWire2 reached normal operation.

0 = Disabled
1 = Enabled

Activation Time-Out Setting

This command sets the activation interval time-out setting.

Value	Option	Description	C Constant
0x00	Fixed 30-Second	The activation interval is set to 30 seconds. This option should be used when running at the standard HDSL1 and HDSL2 data rates (default value).	_ACT_TIME_30SEC
0x01	Reserved	—	—
0x02	Variable Number of Symbols	The activation interval is based on data rate rather than fixed time. This option should be used in multirate systems (see Table 17-2 for startup times).	_ACT_TIME_VARIABLE
0x03	Reserved	—	—

The variable number of symbols option uses a formula that calculates the number of symbols based on data rate. The absolute time is then calculated based on the number of symbols. The formula is derived from the typical startup times. The activation times are set to approximately twice the typical startup times. The results of the formula correspond to the following times.

Table 17-2. HDSL1 Variable Rate Startup Times

Data Rate (kbps)	Typical Startup Time(s)	Activation Time-Out(s)
144	64.4	128.8
288	35.3	70.5
416	26.3	52.6
784	16.8	33.7
1,168	13.3	26.7
1,552	11.5	23.1
2,320	9.8	19.5

Startup Sequence Source

This command selects the data source during the activation sequence. When using the combination ZipWire2 DSP and DSL Framer, the internal framed option should be used to conform to the HDSL1 or HDSL2 standards. In Framer Bypass mode, the source of the data is dependent on the application.

The Internal Scrambled 1s option provides internally generated scrambled 1s that may be used in a standalone transceiver implementation. This option does not meet the standard activation requirements that require HDSL framing information to be included in the activation sequences. When the internal scrambled 1s option is specified, the _DSL_TRANSMIT_EXT_DATA command must be called when the activation process is successfully completed.

The External Data Source option should only be used in conjunction with Framer Bypass mode. The transceiver data source is provided from the TXDAT. The transceiver will modify the magnitude bits to generate the necessary 2-Level, 4-Level, and so on codes. When using the external data source, either the Bit Pump Scrambler (see Section 17.5.6) must be enabled or the data must be scrambled by the external source.

Value	Option	Description	C Constant
0x00	Internal Framed	During the activation training, the ZipWire2 provides the necessary framed and scrambled data as defined in the HDSL standards (default value). This option should be used when the DSL Framer is enabled.	_INTERNAL_FRAMED_SOURCE
0x01	Internal Scrambled 1s	During the activation training, the ZipWire2 provides scrambled 1s. This option should be used in Framer bypass.	_INTERNAL_ONES_SOURCE
0x02	External Data Source	During the activation training, the ZipWire2 uses externally supplied data. This option might be used in Framer bypass applications that use a custom external frame format.	_EXTERNAL_DATA_SOURCE
0x03	Reserved	—	—

Activation Request State

This command sets the activation request flag (ACTREQ). The activation request controls the state of the DSL Loop Manager and Activation State Manager. Setting this bit will enable the activation request and set the ASM state to the Inactive state. Clearing this bit will disable the activation request.

The activation request should be disabled for certain diagnostic test modes and loopbacks, primarily when trying to debug the DSL Framer section of the device.

The default is based on the START Configuration pins.

DSL Loop Manager State

This command sets the DSL Manager Loop State (see Section 12.10). Setting this bit will enable the DSL Loop Manager; clearing this bit will disable the DSL Loop Manager.

The DSL Loop Manager should be disabled for certain diagnostic test modes and loopbacks. When disabled, the bit pump and DSL Framer performs no automatic functions. The host processor must control these devices through the API.

The default is based on the START Configuration pins.

17.3.4 Bit Pump Training Mode

This command configures the Bit Pump Training mode. This command is only required when the DSL Frame Structure is set to Framer Bypass. This command is properly configured when a standard configuration (i.e., HDSL1 2E1 or HDSL2 1T1) is selected. See Chapter 13.0 for more details.

This command will take effect on subsequent startups.

C constant	_DSL_TRAINING_MODE
Opcode	0x03
Type	Control
Incoming Bytes	1
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	Training Mode	See bit field description below.

Bit 7:4	Bit 3:0
DSL Line Code	Training Mode

DSL Line Code Sets the desired final DSL line code. The automatic mode will select the appropriate line code based on the training mode.

Value	Mode	Line Code	C Constant
0x00	3-Bit Trellis Coded	16 PAM (OPTIS/G.shdsl)	_16PAM_CODED_LINE
0x01	2-Bit Trellis Coded	8 PAM	_8PAM_CODED_LINE
0x02	1-Bit Trellis Coded	4 PAM	_4PAM_CODED_LINE
0x03	Invalid	—	—
0x04	4-Bit Uncoded	16 PAM	_16PAM_UNCODED_LINE
0x05	3-Bit Uncoded	8 PAM	_8PAM_UNCODED_LINE
0x06	2-Bit Uncoded	4 PAM (2B1Q)	_4PAM_UNCODED_LINE
0x07	1-Bit Uncoded	2 PAM (startup only)	_2PAM_UNCODED_LINE
0x07–0x0E	—	—	—
0x0F	Automatic	—	—

Training Mode Sets the desired bit pump training mode.

Value	Training Mode	C Constant
0x00	2B1Q	_2B1Q_TRAINING
0x01	HDSL2 OPTIS	_OPTIS_TRAINING
0x02	G.shdsl	_GSHDSL_TRAINING
0x03	IDSL	_IDSL_TRAINING
0x04	ADSL	—
0x05	2B1Q ZipStart	—
0x07–0x0F	Reserved	—

NOTE: The ADSL and 2B1Q ZipStart are provided to be compatible with the existing ZipWire1 AutoBaud; these modes are not supported by the CN8980.

17.3.5 DSL Pre-Activation Mode

This command configures the pre-activation mode. This command will take effect on subsequent startups.

C constant	_BP_PREACTIVATION_MODE
Opcode	0x0F
Type	Control
Incoming Bytes	1
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	Pre-Activation Mode	See bit-field description below.

Bit 7:4	Bit 3:0
Reserved	Pre-Activation Mode

Pre-Activation Mode Sets the desired pre-activation mode.

Value	Pre-Activation Mode	C Constant
0x00	None	_NO_PREACT
0x01	Automatic	_AUTOMATIC_PREACT
0x02	AutoBaud	_AUTO_BAUD_PREACT
0x04	G.hs	_GHS_PREACT
0x05–0x0F	Reserved	—

17.3.6 DSL Startup Mode

This command configures the DSL Startup Mode.

C constant	_DSL_STARTUP_MODE
Opcode	0x07
Type	Control
Incoming Bytes	1
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	DSL Startup Mode	See bit-field description below

Bit 7	Bit 6	Bit 5:2	Bit 1:0
BP Auto Tip/Ring	ZipStartup	Reserved	Nonstandard Startup

17.3.6.1 Bit Pump Auto Tip/Ring Reversal

The Auto Tip/Ring Reversal is used to enable and disable the bit pump automatic tip/ring detection and reversal of the line. When set (enabled), the bit pump activation procedure will automatically detect and correct tip/ring reversal. In order for tip/ring reversal to work, the user must set the internal scrambled 1s during activation (Section 17.3.3), enable the scrambler and descrambler, and have a ZipWire1 or ZipWire2 part on the far-end configured in the same fashion (none of these are part of this API option). Clearing this bit will disable the bit pump auto tip/ring feature.

NOTE: This command should only be used while operating in legacy 2B1Q mode in combination with the DSL Framer Bypass mode.
0 = Disable
1 = Enable

ZipStartup Enable or disable the ZipStartup feature. When enabled, the bit pump algorithm will support the Zip Start code, i.e., save coefficients during the activation, allow coefficients to update during normal operation, and so on. Disabling the ZipStart will force all activation attempts in the cold mode.

NOTE: This command is only supported for legacy 2B1Q applications.
0 = Disable
1 = Enable

Nonstandard Startup In a system where both terminals use the ZipWire2 bit pump, several activation operations can be performed more efficiently relative to the standard requirements. Setting this bit will enable the nonstandard activation procedure; clearing this bit will use the standard activation procedure.

17.3.7 LOST Time (Tsilent) Period

An on-chip timer is restarted when a Loss Of Signal (LOS) condition is detected. When this timer reaches a predefined value, the LOST status bit is turned ON. Once turned on, the status bit will not reset (even if there is no longer an LOS condition). The LOST indication is cleared only when an Activate or Reset command is issued.

The LOST mechanism is active only in the deactivated state. Thus, during activation or normal operation, the LOST status is never set. This implementation is in correspondence with the T1E1/ETSI HDSL activation state diagrams. The LOST time interval is programmable in the range 0–25.5 seconds with a resolution of 1/10 second. The value of the LOST status bit may be checked using the bit pump status command.

NOTE: Tsilent (HDSL2) is synonymous with LOST (HDSL1).

C constant	_DSL_LOST_TIME_PERIOD
Opcode	0x08
Type	Control
Incoming Bytes	1
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	LOST Time Period	1-byte unsigned integer X. The LOST time period is set to X / 10 seconds. Default Value: 10 = 1 second.

17.3.8 DSL Reset

This command issues a reset to the DSL system.

The software reset will set the DSL Reset flag (see Table 15-13). The program will reconfigure the device to their default values (based on the configuration pins). Any custom configuration commands must be reissued. The software reset can be issued to any ZipWire2 device.

The hardware reset can only be issued to the Group Master. The hardware reset will toggle the RESET_OUT pin forcing all other devices (including itself) to reset. Each device will then reperform the download procedure.

C constant	_DSL_RESET_SYSTEM
Opcode	0x00
Type	Control
Incoming Bytes	1
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	Reset	0x00 = Software reset (_DSL_SW_RESET) 0x01 = Hardware reset (_DSL_HW_RESET)

17.3.9 DSL Framer—PCM Configuration

This command configures the DSL Framer PCM block. This only applies when the DSL Framer is enabled.

C constant	_DSL_FR_PCM_CONFIG
Opcode	0x10
Type	Control
Incoming Bytes	1
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	PCM Configuration	See bit-field description below.

Bit 7:5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	RSIG Enabled	PRA Enabled	F-Bit Present	DPLL Mode	PCM Float

Receive Signaling Enabled Only applies in Point to Multi-point applications.

PRA Enabled TBD

F-Bit Present Indicates if the PCM Bus contains an extra F-Bit. Refer to Section 16.3 for details about the Single Pair Configuration commands. Setting this bit indicates the PCM bus does contain the F-Bit; then, the PCM bus is a (N x 64k + 1) serial stream. Clearing this bit indicates the PCM bus does not contain an F-Bit; then, the PCM bus is a (N x 64k) serial stream.

DPLL Mode Configure the PCM DPLL mode. The DPLL clock can operate in either closed-loop or open-loop mode. The PCM Rx clock is typically sourced from the DPLL clock.

0—DPLL operates in a closed loop to recover the PCM receive clock from the master HDSL receive channel. During startup, the DPLL is switched to open-loop mode to provide a stable PCM RCLK. Once startup is completed, the DPLL is switched back to closed-loop mode.

1—DPLL always operates in open-loop mode. The DPLL clock provides a fixed frequency. Use the DSL Framer clock generator to the desired DPLL clock frequency (see Section 17.5.14).

PCM Float Configure the PCM Frame Format. When running in framed format, the sync bit is used to indicate bit 0 of time slot 0 of frame 0. Unframed format allows unframed or asynchronous payload mapping of PCM frames into HDSL frames.

0—Sync aligned with bit 0 of time slot 0.

1—Asynchronous sync versus data alignment.

17.3.10 PCM Multi-Frame Length

This command sets the number of PCM frames per PCM multi-frame. The PCM multi-frame is used to indicate bit 0 of time slot 0.

C constant	_DSL_PCM_MF_LEN
Opcode	0x05
Type	Control
Incoming Bytes	1
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	Multi-Frame Length	A value of 0 implies 1 frame. One frame is 125 µs. X = 0...47 (1–48 frames or 125 µs–6 ms) Default is 6 ms.

17.3.11 Single Pair Configuration

This command configures the Single Pair Configuration parameters. Refer to Section 16.3 for more details about the single pair configuration commands.

C constant	_DSL_SINGLE_PAIR_CONFIG
Opcode	0x1B
Type	Control
Incoming Bytes	4
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	Number of PCM Time Slots	Total number of PCM time slots available. A value of 0 implies 1 time slot. X = 0...255 (1–256 time slots)
2	Number of DSL Time Slots	Total number of DSL time slots available. A value of 0 implies 1 time slot. X = 0...71 (1–72 time slots)
3	Number of Occupied Time Slots	Total number of occupied time slots available. A value of 0 implies 1 time slot. X = 0...MIN (Total PCM, Total DSL) – 1
4	Starting PCM Time Slot Location	Indicates the location of the first time slot to extract from the PCM bus. The ZipWire2 device will then sequential map the number of occupied time slots into the DSL channel.

17.3.12 DSL Data Rate

The Data Rate command sets the bit pump DSL Data Rate. This command should only be used while operating in Framer Bypass Mode. When the DSL Framer is enabled, the DSL Configuration (Section 17.2.1) or Single Pair API commands will calculate the proper DSL data rate based on the number of HDSL bytes available. The Data Rate parameter is determined by the following equation:

$$X = \text{Data Rate} / 8{,}000$$

The supported range of X is 8–580 (or data rate = 64–4,640 kbps respectively).

C constant	_DSL_DATA_RATE
Opcode	0x0E
Type	Control
Incoming Bytes	2
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	Data Rate (Low Byte)	Contains the lower 8 bits of the Symbol Rate parameter X.
2	Data Rate (High Byte)	Contains the upper 2 bits of the Symbol Rate parameter X.

17.3.13 DSL Status—Dynamic

This command queries the DSL Status registers. These status bytes provide dynamic information about the ZipWire2 system.

C constant	_DSL_STATUS
Opcode	0x85
Type	Status
Incoming Bytes	1
Outgoing Bytes	8

Incoming Data Parameter Description

Byte #	Content	Description
1	0	Set to 0 for future compatibility.

Outgoing Data Parameter Description

Byte #	Content	Description
1	STATUS_1	The STATUS_1 contains basic status of the system. The Host Processor should only be required to poll this one status register to see if the system is functioning, i.e., simple Go / No Go. The DSL status bit definitions are listed in Table 17-3.
2	STATUS_2	Undefined
3	STATUS_3	Startup Failure Status bit definitions are listed in Table 17-4.
4	STATUS_4	DSL Framer Status bit definitions are listed in Table 17-5.
5	STATUS_5	Undefined
6	STATUS_6	Undefined
7	STATUS_7	Undefined
8	STATUS_8	Undefined

Table 17-3. STATUS_1: DSL Status Bit Definitions

STATUS_1 Bit	Description	Bit Definition
0	LOS—Loss of Signal	0 = No 1 = Yes
1	LOST—Loss of Signal Timer (Tsilent)	0 = No 1 = Yes
2	LOSW—Loss of Sync Word	0 = No 1 = Yes
3	LOSWT—Loss of Sync Word Timer (Tresync)	0 = No 1 = Yes
4	NMR OK—Line Quality	0 = No (poor line quality) 1 = Yes (good line quality)
5	Fatal Error—need to poll other registers. One example failed 3 consecutive attempts.	0 = No 1 = Yes
7–6	Activation Status	00 = Idle (_ASM_STAT_IDLE) 01 = Normal Operation (_ASM_STAT_SUCCESS) 10 = Deactivated (_ASM_STAT_DEACTIVATED) 11 = In-Progress (_ASM_STAT_IN_PROGRESS)

Table 17-4. STATUS_3: Startup Failure Status Bit Definitions

STATUS_3 Bit	Description	Bit Definition
3–0	Activation Failure—Result will be latched until next successful or failed startup attempt.	0 = None 1 = Bad NMR 2 = Unable to Frequency Lock 3 = Failed Pre-Activation Startup 4 = Unable to detect Sync Word 5 = Failed Pair ID
5–4	Reserved.	—
6	Failed 3 (or n) consecutive startup attempts. N is an API command.	0 = No 1 = Yes
7	Activation Time-Out.	0 = No 1 = Yes

Table 17-5. STATUS_4: DSL Framer Status Bit Definitions

STATUS_4 Bit	Description	Bit Definition
0	DPLL Locked—only valid in DPLL Closed Loop mode. Set when the DPLL Phase Error is less than (TBD).	0 = Not locked 1 = Locked, DPLL Stable
1	DPLL Error—only valid in DPLL Closed Loop mode. Set when the DPLL Error exceeds (TBD).	0 = No 1 = Yes
2	Transmit Stuff Error.	0 = No 1 = Yes
3	Transmit FIFO Error.	0 = No 1 = Yes
4	Receive FIFO Error.	0 = No 1 = Yes
5	Loop Reversal.	0 = No 1 = Yes
7–6	HDSL Sync State.	00 = Out-of-Sync 01 = Acquiring Sync 10 = In-Sync 11 = Losing Sync

Table 17-6. STATUS_8: Acknowledge Status (ISR) Bit Definitions

STATUS_8 Bit	Description	Bit(s) Definition
0	Receive EOC Message—this bit sets when the 8051 processor receives an EOC message that needs to be forwarded to the host processor.	0 = No 1 = Yes
1	Activation State Manager (ASM) Transition—this bit sets when the ASM transitions into the Activate State (normal operation) or when the ASM transitions from the Pending State to the Deactivated State. The host reads the STATUS_1, Activation Status bits to determine the link-up or link-down status. In summary, this bit only provides link-down to link-up transition and link-up to link-down transition.	0 = No 1 = Yes
7–2	Reserved.	0

17.3.14 DSL Status—Static

This command queries the DSL Status registers.

C constant	_DSL_STATUS_STATIC
Opcode	0x86
Type	Status
Incoming Bytes	1
Outgoing Bytes	?

Incoming Data Parameter Description

Byte #	Content	Description
1	0	Set to 0 for future compatibility.

Outgoing Data Parameter Description

Byte #	Content	Description
1	Fatal Error	Fatal Error Status bit definitions are listed in Table 17-7.
2	Transmit State	Transmit State Status bit definitions are listed in Table 17-8.

Table 17-7. Fatal Error Bit Definitions

# of Bits	Description	Bit Definition
0	Device failed power-on self-test.	0 = No 1 = Yes
3–1	Self Test Failure, only if Bit 0 is set.	0 = Failed Boot Load 1 = DSP Not Detected 2 = AFE Not Detected 3 = DSL Framer Not Detected 4 = Failed RAM/ROM test
7–4	Reserved	—

Table 17-8. Tx State Bit Definitions

# of Bits	Description	Bit Definition
3–0	Tx Level	One of 16 transmit levels.
4	Tx State	0 = Tx OFF, 1 = Tx ON; see Tx level.
7–5	Reserved	—

17.3.15 Versions

Requests the ZipWire2 hardware, software, and silicon version numbers.

C constant	_DSL_VERSIONS
Opcode	0x8A
Type	Status
Incoming Bytes	1
Outgoing Bytes	11

Incoming Data Parameter Description

Byte #	Content	Description
1	0	Set to 0 for future compatibility.

Outgoing Data Parameter Description

Byte #	Content	Description
1	Major Software Version	1-byte unsigned integer field returning the major software version.
2	Minor Software Version	1-byte unsigned integer field returning the minor software version.
3	Internal Software Version	1-byte unsigned integer field returning internal software version. Used during S/W development. This field will return a 0 for all official releases.
4	Compiler Build Option Low	Low byte of 2-byte field containing the compiler build option, see below.
5	Compiler Build Option High	Upper byte of the compiler build option field.
6	DSP Silicon Type	1-byte unsigned integer field returning the DSP silicon type.
7	DSP Silicon Revision	1-byte unsigned integer field returning the DSP silicon revision.
8	AFE Silicon Type	1-byte unsigned integer field returning the AFE silicon version type.
9	AFE Silicon Revision	1-byte unsigned integer field returning the AFE silicon revision.
10	DSL Framer Silicon Type	1-byte unsigned integer field returning the DSL Framer silicon type.
11	DSL Framer Silicon Revision	1-byte unsigned integer field returning the DSL Framer silicon revision.

DSP Silicon Type	
Byte Value	**Type**
0x00	Bt8952
0x01	Bt8960
0x02	Bt8970
0x03	RS8973
0x80	CN8980

NOTE: The legacy 2B1Q devices are listed because this information may be retrieved across the EOC channel.

DSP Silicon Revision	
Byte Value	**Revision**
0x00	Rev A
0xFF	Rev B
0xFE	Rev C
0x01	Rev D

AFE Silicon Type	
Byte Value	**Type**
14	CN8980

DSL Framer Silicon Type	
Byte Value	**Type**
0	CN8980
255–1	Undefined

A 1 in the bit-field denotes the compiler option is enabled.

Compiler Build Option	
Bit #	**Description**
0	HTU-C
1	HTU-R
2	Regenerator
3	HDSL2 (OPTIS)
4	HDSL1 (2B1Q)
5	DSL Framer
6	T1/E1 Framer
7	EVM Code
8	Group Talk
9	TDEBUG
10–15	Reserved

17.3.16 Line Attenuation

Requests a value of the far-end signal attenuator. This value is based on measuring the average far-end signal level after echo cancellation. The return value is already adjusted to match the analog gain value (AAGC). The attenuation is calibrated against a 150k Sine Wave.

For HDSL1, the far-end signal attenuation is calibrated for a 13.5 dBM transmit power at the far-end.

For HDSL2, the far-end signal attenuation is calibrated for a 17.0 dBM transmit power at the far-end.

C constant	_DSL_FAR_END_ATTEN
Opcode	0x82
Type	Status
Incoming Bytes	1
Outgoing Bytes	1

Incoming Data Parameter Description

Byte #	Content	Description
1	0	Set to 0 for future compatibility.

Outgoing Data Parameter Description

Byte #	Content	Description
1	Attenuation	1-byte unsigned integer X ($0 \leq X \leq 255$) indicating the signal power attenuation in units of 0.5 dB. For example, a value of 35 means a total cable attenuation of 17.5 dB.

17.3.17 Noise Margin

Requests a value of the Noise Margin of the Receiver (NMR). 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}. The value is based on measuring the average absolute level of the noise at the input to the slicer.

The noise margin format matches the definition in the HDSL1 and HDSL2 standards.

C constant	_DSL_NOISE_MARGIN
Opcode	0x83
Type	Status
Incoming Bytes	1
Outgoing Bytes	1

Incoming Data Parameter Description

Byte #	Content	Description
1	0	Set to 0 for future compatibility.

Outgoing Data Parameter Description

Byte #	Content	Description
1	NMR	1-byte signed integer X ($-128 \leq X \leq 127$) indicating the noise margin in units of 0.5 dB. For example, a value of –4 means a noise margin of –2.0 dB.

17.3.18 EOC Send Command

This command sends a standard EOC-request command across the EOC channel. The message ID and destination are placed into the EOC transmit queue.

C constant	_EOC_SEND_COMMAND
Opcode	0xB0
Type	Status
Incoming Bytes	2
Outgoing Bytes	1

Incoming Data Parameter Description

Byte #	Content	Description
1	Destination Address	EOC destination address (see Table 11-4).
2	Message ID	EOC message ID (see Table 11-5).

Outgoing Data Parameter Description

Byte #	Content	Description
1	Tx Queue Index	1-byte field returning index into TxQueue[]. This allows the host to query the status for a particular message.

17.3.19 EOC Get Message Status

This command gets the status of the EOC message. This command only queries the status for messages placed into the transmit queue.

C constant	_EOC_GET_MSG_STATUS
Opcode	0xB1
Type	Status
Incoming Bytes	1
Outgoing Bytes	1

Incoming Data Parameter Description

Byte #	Content	Description
1	TxQueue Index	TxQueue[] index returned from the EOC Send Command.

Outgoing Data Parameter Description

Byte #	Content	Description
1	Message Status	1-byte field returning index into the TxQueue[]. This allows the host to query the status for a particular message.

17.3.20 EOC Set Database Data

This command fills in the EOC database contents.

C constant	_EOC_SET_DATABASE
Opcode	0x60
Type	Control
Incoming Bytes	1 + L—depends on EOC command
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	Message ID	EOC message ID (see Table 11-5).
2 + L	Database Contents	Database information; refer to the specific EOC command for message format.

17.3.21 EOC Get Database Data

This command extracts the current contents of an EOC database.

C constant	_EOC_GET_DATABASE
Opcode	0xB2
Type	Status
Incoming Bytes	1
Outgoing Bytes	L – depends on EOC command

Incoming Data Parameter Description

Byte #	Content	Description
1	Message ID	EOC message ID (see Table 11-5).

Outgoing Data Parameter Description

Byte #	Content	Description
1–L	Database Contents	Database information; refer to the specific EOC command for message format.

17.3.22 EOC Set Message Control

This command sets the message control to determine how each EOC message is processed.

C constant	_EOC_SET_MSG_CONTROL
Opcode	0x61
Type	Control
Incoming Bytes	2
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	Message ID	EOC message ID (see Table 11-5).
2	Message Control	Message control for specified message ID.

17.3.23 EOC Read Receive Queue

This command reads the RxQueue to determine which EOC messages were received. For each message received, two bytes are returned to the host processor: source/destination address and message ID.

C constant	_EOC_READ_RX_QUEUE
Opcode	0xB3
Type	Status
Incoming Bytes	1
Outgoing Bytes	2 × number of messages received (N = number of messages)

NOTE: The host can determine the number of messages by using the following formula:

$$N = (\text{Data Length} + 1) / 2$$

Incoming Data Parameter Description

Byte #	Content	Description
1	0	Set to 0 for future compatibility

Outgoing Data Parameter Description

Byte #	Content	Description
1	Src/Dest Address #1	Source/destination address of first message in RxQueue
2	Message ID #1	Message ID of first message in RxQueue
3	Src/Dest Address #2	Source/destination address of second message in RxQueue
4	Message ID #2	Message ID of second message in RxQueue
(N × 2) + 1	Src/Dest Address #N	Source/destination address of Nth message in RxQueue
(N × 2) + 2	Message ID #N	Message ID of Nth message in RxQueue

17.3.24 EOC Set Proprietary Length

This command sets the length of the EOC proprietary messages. This command can be used to override the User-Defined Message and API over EOC proprietary messages.

C constant	_EOC_SET_PROPRIETARY_LEN
Opcode	0x62
Type	Control
Incoming Bytes	2
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	Message ID	EOC proprietary message ID, 0x70 to 0x78 and 0xF0 to 0xF8 (see Table 11-5).
2	Proprietary Length	Length of proprietary message; the maximum length is 75 bytes.

17.3.25 Download Start (Length)

This command is used to begin a new download. The size of program code (length) is used to validate the download procedure. This command must be issued before the Download Data command is issued.

Issuing the Download Start command in the middle of a download will reset the download process.

NOTE: This command is only supported when the ZipWire2 device is in boot code.

C constant	_DSL_DOWNLOAD_START
Opcode	0x53
Type	Control
Incoming Bytes	2
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1–2	Length	2-byte value specifying the PRAM length. Low byte is programmed first.

17.3.26 Download Data

This command transfers the next block of the program data. All download packets should have a data parameter length of 75 bytes until the last packet, which will contain only the length of byte necessary to complete the download.

NOTE: This command is only supported when the ZipWire2 device is in boot code.

C constant	_DSL_DOWNLOAD_DATA
Opcode	0x54
Type	Control
Incoming Bytes	Up to 75 (L)
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1–L	Download Data	Download data. Data is copied sequentially into the next PRAM location.

17.3.27 Download End (Checksum)

This command is used to indicate the end of the download. The download data checksum is passed in so the 8051 can validate the download contents.

NOTE: This command is only supported when the ZipWire2 device is in boot code.

C constant	_DSL_DOWNLOAD_END
Opcode	0x55
Type	Control
Incoming Bytes	1
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	Checksum	Checksum of download data contents, does not include the length or checksum bytes. The formula to calculate the checksum is listed in Section 4.6.11.3.

17.3.28 Download Slave

This command is used to trigger the Group Master to download its PRAM contents to the slave devices.

NOTE: This command is only supported by the Group Master when it is in operational mode.

C constant	_DSL_DOWNLOAD_SLAVE
Opcode	0x56
Type	Control
Incoming Bytes	1
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	Slave ID	Set to 0 for future compatibility.

17.4 Level 2 API Commands

17.4.1 Read DSL Control Commands

The Read DSL Control Commands API command is a read-back of the current setting for each of API control commands.

C constant	_DSL_READ_CONTROL
Opcode	0x80
Type	Status
Incoming Bytes	1
Outgoing Bytes	Depends on control command

Incoming Data Parameter Description

Byte #	Content	Description
1	Control Command Opcode	Return the configuration for the specified control command. Input range is 0–127 (0x7F).

Outgoing Data Parameter Description

Byte #	Content	Description
N	—	Refer to the specific control command

17.4.2 Stage Number

Queries for the stage number of the various state machines.

C constant	_DSL_STAGE_NUMBER
Opcode	0x8F
Type	Status
Incoming Bytes	1
Outgoing Bytes	5

Incoming Data Parameter Description

Byte #	Content	Description
1	0	Set to 0 for future compatibility

Outgoing Data Parameter Description

Byte #	Content	Description
1	DSL Loop Manager Stage	1-Byte unsigned integer field returning the DSL Loop Manager stage number.
2	ASM Stage	1-Byte unsigned integer field returning the Activation State Manager (ASM) stage number.
3	DSP Startup Stage	1-Byte unsigned integer field returning the DSP startup stage number.
4	DSL Framer Stage	1-Byte unsigned integer field returning the DSL Framer stage number.
5	DPLL Handler Stage	1-Byte unsigned integer field returning the DPLL Handler stage number.

17.4.3 Read Configuration Pins

Requests the current START and DEVADR/BOOTOP configuration pins.

C constant	_DSL_CONFIG_PINS
Opcode	0x8B
Type	Status
Incoming Bytes	1
Outgoing Bytes	3

Incoming Data Parameter Description

Byte #	Content	Description
1	0	Set to 0 for future compatibility

Outgoing Data Parameter Description

Byte #	Content	Description
1	START Pins	Request START configuration pins.
2	DEVADR and BOOTOP Pins	Request Device Address / BOOT Mode configuration pins (see Table 17-9).
3	Number of Devices Found	Request the number of devices found within the group. This only applies to the Group Master. A Group Slave will return a 1.

Table 17-9. DEVADR / BOOPOP Bit Definitions

Bit 7	Bit 6:4	Bit 3:0
Reserved	DEAVADR[2:0]	BOOTOP[3:0]

17.4.4 AFE Setting

Reads the current setting of the AFC and hybrid selection. The DSP startup algorithm determines these results. The AFE value should only be read during normal operation.

C constant	_DSL_AFC_SETTING
Opcode	0x90
Type	Status
Incoming Bytes	1
Outgoing Bytes	1

Incoming Data Parameter Description

Byte #	Content	Description
1	0	Set to 0 for future compatibility.

Outgoing Data Parameter Description

Byte #	Content	Description
1	AFE Value	1-Byte unsigned (see Table 17-10).

Table 17-10. AFE Bit Definitions

AFE Bit	Description	Bit(s) Definition
0	Hybrid Selection	0 = Hybrid 1 1 = Hybrid 2
3–1	Reserved	0
6–4	AFC setting corresponding to the Absolute Gain (dB) setting	0 = 0.0 dB 1 = 3.5 dB 2 = 6.0 dB 3 = 7.9 dB 4 = 10.0 dB 5 = 11.6 dB 6 = 13.3 dB 7 = 15.0 dB
7	Reserved	0

17.4.5 Analog Front End (AFE) Configuration

Configures the ZipWire2 AFE chip.

C constant	_DSL_AFE_CONFIG
Opcode	0x02
Type	Control
Incoming Bytes	1
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	AFE Configuration	See bit-field description below.

Bit 7:2	Bit 1	Bit 0
Reserved	Hybrid 2 Enable	Hybrid 1 Enable

Hybrid 2 Enable Enables or disables the Hybrid 2 input.
0—Hybrid 2 Disabled
1—Hybrid 2 Enabled (Default)

Hybrid 1 Enable Enables or disables the Hybrid 1 input.
0—Hybrid 1 Disabled
1—Hybrid 1 Enabled (Default)

17.4.6 DSL Force Deactivate

This command will force the ZipWire2 system to deactivate. If the DSL Manager and Activation State Manager are enabled, the system will then reperform the startup.

If in normal operation, the Activation State Manager will proceed through the pending deactivated state. The force-deactivate flag is then cleared when the Activation State Manager reaches the deactivated state. If the system is in the process of a startup, the system will immediate fail that startup attempt by asserting the activation time-out (Tact) flag.

This command allows the host processor to easily control the startup state machine if the host processor detects a higher-level error.

C constant	_DSL_FORCE_DEACTIVATE
Opcode	0x0C
Type	Control
Incoming Bytes	1
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	1	Set to 1 for future compatibility.

17.4.7 Transmit External Data

This command should be used only when the Startup Sequence Source (Section 17.3.3) is set to internal scrambled 1s (value 0x01). This command should only be used while operating in Framer Bypass Mode.

When issued, this command causes the bit pump to start transmission of externally supplied data symbols. This command should be issued upon the successful completion of activation, which is determined by the application based on bit pump status responses. The transmitted data should be supplied to the bit pump prior to issuing this command.

This function is also controlled by autocommand.

C constant	_DSL_TRANSMIT_EXT_DATA
Opcode	0x0A
Type	Control
Incoming Bytes	1
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	0	Set to 0 for future compatibility.

17.4.8 Test Modes

Operates the bit pump in special test modes. For all test modes, the DSL Manager and Activation State Manager (ASM) will be disabled. To turn off any of the special test modes, use the Test Mode command with an _EXIT_TEST_MODE (0 value) parameter. Executing any test mode will be destructive to the current bit pump link (bring the link down). All test modes require a complete activation procedure to be repeated (assuming normal operation is required) after exiting the test mode; this is accomplished by enabling the DSL Manager and ASM. When exiting all these tests, the bit pump is initialized to a reset state and goes to the IDLE State, where it awaits further commands.

NOTE: The duty cycle of the Isolated Pulses and Alternating Symbols Test Modes are dependent on the meter interval register which defaults to 0xTBD (decimal TBD) when the test mode API is issued.

C constant	_DSL_TEST_MODE
Opcode	0x0D
Type	Control
Incoming Bytes	1
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	Test Mode Option	See Table 17-11

Table 17-11. Test Mode Options (1 of 2)

Option	Description	Parameter (C Constant)
Exit Test Mode	Cancel current test mode.	0x00 (_EXIT_TEST_MODE)
Transmit Isolated Pulse	Transmit (repeatedly) an isolated pulse. This is useful for testing the transmit pulse template. Use the _BP_TX_ISOLATED_PULSE API command to set the desired pulse code.	0x01 (_TM_TX_ISOLATED_PULSE)
Transmit Alternating Symbols	Transmit (repeatedly) an alternating symbols pattern. Use the _BP_TX_ISOLATED_PULSE API command to set the desired symbol pattern.	0x02 (_TM_TX_ ALTERNATE_SYM)
Transmit Fixed Pattern	Transmit (repeatedly) a fixed 8-bit pattern. Use the _AFE_TX_FIXED_PATT API command to set the desired pulse code.	0x03 (_TM_TX_FIXED_PATT)
Transmit AFE White Noise Generator	Transmit a white noise source (fixed PRBS pattern) generated by the ZipWire2 AFE.	0x04 (_TM_AFE_TX_WHITE)
Transmit AFE Sine Wave	Transmit a sine wave generated by the ZipWire2 AFE.	0x05 (_TM_AFE_TX_SINE)
Transmit Continuous 16-Level	Transmit continuous 16-level scrambled 1s. This is useful for measuring PSD and transmit power.	0x09 (_SIXTEEN_LEVEL_SCR)

Table 17-11. Test Mode Options (2 of 2)

Option	Description	Parameter (C Constant)
Transmit Continuous 8-Level	Transmit continuous 8-Level scrambled 1s. This is useful for measuring PSD and transmit power.	0x0A (_EIGHT_LEVEL_SCR)
Transmit Continuous 4-Level	Transmit continuous 4-level scrambled 1s. This is useful for measuring PSD and transmit power.	0x0B (_FOUR_LEVEL_SCR)
Transmit Continuous 2-Level	Transmit continuous 2-level scrambled 1s. This is useful for measuring PSD and transmit power.	0x0C (_TWO_LEVEL_SCR)
Set Nominal Crystal Frequency	Set frequency control word to its nominal (center) frequency. This is useful for measuring the crystal center frequency.	0x0D (_VCXO_NOMINAL)
Set Minimum Crystal Frequency	Set frequency control word to its minimum frequency. This is useful for measuring the crystal pull range.	0x0E (_VCXO_MIN)
Set Maximum Crystal Frequency	Set frequency control word to its maximum frequency. This is useful for measuring the crystal pull range.	0x0F (_VCXO_MAX)
ERLE Test	Start the ERLE test with the current ERLE options (see Section 17.4.14). Use the Section 17.4.15 API command to query the results.	0x10 (_TM_ERLE)

17.4.9 Bit Pump Transmit Isolated Pulses Test Mode

This command selects the desired output level while in the Transmit Isolated Pulses or Transmit Alternating Symbols test modes.

When the Transmit Alternating Symbols test mode is active, setting the bit-level map to either +15 or –15 will set the alternating +15/–15 symbol pattern.

Default is 0x00 (Transmit Isolated –15 Pulse).

C constant	_DSL_TX_ISO_PULSE
Opcode	0x16
Type	Control
Incoming Bytes	1
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	Isolated Pulse Option	Set the Isolated Pulse level, i.e., +15, +13, and so on, or Alternating Symbol level, i.e., +15/–15, +13/–13, and so on. See Section 13.1 for bit-level code values.

17.4.10 Bit Pump Transmit Fixed Pattern Test Mode

This commands sets the desired pattern while in the Transmit Fixed Pattern test mode.

C constant	_DSL_TX_FIXED_PATT
Opcode	0x17
Type	Control
Incoming Bytes	1
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	Fixed Pattern	8-bit value

17.4.11 Loopbacks

Operates the bit pump in loopback modes (see Figure 11-2). To turn off any of the loopback modes, use the loopback command with the _EXIT_LOOPBACK (0 value) parameter.

The AFE Analog and Bit Pump Transmit Loopbacks are destructive to the current bit pump link, that is, will bring the link down since the ZipWire2 system performs a mini-startup to adapt the DSP receiver section. The Normal Operation and Activation Failure bits in the DSL status should be used to determine when status of the loopback (similar concept to a normal startup). On exit of these loopbacks, the bit pump is initialized to a reset state and goes to the idle state, where it awaits further commands. A complete activation procedure should be repeated if normal operation is required. The DSL Loop Manager is disabled when performing these analog loopbacks.

The other loopbacks can be issued (or exited) at any time during normal operation without affecting the bit pump link. Only the throughput data will be affected to match the desired loopback condition. The software will automatically handle swapping scrambler and descrambler taps on entry and exit of certain loopbacks.

All loopbacks can be issued while in the Out-of-Service (Idle) state to facilitate development and debugging of other devices in the system. For example, the ZipWire2 can be placed in the _FR_PCM_ON_PCM_LB to develop (and debug) code for the T1/E1 framer.

C constant	_DSL_LOOPBACK
Opcode	0x09
Type	Control
Incoming Bytes	1
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	Loopback Option	See Table 17-12

Table 17-12. Loopback Options (1 of 2)

Option	Description	Parameter (C Constant)
Exit Loopback	Cancel current loopback.	0x00 (_EXIT_LOOPBACK)
Destructive to HDSL Link		
Transmitting Analog Loopback	The data is transmitted out the AFE line driver and looped back into the AFE hybrid input. The AFE Receiver inputs are bypassed.	0x01 (_AFE_HYBRID1_LB) 0x02 (_AFE_HYBRID2_LB)
Silent Analog Loopback	The data is looped back internally before the AFE line driver into the AFE A/D converter. The AFE Receiver and Hybrid inputs are bypassed. The AFE Line Driver is disabled.	0x03 (_AFE_SILENT_LB)

Table 17-12. Loopback Options (2 of 2)

Option	Description	Parameter (C Constant)
AFE Transmit Digital Loopback	The data is looped back internally before the AFE to Bit pump interface back into the AFE line driver. The AFE Line Driver will output the proper levels. This option is provided for internal testing only.	0x04 (_AFE_TX_DIG_LB)
Bit Pump Transmit Loopback	The data is looped back internally before the Bit Pump to AFE interface back into the Bit Pump AFE serial inputs. The AFE line driver will still output the proper levels. The AFE Receiver and Hybrid inputs are bypassed.	0x05 (_BP_TX_LB)
Not Destructive to HDSL Link		
Bit Pump Near Loopback	The data is looped back internally before the Bit Pump DSP transmit section is looped back into the DSL Framer HDSL receive section. When the DSL Framer is present, the DSL Framer scrambler and descrambler are set to use the same tap.	0x06 (_BP_DIGITAL_NEAR_LB)
Bit Pump Far Loopback	The data is looped back internally before the DSL Framer HDSL receive section is back into the Bit Pump DSP transmit section. The bit pump scramblers will be enabled. The far-end scrambler and descrambler must also be enabled.	0x07 (_BP_DIGITAL_FAR_LB)
DSL Framer HDSL on HDSL Loopback	The data is looped back internally before the Bit Pump DSP receive section back into the DSL Framer HDSL transmit section. The bit pump scramblers will be enabled and the DSL Framer scramblers will be disabled.	0x08 (_FR_HDSL_ON_HDSL_LB)
DSL Framer PCM on HDSL Loopback	The data is looped back internally before the DSL Framer HDSL transmit section back into the Bit Pump DSP receive section. The DSL Framer scrambler and descrambler are set to use the same tap.	0x09 (_FR_PCM_ON_HDSL_LB)
DSL Framer HDSL on PCM Loopback	The clock, data, and sync signals are looped back internally before the DSL Framer PCM receive inputs back into the DSL Framer PCM transmit inputs.	0x0A (_FR_HDSL_ON_PCM_LB)
DSL Framer PCM on PCM Loopback	The clock, data, and sync signals are looped back internally from the DSL Framer PCM transmits inputs to the DSL Framer PCM receive inputs.	0x0B (_FR_PCM_ON_PCM_LB)

17.4.12 Bit Pump BER Meter State

Activates or deactivates the internal Bit Pump BER Meter.

When activated, the bit pump is configured to transmit an internal scrambled 1s pattern. This command should only be called during the Bit Pump's normal operation. If the DSL Framer is present, then the Activation State Manager and DSL Loop Manager will ignore all DSL Framer Out-of-Sync errors.

When deactivated, the bit pump is set to transmit external data, and the transmit scrambler and receive descrambler are put back to their previous value before the Activate BER Meter was issued. The Bit Pump BER Meter results are unmodified so they can be still read.

NOTE: This command should only be used while operating in Framer Bypass mode. It is recommended that the customer use the DSL Framer BER Meter when the DSL Framer is present.

C constant	_BP_BER_METER_STATE
Opcode	0x15
Type	Control
Incoming Bytes	1
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	Bit Pump BER State	0 = Deactivate the Bit Pump BER Meter (default) 1 = Activate the Bit Pump BER Meter

17.4.13 Bit Pump BER Meter Results

Requests the BER Meter Status. Reading the BER Meter status commands while the BER Meter is enabled does not effect the BER meter operation.

C constant	_BP_BER_RESULTS
Opcode	0x92
Type	Status
Incoming Bytes	1
Outgoing Bytes	10

Incoming Data Parameter Description

Byte #	Content	Description
1	0	Set to 0 for future compatibility

Outgoing Data Parameter Description

Byte #	Content	Description
1	BER Status	1-byte value indicating the BER Meter Status, see bit-field definitions below.
2–3	Number of Bit Errors	16-bit value specifying the number of bit-errors. The low byte is sent first followed by the high byte.
4–5	Number of Meter Intervals	16-bit value specifying the number of meter intervals. The low byte is sent first followed by the high byte.
6–9	Meter Interval Length	32-bit value specifying the meter interval length. The low byte is sent first.
10	Bits per Symbol	8-bit value specifying the number of bits per symbol.

NOTE(S):
1. 16-bit value = (high byte << 8) + (low byte)
2. 32-bit value = (byte 4 << 24) + (byte 3 << 16) + (byte 2 << 8) + (byte 1)

BER Status Bits

Status Bit	Indicates	Value = 0	Value = 1
0	BER Meter Enabled	Not Active	Active
7–1	Reserved	—	—

(Need to validate the formulas and assumptions for ZipWire2. TBD)
The following formulas are used to calculate the Avg BER and Elapsed Time:

$$AvgBER = \frac{\#ofBitErrors}{\#ofMeterIntervals \times MeterIntervalLength \times BitsPerSymbol}$$

$$ElapsedTime = \frac{\#ofMeterIntervals \times MeterIntervalLength \times BitsPerSymbol}{DataRate}$$

Variable	How Derived
# of Bit-Errors	Read the # of Bit-Errors Low and High Byte API commands and build a 16-bit unsigned integer.
# of Meter Intervals	Read the # of Meter Intervals Low and High Byte API commands and build a 16-bit unsigned integer.
Meter Interval Length	Read the Meter Interval register and build a 32-bit unsigned integer. During normal operation, these registers should always read TBD.
Data Rate	Data Rate of the system, i.e., 2,320,000 or 784,000.
Bits per Symbol	Specify the number of bits per symbol. This will be either 2, 3, or 4.

17.4.14 ERLE Test Mode

This command activates the ERLE Test Mode. This command will disable the DSL Manager. To abort the ERLE test mode before completion, use the _DSL_TEST_MODE–_EXIT_TEST_MODE, API command.

NOTE: The ERLE test is typically run with the transmitter set to 2-Level, and the AAGC set to 0.0 dB.

C constant	_BP_ERLE_TEST_MODE
Opcode	0x18
Type	Control
Incoming Bytes	1
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	ERLE Configuration	See bit-field description below.

Bit 7	Bit 6:4	Bit 3	Bit 2:1	Bit 0
Reserved	AGAIN[2:0]	Reserved	Transmit Level[1:0]	Hybrid Selection

AAGC Bits	Decimal	Gain (dB)
000	0	0.0
001	1	3.5
010	2	6.0
011	3	7.9
100	4	10.0
101	5	11.6
110	6	13.3
111	7	15.0

Tx Level Bits	Decimal	Description
00	0	Set transmitter to 2-Level
01	1	Set transmitter to 4-Level
10	2	Set transmitter to 8-Level
11	3	Set transmitter to 16-Level

Hybrid	Description
0	Select Hybrid 1
1	Select Hybrid 2

17.4.15 ERLE Results

This command queries for the ERLE Test Mode results.

C constant	_BP_ERLE_RESULTS
Opcode	0x93
Type	Status
Incoming Bytes	1
Outgoing Bytes	16

Incoming Data Parameter Description

Byte #	Content	Description
1	0	Set to 0 for future compatibility.

Outgoing Data Parameter Description

Byte #	Content	Description
1–4	Noise	32-bit value specifying the background noise floor. The low byte is sent first.
5–8	SLM	32-bit value specifying the SLM value. The low byte is sent first.
9–12	FELM	32-bit value specifying the FELM value. The low byte is sent first.
13–16	SLM2	32-bit value specifying the SLM2 value. The low byte is sent first.
NOTE(S): 32-bit value = (byte 4 << 24) + (byte 3 << 16) + (byte 2 << 8) + (byte 1)		

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

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

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

17.4.16 Auxiliary CLK Select

Selects the Auxiliary CLK output frequency.

C constant	_DSL_AUX_CLK_SELECT
Opcode	0x21
Type	Control
Incoming Bytes	1
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	Aux Clock Select	TBD

17.4.17 PCM Clock Configuration

Configures the PCM Transmit and Receive Clock Sources.

C constant	_DSL_PCM_CLK_CONF
Opcode	0x12
Type	Control
Incoming Bytes	1
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	PCM Clock Configuration	See bit-field description below.

Bit 7:6	Bit 5	Bit 4	Bit 3:2	Bit 1:0
Reserved	RP_CLK_POL	TP_CLK_POL	RP_CLK_SOURCE	TP_CLK_SOURCE

RP_CLK_POL Receive PCM clock polarity.
- 0—Normal clock selected by *RP_CLK_SOURCE* (rising edge outputs, falling edge inputs).
- 1—Inverted clock selected by *RP_CLK_SOURCE* (falling edge outputs, rising edge inputs).

TP_CLK_POL Transmit PCM clock polarity.
- 0—Normal clock selected by *TP_CLK_SOURCE* (rising edge outputs, falling edge inputs).
- 1—Inverted clock selected by *TP_CLK_SOURCE* (falling edge outputs, rising edge inputs).

RP_CLK_SOURCE Receive PCM clock source.
- 00—TPCLK input pin.
- 01—PEXTCLK input.
- 10—DPLL recovery clock.
- 11—Invalid, do not use.

TP_CLK_SOURCE Transmit PCM clock source.
- 00—TPCLK input pin.
- 01—PEXTCLK input.
- 10—DPLL recovery clock.
- 11—Invalid, do not use.

17.4.18 DSL Framer Transmit PCM BER Meter Results

Requests the DSL Framer Transmit PCM BER Meter Status. Reading the BER Meter status commands while the BER Meter is enabled does not effect the BER meter operation.

C constant	_DSL_TP_BER_RESULTS
Opcode	0x8C
Type	Status
Incoming Bytes	1
Outgoing Bytes	5

Incoming Data Parameter Description

Byte #	Content	Description
1	0	Set to 0 for future compatibility.

Outgoing Data Parameter Description

Byte #	Content	Description
1	BER Status	1-byte value indicating the BER Meter Status (see Table 17-13).
2–3	Number of Bit Errors	16-bit value specifying the number of bit-errors. The low byte is sent first followed by the high byte.
4–5	Elapsed Time	16-bit value specifying the elapsed time in seconds. The low byte is sent first followed by the high byte.
NOTE(S): 16-bit value = (high byte << 8) + (low byte).		

Table 17-13. DSL Framer BER Status Bits

Status Bit	Description	Bit Definition
1–0	Qualification Phase Status	00 - IDLE 01 - Complete 10 - Failed 11 - In Progress
3–2	Measurement Phase Status	00 - IDLE 01 - Complete 10 - Failed 11 - In Progress
7–4	Reserved	—

The following formulas are used to calculate the Average BER:

$$AvgBER = \frac{\#ofBitErrors}{\#ofBitsProcessed}$$

where:

When the DSL Framer is complete, use:

$$\#ofBitsProcessed = BERScale$$

When the DSL Framer BER is in progress, use:

$$\#ofBitsProcessed = ElapsedTime \times DataRate \times \frac{\#MappedBERBitsperFrame}{\#BitsperFrame}$$

17.4.19 DSL Framer Receive PCM BER Meter Results

Requests the DSL Framer Receive PCM BER Meter status. Reading the BER Meter status commands while the BER Meter is enabled does not effect the BER meter operation.

NOTE: The description of the return results and formulas are described in the Transmit PCM BER Meter results command.

C constant	_DSL_RP_BER_RESULTS
Opcode	0x8D
Type	Status
Incoming Bytes	1
Outgoing Bytes	5

Incoming Data Parameter Description

Byte #	Content	Description
1	0	Set to 0 for future compatibility.

Outgoing Data Parameter Description

Byte #	Content	Description
1	BER Status	1-byte value indicating the BER Meter status.
2–3	Number of Bit Errors	16-bit value specifying the number of bit errors. The low byte is sent first followed by the high byte.
4–5	Elapsed Time	16-bit value specifying the elapsed time in seconds. The low byte is sent first followed by the high byte.
NOTE(S): 16-bit value = (high byte << 8) + (low byte).		

17.4.20 Transmit PCM BER State

Enable or Disable the DSL Framer Transmit PCM BER meter. The Transmit and Receive PCM BER meters can be operated independently.

C constant	_DSL_TP_BER_STATE
Opcode	0x23
Type	Control
Incoming Bytes	1
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	BER State	0 = Disable the Transmit PCM BER Meter. All results are unmodified so they can be still read (Default Value). 1 = Enable the Transmit PCM BER meter. All results are reset to 0. Issuing the enable option forces the BER meter to reperform the BER sync qualification period. The error counter and elapsed time counters are reset to 0. The enable option can be used as a BER meter reset.

17.4.21 Receive PCM BER State

Enable or disable the DSL Framer Receive PCM BER meter. The Transmit and Receive PCM BER meters can be operated independently.

C constant	_DSL_RP_BER_STATE
Opcode	0x24
Type	Control
Incoming Bytes	1
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	BER State	0 = Disable the Receive PCM BER meter. All results are unmodified so they can still be read (Default Value). 1 = Enable the Receive PCM BER meter. All results are reset to 0. Issuing the enable option forces the BER meter to reperform the BER sync qualification period. The error counter and elapsed time counters are reset to 0. The enable option can be used as a BER meter reset.

17.4.22 PRBS Configure

Configure the PRBS generator to one of the selected patterns. The Transmit PCM and Receive PCM share a common PRBS generator. This command is only necessary when either the Transmit or Receive PCM BER Configure data is sourced from the PRBS generator.

Issuing this command with the same data pattern parameter value will force a reset on the PRBS polynomial.

C constant	_DSL_PRBS_CONFIGURE
Opcode	0x25
Type	Control
Incoming Bytes	1
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	PRBS Configuration	See bit-field description below. Default is 0x08.

Bit 7:6	Bit 5	Bit 4	Bit 3:0
BER Scale	PRBS Invert	PRBS Source	PRBS Data Pattern

BER Scale BER Test Interval (or number of bits) where bit errors are accumulated.

00—2^{31} Bits.
01—2^{28} Bits.
10—2^{25} Bits.
11—2^{21} Bits.

PRBS Invert 0—PRBS Data Normal (not inverted)
1—PRBS Data Inverted

PRBS Source 0—Enable random pattern, data generated from Table 17-14
1—Enable fixed pattern, data generated from CONST_PATT register

Table 17-14. PRBS Data Pattern

Data Pattern	Description	Parameter (C Constant)
All 0s (SPACE)	Outputs an all-0s pattern.	0x00 (_DSL_PRBS_ZERO)
All 1s (MARK)	Outputs an all-1s pattern.	0x01 (_DSL_PRBS_ONE)
1:1	Alternating 0s and 1s.	0x02 (_DSL_PRBS_1_1)
2^6–1	Repeats every 2^6–1 (63) bits. The polynomial is $x^6 + x + 1$.	0x03 (_DSL_PRBS_2_6)
2^9–1	Repeats every 2^9–1 (511) bits. The polynomial is $x^9 + x^4 + 1$.	0x04 (_DSL_PRBS_2_9)
2^{11}–1	Repeats every 2^{11}–1 (2047) bits. The polynomial is $x^{11} + x^2 + 1$.	0x05 (_DSL_PRBS_2_11)
2^{15}–1	Repeats every 2^{15}–1 bits. The polynomial is $x^{15} + x^{14} + 1$.	0x06 (_DSL_PRBS_2_15)
QRSS	Repeats every 2^{20}–1 bits with 14-bit zero suppression. The polynomial is $x^{20} + x^{17} + 1$.	0x07 (_DSL_PRBS_QRSS)
2^{23}–1	Repeats every 2^{23}–1 bits. The polynomial is $x^{23} + x^{18} + 1$.	0x08 (_DSL_PRBS_2_23)
NOTE(S): All-0s, All-1s, and alternating 0s and 1s set the PRBS source to the fixed pattern option and program the Fill Pattern (Section 17.4.23) to the appropriate value.		

17.4.23 Fill Pattern (CONST_FILL)

Sets the Fill Pattern (Constant Pattern) used when the Transmit or Receive PCM BER Configure data is sourced from the Constant Pattern. The Constant Pattern is a useful debugging tool because the PRBS can generate a known data pattern in a given time slot to debug sync versus data alignment problems.
Writing the fill pattern while the device is in the PRBS mode will corrupt the PRBS pattern.

C constant	_DSL_CONST_FILL
Opcode	0x26
Type	Control
Incoming Bytes	1
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	Fill Pattern	1-byte field, least significant bit is sent first. The default is 0x55.

17.4.24 Data Bank Contents

Sets the 3 Data Bank patterns.

Data Bank 1 is used to fill unused PCM and HDSL time slots in the PCM and HDSL mappers. The default value is 0xFF which provides an AIS (all 1s) code for these unused time slots. The AIS pattern conforms to the HDSL standards.

The software current does not use Data Banks 2 and 3.

C constant	_DSL_DBANK
Opcode	0x27
Type	Control
Incoming Bytes	3
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	Data Bank 1	1-byte field, least significant bit is sent first. The default is 0xFF.
2	Data Bank 2	1-byte field, least significant bit is sent first. The default is 0x00.
3	Data Bank 3	1-byte field, least significant bit is sent first. The default is 0x55.

17.4.25 Transmit PCM Mapper Value

The Transmit PCM Mapper Value command allows the user to control the data going into the Transmit PCM FIFO. The Transmit PCM FIFO Mapper Table is implemented with an 8 x 64 RAM that can accommodate up to 64 different table entries. Each table entry can then process from 1–8 time slots, thus providing up to 512 (8 x 64) time slots per Transmit PCM frame. The 64 table entries are indexed from 0–63 and are stored in data RAM. The Transmit PCM Mapper Write command is used to write the table from the data RAM to the device.

C constant	_DSL_TP_MAPPER_VALUE
Opcode	0x30
Type	Control
Incoming Bytes	65
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	Starting Address	1-byte value indicating the table entry starting address. The starting address must be set to 0 for the current silicon.
2–65	Table Entry	See bit-field description below. The default is based on the system configuration.

Bit 7:5	Bit 4	Bit 3	Bit 2:0
Number of Time Slots (NTS)	Reserved	BER Enable	Data Source

Number of Time Slots Indicates the number of time slots to be controlled in each table line (000 stands for 1 and 111 stands for 8). The NTS should be consistent with the *TP_TS_SIZE* register value, because the total number of bits being controlled in each line are NTS * *TP_TS_SIZE.*

BER EN Enables the BER meter (towards PCM) for the specified time slots:
0—Discard
1—Enable BER meter

Data Source Specifies the Transmit PCM data source:
000—Disregards data
001—DATA from serial input TPDAT
010—PRBS generator (towards HDSL)
011—Assert TPINSEN pin and inserts data from TPINSDAT pin
100—Previous time slot
101–111—Not used

17.4.26 Transmit PCM Mapper Write

Writes the Transmit PCM Mapper table to the device. This should only be called after the Transmit PCM Mapper table has been completely filled in. The appropriate resets are issued.

C constant	_DSL_TP_MAPPER_WRITE
Opcode	0x31
Type	Control
Incoming Bytes	1
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	0	Set to 0 for future compatibility.

17.4.27 Receive PCM Mapper Value

The Receive PCM Mapper Value command allows the user to control the data going into the Receive PCM FIFO. The Receive PCM FIFO Mapper table is implemented with an 8 x 64 RAM that can accommodate up to 64 different table entries. Each table entry can then process from 1–8 time slots, thus providing up to 512 (8 × 64) time slots per Receive PCM frame. The 64 table entries are indexed from 0–63 and are stored in data RAM. The Receive PCM Mapper Write command is used to write the table from the data RAM to the device.

C constant	_DSL_RP_MAPPER_VALUE
Opcode	0x32
Type	Control
Incoming Bytes	65
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	Starting Address	1-byte value indicating the table entry starting address. The starting address must be set to 0 for the current silicon.
2–65	Table Entry	See bit-field description below. The default is based on the system configuration

Bit 7:5	Bit 4	Bit 3	Bit 2:0
Number of Time Slots (NTS)	Drop	BER Enable	Data Source

Number of Time Slots Indicates the Number of Time Slots (NTS) to be controlled in each table line (000 stands for 1 and 111 stands for 8). The NTS should be consistent with the *RP_TS_SIZE* register value, because the total number of bits being controlled in each line are NTS * *RP_TS_SIZE*.

DROP When enabled, asserts RPDROP output to mark specific time slots in RPDAT. When disabled and *RPDAT_MODE* = 1, RPDAT is three-stated.
0—Disable
1—Enable

BER EN Enables the BER meter (from HDSL) for the specified time slots.
0—Discard
1—Enable BER Meter

Data Source Specifies the Receive PCM data source for each time slot.
000—RX FIFO1
001—PRBS generator (towards PCM)
010—DATA BANK Register 1 (DBANK_1)
011—DATA BANK Register 2 (DBANK_2)
100—DATA BANK Register 3 (DBANK_3), or Signaling Table Enable (When *SIG_EN* is set to 1)
101—Data from RPEXTDAT input (used in multi-pair configuration)
110—Not used
111—Not used

17.4.28 Receive PCM Mapper Write

Writes the Receive PCM Mapper table to the device. This should only be called after the Receive PCM Mapper table has been completely filled in. The appropriate resets are issued.

C constant	_DSL_RP_MAPPER_WRITE
Opcode	0x33
Type	Control
Incoming Bytes	1
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	0	Set to 0 for future compatibility.

17.4.29 Transmit HDSL Mapper Value

The Transmit HDSL Mapper Value command allows the user to control the data transmitted to the HDSL channel. The Transmit HDSL Payload Mapper table is implemented with an 8x128 RAM that can accommodate up to 128 table entries. Each table entry can then process from 1–4 time slots, thus providing up to 512 (4 x 128) time slots per Transmit HDSL frame. The 128 table entries are indexed from 0–127 and are stored in data RAM. The Transmit HDSL Mapper Write command is used to write the table from the data RAM to the device.

The Transmit HDSL Mapper value configures the current Transmit HDSL Mapper table location with the specified data value. The table pointer is incremented to the next table entry location.

C constant	_DSL_TH_MAPPER_VALUE
Opcode	0x34
Type	Control
Incoming Bytes	65
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	Starting Address	1-byte value indicating the table entry starting address.
2–65	Table Entry	See bit-field description below. The default is based on the system configuration.

Bit 7:6	Bit 5:4	Bit 3	Bit 2:0
Reserved	Number of Time Slots (NTS)	Reserved	Data Source

Number of Time Slots Indicates the number of time slots to be controlled in each table entry (00 stands for 1 and 11 stands for 4). The NTS should be consistent with the *TH_TS_SIZE* register value, because the total number of bits being controlled in each line are NTS * *TH_TS_SIZE.*

Data Source Specifies the Transmit HDSL data source for each time slot.

000—Override/Insert Data Bank register 1 (DBANK_1)
001—Payload–Read data from Transmit FIFO
010—Reserved
011—If *AUX_EN* = 0—Override/Insert Data Bank register 2 (DBANK_2).
If *AUX_EN* = 1—Payload 3 (Auxiliary channels).
100–111—Reserved

17.4.30 Transmit HDSL Mapper Write

Writes the Transmit HDSL Mapper table to the device. This should be called only after the Transmit HDSL Mapper table has been completely filled in. The appropriate resets are issued.

C constant	_DSL_TH_MAPPER_WRITE
Opcode	0x35
Type	Control
Incoming Bytes	1
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	0	Set to 0 for future compatibility.

17.4.31 Receive HDSL Mapper Value

The Receive HDSL Data Mapper Value command allows the user to control the data coming from the HDSL channel. The Receive HDSL Payload Mapper table is implemented with an 8x128 RAM that can accommodate up to 128 table entries. Each table entry can then process from 1–4 time slots, thus providing up to 512 (4 x 128) time slots per Receive HDSL frame. The 128 table entries are indexed from 0–127 and are stored in data RAM. The receive HDSL Mapper Write command is used to write the table from the data RAM to the device.

The receive HDSL Mapper value configures the current receive HDSL Mapper table location with the specified data value. The table pointer is incremented to the next table entry location.

C constant	_DSL_RH_MAPPER_VALUE
Opcode	0x36
Type	Control
Incoming Bytes	65
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	Starting Address	A 1-byte value indicating the table entry starting address.
2–65	Table Entry	See bit-field description below. The default is based on the system configuration

Bit 7:6	Bit 5:4	Bit 3	Bit 2:0
Reserved	Number of Time Slots (NTS)	Reserved	Data Source

17.4.31.1 Number of Time Slots

Indicates the number of time slots to be controlled in each table line (00 stands for 1 and 11 stands for 4). The NTS should be consistent with the *RH_TS_SIZE* register value, because the total number of bits being controlled in each line are NTS * *RH_TS_SIZE.*

Data Source

Specifies the Receive HDSL data destination for each time slot.
000—Discard
001—Insert into Receive FIFO
010—Reserved
011—Output to RHAUX pin, auxiliary channel
100–111—Reserved

17.4.32 Receive HDSL Mapper Write

Writes the receive HDSL Mapper table to the device. This should be called only after the receive HDSL Mapper table has been completely filled in. The appropriate resets are issued.

C constant	_DSL_RH_MAPPER_WRITE
Opcode	0x37
Type	Control
Incoming Bytes	1
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	0	Set to 0 for future compatibility.

17.4.33 Clear ZipWire2 Error Counters

Clears the specified ZipWire2 Error counters to 0. This command can either clear all of the error counters or clear error counter blocks.

C constant	_DSL_CLEAR_ERROR_CTRS
Opcode	0x40
Type	Control
Incoming Bytes	1
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	Clear Error Counter Option	See Table 17-15.
NOTE(S): The Clear All option does not clear the system performance error counters.		

Table 17-15. Clear Error Counter Options

Option	Description	Parameter (C Constant)
Clear All	Clears all error counters.	0x00 (_CLEAR_ALL_COUNTERS)
Clear Operational	Clears the ZipWire2 operational error counter.	0x01 (_CLEAR_OPER_ERR_CTRS)
Clear HDSL Performance	Clears the ZipWire2 HDSL Performance error counter.	0x02 (_CLEAR_HDSL_ERR_CTRS)
Clear PCM Performance	Clears the ZipWire2 PCM Performance error counter.	0x03 (_CLEAR_PCM_ERR_CTRS)
Clear System Performance	Clears the ZipWire2 System Performance error counter. This is not cleared when the Clear All option is selected.	0x04 (_CLEAR_SYSTEM_ERR_CTRS)
Clear Error History	Clears the ZipWire2 Error History counters.	0x05 (_CLEAR_ERROR_HISTORY)

17.4.34 Read ZipWire2 Operational Error Counters

Queries the ZipWire2 Operational Error counters. These error counters are accumulated when the ZipWire2 device reaches normal operation or when the Clear Error Counter command was issued. The operational error counters are 1-byte wide.

C constant	_DSL_OPER_ERR_CTRS
Opcode	0x9C
Type	Status
Incoming Bytes	1
Outgoing Bytes	10

Incoming Data Parameter Description

Byte #	Content	Description
1	0	Set to 0 for future compatibility.

Outgoing Data Parameter Description

Byte #	Content	Description
1	RFIFO Full	8-bit value specifying the number of Rx FIFO Full errors.
2	RFIFO Empty	8-bit value specifying the number of Rx FIFO Empty errors.
3	RFIFO Slip	8-bit value specifying the number of Rx FIFO Slip errors.
4	TFIFO Full	8-bit value specifying the number of Tx FIFO Full errors.
5	TFIFO Empty	8-bit value specifying the number of Tx FIFO Empty errors.
6	TFIFO Slip	8-bit value specifying the number of Tx FIFO Slip errors.
7	Transmit Stuff	8-bit value specifying the number of Tx Stuff errors.
8	DPLL	8-bit value specifying the number of DPLL errors.
9	TFIFO Water Level	8-bit value specifying the number of Tx FIFO Water Level errors.
10	RFIFO Water Level	8-bit value specifying the number of Rx FIFO Water Level errors.

17.4.35 Read ZipWire2 HDSL Performance Error Counters

Queries the ZipWire2 HDSL Performance Error counters. These error counters are accumulated when the ZipWire2 device reaches normal operation or when the Clear Error Counter command was issued. The HDSL performance error counters are 2-bytes wide.

C constant	_DSL_HDSL_PERF_ERR_CTRS
Opcode	0x9E
Type	Status
Incoming Bytes	1
Outgoing Bytes	10

Incoming Data Parameter Description

Byte #	Content	Description
1	0	Set to 0 for future compatibility.

Outgoing Data Parameter Description

Byte #	Content	Description
1–2	Out-of-Sync (LOSW)	16-bit value specifying the number of Out-of-Sync errors. The low byte is sent first followed by the high byte.
3–4	SEGD	16-bit value specifying the number of SEGD errors. The low byte is sent first followed by the high byte.
5–6	CRC	16-bit value specifying the number of CRC errors. The low byte is sent first followed by the high byte.
7–8	SEGA	16-bit value specifying the number of SEGA errors. The low byte is sent first followed by the high byte.
9–10	LOSWT	16-bit value specifying the number of LOSWT errors. The low byte is sent first followed by the high byte.
NOTE(S): 16-bit value = (high byte << 8) + (low byte)		

17.4.36 Read ZipWire2 PCM Performance Error Counters

Queries the ZipWire2 PCM Performance Error counters. These error counters are accumulated when the ZipWire2 device reaches normal operation or when the Clear Error Counter command was issued. The PCM performance error counters are 2-bytes wide.

C constant	_DSL_PCM_PERF_ERR_CTRS
Opcode	0x9F
Type	Status
Incoming Bytes	1
Outgoing Bytes	8

Incoming Data Parameter Description

Byte #	Content	Description
1	0	Set to 0 for future compatibility.

Outgoing Data Parameter Description

Byte #	Content	Description
1–2	LCV	16-bit value specifying the number of Line Code Violations (LCV) errors. The low byte is sent first followed by the high byte.
3–4	LOS	16-bit value specifying the number of T1/E1 LOS errors. The low byte is sent first followed by the high byte.
5–6	LOSD	16-bit value specifying the number of LOSD errors. The LOSD errors are accumulated using the HDSL indictor bit. The low byte is sent first followed by the high byte.
7–8	CRC	16-bit value specifying the number of PCM CRC errors. The low byte is sent first followed by the high byte.
NOTE(S): 16-bit value = (high byte << 8) + (low byte)		

17.4.37 Read ZipWire2 System Performance Error Counters

Queries the ZipWire2 PCM Performance Error counters. These error counters are accumulated when the ZipWire2 device reaches normal operation or when the Clear Error Counter command was issued. The system performance error counters are 2-bytes wide.

C constant	_DSL_SYSTEM_PERF_ERR_CTRS
Opcode	0xA2
Type	Status
Incoming Bytes	1
Outgoing Bytes	6

Incoming Data Parameter Description

Byte #	Content	Description
1	0	Set to 0 for future compatibility.

Outgoing Data Parameter Description

Byte #	Content	Description
1–2	Startup Attempts	16-bit value specifying the number of startup attempts. The low byte is sent first followed by the high byte.
3–4	Startup Successful	16-bit value specifying the number of successful startup attempts. The low byte is sent first followed by the high byte.
5–6	Framer Interrupt Watch-Dog	16-bit value specifying the number of Framer Interrupt Watch-Dog errors. The low byte is sent first followed by the high byte.
NOTE(S): 16-bit value = (high byte << 8) + (low byte)		

17.4.38 Available Seconds and Total Seconds

Requests the available seconds and total seconds since power-on or reset.

Available Seconds refers to the total accumulated time the system is in normal operation (active transmit/receive state) and is passing data across the link. Total Seconds refers to the total time the system has been in operation. Error Seconds refer to the number of seconds that detected a CRC error. Error Seconds are only accumulated while the system is in normal operation.

C constant	_DSL_TIME
Opcode	0x9D
Type	Status
Incoming Bytes	1
Outgoing Bytes	12

Incoming Data Parameter Description

Byte #	Content	Description
1	0	Set to 0 for future compatibility.

Outgoing Data Parameter Description

Byte #	Content	Description
1–5	Available Seconds	32-bit value; low byte first, most significant byte last which corresponds to byte 0, byte 1, byte 2, and byte 3.
5–8	Total Seconds	32-bit value; low byte first, most significant byte last which corresponds to byte 0, byte 1, byte 2, and byte 3.
9–12	Error Seconds	32-bit value; low byte first, most significant byte last which corresponds to byte 0, byte 1, byte 2, and byte 3.
NOTE(S): 32-bit value = (byte 4 << 24) + (byte 3 << 16) + (byte 2 << 8) + (byte 1)		

17.4.39 Inject DSL CRC Error

Inject a CRC error in the next N number of DSL frames or continuously inject CRC in all frames. All six CRC bits are inverted based on the calculated CRC values. The user can issue the Inject CRC Error OFF option before all N frames are completed.

C constant	_DSL_INJECT_CRC_ERROR
Opcode	0x41
Type	Control
Incoming Bytes	1
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	Inject CRC Configuration	See Table 17-16

Table 17-16. Inject CRC Error Options

Option	Description	Parameter (C Constant)
Off	Normal CRC value (Default Value).	0x00 (_INJECT_CRC_OFF)
Continuous Error	Continuously inject CRC error in all DSL frames.	0xFF (_INJECT_CRC_CONT)
Inject N Errors	Inject CRC error in next N number of DSL frames. A value of 1 equals 1 frame.	1–254

17.4.40 Set CRC/FEBE Error History State

Enable or disable the CRC and FEBE Error History accumulation.

C constant	_DSL_CRC_FEBE_ERR_STATE
Opcode	0x42
Type	Control
Incoming Bytes	1
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	Error History Configuration	See bit-field description below

Bit 7:1	Bit 0
Reserved	CRC/FEBE State

CRC/FEBE State:
Enables or disables the CRC/FEBE error history.
0—Disabled
1—Enabled

17.4.41 Query CRC/FEBE History Interval In-Progress

Requests the current CRC and FEBE interval error counter for Interval 1, Interval 2, and Interval 3. Interval 1 records the number of CRC and FEBE errors occurring in 1-second intervals for a 15-minute time period and is updated every second. Interval 2 records the number of CRC and FEBE errors occurring in each 15-minute interval for a 24-hour time period and is updated every 15 minutes. Interval 3 records the number of CRC and FEBE errors occurring in each 24-hour interval for a 7-day period and is updated daily.

The Interval 1 records are 1-byte wide. The Interval 2 and Interval 3 records are 2-bytes wide.

C constant	_DSL_CRC_FEBE_IN_PROGRESS
Opcode	0x95
Type	Status
Incoming Bytes	1
Outgoing Bytes	10

Incoming Data Parameter Description

Byte #	Content	Description
1	0	Set to 0 for future compatibility.

Outgoing Data Parameter Description

Byte #	Content	Description
1	Current CRC at Interval 1	8-bit value specifying the CRC error at the Interval 1 counter.
2–3	Current CRC at Interval 2	16-bit value specifying the CRC error at the Interval 2 counter. The low byte is sent first followed by the high byte.
4–5	Current CRC at Interval 3	16-bit value specifying the CRC error at the Interval 3 counter. The low byte is sent first followed by the high byte.
6	Current FEBE at Interval 1	8-bit value specifying the FEBE error at the Interval 1 counter.
7–8	Current FEBE at Interval 2	16-bit value specifying the FEBE error at the Interval 2 counter. The low byte is sent first followed by the high byte.
9–10	Current FEBE at Interval 3	16-bit value specifying the FEBE error at the Interval 3 counter. The low byte is sent first followed by the high byte.
NOTE(S): 16-bit value = (high byte << 8) + (low byte).		

17.4.42 CRC Error History at Interval 1

Requests the CRC Error History at Interval 1. Interval 1 records the number of CRC errors occurring in 1-second intervals for a 15-minute time period and is updated every second. Each 1-second entry is 1-byte wide and can record up to 255 errors. The error counter will stop incrementing when it reaches the maximum of 255 errors. Record entry 0 corresponds to the previous second while entry 899 corresponds to 900 seconds prior.

Because the API protocol cannot transfer all 900 bytes at once, the complete error history must be downloaded in blocks of 50 bytes at a time. The incoming byte specifies which block to download

C constant	_DSL_CRC_ERR_INTERVAL 1
Opcode	0x96
Type	Status
Incoming Bytes	1
Outgoing Bytes	50

Incoming Data Parameter Description

Byte #	Content	Description
1	Which CRC Block	0 = First block, records 0–49 1 = Second block, records 50–99 N = Nth block, where N = 0–17 (18 blocks) which represents records (N × 50) to (N × 50 + 49)

Incoming Data Parameter Description

Byte #	Content	Description
1–50	CRC at Interval 1 Block	8-bit value specifying the CRC error at the Interval 1 counter.

17.4.43 CRC Error History at Interval 2

Requests the CRC Error History at Interval 2. Interval 2 records the number of CRC errors occurring in each 15-minute interval for a 24-hour time period and is updated every 15 minutes. Each 15-minute entry is 2-bytes wide and can record up to 65,535 errors. The error counter will stop incrementing when it reaches the maximum of 65,535 errors. Record entry 0 corresponds to the previous 15-minute interval while entry 95 corresponds to 96, 15-minute intervals prior.

Because the API protocol cannot transfer all 192 (96 × 2) bytes at once, the complete error history must be downloaded in blocks of 48 bytes at a time. The incoming byte specifies which block to download.

C constant	_DSL_CRC_ERR_INTERVAL 2
Opcode	0x97
Type	Status
Incoming Bytes	1
Outgoing Bytes	48

Incoming Data Parameter Description

Byte #	Content	Description
1	Which CRC Block	0 = First block, records 0–23 1 = Second block, records 24–47 2 = Third block, records 48–71 3 = Second block, records 72–95

Outgoing Data Parameter Description

Byte #	Content	Description
1–48	CRC at Interval 2 Block	16-bit value specifying the CRC Error at the Interval 2 counter. The low byte is sent first followed by the high byte. Because 2 bytes comprise one record entry, bytes 0–1 = Entry 0, bytes 2–3 = Entry 1, and so on.
NOTE(S): 16-bit value = (high byte << 8) + (low byte)		

17.4.44 CRC Error History at Interval 3

Requests the CRC Error History at Interval 3. Interval 3 records the number of CRC errors occurring in each 24-hour interval for a 7-day period and is updated daily. Each 1-day entry is 2-bytes wide and can record up to 65,535 errors. The error counters will stop incrementing when it reaches the maximum of 65,535 errors. Record entry 0 corresponds to the previous 1-day interval while entry 6 corresponds to 7, 1-day intervals prior.

C constant	_DSL_CRC_ERR_INTERVAL 3
Opcode	0x98
Type	Status
Incoming Bytes	1
Outgoing Bytes	14

Incoming Data Parameter Description

Byte #	Content	Description
1	0	Set to 0 for future compatibility.

Outgoing Data Parameter Description

Byte #	Content	Description
1–14	CRC at Interval 3 Block	16-bit value specifying the CRC Error at the Interval 3 counter. The low byte is sent first followed by the high byte. Because 2 bytes comprise one record entry, bytes 0–1 = Entry 0, bytes 2–3 = Entry 1, and so on.
NOTE(S): 16-bit value = (high byte << 8) + (low byte)		

17.4.45 FEBE Error History at Interval 1

Requests the FEBE Error History at Interval 1. Interval 1 records the number of FEBE errors occurring in 1-second intervals for a 15-minute time period and is updated every second. Each 1-second entry is 1-byte wide and can record up to 255 errors. The error counter will stop incrementing when it reaches the maximum of 255 errors. Record entry 0 corresponds to the previous second while entry 899 corresponds to 900 seconds prior.

Because the API protocol cannot transfer all 900 bytes at once, the complete error history must be downloaded in blocks of 50 bytes at a time. The incoming byte specifies which block to download

C constant	_DSL_FEBE_ERR_INTERVAL 1
Opcode	0x99
Type	Status
Incoming Bytes	1
Outgoing Bytes	50

Incoming Data Parameter Description

Byte #	Content	Description
1	Which FEBE Block	0 = First block, records 0–49 1 = Second block, records 50–99 N = Nth block, where N = 0–17 (18 blocks) which represents records (N × 50) to (N × 50 + 49)

Incoming Data Parameter Description

Byte #	Content	Description
1–50	FEBE at Interval 1 Block	8-bit value specifying the FEBE Error at the Interval 1 counter.

17.4.46 FEBE Error History at Interval 2

Requests the FEBE Error History at Interval 2. Interval 2 records the number of FEBE errors occurring in each 15-minute interval for a 24-hour time period and is updated every 15 minutes. Each 15-minute entry is 2-bytes wide and can record up to 65,535 errors. The error counter will stop incrementing when it reaches the maximum of 65,535 errors. Record entry 0 corresponds to the previous 15-minute interval while entry 95 corresponds to 96, 15-minute intervals prior.

Because the API protocol cannot transfer all 192 (96 × 2) bytes at once, the complete error history must be downloaded in blocks of 48 bytes at a time. The incoming byte specifies which block to download

C constant	_DSL_FEBE_ERR_INTERVAL 2
Opcode	0x9A
Type	Status
Incoming Bytes	1
Outgoing Bytes	48

Incoming Data Parameter Description

Byte #	Content	Description
1	Which FEBE Block	0 = First block, records 0–23 1 = Second block, records 24–47 2 = Third block, records 48–71 3 = Second block, records 72–95

Outgoing Data Parameter Description

Byte #	Content	Description
1–48	FEBE at Interval 2 Block	16-bit value specifying the CRC Error at the Interval 2 counter. The low byte is sent first followed by the high byte. Because 2 bytes comprise one record entry, bytes 0–1 = Entry 0, bytes 2–3 = Entry 1, and so on.
NOTE(S): 16-bit value = (high byte << 8) + (low byte)		

17.4.47 FEBE Error History at Interval 3

Requests the FEBE Error History at Interval 3. Interval 3 records the number of FEBE errors occurring in each 24-hour interval for a 7-day period and is updated daily. Each 1-day entry is 2-bytes wide and can record up to 65,535 errors. The error counters will stop incrementing when it reaches the maximum of 65,535 errors. Record entry 0 corresponds to the previous 1-day interval while entry 6 corresponds to the 7, 1-day intervals prior.

C constant	_DSL_FEBE_ERR_INTERVAL 3
Opcode	0x9B
Type	Status
Incoming Bytes	1
Outgoing Bytes	14

Incoming Data Parameter Description

Byte #	Content	Description
1	0	Set to 0 for future compatibility.

Outgoing Data Parameter Description

Byte #	Content	Description
1–14	FEBE at Interval 3 Block	16-bit value specifying the FEBE Error at the Interval 3 counter. The low byte is sent first followed by the high byte. Because 2 bytes comprise one record entry, bytes 0–1 = Entry 0, bytes 2–3 = Entry 1, and so on.
NOTE(S): 16-bit value = (high byte << 8) + (low byte).		

17.5 Level 3 API Commands

17.5.1 Signal Level Meter

Requests the level of the average signal level at the ADC input.

NOTE: The signal at the ADC input consists of a large transmitted echo component and a smaller far-end signal component. Thus, no cable attenuation data may be extracted from this information.

The input voltage is calculated using the following formula: (formula to be determined)

C constant	_DSL_SLM
Opcode	0x8E
Type	Status
Incoming Bytes	1
Outgoing Bytes	1

Incoming Data Parameter Description

Byte #	Content	Description
1	0	Set to 0 for future compatibility.

Outgoing Data Parameter Description

Byte #	Content	Description
1	Attenuation	1-byte unsigned integer X ($0 \leq X \leq 255$) 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.

17.5.2 Timing Recovery Offset

Requests the offset value of the Timing Recovery Control circuit. This value indicates the timing recovery frequency relative to its center frequency, which does not necessarily equal the nominal transmission frequency. On an HTU-C terminal, this response will always be zero since the control circuit is set to its nominal value. On an HTU-R terminal, this value gives an estimate of the frequency offset relative to the center frequency.

The relation of the given value to the frequency offset in Hz can be calculated using the formula: (formula to be determined)

C constant	_DSL_TIMING_RECOVERY
Opcode	0x84
Type	Status
Incoming Bytes	1
Outgoing Bytes	2

Incoming Data Parameter Description

Byte #	Content	Description
1	0	Set to 0 for future compatibility.

Outgoing Data Parameter Description

Byte #	Content	Description
1–2	Timing Recovery	2-byte signed integer X ($-32,768 \leq X \leq +32,767$ indicating the Timing Recovery control word. The low byte is sent first followed by the high byte.
NOTE(S): 16-bit value = (high byte << 8) + (low byte)		

17.5.3 Bit Pump Reverse Tip/Ring

Reverses the tip/ring polarity on the received signal in the DSP receiver. This does not reverse the tip/ring polarity on the transmitted signal. This command should be used only while operating in Framer Bypass mode because the ZipWire2 framer can automatically handle tip/ring reversal.

NOTE: This command is useful in applications where the external framer has the ability to detect tip/ring reversal but cannot correct the tip/ring reversal. Since this command only reverses the received signal, it is necessary to call this command on both the central and remote terminals when tip/ring reversal is detected.

C constant	_BP_REVERSE_TIP_RING
Opcode	0x14
Type	Control
Incoming Bytes	1
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	Tip/Ring Configuration	0 = Sets the Tip/Ring polarity on the received signal to normal (not-reversed). 1 = Reverses the Tip/Ring polarity on the received signal.

17.5.4 Scrambler/Descrambler Configuration

Configures the DSL scramblers and descramblers. See Section 16.2.1.

C constant	_DSL_SCR_DESCR_CONFIG
Opcode	0x04
Type	Control
Incoming Bytes	1
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	Scr/Descr Configuration	See bit-field description below. Default value = 0x22.

Bit 7:6	Bit 5:4	Bit 3:2	Bit 1:0
Tap Select	Rx Descrambler	Reserved	Tx Scrambler

Tap Select Configure the ZipWire2 Framer to use the same scrambler and descrambler tap in the Transmit and Receive HDSL Framer blocks. This is necessary in certain loopbacks where the DSL transmitted data is looped back into the DSL receiver without a separate scramble or descramble operation being performed on the data. However, the appropriate loopbacks will automatically take care of switching the scrambler and descrambler taps.

Value	Option	Description	C Constant
0x00	Normal	The Tx scrambler and Rx descrambler use the default (different) taps (Default Value).	—
0x01	Use Tap 5	Set both to use Tap 5.	—
0x02	Use Tap 18	Set both to use Tap 18.	—
0x03	Reserved	—	—

Transmit Scrambler Selects the internal Transmit Scrambler options. The ZipWire2 device has separate scramblers in the bit pump DSP and DSL Framer blocks. The ZipWire2 must have scrambled data in order for the transceiver to function properly. The scrambler function can be generated internally or must be provided by the external data source.

Value	Option	Description	C Constant
0x00	Bypass	Disables both the Bit Pump and DSL Framer scramblers. The external data source must provide the necessary scrambling. This mode should only be used in Framer Bypass mode. All data bits are transmitted unchanged.	_SCR_DESCR_BYPASS
0x01	Bit Pump Only	The Bit Pump scrambler is enabled and the DSL Framer scrambler is disabled. This mode should only be used in Framer Bypass mode. All data bits are scrambled.	_SCR_DESCR_BIT_PUMP
0x02	DSL Framer Only (Framed)	The DSL Framer scrambler is enabled and the Bit Pump scrambler is disabled. This mode should be used when the ZipWire2 framer is enabled. This mode provides the proper framed data structure that conforms to the HDSL standards (Default Value).	SCR_DESCR_FRAMED
0x03	Bit Pump and DSL Framer	Enables both the Bit Pump and DSL Framer scramblers. The mode is provided for debugging purposes only and should not be used in customer applications.	_SCR_DESCR_BOTH

Receive Descrambler Selects the internal Receive Descrambler options. The ZipWire2 device has separate descramblers in the bit pump DSP and DSL Framer blocks. The Receive Descrambler source should match the far-end's Transmit Scrambler source.

Value	Option	Description	C Constant
0x00	Bypass	Disables both the Bit Pump and DSL Framer descramblers. All received data bits are unchanged.	_SCR_DESCR_BYPASS
0x01	Bit Pump Only	The Bit Pump descrambler is enabled and the DSL Framer descrambler is disabled. This mode should only be used in Framer Bypass mode. All received data bits are descrambled.	_SCR_DESCR_BIT_PUMP
0x02	DSL Framer Only (Framed)	The DSL Framer descrambler is enabled and the Bit Pump descrambler is disabled. This mode should be used when the ZipWire2 framer is enabled. This mode receives the proper framed data structure that conforms to the HDSL standards (Default Value).	_SCR_DESCR_FRAMED
0x03	Bit Pump and DSL Framer	Enables both the Bit Pump and DSL Framer descramblers. This mode is provided for debugging purposes only and should not be used in customer applications.	_SCR_DESCR_BOTH

17.5.5 Write AFE Transmit Gain

Writes the AFE Line Driver Transmitter Gain register. The Transmitter Gain register is a 5-bit unsigned value; the upper 3 bits are ignored. The Tx Gain adjusts the nominal transmit power of the ZipWire2 device. The Tx Gain ranges from +1.6 dBm (0x00) to –1.6 dBm (0x1F). Each code range is 0.1 dBm steps.

C constant	_AFE_TX_GAIN
Opcode	0x13
Type	Control
Incoming Bytes	1
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	Tx Gain	5-bit unsigned integer value. The default is 0x10.

17.5.6 Read AFE Transmit Gain

Read the AFE Line Driver Transmitter Calibration/Gain value. The transmitter gain value is a 5-bit unsigned value. The upper 3-bits of this field are ignored.

C constant	_AFE_READ_TX
Opcode	0x91
Type	Status
Incoming Bytes	1
Outgoing Bytes	2

Incoming Data Parameter Description

Byte #	Content	Description
1	0	Set to 0 for future compatibility.

Outgoing Data Parameter Description

Byte #	Content	Description
1	Tx Calibration	5-Bit unsigned integer value indicating the transmit gain calibration value.
2	Tx Gain	5-Bit unsigned integer value indicating the transmit gain value.

17.5.7 DSL Framer Transmit Path Reset

Writing a 1 to a single bit in this command resets the operation of the corresponding module in the transmit path. After the reset operation is completed, the device will clear the bit; therefore, the read-back will read 0.

C constant	_DSL_FR_TX_RESET
Opcode	0x4E
Type	Control
Incoming Bytes	1
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	Transmit Mask	See bit-field description below.

Bit #	Block	Description
0	TP_PRBS_RST	Writing 1 to this bit will reset the transmit PCM PRBS process (towards HDSL).
1	TP_BER RST	Writing 1 to this bit will reset the transmit PCM BER Meter process (from PCM).
2	TX_WL_START	Writing 1 to this bit will start the process of transmit water level measurement.
3	TX_WL_RST	Writing 1 to this bit will initialize the transmit water level depth (due to TX_WL_VAL_IN).
4	TX_FIFO_RST	Writing 1 to this bit will initialize the transmit FIFO pointers.
5	TP_SD_RST	Writing 1 to this bit will initialize the transmit BSP Sync Detector process.
7:6	Reserved	—

17.5.8 DSL Framer Receive Path Reset

Writing a 1 to a single bit in this command resets the operation of the corresponding module in the receive path. After the reset operation is completed, the device will clear the bit; therefore, the read-back will read 0.

C constant	_DSL_FR_RX_RESET
Opcode	0x4F
Type	Control
Incoming Bytes	1
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	Receive Mask	See bit-field description below.

Bit #	Block	Description
0	RP_PRBS_RST	Writing 1 to this bit will reset the receive PCM PRBS process (towards PCM).
1	RP_BER RST	Writing 1 to this bit will reset the receive PCM BER Meter process (from HDSL).
2	RX_WL_START	Writing 1 to this bit will start the process of receive water level measurement.
3	RX_WL_RST	Writing 1 to this bit will initialize the receive water level depth (due to RX_WL_VAL_IN).
4	RX_FIFO_RST	Writing 1 to this bit will initialize the receive FIFO pointers.
5	RP_SD_RST	Writing 1 to this bit will initialize the receive BSP Sync Detector process.
6	DPLL_RST	Writing 1 to this bit will reset the DPLL state machine.
7	HSYNC_RST	Writing 1 to this bit will reset the HDSL SYNC Detector state machine.

17.5.9 DSL Framer—HDSL Configuration

Configure the DSL Framer HDSL block. This only applies when the DSL Framer is enabled.

C constant	_DSL_FR_HDSL_CONFIG
Opcode	0x11
Type	Control
Incoming Bytes	1
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	HDSL Configuration	See bit-field description below.

Bit 7:2	Bit 1	Bit 0
Reserved	HXP Polarity	AUX Mode

HXP Polarity In HDSL 2B1Q applications, the HXP_POL shall be set to 0 to be compliant (HXP = 0 is sign and HXP = 1 is magnitude).
0—Active low
1—Active high

AUX Mode Auxiliary Channel Mode (see Section 6.3.1.3).
0—THLOAD and RHMARK pins are active-high during the relevant data
1—THLOAD and RHMARK pins are clock-gated. Coincides with the relevant data

17.5.10 Mask Host Port Interrupt (INTR_HOST)

Masks the INTR_HOST pin on the Host Port Interface. When masked, the INTR_HOST will always be inactive (high) but the API Status Acknowledge and API results can still be polled.

C constant	_DSL_MASK_INTR_HOST
Opcode	0x52
Type	Control
Incoming Bytes	1
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	Host Port Mask	See bit-field description below. TBD

Bit 7:3	Bit 2	Bit 1	Bit 0
Reserved	System Status	EOC Status	API Result

17.5.11 DSL Framer Auto Water Level

Configures the DSL Framer Auto Water Level.

C constant	_DSL_AUTO_WATER_LEVEL
Opcode	0x2B
Type	Control
Incoming Bytes	1
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	Water Level Configuration	See bit-field description below.

Bit 7:2	Bit 1	Bit 0
Reserved	AUTO_RX_WL	AUTO_TX_WL

AUTO_RX_WL Enables or disables the Auto Receive Water Level.
0—Disable; use DSL Framer Rx Water Level to set desired water level
1—Enable (default)

AUTO_TX_WL Enables or disables the Auto Transmit Water Level.
0—Disable; use DSL Framer Tx Water Level to set desired water level
1—Enable (default)

17.5.12 DSL Framer Transmit Water Level

Sets the DSL Framer Transmit Water Level when the Auto Water Level is disabled. The transmit water level is a 10-bit value with a value range of 1–1,024 bits where a 0 corresponds to 1 bit.

C constant	_DSL_TX_WATER_LEVEL
Opcode	0x2C
Type	Control
Incoming Bytes	2
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	Water Level (Low Byte)	Contains the lower 8 bits of the water level parameter.
2	Water Level (High Byte)	Contains the upper 2 bits of the water level parameter.

17.5.13 DSL Framer Receive Water Level

Sets the DSL Framer Receive Water Level when the Auto Water Level is disabled. The receive water level is a 10-bit value with a value range of 1–1024 bits where a 0 corresponds to 1 bit.

C constant	_DSL_RX_WATER_LEVEL
Opcode	0x2D
Type	Control
Incoming Bytes	2
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	Water Level (Low Byte)	Contains the lower 8 bits of the water level parameter.
2	Water Level (High Byte)	Contains the upper 2 bits of the water level parameter.

17.5.14 DSL Framer DPLL Clock Generator

When the DSL Framer DPLL is in NCO Open mode, the DPLL Clock output (signal RPDPLLCLK) can be used as a clock generator. Based on the desired output clock frequency, use the formula to calculate the integer and fractional parts of the DPLL configuration register:

$$INTEGER.FRACTION = \left(\frac{Frefclk \times LPLL\_MUL}{Fclkout}\right)$$

where:

Fclkout	*= Desired Clock Frequency Output*
Frefclk	*= Input Reference Clock, typically 19.6608 MHz*
LPLL_MUL	*= LPLL multiplication factor, typically 5 for Frefclk = 19.6608 MHz*
INTEGER	*= Integer part of result*
FRACTION	*= Fractional part of result*

Frefclk × PLL_MUL creates the internal HFCLK clock. The HFCLK frequency should be 100–130 MHz. Only the INTEGER (2 bytes), FRACTION (2 bytes), and LPLL_MUL (1 byte) require programming. The API commands are then programmed as follows:

_DSL_DPLL_CLK_LPLL_MUL = LPLL_MUL – 1
_DSL_DPLL_CLK_INT = INTEGER – 1
_DSL_DPLL_CLK_FRAC = round(FRACTION × 65,535)

Example:
Fclkout = 2.048 MHz
Frefclk = 22.1184 MHz
LPLL_MUL = 5

$$INTEGER.FRACTION = \left(\frac{(22.1184 \times 10^6) \times 5}{2.048 \times 10^6}\right)$$

$$INTEGER.FRACTION = 54.5325$$

_DSL_DPLL_CLK_LPLL_MUL = 5 – 1 = 4
_DSL_DPLL_CLK_INT = 54 – 1 = 53
_DSL_DPLL_CLK_FRAC = round(0.5325 × 65,535) = 34,897 = 0x8,851

C constant	_DSL_DPLL_CLOCK_GEN
Opcode	0x58
Type	Control
Incoming Bytes	5
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	LPLL Multiplication Factor	1-byte value specifying the LPLL Multiplication Factor. Program this value to one less than the desired multiplication factor.
2–3	Integer Part	2-byte value specifying the integer part. Program this to one less than the desired integer value. Low byte is programmed first.
4–5	Fractional Part	2-byte value specifying the fractional part. The low byte is programmed first.

17.6 Read/Write Register Commands

The Read/Write Register commands are per ZipWire2 device and thus require the _ZIP_WIRE0 to _ZIP_WIRE7 (0x00–0x07) destination (see Table 15-1).

There are 2 sets of Read/Write Register commands:

- Internal 8051 Addressable Data Space (see Section 4.1)—The data space range is a 16-bit address field, 0x0000–0xFFFF (0–65,535).
- ZipWire2 AFE Data Space—The AFE is accessed via indirect registers through the DSP. The AFE data space range is a 7-bit address field, 0x00–0x7F (0–127).

NOTE: The ZipWire2 Register map is not provided. These commands are primarily provided for internal development and characterization. However, to isolate a problem, Conexant engineering may at times ask the customer to provide an internal register dump.

17.6.1 Write Register

This command writes the specified block of data to the specified address.

C constant	_DSL_WRITE_REG
Opcode	0x75
Type	Control
Incoming Bytes	3 + block size (length)
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1–2	Address	2-byte value specifying the address. Low byte is programmed first.
3	Length (L)	1-byte specifying the block size up to 64 bytes. A length of 0 corresponds to 1 byte and a 63 corresponds to 64 bytes.
4 + L	Data	Block of data. The first byte is written to the specified address, the second byte to the address + 1, and so on.
NOTE(S): Address = (high << 8) + low.		

17.6.2 Read Register

This command reads the specified block of data from the specified address.

C constant	_DSL_READ_REG
Opcode	0xA0
Type	Control
Incoming Bytes	3
Outgoing Bytes	Block size (length)

Incoming Data Parameter Description

Byte #	Content	Description
1–2	Address	2-byte value specifying the address. Low byte is programmed first.
3	Length (L)	1-byte specifying the block size up to 64 bytes. A length of 0 corresponds to 1 byte and a 63 corresponds to 64 bytes.
NOTE(S): Address = (high << 8) + low.		

Outgoing Data Parameter Description

Byte #	Content	Description
1–L	Data	Block of data. The first byte corresponds to the specified address, the second byte to the address + 1, and so on.

17.6.3 Write AFE Register

This command writes the specified block of data to the specified address of the AFE device.

C constant	_DSL_WRITE_AFE
Opcode	0x76
Type	Control
Incoming Bytes	2 + block size (length)
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	Address	1-byte value specifying the address. The AFE only has 128 registers.
2	Length (L)	1-byte specifying the block size up to 64 bytes. A length of 0 corresponds to 1 byte and a 63 corresponds to 64 bytes.
4 + L	Data	Block of data. The first byte is written to the specified address, the second byte to the address + 1, and so on.

17.6.4 Read AFE Register

This command reads the specified block of data from the specified address of the AFE device.

C constant	_DSL_READ_AFE
Opcode	0xA1
Type	Control
Incoming Bytes	2
Outgoing Bytes	Block size (length)

Incoming Data Parameter Description

Byte #	Content	Description
1	Address	1-byte value specifying the address. The AFE only has 128 registers.
2	Length (L)	1-byte specifying the block size up to 64 bytes. A length of 0 corresponds to 1 byte and a 63 corresponds to 64 bytes.

Outgoing Data Parameter Description

Byte #	Content	Description
1–L	Data	Block of data. The first byte is corresponds to the specified address, the second byte to the address + 1, and so on.

17.7 T1/E1 Framer Commands

All T1/E1 Framer Commands require the _T1E1_FRAMER (0x09) destination (see Table 15-1).

17.7.1 T1/E1 Framer Configure

Configures the T1/E1 Framer PCM mode; all registers are programmed. When the T1/E1 framer is present, the PCM mode is configured based on the DSL Configuration API command.

NOTE: The Receive Termination pins in the Miscellaneous Output API command are set accordingly.

C constant	_T1E1_CONFIGURE
Opcode	0x01
Type	Control
Incoming Bytes	1
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	T1/E1 Configuration	0 = E1 Mode 1 = T1 Mode

17.7.2 T1/E1 Framer Frame Format

Configures the T1/E1 Framer frame format mode.

C constant	_T1E1_FRAME_FORMAT
Opcode	0x02
Type	Control
Incoming Bytes	1
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	Frame Format	0 = Framed Mode (Default Value) 1 = Unframed Mode

17.7.3 T1/E1 Framer Transmit AIS

Configures the T1/E1 Framer to transmit an AIS (all-1s) pattern.

C constant	_T1E1_TRANSMIT_AIS
Opcode	0x03
Type	Control
Incoming Bytes	1
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	Transmit AIS	0 = Disabled, transmit normal payload data (Default Value) 1 = Enabled, transmit AIS pattern

17.7.4 T1/E1 Framer Output Mode

Configures the T1/E1 Framer output mode. The output mode allows the T1/E1 framer outputs to be three-stated so the ZipWire2 PCM bus can be connected to a different interface (i.e., V.35).

C constant	_T1E1_OUTPUT_MODE
Opcode	0x03
Type	Control
Incoming Bytes	1
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	Output Mode	0 = Output enabled (Default Value) 1 = Output disabled

17.7.5 T1/E1 Framer Receive Termination

Configures the T1/E1 Framer Receive Termination. The default value is based on the T1/E1 configuration.

C constant	_T1E1_RX_TERM
Opcode	0x05
Type	Control
Incoming Bytes	1
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	Receive Termination	0 = Set to 120 Ω termination, used in E1 (twisted pair) applications. 1 = Set to 100 Ω termination, used in T1 applications (default for T1). 2 = Set to 75 Ω termination, used in E1 (BNC) applications (default for E1).

17.7.6 T1/E1 Framer Loopbacks

Operates the T1/E1 framer in a loopback mode (refer to Figure 11-2). To turn off any of the loopback modes, use the loopback command with the _EXIT_LOOPBACK (0 value) parameter.

The framer loopbacks can be issued (or exited) at any time during normal operation without effecting the ZipWire2 link. Only the throughput data will be effected to match the desired loopback condition.

All loopbacks can be issued while in the Out-of-Service (IDLE) state to facilitate development and debugging of other devices in the system.

C constant	_T1E1_LOOPBACK
Opcode	0x09
Type	Control
Incoming Bytes	1
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	Loopback Option	See Table 17-17.

Table 17-17. T1/E1 Framer Loopback Options

Option	Description	Parameter (C Constant)
Exit Loopback	Cancel current loopback (default value).	0x00 (_EXIT_LOOPBACK)
PCM Local	The T1/E1 PCM transmit clock, data, and sync inputs are internally looped back to the T1/E1 PCM receive clock, data, and sync outputs.	0x00 (_T1E1_PCM_LOCAL)
PCM Line	The T1/E1 PCM receive clock, data, and sync outputs are internally looped back to the T1/E1 PCM transmit clock, data, and sync inputs.	0x01 (_T1E1_PCM_LINE)
LIU Local	The T1/E1 analog transmit signal is internally looped back to the T1/E1 analog receive path.	0x02 (_T1E1_LIU_LOCAL)
LIU Line	The T1/E1 analog receive signal is internally looped back to the T1/E1 analog transmit path.	0x03 (_T1E1_LIU_LINE)

17.7.7 T1/E1 Read Framer Control Commands

The Read Framer Control Commands API command is a read-back of the current setting for each of T1/E1 Framer API control commands.

C constant	_T1E1_READ_CONTROL
Opcode	0x80
Type	Status
Incoming Bytes	1
Outgoing Bytes	Depends on control command

Incoming Data Parameter Description

Byte #	Content	Description
1	Control Command Opcode	Return the configuration for the specified control command. Input range is 0–127 (0x7F).

Outgoing Data Parameter Description

Byte #	Content	Description
N	—	Refer to the specific control command.

17.7.8 T1/E1 Framer Versions

Requests the T1/E1 Framer hardware and silicon revision numbers.

C constant	_T1E1_VERSIONS
Opcode	0x8A
Type	Status
Incoming Bytes	1
Outgoing Bytes	2

Incoming Data Parameter Description

Byte #	Content	Description
1	0	Set to 0 for future compatibility.

Outgoing Data Parameter Description

Byte #	Content	Description
1	Framer Type	1-byte unsigned integer field returning the T1/E1 framer silicon type
2	Framer Revision	1-byte unsigned integer field returning the T1/E1 silicon revision

NOTE: The T1/E1 framer type and revision matrix is dependent on which versions of the T1/E1 Framer the ZipWire2 EVMs can support.

Table 17-18. Framer Silicon Types

Byte Value	Type
0	Bt8370 Detected
3	Bt8373 Detected
5	Bt8375 Detected
6	Bt8376 Detected
8	Bt8398 Detected

17.8 EVM Specific Commands

All EVM commands require the _EVM (0x08) destination (see Table 15-1).

17.8.1 EVM Set LED Bank

Sets the LED Bank value. Should only be issued when LED Update is disabled.

NOTE: The LED Bank 2 and Miscellaneous Output register are combined to generate an 8-bit field. However, the current register value is stored internally in the software so that writing to either the LED Bank 2 or the Miscellaneous Output Register will not corrupt the other nibble. In simpler terms, writing to the LED Bank 2 register will not corrupt the Miscellaneous Output register.

C constant	_EVM_SET_LED
Opcode	0x01
Type	Control
Incoming Bytes	2
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	LED Bank 1	1-byte value (see Table 9-3)
2	LED Bank 2	1-byte value (see Table 9-4)

17.8.2 EVM Set Miscellaneous Output

Sets the Miscellaneous Output Port value. The Miscellaneous Output bits can be set regardless of the LED Update State.

The Receive Termination pins in the Miscellaneous Output API command are set by the DSL Configuration Mode API command (Section 17.2.1).

NOTE: The LED Bank 2 and Miscellaneous Output register are combined to generate an 8-bit field. However, the current register value is stored internally in the software so that writing to either the LED Bank 2 or the Miscellaneous Output register will not corrupt the other nibble. In simpler terms, writing to the Miscellaneous Output register will not corrupt the LED Bank 2 register.

C constant	_EVM_SET_MISC_OUT
Opcode	0x02
Type	Control
Incoming Bytes	1
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	Miscellaneous Output	1-byte value (see Table 9-4)

17.8.3 EVM LED Update State

Sets the LED Update State. The Disable option prevents the internal 8051 from updating the LEDs. This allows the external Host Processor to control the LEDs. The option is used in EVM production testing.

C constant	_EVM_LED_UPDATE
Opcode	0x03
Type	Control
Incoming Bytes	1
Outgoing Bytes	None

Incoming Data Parameter Description

Byte #	Content	Description
1	LED Update	0 = Disabled, the 8051 does not update the LEDs 1 = Enabled, the 8051 updates the LEDs

17.8.4 EVM LED and DIP Switch Status

Requests the current LED and DIP Switch settings. The LED query provides a read-back of what software set the LEDs. Reading the LED state is not possible.

C constant	_EVM_LED_DIP_SW_STATUS
Opcode	0x81
Type	Status
Incoming Bytes	1
Outgoing Bytes	6

Incoming Data Parameter Description

Byte #	Content	Description
1	0	Set to 0 for future compatibility.

Outgoing Data Parameter Description

Byte #	Content	Description
1	LED 1	1-byte unsigned integer field returning the LED 1.
2	LED 2	1-byte unsigned integer field returning the LED 2. Only the lower 4 bits are valid.
3	Misc. Output	1-byte unsigned integer field returning the miscellaneous output register. Only the lower 4 bits are valid.
4	LED Update State	0 = Disabled 1 = Enabled
5	DIP Switch 3	1-byte unsigned integer field returning the DIP Switch 3 setting.
6	DIP Switch 4	1-byte unsigned integer field returning the DIP Switch 4 setting.

18.0 Electrical and Mechanical Specifications

The following specifications apply to both the ZipWire2 Transceiver/Framer and the ZipWire2 AFE.

18.1 Specifications for the ZipWire2 Transceiver/Framer and ZipWire2 AFE

18.1.1 Recommended Operating Conditions

Table 18-1. Recommended Operating Conditions

Parameter	Symbol	Min	Typ	Max	Units
VAA (5.0)—Analog Supply	VAA	4.75	5.0	5.25	V
VIO (3.3 V)—Input/Output	VIO	3.00	3.3	3.60	V
VDD (2.5 V)—Digital Core	VDD	2.3	2.5	2.75	V
NOTE(S): AGND = IOGND = DGND = 0 V; other voltages with respect to 0 V.					

18.1.2 Absolute Maximum Ratings

Table 18-2. Absolute Maximum Ratings

Parameter	Symbol	Min	Typ	Max	Units
VAA	VAA	–0.3	5.0	6.0	V
VIO	VIO	–0.3	3.3	4.6	V
VDD	VDD	–0.3	2.5	3.3	V
Voltage on any Signal Pin	—	GND–0.3	—	VIO + 0.3 V	—
Input Current, any pin except supplies	IMAX	—	—	±10	mA
Analog Input Voltage	—	–0.3	—	VAA + 0.3	V
Digital Input Voltage	—	–0.3	—	VIO + 0.3	V
Ambient Operating Temperature	T_A	–40	25	+85	^oC
Junction Temperature	T_J	—	—	125	^oC
Storage Temperature (ambient)	T_{SA}	–65	—	150	^oC
Soldering Temperature	TSOL	–65	—	260	^oC
Vapor Phase Soldering	TVSOL	–65	—	220	^oC
Air Flow	0	—	—	—	L.F.P.M.

CAUTION

NOTE(S): Operation beyond these limits may cause permanent damage to the device. Normal operation is not guaranteed at these extreme conditions.

18.2 Thermal Characteristics

18.2.1 ZipWire2 AFE

For the 80-pin TQFP (AFE) with .2 m/s of airflow, θ_{JA}~ 23 °C/W.

18.2.2 ZipWire2 Transceiver

For the 15 x 15 mm CABGA (DSP) with .2 m/s of airflow, θ_{JA}~ 40 °C/W.

18.2.3 ZipWire2 Framer

For the 9 x 9 mm CABGA (Framer) with .2 m/s of airflow, θ_{JA}~ 50 °C/W.

18.2.4 ZipWire2 Transceiver/Framer

For the 27 x 27 mm BGA (Transceiver/Framer) with .2 m/s of airflow, θ_{JA}~ 26 °C/W.

18.3 Specifications for ZipWire2 Transceiver/Framer Only

18.3.1 Power Dissipation

Table 18-3 shows the breakdown for the ZipWire2 Transceiver/Framer power dissipation.

Table 18-3. ZipWire2 Transceiver/Framer Power Dissipation

Parameter	Symbol	Min	Typ	Max	Units
DSP, +2.5 V	$PD_{DSP\ 2.5}$	—	900	—	mW
DSP, +3.3 V	$PD_{DSP\ 3.3}$	—	115	—	mW
FRAMER, +3.3 V	PD_{FRAMER}	—	53	—	mW

18.3.2 DC Characteristics

Table 18-4 lists transceiver DC characteristics.

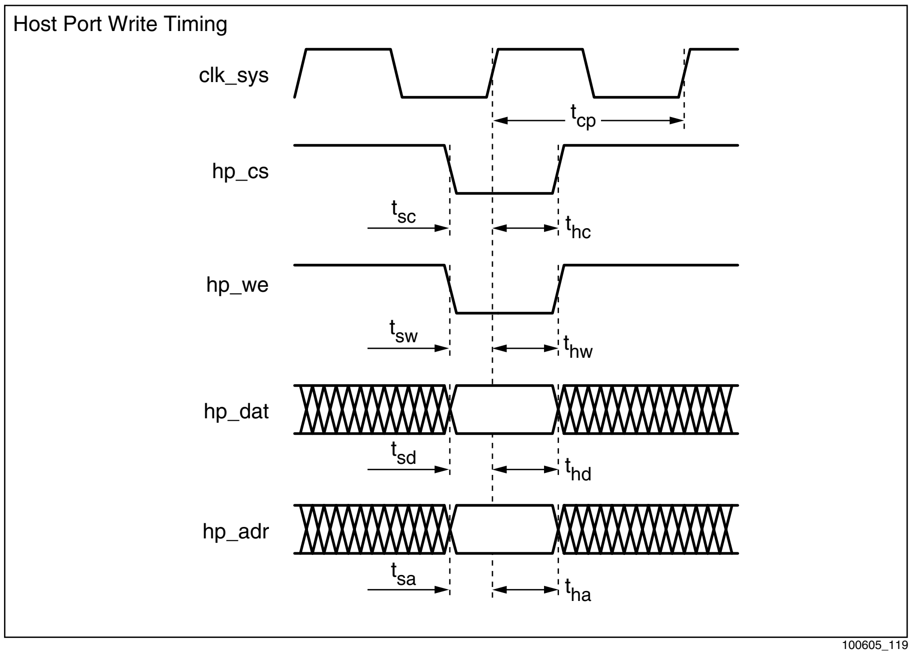
Table 18-4. Transceiver/Framer DC Characteristics

Parameter	Symbol	Min	Typ	Max	Units
Digital Inputs					
Input High Voltage	VIH	2.0	—	5.25	V
Input Low Voltage	VIL	0	—	0.8	V
Input Leakage Current	IIL/IIH	—	—	—	µA
Input Capacitance	CIN	—	—	—	pF
Digital Outputs					
Output High Voltage	VOH	2.4	—	—	V
Output Low Voltage	VOL	—	—	0.4	V
Three-State Output Leakage	ILK	—	—	—	µA
Output Capacitance	C_{OUT}	—	—	—	pF
Digital Bi-directionals					
Three-State Output Leakage	ILK	—	—	—	µA
Input/Output Capacitance	CINOUT	—	—	—	pF

18.3.3 Host Port RAM Interface Timing

Figure 18-1 illustrates the Host Port RAM interface timing. Table 18-5 lists the Host Port RAM interface timing.

Figure 18-1. Host Port Interface Timing Diagrams



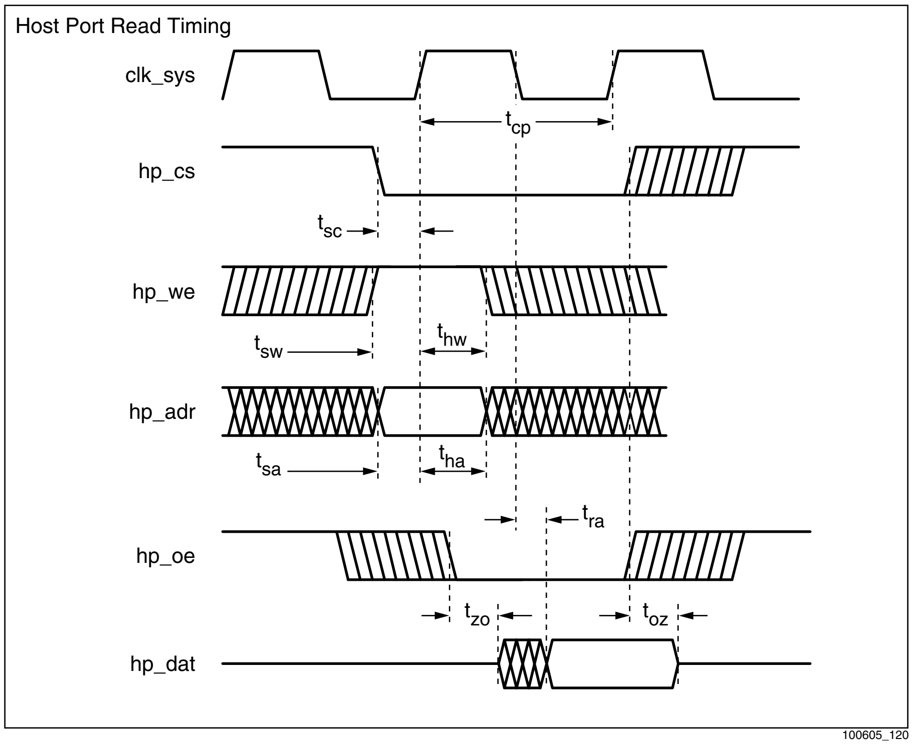


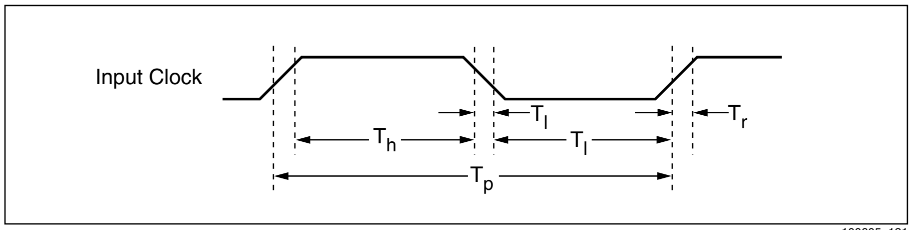
Table 18-5. Host Port Ram Interface Timing Table

Symbol	Parameter	Minimum	Maximum	Units
t_{cp}	System clock period	?	?	ns
t_{sc}	Chip select setup to clk_bp_o	2	—	ns
t_{hc}	Chip select hold after clk_bp_o	4	—	ns
t_{sw}	Write enable setup to clk_bp_o	2	—	ns
t_{hw}	Write enable hold after clk_bp_o	4	—	ns
t_{sd}	Data setup to clk_bp_o	2	—	ns
t_{hd}	Data hold after clk_bp_o	4	—	ns
t_{sa}	Address setup to clk_bp_o	2	—	ns
t_{ha}	Address hold after clk_bp_o	4	—	ns
t_{ra}	Read Access Time	—	8	ns
t_{zo}	Output enable to data driven	—	5	ns
t_{oz}	Output enable to data tristate	6	—	ns

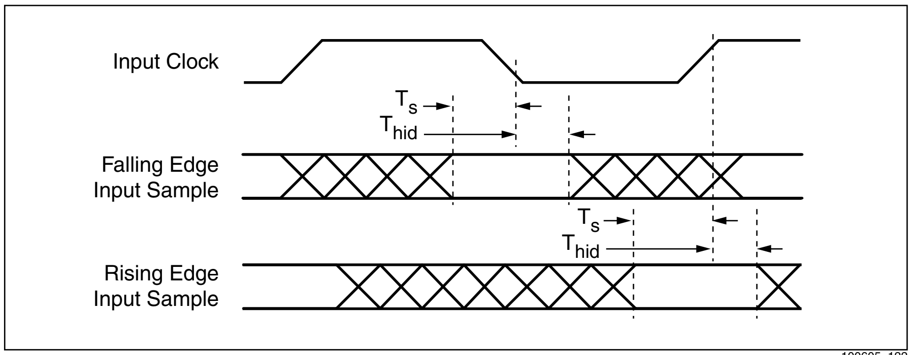
18.3.4 Master Bus Interface Timing

TBD

18.3.5 DSL Framer Timing Requirements

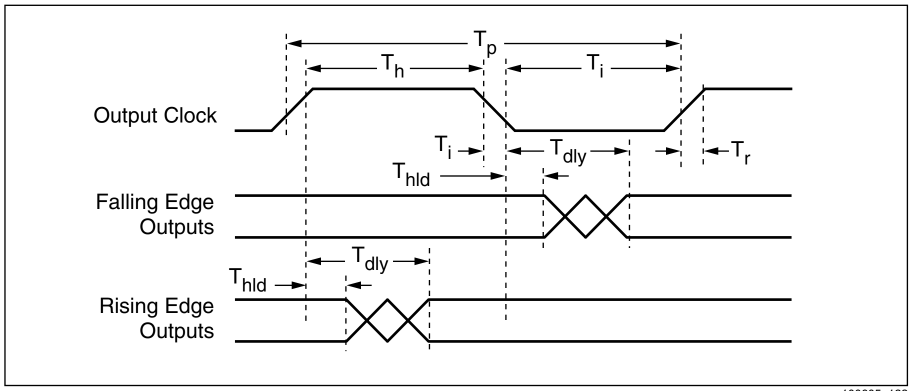


Symbol	Parameter	Minimum	Maximum	Units
—	TPCLK, PEXTCLK	—	—	—
1/Tp	Frequency	0.064	8.196	MHz
Th	Clock Width High	$0.4 \times$ Tp	$0.6 \times$ Tp	ns
T1	Clock Width Low	$0.4 \times$ Tp	$0.6 \times$ Tp	ns
Tr	Clock Rise Time	—	20	ns
Tf	Clock Fall Time	—	20	ns



Symbol	Parameter	Minimum	Maximum	Units
—	TPDAT, TPINSDAT, TPMFSYNC	—	—	—
Ts	Input Setup Time	35	—	ns
Thld	Input Hold Time	10	—	ns

18.3.6 DSL Framer Switching Characteristics



Symbol	Parameter	Minimum	Maximum	Units
—	RPCLK, RPDPLLCLK	—	—	—
1/Tp	Frequency	0.064	8.196	MHz
Th	Clock Width High	Tp/2 – 20	Tp/2 + 20	ns
T1	Clock Width Low	Tp/2 – 20	Tp/2 + 20	ns
Tr	Clock Rise Time	—	15	ns
Tf	Clock Fall Time	—	15	ns
—	RPDAT, RPDROP, RPMFSYNC, TPMFSYNC	—	—	—
Thld	Output Data Hold	0	—	ns
Tdly	Output Data Delay	—	25	ns

18.4 Specifications for ZipWire2 AFE Only

The following specifications apply only to the ZipWire2 AFE.

18.4.1 Power Dissipation

Table 18-6 shows the breakdown for the ZipWire2 AFE power consumption for 1,552 kbps OPTIS and 2,320 kbps G.shdsl Symmetric Mask mode. Power consumption includes the power delivered to the line, ~100 mW.

The power dissipation for other data rates is: TBD.

Table 18-6. ZipWire2 AFE Power Dissipation

Parameter	Condition	Symbol	Min	Typ	Max	Units
AFE, +5.0 V	OPTIS	PD_{AFE} 5.0	—	1,000	—	mW
	G.shdsl		—	700	—	mW
AFE, +3.3 V	—	PD_{AFE} 3.3	—	129	—	mW

18.4.2 DC Characteristics

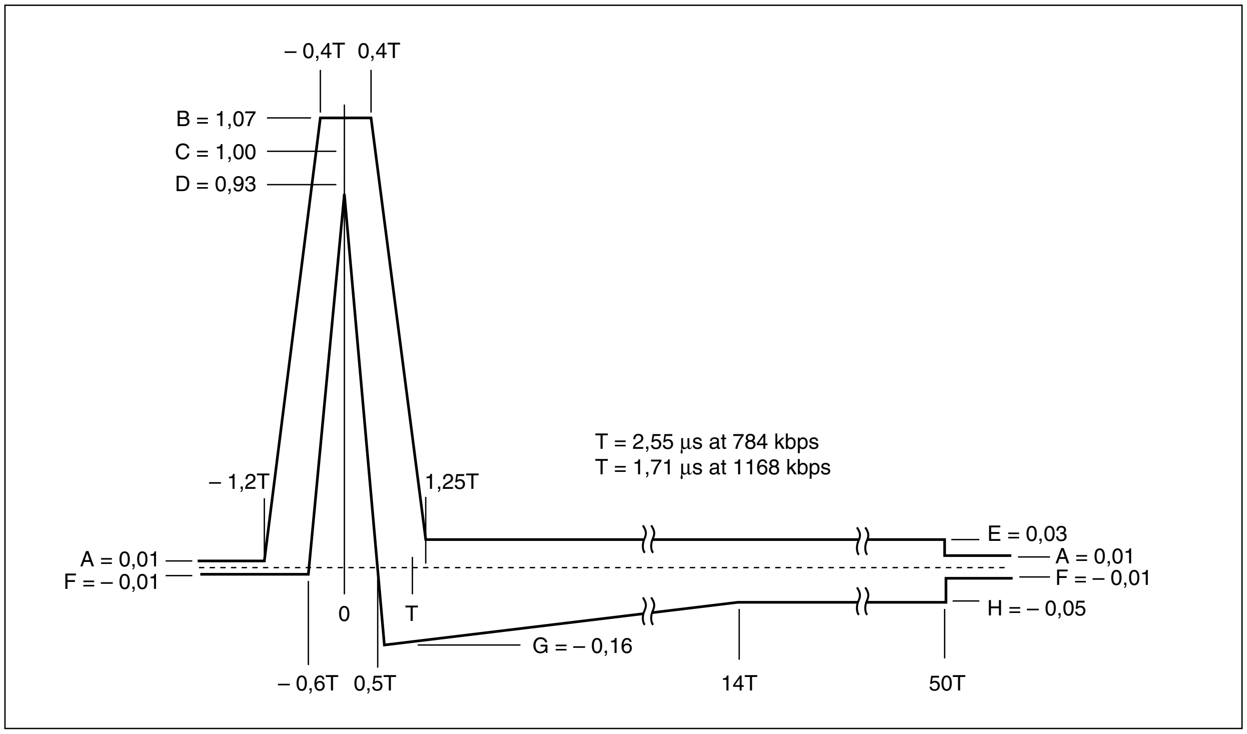
Table 18-7. AFE DC Characteristics

Parameter	Symbol	Min	Typ	Max	Units
Digital Inputs					
Input High Voltage	VIH	0.8 VIO	—	5.5	V
Input Low Voltage	VIL	GND	—	0.1 VDD	V
Input Leakage Current	IIL/IIH	–10	—	10	µA
Input Capacitance	CIN	—	2.9	—	pF
Digital Outputs					
Output High Voltage	VOH	0.9 VIO	—	VIO	V
Output Low Voltage	VOL	GND	—	0.1 VIO	V
Three-State Output Leakage	ILK	10	—	10	µA
Output Capacitance	CIN	—	3.1	—	pF
Digital Bidirectionals					
Three-State Output Leakage	ILK	–10	—	10	µA
Input/Output Capacitance	CINOUT	—	3.0	—	pF

18.4.3 PSD Specifications

18.4.3.1 2B1Q PSD

Figure 18-2. Transmit Pulse Template for Two- and Three-Pair Systems; Normalized Pulse Mask (Source ETSI TS 101 135, Formerly ETR 152)



100605_132

Table 18-8. Transmit Pulse Template for Two- and Three-Pair Systems (Source ETSI TS 101 135, Formerly ETR 152)

Normalized Level		Quaternary Symbols			
		+ 3	+ 1	– 1	– 3
A	0.01	0.0264	0.0088	–0.0088	–0.0264
B	1.07	2.8248	0.9416	–0.9416	–2.8248
C	1.00	2.6400	0.8800	–0.8800	–2.6400
D	0.93	2.4552	0.8184	–0.8184	–2.4552
E	0.03	0.0792	0.0264	–0.0264	–0.0792
F	–0.01	–0.0264	–0.0088	0.0088	0.0264
G	–0.16	–0.4224	–0.1408	0.1408	0.4224
H	–0.05	–0.1320	–0.0440	0.0440	0.1320

Figure 18-3. Transmit Pulse Template for One-Pair Systems (Source ETSI TS 101 135, Formerly ETR 152)

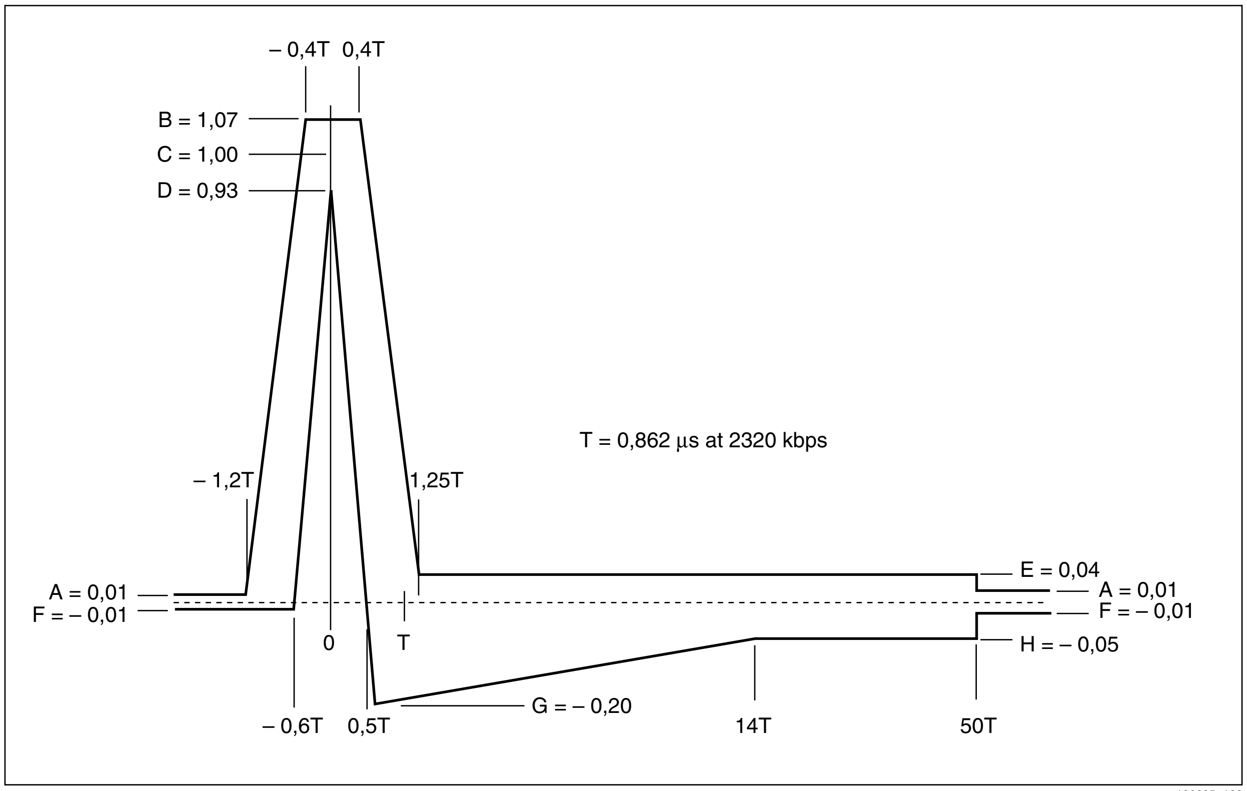


Table 18-9. Transmit Pulse Template for One-Pair Systems (Source ETSI TS 101 135, Formerly ETR 152)

Normalized Level		Quaternary Symbols			
		+ 3	**+ 1**	**– 1**	**– 3**
A	0.01	0.0250 V	0.0083 V	–0.0083 V	–0.0250 V
B	1.07	2.6750 V	0.8917 V	–0.8917 V	–2.6750 V
C	1.00	2.500 V	0.8333 V	–0.8333 V	–2.5000 V
D	0.93	2.3250 V	0.7750 V	–0.7750 V	–2.3250 V
E	0.04	0.1000 V	0.0333 V	–0.0333 V	–0.1000 V
F	–0.01	–0.0250 V	–0.0083 V	0.0083 V	0.0250 V
G	–0.20	–0.5000 V	–0.1667 V	0.1667 V	0.5000 V
H	–0.05	–0.1250 V	–0.0417 V	0.0417 V	0.1250 V

18.4.3.2 OPTIS

TBD

18.4.3.3 G.shdsl

TBD

18.4.4 Pulse Template Specifications

These specifications are only required for the HDSL1 (2B1Q).

Figure 18-4. Upper Bound of the Average PSD of a 392 kbaud System (Source ETSI TS 101 135, Formerly ETR 152)

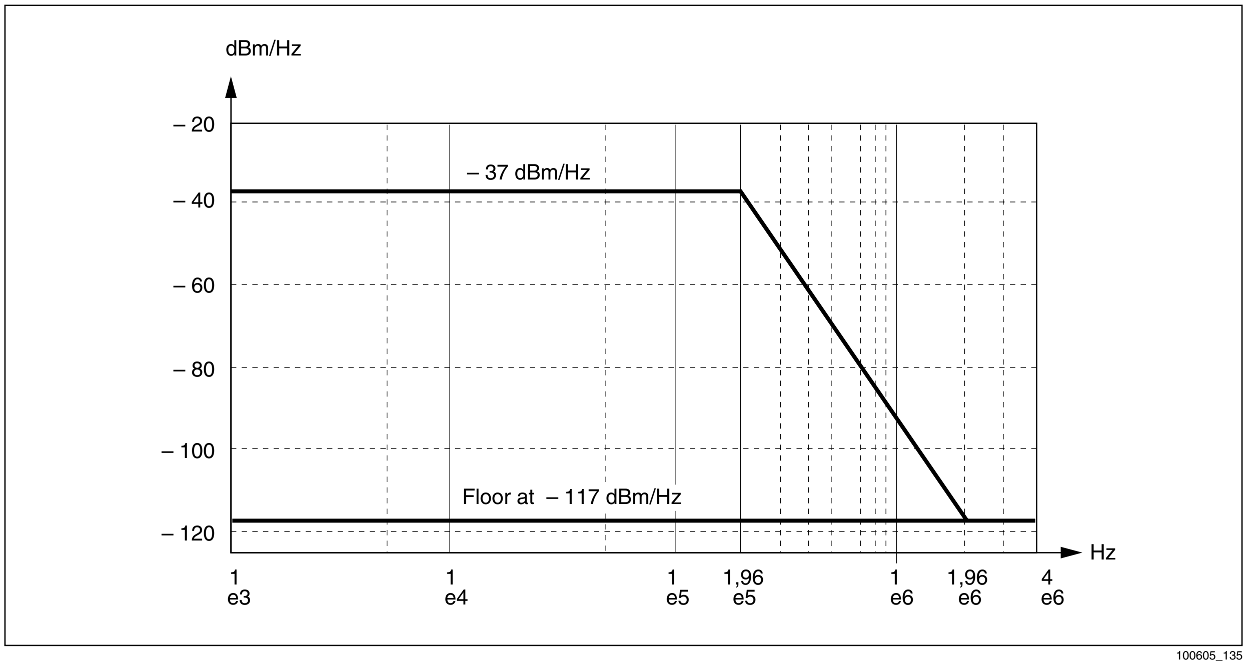


Figure 18-5. Upper Bound of the Average PSD of a 584 kbaud System (Source ETSI TS 101 135, Formerly ETR 152)

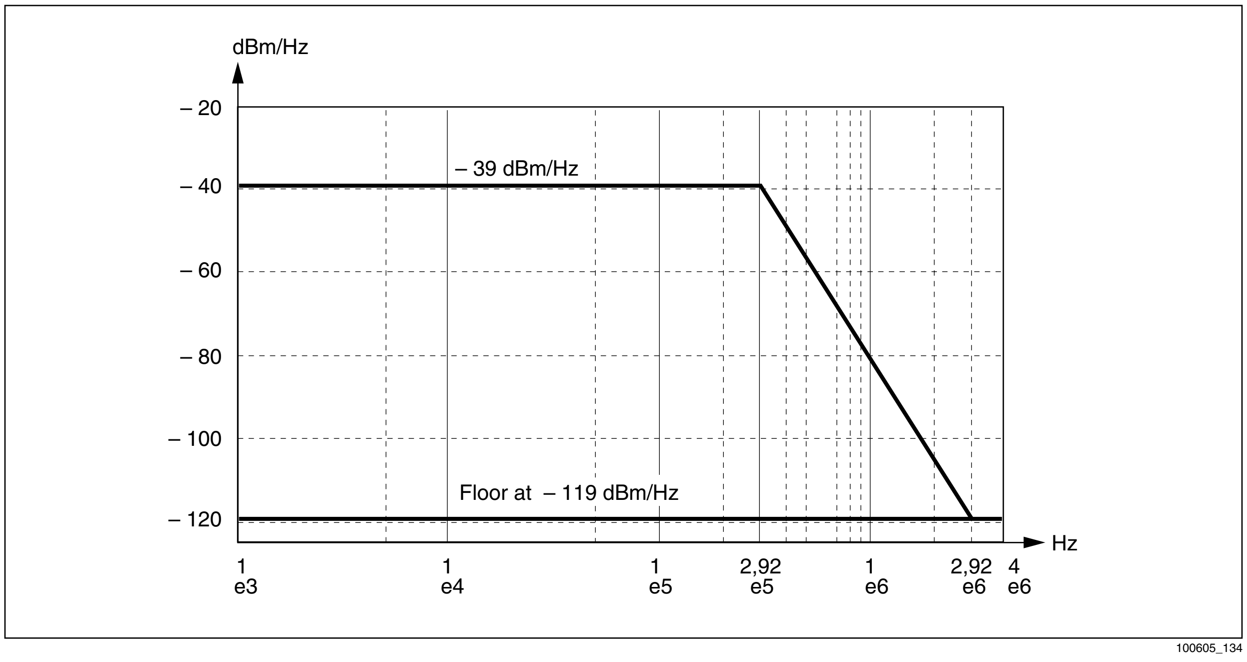
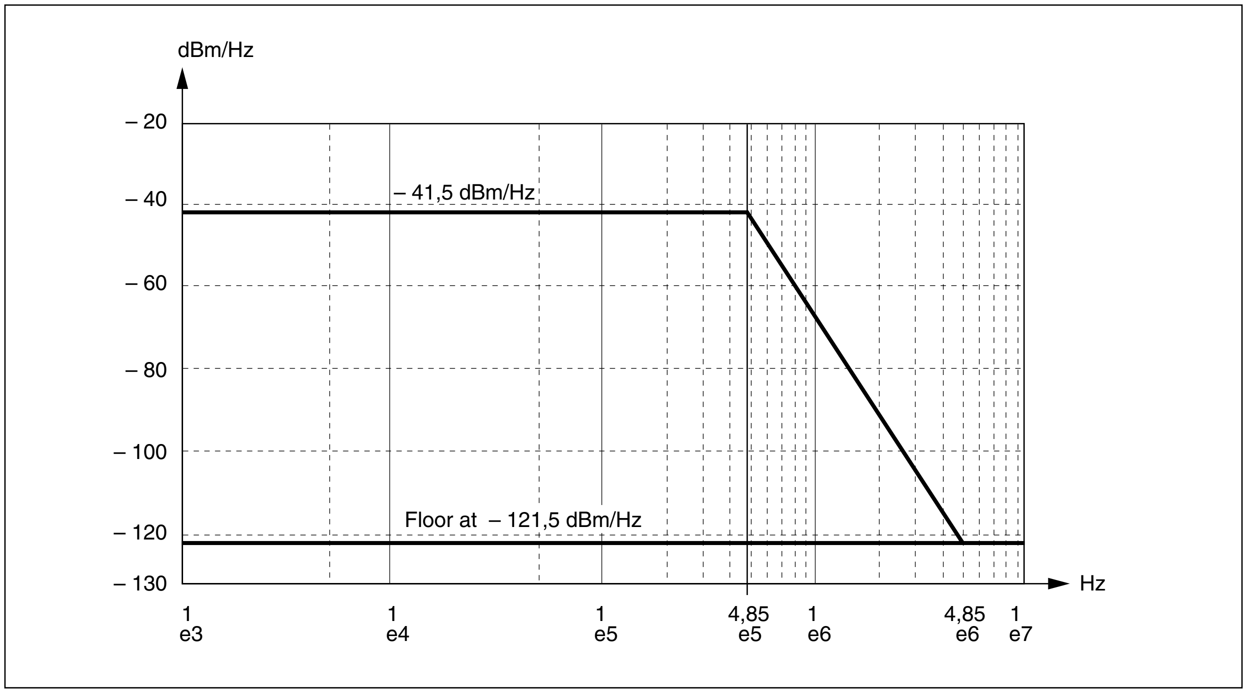


Figure 18-6. Upper Bound of the Average PSD of a 1,160 kbaud System (Source ETSI TS 101 135, Formerly ETR 152)



100605_136

18.5 Mechanical Specifications

Figure 18-7. Package Outline, 27 x 27 mm, Two-Layer Chip, 314-Pin Ball Grid Array (BGA)

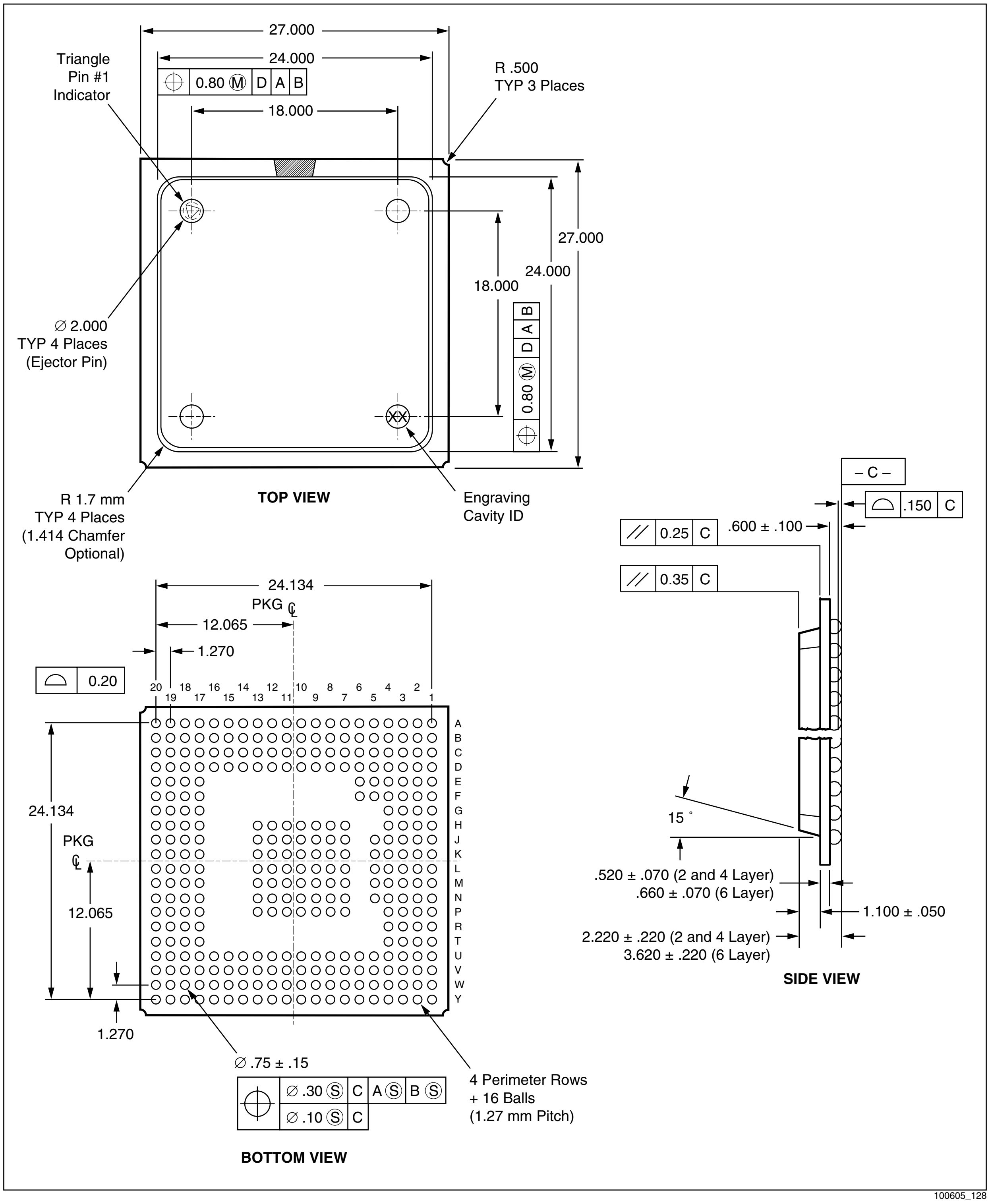
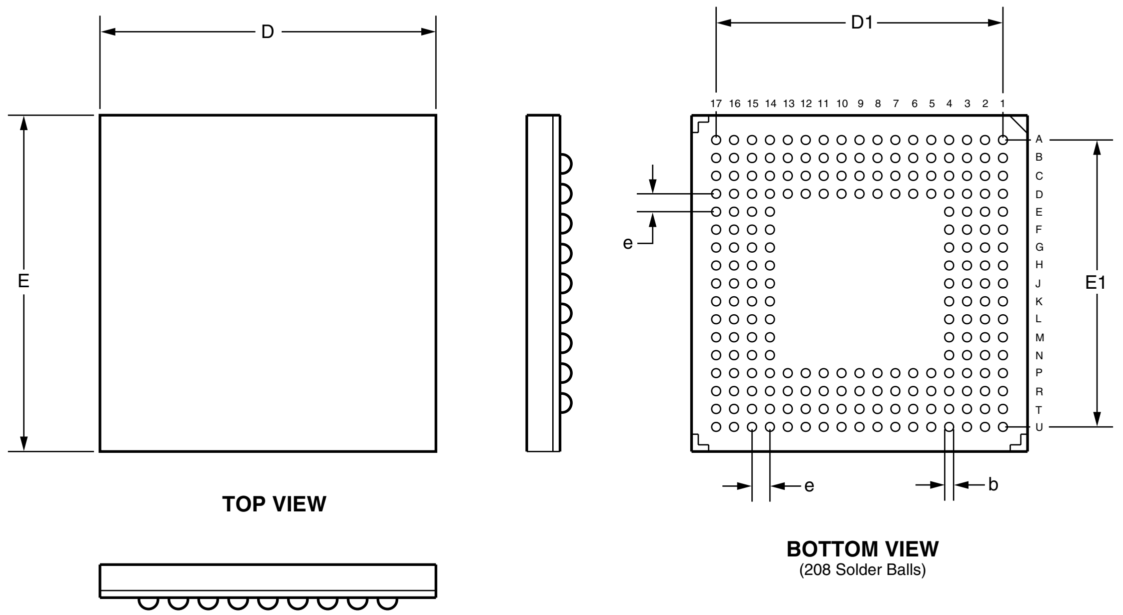
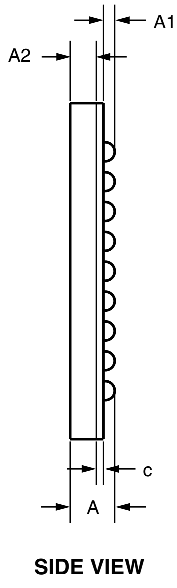


Figure 18-8. Package Outline, 15 x 15 mm, Two-Layer Chip, 208-Pin CABGA

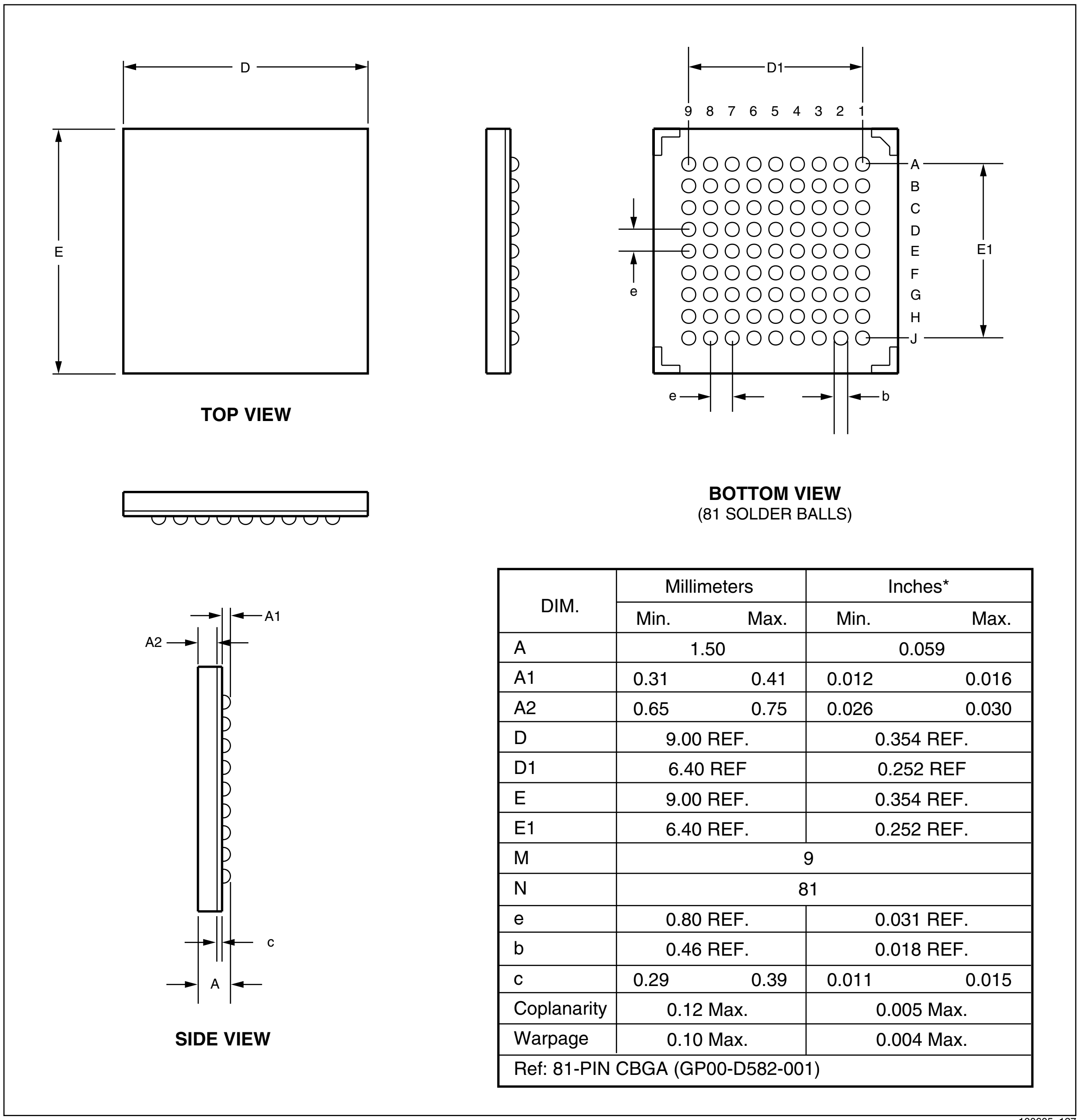




Dim.	Millimeters		Inches	
	Min.	Max.	Min.	Max.
A	1.50		1.059	
A1	0.31	0.41	0.12	0.016
A2	0.65	0.76	0.026	0.030
D	15.00 REF.		0.591 REF.	
D1	12.80 REF.		0.504 REF.	
E	15.00 REF.		0.591 REF.	
E1	12.80 REF.		0.504 REF.	
M	17			
N	208			
e	0.80 REF.		0.031 REF.	
b	0.48 REF.		0.018 REF.	
c	0.29	0.39	0.011	0.015
Coplanarity	0.12 MAX.		0.005 MAX.	
Warpage	0.10 MAX.		0.004 MAX.	
Ref: 208-Pin CABGA (GP00-D576-001				

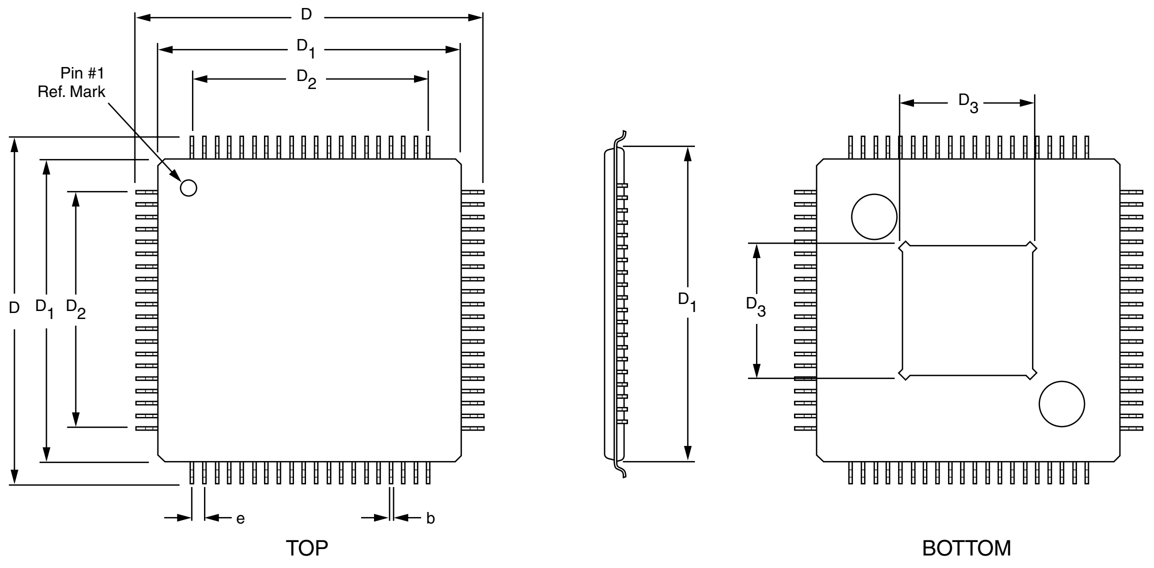
100605_129

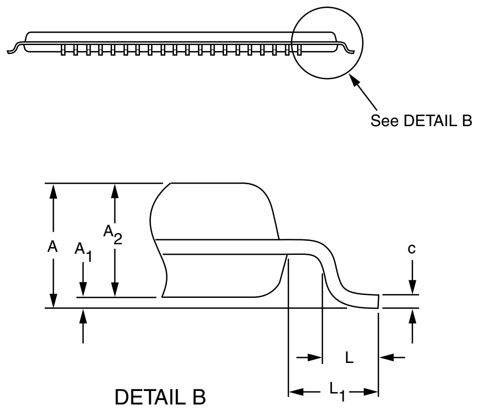
Figure 18-9. Package Outline, 9 x 9 mm Two-Layer Chip, 81-Pin CABGA



DIM.	Millimeters		Inches*	
	Min.	Max.	Min.	Max.
A	1.50		0.059	
A1	0.31	0.41	0.012	0.016
A2	0.65	0.75	0.026	0.030
D	9.00 REF.		0.354 REF.	
D1	6.40 REF		0.252 REF	
E	9.00 REF.		0.354 REF.	
E1	6.40 REF.		0.252 REF.	
M	9			
N	81			
e	0.80 REF.		0.031 REF.	
b	0.46 REF.		0.018 REF.	
c	0.29	0.39	0.011	0.015
Coplanarity	0.12 Max.		0.005 Max.	
Warpage	0.10 Max.		0.004 Max.	
Ref: 81-PIN CBGA (GP00-D582-001)				

Figure 18-10. Package Outline for the 80-Pin Thin Quad Flat Pack (TQFP)





	Millimeters		Inches	
Dim.	Min.	Max.	Min.	Max.
A	1.20 MAX.		0.047 MAX.	
A_1	0.05	0.15	0.002	0.006
A_2	0.95	1.05	0.040	0.041
D	15.75	16.25	0.620	0.640
D_1	13.90	14.10	0.547	0.555
D_2	12.35 REF.		0.486 REF.	
D_3	6.50 REF.		0.256 REF.	
L	0.45	0.75	0.018	0.030
L_1	1.00 REF.		0.039 REF.	
e	0.65 REF.		0.026 REF.	
b	0.32 REF.		0.013 REF.	
c	0.09	0.20	0.004	0.008
Coplanarity	0.10 MAX.		0.004 MAX.	
Ref. 80-Pin ETQFP (GP00-D537)**				

100605_124

Appendix A: Acronyms and Abbreviations

ADC (A/D)	Analog-to-Digital Converter
AFE	Analog Front End
AIS	Alarm Indication Signal
API	Application Programming Interface
BER	Bit Error Rate
BGA	Ball Grid Array
BP	Bit Pump
BT	Bit Pump Transceiver
Channel Unit	HDSL Framer (name comes from HDSL1 Framer)
CRC-N	Cyclic Redundancy Check-N
CU	Channel Unit or HDSL Framer
DAC (D/A)	Digital-to-Analog Converter
DFE	Decision Feedback Equalizer
DIP	Dual In-Line Package
Downstream	From the HTU-C towards the HTU-R (includes regenerators)
DPLL	Digital Phase Lock Loop
DSL	Digital Subscriber Line
DSL Framer	ZipWire2 DSL Framer Block
DSP	Digital Signal Processing
EC	Echo Canceller
EOC	Embedded Operations Channel
EVM	Evaluation Module
FEBE	Far End Block Error (the far end reported a CRC error)
FEXT	Far End Cross Talk
FFE	Feed Forward Equalizer
FIFO	First-In First-Out
FR	Framer
H2TU	HDSL2 Terminal Unit

HDLC	High-Level Data Link Controller
HDSL	High-Bit-Rate Digital Subscriber Line
HTU	HDSL Terminal Unit
HTU-C or COT or LTU	Central Office Terminal or Local Terminal Unit
HTU-R or RT or NTU	Remote Terminal or Network Terminal Unit
LED	Light Emitting Diode
LOS	Loss of Signal
NEXT	Near End Cross Talk
OOF	Out of Frame
P2MP	Point to Multipoint
PAM	Pulse Amplitude Modulation
PCM	Pulse Code Modulation
PLL	Phase Lock Loop
PRA	Primary Rate Access
PRBS	Pseudo-Random Bit Sequence
RAM	Random Access Memory
ROM	Read Only Memory
TCM	Time Code Modulation
TQFP	Thin Quad Flat Pack
Transceiver	ZipWire2 DSP/Transceiver Block
UART	Universal Asynchronous Receive Transmit
UIP	User Interface Program
Upstream	From the HTU-R towards the HTU-C (includes regenerators)

CONEXANT™
What's next in communications technologies

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

www.conexant.com