Bt8960

Software Users Guide

100251B
September 2000

CONEXANT™

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

Table of Contents

List of Figures

List of Tables

1.0 Introduction

The Bt8960 Single Chip 2B1Q Transceiver (also referred to as bit-pump) chip is designed to be controlled by an external microprocessor. The microprocessor sets and controls all Bt8960 internal operation modes during activation, monitors the bit-pump's performance during normal operation, and implements steady-state activities such as micro-interruptions recovery.

A single processor may control up to six bit-pump chips. The Bt8960 software implements all the necessary bit-pump control and monitoring operations, including activation process, performance monitoring, and test modes. This software package is an integral part of the Bt8960 bit-pump, and must be used without any significant changes in order to guarantee correct operation and optimal bit-pump performance.

The software is supplied in both C source code format and Intel 8032 object code format. The object code software may be used in a system employing Intel's 8032 controller to control the Bt8960 chip, or in a specific hardware configuration which employs a second "host" processor that runs the user code and communicates with the 8032 through the serial communication channel (similar to Brooktree's HDSL EVM system). Such a design will require no user code modifications.

If a non-8032 processor is used, or if an 8032-based system implements different hardware design, the C source code version should be compiled together with necessary modifications for the target processor of choice.

A major design goal for the Bt8960 software is portability and ease of conversion for different hardware and compiler environments. However, this cannot completely eliminate the need to "tune" the software for specific implementations because of the hardware-dependent nature of the code. This manual includes detailed instructions to allow for modification of this code.

All application interactions with the Bt8960 are done via a user interface protocol, which is implemented either at the code level (for applications integrating the software in a "single processor" configuration) or by using a serial communication channel (for systems employing an 8032 controller and an additional host processor in a "dual processor" configuration).

The Bt8960 software is conceptually broken up into two sections of code: Application Code and Bit-pump Code.

The application code section is inevitably the responsibility of the end user. Brooktree offers a complete evaluation system (EVM) which implements an example of this application code. To obtain the evaluation system, contact your local sales representative.

The Bt8960 bit-pump software handles the complete activation process without any need for application intervention. All bit-pumps may go through activation in parallel, with practically no degradation of activation time.

The user interface allows access to all parameter status required to implement an HDSL system, including noise margin, pulse attenuation, loss of signal (LOS), etc.

Although no DSP tasks are performed by the control software, some sections of the code involve real-time operations, especially during activation. Applications that integrate the Bt8960 software should take this into account and follow the software integration instructions.

1.1 How to Use this Manual

This manual contains all the necessary information required to successfully modify, compile, and use the Bt8960 software. It is highly recommended that this manual is read prior to writing application code for the Bt8960.

The chapters "Software Structure" through "Software Integration" are particularly important for those using the source code version of the software to integrate the code with their own software. Use the "Compiling the Source Code" chapter when modifying and compiling the source code. Refer to the "Software Integration" chapter when integrating the application software with the Bt8960 code.

The Bt8960 is controlled using the software user interface. Use the detailed information given in the "User Interface" chapter when writing the Bt8960 application software. Appendix A serves as a reference to the control and status command set.

The "Software Application" chapter contains information on how to use the Bt8960 software and User Interface command set in a transceiver application. It outlines the required initialization and parameter setting sequence, accompanied by examples of a typical application.

1.2 Contents of this Manual

Chapter	Contents
Overview	Overview of the software design principles, portability considerations, hardware-related issues, application interface principles, and application code integration constraints.
Software Structure	Description of the code structure, main routines, file names, and reserved names.
Software Operation	Software operation principles, including the application interface and activation code sections.
Compiling the Source Code	Explains how to modify and compile the C source code to match the application hardware/software environment.
Software Integration	Explains how to integrate the application software with the Bt8960 software.
User Interface	Details of the User Interface operation, command structure, and status responses.
Software Application	How to use the Bt8960 software and user interface in an HDSL transceiver application.
Command Set Reference	Lists control and status request commands, including opcodes, parameter values, etc.

2.0 Overview

2.1 Software Structure and Operation

The Bt8960 software is responsible for handling and monitoring the activation and normal operation activities of up to six Bt8960 HDSL bit-pumps.

The Bt8960 software includes the following five major functional blocks:

1. Application—Contains the application specific code to initialize the system and to control any system level tasks.
2. BtMain—Activates the bit-pump control process for each of the Bt8960 chips present in the system. It also calls routines that handle messages received on the serial communication channel.
3. Bt8960 Control—The principal bit-pump control routine. The Control routine goes through a series of states that implement the various activation and normal operation activities. Every call to the Control routine executes the operations relating to a specific state, and the state variable is advanced to the next state. The same routine is used for all active bit-pump chips, only the state variable being bit-pump specific.
4. User Interface—Receives and executes user commands, and sending back status information. The interface is implemented either at the code level, called the Application-Program Interface (API) (when an application is integrated with the Bt8960 software) or via a serial communication protocol. In both control options, the interface is based on a set of commands issued and parameter status returned.
5. Interrupt Handler—This routine is entered every time one of the Bt8960 chips initiates an interrupt. Most interrupts are handled by setting appropriate software flags.

2.2 Portability

The Bt8960 software is designed with code portability as a major goal. This approach creates a C code that can be easily modified to match different hardware and software environments.

The code itself has hardware-related parameters which must be defined. For example, the absolute address space allocated to each of the Bt8960 chips depends on the specific hardware design and address decoding details.

Another example of a hardware/software-dependent code section is the definition of the Bt8960 interrupt handling routine. The routine itself is part of the supplied software (and should not be modified) but the routine header definition and notation is both processor and compiler dependent. These examples demonstrate the kind of modifications required in the source code in order to successfully compile and run the Bt8960 software.

The "Compiling the Source Code" chapter describes all of the required modifications. In addition, these modifications are clearly marked and commented in the source files.

2.3 Integration with Application-Specific Software

The Bt8960 software controls and sequences all the activation operations which include real time operations. In a system where the Bt8960 software shares processing resources with the user's application software, care must be taken to not impair bit-pump performance. Guidelines to handle real-time operations within applications are described and explained in detail in the "Software Integration" chapter of this manual.

2.4 Software Interfacing with the Bt8960

Since all bit-pump operations (activation, performance monitoring, etc.) are handled by the Bt8960 software, all user interaction with the bit-pumps is done through the Bt8960 software and not by directly accessing the Bt8960 chip. In some cases, performing a read/write operation can impair the proper operation of the control software.

The Bt8960 software contains a User Interface (UI) module that allows full control over the operation of the bit-pumps and performance monitoring. The UI is based on a set of issued commands, and the Bt8960 software responds by performing the required action or supplying the required status. When used in the single processor configuration, it is highly recommended that the API and UI be used to ensure easy integration with code updates.

There are two configurations for transferring commands and parameter status between an application and the Bt8960 software. In a single processor design, the interface is done using two API (Application-Program Interface) routines that are part of the Bt8960 software. The application issues commands and receives status by calling these routines.

In a dual processor design, the Bt8960 bit-pumps are controlled by an 8032 processor and the application interfaces to the bit-pumps through the serial communication (UART) channel. In this configuration, the commands and status are sent as messages over the serial channel.

Both configurations use exactly the same set of commands. This command set is designed for maximum flexibility in operating the bit-pumps, and supplies all the necessary performance and status monitoring information. Each bit-pump can be controlled and monitored individually via the UI.

The "User Interface" chapter describes the structure of the commands and the communication protocol for both UI configurations in detail. Appendix A is a reference guide to the set of commands and status bits, with details on command use, syntax, and operation.

3.0 Software Application

Interaction with the Bt8960 bit-pumps is done by using the set of control and status request commands. This chapter addresses the issue of applying these commands in a complete HDSL transceiver system.

3.1 System Configuration

Before any other command or status request may be issued to a bit-pump, the system configuration needs to be defined, using the Bit-pump ON/OFF command. This informs the Bt8960 software which bit-pump is ON. Following power up, all bit-pumps are declared to be OFF. All commands (other than system configuration) sent to a bit-pump that is turned OFF are ignored by the Bt8960 software.

Turning ON a bit-pump means that the corresponding Bt8960 is installed in the system and is intended to be used. Not all Bt8960s that are installed in the system must be turned ON. A bit-pump may be turned ON or OFF at any time, and any combination of bit-pumps turned ON or OFF is allowed. Turning a bit-pump ON (even if it is turned ON already) resets the corresponding Bt8960, and all configured parameters must be reprogrammed.

3.2 Setting Bit-Pump Parameters

For proper operation of the bit-pumps, all configuration and operation parameters must be set to their correct values (depending on the specific application) prior to activating the bit-pumps. These parameters include design-specific information (such as analog AGC configuration, data transfer format, activation sequence source, etc.) and application specific parameters (such as symbol rate, terminal type, etc.).

All parameters and configuration options have default values (see Appendix A). Only those parameters that have values different than the default need to be programmed. Table 3-1 lists all the user-programmable parameters together with their default values. See the "User Interface" chapter and Appendix A for details regarding the use and syntax of the specific commands used to set these parameters.

NOTE: The _SYM_RATE command must be immediately issued after the _SYSTEM_CONFIG ON command.

Table 3-1. User Configurable Parameters and their Default Values

Parameter	Default
Bit-pump On/Off	Off
Terminal Type	HTU-C
Other Side Bt8960	Non-Bt8960
Analog AGC Configuration	No AGC
Symbol Rate	144 K Symbol/Sec
LOST Time Period	1sec
Data Transfer Format	Serial
Start-up Sequence Source	External
Transmitter Scrambler On / Bypass	Bypass
Receiver Descrambler On / Bypass	Bypass
Operate Non Linear EC	Off
Transmitter Gain	Calibration Value

3.3 Activation

Upon power-up, the bit-pumps move to an IDLE state where no activities take place. No signal is transmitted and the bit-pumps will not respond in any way to input signal detection. To initiate a activation procedure, an Activate command should be issued at the HTU-C and HTU-R terminals. This results in the transmission of a 2-level activation sequence, and begins the sequence of activation activities on the HTU-C.

Under normal operating conditions, the HTU-R should detect the incoming signal and respond according to the standard activation protocol. The HTU-R bit-pump will respond in this way only if an Activate command was issued on the HTU-R terminal. Note that the Activate command on the HTU-R does not initiate a response, but enables such a response once a signal is detected.

The order in which the Activate commands are issued at the HTU-C and HTU-R terminals is of no importance, but the complete activation procedure will take place only when both terminals are activated. When the HTU-C is activated first, it starts a 2-level transmission, but will get no response from the HTU-R until the HTU-R has been activated. It will therefore wait until an HTU-R signal is detected, and then proceed with the activation. When the HTU-R is activated first, it will wait for the detection of an HTU-C signal. Once the HTU-C has been activated it starts transmitting, and the HTU-R, upon detecting this transmission, will proceed with the activation.

Once both terminals are activated, the complete sequence of activation activities is carried out on both terminals according to the standard protocol. No user intervention is required during this process.

It is the responsibility of the application software to monitor the appropriate status responses, in addition to Channel Unit frame sync status, during activation, and decide whether or not the activation has been successfully completed. The important status parameters for activation monitoring include noise margin, loss of signal (LOS, LOST), transmit 4-Level indicator, and activation timers expiration. Table 3-2 describes these together with the specific status request commands that are used to access each status parameter. Table 3-3 shows additional bit-pump responses that do not relate directly to the activation procedure. Further information may be found in the "User Interface" chapter and in Appendix A.

Table 3-2. Start-up Status Parameters

Indication	Description	Command Name
Noise Margin	An estimate of the noise margin (in dB), indicating the tolerable increase in noise level while still maintaining a BER<1E–7.	Noise Margin
Noise Margin OK	1 indicates noise margin higher than –5 dB.	Bit-pump Status
LOS	0 Indicates the presence of an incoming signal, 1 indicates no input signal present. Status is based on average far-end signal power measurement. This status is valid at all times.	Bit-Pump Status
LOST	1 indicates the presence of a LOS condition for more than the LOST time period (programmable). The LOST response is valid only after a Deactivate command is issued.	Bit-Pump Status
Activation Timer	1 indicates expiration of the activation interval.	Bit-Pump Status
Transmit 4-Level	1 indicates the bitpump is transmitting a 4-Level signal.	Bit-Pump Status
Normal Operation	1 indicates the bitpump has completed the stages of the activation process.	Bit-Pump Status

Table 3-3. Additional Status Parameters

Indication	Description	Command Name
Total Input Signal Level	Average absolute level of signal at the input to the A/D. This signal is composed of both the far-end and echo signals.	Input Signal Level
Input DC Offset	Average DC offset at the input to the A/D. A high value may cause performance degradation.	Input DC Offset
Cable Attenuation	Total signal attenuation (in dB).	Far-end Signal Attenuation
Frequency Deviation	Value of the (digital) VCXO control signal.	VCXO Control Voltage

3.4 Test Modes

The Bt8960 may be operated in special test modes, used during development and field operation, for maintenance and fault identification. All test modes are activated using the Test Mode command, with the command parameter value selecting the specific mode.

To exit a test mode, issue the Test Mode command with the *_EXIT_TEST_MODE (0x00 value)* parameter. The action taken when exiting a test mode condition depends on the specific test mode selected. Activating most test modes while the bit-pump is synchronized will cause the bit-pump to lose sync. Exiting the test mode and going back to normal data transfer condition will require full restart. In these cases, the bit-pump goes to the IDLE state when exiting the test mode.

Other test modes do not create a sync loss condition, and normal operation may continue when exiting the test condition. When exiting one of these test modes, the bit-pump goes back to the state that was active when the Test Mode command was issued.

Table 3-4 describes the available test modes, and shows for each mode the ability to go back to normal operation. Further information may be found in the User Interface section and in Appendix A.

Table 3-4. Bt8960 Test Modes *(1 of 2)*

Test Mode	Description	Sync Loss
External Analog loopback	Transmits the externally supplied TX symbols, use the echo signal as a "received" signal, and detect the symbols using the standard equalizer.	√
Digital "near" loopback	TX symbols supplied to the bit-pump (by the framer) are looped back as the RX symbols going from the bit-pump to the framer. Useful for testing the framer and the framer-bit-pump connection.	
Digital "far" loopback	Detected (RX) symbols are transmitted back on the loop. Useful for testing full 2-way transmission over a loop. Note: the operation of a digital "far" loopback requires activating the Bt8960 internal TX scrambler and RX descrambler.	
Transmit isolated +3 pulse	Transmits (repeatedly) an isolated +3 level pulse. Useful for testing the transmitted pulse shape.	√
Transmit isolated +1 pulse	Transmits (repeatedly) an isolated +1 level pulse.	√
Transmit isolated –1 pulse	Transmits (repeatedly) an isolated –1 level pulse.	√
Transmit isolated –3 pulse	Transmits (repeatedly) an isolated –3 level pulse.	√
Continuous 4-level transmission	Continuous transmission of a 4-level scrambled 1's sequence (internally generated). Useful for measuring transmitted power and spectral shape.	√
Continuous 2-level transmission	Continuous transmission of a 2-level scrambled 1's sequence (internally generated).	√
Set minimum VCXO frequency	Sets VCXO control word to its minimum value.	√
Set nominal VCXO frequency	Sets VCXO control word to its nominal value. Useful for measuring VCXO center frequency.	√

Table 3-4. Bt8960 Test Modes *(2 of 2)*

Test Mode	Description	Sync Loss
Set maximum VCXO frequency	Sets VCXO control word to its maximum value.	√
Internal Analog loopback	Transmits the externally supplied Tx symbols out the TXP and TXN pins and detects the symbols on the hybrid inputs (RXBP, RXBN), the receive inputs (RXP and RXN) are bypassed.	√
Isolated Analog loopback	The externally supplied Tx symbols are internally looped back in the Bt8960. The transmitter (TXP and TXN) is turned off (silent).	√

Table 3-5 shows the relationship between the Chip Revision, Software Version, and Analog Loopbacks supported.

Table 3-5. Analog Loopback vs. Version Relationships

Version	External	Internal	Isolated
V 1.x	None	None	None
V 2.0+	Rev. B and C	Rev C only	Rev C only

3.5 Internal BER Meter Operation

This section describes how to use the Bt8960's Internal BER Meter. The BER meter code is only accessible when the BER_METER compiler flag is declared. The BER Meter can be used to verify the integrity of the line, and as a diagnostic tool during production/field testing or hardware/software development.

The BER Meter uses its internal scrambled ones generator and de-scrambler to detect bit-errors. For the BER Meter to function properly, both the HTU-C and HTU-R must issue the _BER_METER_START API command. Since the BER Meter uses its own internal scrambled ones generator, it cannot be used when transporting real payload data. The BER Meter is only operational after the bit-pump has successfully completed startup. All access to the BER meter is done through the API, see Appendix A.

The interrupt handler reads the Bit Error Rate Meter Register (Address 0x4C, 0x4D) and updates the Number of Bit-Errors and Number of Meter Intervals counter after every meter interval. This causes the interrupt handler to be slightly longer, ~55uS per bit-pump on an 11.0592MHz Intel 8032.

3.5.1 Calculating Avg BER and Elapsed Time

The following formulas are used to calculate the Avg BER and Elapsed Time.

$$AvgBER = \frac{\#ofBitErrors}{\#ofMeterIntervals * MeterIntervalLength * 2}$$

$$ElapsedTime = \frac{\#ofMeterIntervals * MeterIntervalLength * 2}{DataRate}$$

Variable	How Derived
# of Bit-Errors	Read the # of Bit-Errors Low & High Byte API commands and build a 16-bit unsigned integer.
# of Meter Intervals	Read the # of Meter Intervals Low & High Byte API commands and build a 16-bit unsigned integer.
Meter Interval Length	Read the Bt8960 Meter Interval Register (Address 0x18, 0x19) and build a 16-bit unsigned integer. During normal operation, these registers should always read 0x8000 (32768).
Data Rate	Data Rate of the system, i.e. 288000 or 416000.
2	The '*** 2**' is necessary because there are 2 bits per symbol and the meter interval length is based on the number of symbols.
NOTE(S): 16-bit value = (high byte << 8) + (low byte)	

Figure 3-1. Example

```
/*
 * Assuming 288kbps Data Rate, Normal Operation, and BER Meter Active
 * Also assumes using compiler/linker that supports floating point.
 */
void get_ber_meter_status (unsigned char no)
{
  unsigned char temp, temp1;
  unsigned int errors, intervals;
  float avg_ber, elapsed_time;

  _BtStatus(no, _BER_METER_STATUS, _BER_BIT_ERRORS_LOW, &temp);
  _BtStatus(no, _BER_METER_STATUS, _BER_BIT_ERRORS_HIGH, &temp1);
  errors = (unsigned)BYTE2WORD(temp1, temp);

  _BtStatus(no, _BER_METER_STATUS, _BER_METER_INTERVALS_LOW, &temp);
  _BtStatus(no, _BER_METER_STATUS, _BER_METER_INTERVALS_HIGH, &temp1)
  intervals = (unsigned)BYTE2WORD(temp1, temp);

  avg_ber = (errors) / (intervals * 0x8000 * 2)   /* equations don't show necessary type casting */
  elapsed_time = (intervals * 0x8000 * 2) / (288000)

#if TDEBUG
  printf("# Bit Errors = %u\n", errors);
  printf("# Meter Intervals = %u\n", intervals);
  printf("Avg Ber = %.2e\n", avg_ber);
  printf("Elapsed Time = %.1f seconds\n", elapsed_time);
#endif

  return;
}
```

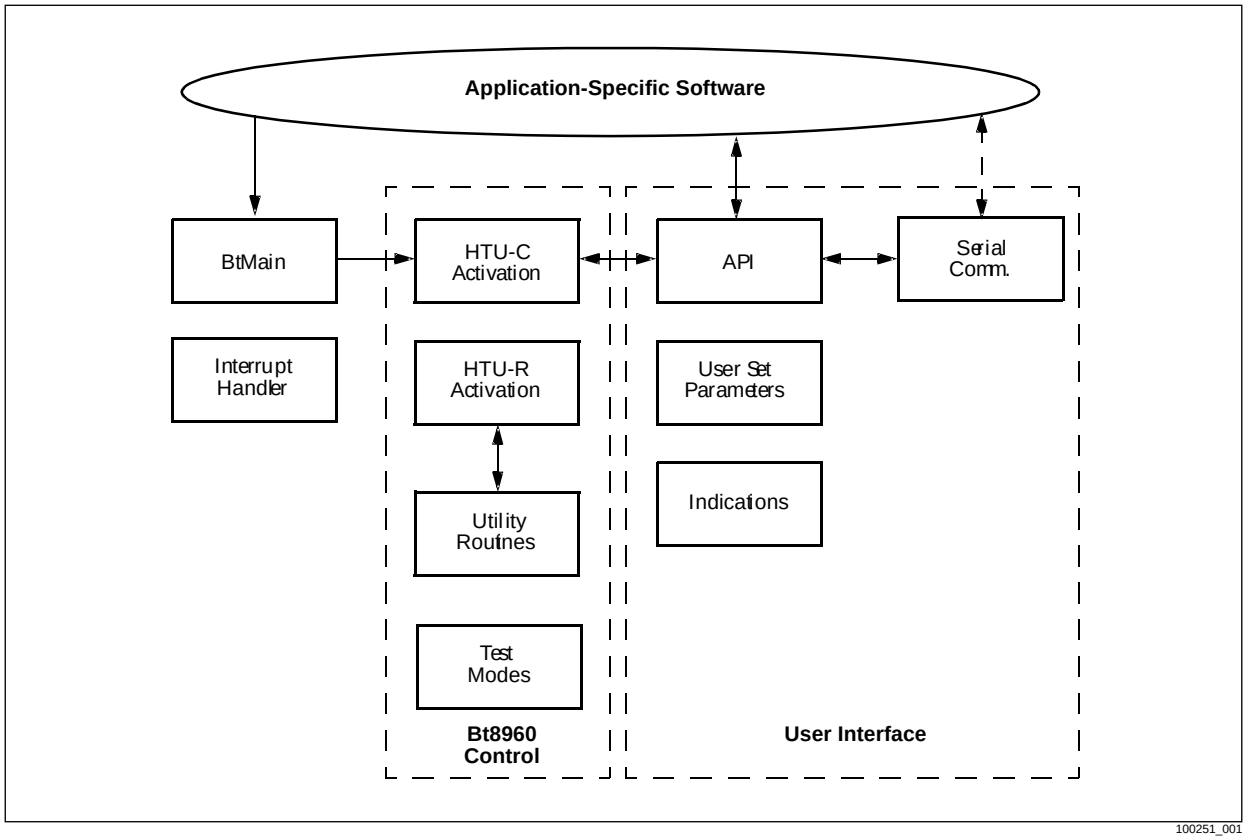
4.0 Software Structure

This chapter describes the structure of the Bt8960 control software. The information is useful for understanding the Bt8960 operation and the proper integration of Bt8960 code with the application software.

4.1 Software Block Diagram

This section describes the major Bt8960 software logical blocks and briefly outlines their function. Figure 4-1 shows the major software modules and their relation with application specific software.

Figure 4-1. Bt8960 Software Modules



The Application block contains the application's high level system tasks, i.e. activation state diagram, diagnostics, controlling other devices, etc. The *_BtMain()* module is called by the application's main program, and it is important to ensure a periodic execution of this module within specified real-time constraints (see the "Software Integration" chapter for details).

The *_BtMain()* module is responsible for executing all the specific Bt8960 tasks, including the bit-pump control operations, and the received/transmitted message handling. This is the highest level block in the Bt8960 software hierarchy, and should be executed repeatedly and indefinitely to ensure proper operation of the bit-pumps.

The Bt8960 Control block contains the bit-pump activation control routines (for the central office and remote terminals) and special test mode routines. The activation control routine is executed periodically for every active Bt8960 chip. It goes through the complete activation procedure and handles normal operation monitoring activities. The test modes module is responsible for operating the bit-pump in a specified test mode, such as loopback, isolated pulse transmission, etc.

The User Interface block implements all the tasks related to communication with an application. This communication is either done through a serial communication protocol or by using code level API routines. The UI block interprets received commands, performs the required operation, and sends back status reports.

The Interrupt Handler block responds to Bt8960 interrupts, sets the software interrupt flags, and in special cases, performs bit-pump operations that cannot be delayed.

4.2 User-Called Routines

The routines found in Table 4-1 must be called or accessed by the application software. Further details regarding these specific routines are given in the "User Interface" and the "Software Application" chapters and Appendix A. Further information and documentation may be found in the source files.

Table 4-1. Bt8960 Software Routines

Routine / Macro	File Name	Description
_BtSwPowerUp()	btmain.c	Bt8960 software initialization. Must be executed prior to any other Bt8960 related operation.
_MaskBtHomerInt()	btmain.c	Masks all Bt8960 interrupts for all present Bt8960 chips.
_Init8051()	init51.c	Used only in 8032 environment. Initializes 8032-specific parameters.
_InitVirtualTimers()	intbug.c	Used only with the INT_BUG compiler flag. Initializes Virtual Timers.
_BtControl()	api.c	API routine. Called by the application to issue a control command.
_BtStatus()	api.c	API routine. Called by the application to issue a status request command and to get the requested status.
_BtMain()	btmain.c	Main Bt8960 control routine. Calls all Bt8960 tasks. Should be executed repeatedly and indefinitely.

4.3 Memory Requirements

The code ROM requirements depend on the compiler, processor in use, and compiler flags. The RAM and ROM requirements for four common build options with their associated compiler flags are shown in Table 4-2.

4.3.0.1 BT8960CR w/o BER_METER

C51,ADD_DELAY,PDATA_MODE,SER_COM,HTUC,HTUR,SINGLE_LOOP,CHAN_UNIT

4.3.0.2 BT8960CR w/ BER_METER

C51,ADD_DELAY,PDATA_MODE,SER_COM,HTUC,HTUR,SINGLE_LOOP,BER_METER, CHAN_UNIT

4.3.0.3 BT8960C w/ BER_METER

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

4.3.0.4 BT8960R w/ BER_METER

C51,ADD_DELAY,PDATA_MODE,SER_COM,HTUR,SINGLE_LOOP,BER_METER,CHAN_ UNIT

Table 4-2. Memory Requirements

Build Option	RAM	ROM
BT8960CR w/o BER_METER	111	24.0 k
BT8960CR w/ BER_METER	117	24.4 k
BT8960C w/ BER_METER	117	20.0 k
BT8960R w/ BER_METER	117	20.6 k

The RAM requirements do not include the stack size, an additional 24 bytes is required for the run-time stack. These requirements include the Activation State Diagram application example code.

5.0 Compiling the Source Code

This chapter contains detailed information on how to successfully modify and compile the source code to match the specific application environment. The code modifications are required in order to match the Bt8960 code to the processor/compiler environment and to the details of the system hardware implementation.

In addition to the source code modifications, compilation directives must be correctly declared at compile time.

To get a fully working executable code, perform the following steps:

1. Determine the correct compiler directives definitions suitable for the environment (see the Compilation Directives section).
2. Modify or write all the application modifiable sections by following the instructions in this chapter.
3. Compile and link all the source files. Use the file list and dependency information given in the File Names and Dependencies section.

NOTE: If the C51 flag is declared (indicating an 8032 family processor and Keil & Franklin C51 compiler), modify or write only specific application sections (see the Code Modifications section).

5.1 Directory Structure

The directory structure is partitioned into two directories. The main (application specific) directory as the root and the bit-pump code as a sub-directory labeled BITPUMP. The application examples, hex files, build script files, etc. are found in the main directory. All of the bit-pump source code and header files are found in the BITPUMP subdirectory. However, the final implementation of the directory and file structure is up to the user.

5.2 File Names and Dependencies

Table 5-1 lists all the files that should be compiled and linked in order to get a complete, executable Bt8960 control code. The table also shows the dependency between the various files.

Table 5-1. Source File Names and Dependencies

File to be Compiled	File Depends on…
api.c	bthomer.h
bitpump.c	bthomer.h
btint.c	bthomer.h
btmain.c	bthomer.h
init51.c	bthomer.h
intbug.c	bthomer.h
mail.c	bthomer.h
main.c	bthomer.h
serint.c	bthomer.h
suc.c	bthomer.h
sur.c	bthomer.h
suutil.c	bthomer.h
testmode.c	bthomer.h
user.c	bthomer.h
util.c	bthomer.h
erle.c	bthomer.h

Other Files	File Depends on…
bthomer.h	btmain.h, mail.h, init51.h, intbug.h, serint.h, bitpump.h, suc.h, sur.h, suutil.h, util.h, api.h, testmode.h, user.h, ptrdef.h, reg51.h, erle.h

5.2.1 Compilation Directives

The compilation directives select different code sections for compilation, and are used in code sections that are either hardware dependent or compiler dependent. Accurate declaration of these directives is necessary to generate a correctly working executable code.

NOTE: It is extremely important that after declaring these directives, all of the source modules be recompiled to ensure the changes propagate throughout the entire executable. Failure to do so will result in unpredictable behavior.

5.2.1.1 #SINGLE_LOOP; #TWO_LOOPS; #THREE_LOOPS; #FOUR_LOOPS; #SIX_LOOPS

These compiler flags allow the developer to specify the maximum number of loops supported in their system. One and only one of these flags must be defined at a time. The bit-pumps are allocated in a sequential order starting with _BIT_PUMP0 (1st bit-pump) and ending with _BIT_PUMP5 (6th bit-pump)

Only the bit-pumps that are supported by the specified compiler flag may be accessed during run-time, otherwise unpredictable behavior will result. For example, if THREE_LOOPS is defined, then only _BIT_PUMP0, _BIT_PUMP1, and _BIT_PUMP2 may be addressed.

Each subsequent loop adds ~8 bytes of ROM space and ~4 bytes of RAM.

5.2.1.2 #C51

The C51 flag should be declared when the Bt8960 software is executed on an Intel 8032 family microprocessor and the Keil/Franklin C compiler is used to compile the source code.

If another Intel 8032 family compiler is used, the C51 compiler flag may be defined. The user must be aware there may be compatibility issues between the compilers.

If any non Intel 8032 family microprocessor and C compiler are used, do not declare the C51 flag.

5.2.1.3 #PDATA_MODE, #XDATA_MODE

The PDATA_MODE and XDATA_MODE flags can only be declared together with the C51 flag. The PDATA_MODE and XDATA_MODE flags set the addressing mode used to access the Bt8960 chips. One and only one of these flags must be declared in a C51 system.

The PDATA_MODE flag selects page mode addressing which results in reduced ROM requirements for code space and faster execution. The XDATA_MODE flag selects external non-paged addressing. This results in greater code size and ROM requirements, and somewhat slower execution of the control code.

For PDATA_MODE, in each of the bit-pump functions, the P2 port is set to address the specified bit-pump. In addition, the bit-pump interrupt saves the P2 port value before addressing the bit-pumps, then restores the previous P2 port value on exit of the interrupt function. Special care must be taken so that the P2 port value does not get erroneously modified by any other interrupt handler or another section of code.

NOTE: If the Franklin & Keil Compiler's Large memory model is defined, the XDATA_MODE compiler flag should be set since all variables are declared in **xdata** space.

5.2.1.4 #SER_COM

The SER_COM flag enables the serial communication protocol of the user interface. It is used in the dual processor mode of operation. It should be declared in an 8032-based system that employs the serial communication configuration for controlling the Bt8960. The SER_COM flag may be declared only if the C51 flag is declared.

5.2.1.5 #ADD_DELAY

Defining the ADD_DELAY flag causes the addition of a 2-symbol delay before accessing specific addresses in the Bt8960. This additional delay is required when a read/write operation to the Bt8960 is performed in less than a symbol time period. The delay is added only for very specific Bt8960 addresses where the read/write operations are limited to one access per symbol. The ADD_DELAY flag should be declared only when the external read/write cycle time of the microprocessor is faster than the symbol rate.

5.2.1.6 #HTUR; #HTUC

The HTUR and HTUC flags control whether HTU-R or HTU-C code is included in the executable. For applications where there is no distinction between HTU-R and HTU-C, except at the time of installation, having a single piece of code that serves as HTU-R and HTU-C is useful. For these applications both HTUR and HTUC should be declared.

If code space is a constraint, the HTU-R and HTU-C code can be customized for each terminal type. Since the control process for each terminal type is in separate code, and represents a substantial portion of the code, it is possible to save ROM space by only declaring HTUR for the HTU-R terminal code and HTUC for the HTU-C terminal code. In this case the _TERMINAL_TYPE command would be fixed in code with its parameter set in accordance with the terminal type for which the code is being compiled.

The HTUC code adds ~4 k of ROM and ~5 bytes of RAM.

The HTUR code adds ~4.6 k of ROM.

5.2.1.7 #CHAN_UNIT

The CHAN_UNIT flag provides hooks into the Serial Communication Interface to allow the user to access user defined API commands. In the Brooktree EVM Systems, these API access the Channel Unit EVM boards (Channel Unit, Framer, & LIU). Please refer to the User Interface: Code Level Interface section for details.

NOTE: The CHAN_UNIT flag may not be used when the #FOUR_LOOPS or #SIX_LOOPS flags are specified unless the User Defined API Commands section is modified.

5.2.1.8 #BER_METER

The BER_METER flag compiles in the BER Meter code. The BER Meter code adds ~400 bytes of ROM and 1 byte of RAM plus an additional 5 bytes of RAM for each bit-pump. Note that the BER_METER flag cannot be declared when the INT_BUG compiler flag is specified. See the Internal BER Meter section in the Software Application chapter.

5.2.1.9 #ERLE

The ERLE flag enables the ERLE diagnostic code. Please see Appendix C for details.

5.2.1.10 #INITIATED_INTERRUPT

The INITIATED_INTERRUPT flag should not be declared. It provides minimal support for some multi-tasking operating systems. Multi-tasking operating system use with the activation code is not supported by Brooktree.

5.2.1.11 #TDEBUG

The TDEBUG flag should not be declared. TDEBUG provides printed text statements about the current status of the bit-pumps during activation. The messages are printed to the 8032's serial port. Because of this, TDEBUG cannot be used when SERCOM is declared. This flag can be useful for identifying problem areas in the activation sequence.

5.2.1.12 #INT_BUG

The INT_BUG flag should not be declared. The INT_BUG flag uses the microprocessor's timer and RAM instead of the bit-pump's internal timers. This was needed in an earlier HDSL bit-pump product.

5.2.2 Code Modifications

The exact location of code sections to be modified are designated by comments of specific form in the source files. The format of these comments is shown in Figure 5-1.

Figure 5-1. User-Modifiable Code Section Format

```
    /*-------------------------------------------------------*/
    /*>>>   User Modifiable Section user.1 Begins         <<< */
    /*-------------------------------------------------------*/
    /* Modification Description:                             */
    /* Write the contents of routine EnableUserInterrupts().   */
    /* Calling this routine should enable all user interrupts. */
    /*                                                       */
    /* Reference:                                            */
    /* "Bt8960 Software User's Manual".                      */
    /*-------------------------------------------------------*/
Insert code here

    /*-------------------------------------------------------*/
    /*>>>   User Modifiable Section user.1 Ends            <<<*/
    /*-------------------------------------------------------*/
```

These comments contain basic information describing the code section that should be modified (or written). All the modifications are referenced according to their source file name and the order in which they appear in the file. This reference ("*filename*.X") appears both in source file comments and in this manual.

Each of the following sections refers to a single code section which must be modified or written. All necessary modifications are covered, and no other changes in the source code should be made.

The bit-pump source files that contain modifiable sections are *btint.c, init51.c, serint.c, suc.h and user.c*. No modifications should be made in any other file.

5.2.2.1 Application Code

The EVM Application Code contains templates (examples) for the *main()* routine of the application specific code. This main routine should execute both the application tasks and the Bt8960 software task. Please refer to the *Bt8960 Single-Chip 2B1Q Transceiver EVM Software Manual* for details.

Extreme care must be taken to ensure that the application code (the body of the infinite loop) meets the real-time requirements outlined in the "Software Integration" chapter. The concept is that the time period between successive activations of *_BtMain()* must be kept lower than a specified limit. This requirement puts constraints on the maximum execution time of the application code

5.2.2.2 btint.c.1

Code section *btint.c.1* should contain the header of the Bt8960 interrupt handling routine. The body of the interrupt routine is already written, and should not be modified in any way. However, the routine header syntax is hardware, microprocessor, and compiler specific and must be written for the specific configuration.

This routine should be invoked every time a Bt8960 interrupt signal is asserted (the Bt8960 INT pin is an active-low-level triggered signal). This interrupt should be enabled on power-up. It is necessary to ensure the interrupt vector is initialized within the application.

When using the Borland C compiler, the interrupt routine header is defined as follows:

```
void interrupt _HandleBtInterrupt(void)
```

The application specific power-up initialization code should set the *_HandleBtInterrupt()* routine to function as the Bt8960 interrupt.

5.2.2.3 init51.c

Module *init51.c* contains the Intel 8032 microprocessor initialization routines.

5.2.2.4 init51.c.1

Code Section *init51.c.1* should contain the baud rate timer initialization value. The initialization value is dependent on the microprocessor and the crystal frequency.

5.2.2.5 serint.c.1

Code section *serint.c.1* should contain the header of the serial interrupt handling routine. The body of the interrupt routine is already written. However, the routine header syntax and some variables are hardware, microprocessor, and compiler specific and must be written for the specific configuration.

This routine should be invoked every time a serial interrupt signal is asserted. This interrupt should be enabled on power-up. It is necessary to ensure the interrupt vector is initialized within the application.

When using the Borland C compiler, the interrupt routine header is defined as follows:

```
void interrupt _HandleSerialInterrupt(void)
```

The application specific power-up initialization code should set the *_HandleSerialInterrupt()* routine to function as the serial interrupt.

5.2.2.6 suc.h.1

Code Section *suc.h.1* should contain the activation interval timer initialization value. The ACTIVATION_INTERVAL macro determines the time to complete the activation state diagram in seconds. The initialization value is dependent on the application symbol rate. This value should be modified so the activation does not prematurely time out during a successful attempt or does not take excessive time to time out during an unsuccessful attempt.

The activation interval timer uses the SUT4 activation timer (symbol rate / 1024). The _SYM_RATE API command sets the 'symbol_rate' variable to symbol rate / 4096. Therefore, the value is 4 * desired activation time in seconds * the 'symbol_rate' variable.

Example:

For a desired activation interval timer of 50. The ACTIVATION_INTERVAL macro needs to be programmed to 4 * 50 * symbol_rate, or 4 * (32 + 16 + 2) * symbol_rate. The macro is implemented as follows:

((((short) symbol_rate << 5) + ((short)symbol_rate << 4) + ((short)symbol_rate)<<1)) << 2)

5.2.2.7 user.c.1

Code section *user.c.1* should contain the absolute I/O address definitions of the specified number of Bt8960 chips as defined by the compiler flags. Any value may be defined for non-existent Bt8960 chips.

The address value depends on the address decoding scheme implemented in the system. The defined address should reflect the value that accesses byte #0 on the Bt8960.

5.2.2.8 user.c.2

Code section *user.c.2* should contain the noise margin calibration table. This table must be modified by the user to be accurate when using different noise source types. Refer to Appendix B for details.

5.2.2.9 user.c.3

Code section *user.c.3* contains code that initializes the C pointers used to access the Bt8960 chips. The initialization of a pointer with an absolute external address value is done differently in some C compilers.

The values of the array should be initialized as follows:

- _bit_pump[_BIT_PUMP0] should be initialized with the external address BIT_PUMP0_ADD.
- _bit_pump[_BIT_PUMP1] should be initialized with the external address BIT_PUMP1_ADD.
- _bit_pump[_BIT_PUMP2] should be initialized with the external address BIT_PUMP2_ADD.

In an environment that allows direct assignment of absolute addresses to pointer variables, the *user.c.3* code section should be written as follows:

```
_bit_pump[_BIT_PUMP0] = BIT_PUMP0_ADD;
_bit_pump[_BIT_PUMP1] = BIT_PUMP1_ADD;
_bit_pump[_BIT_PUMP2] = BIT_PUMP2_ADD;
```

When using the Borland C compiler, initializing pointer addresses with an absolute value is done using a special C macro in the following way:

```
_bit_pump[_BIT_PUMP0] = MK_FP(BIT_PUMP_SEGMENT,
                BIT_PUMP0_ADD);
_bit_pump[_BIT_PUMP1] = MK_FP(BIT_PUMP_SEGMENT,
                BIT_PUMP1_ADD);
_bit_pump[_BIT_PUMP2] = MK_FP(BIT_PUMP_SEGMENT,
                BIT_PUMP2_ADD);
```

This assumes all bit-pumps use the same segment, defined by `BIT_PUMP_SEGMENT`, and the bit-pump address constants define the address offset of each bit-pump.

5.2.2.10 user.c.4

The contents of *_Delay2Symbols()* routine are application specific. This routine should implement a delay of 2 symbol periods.

NOTE: Depending on the microprocessor and compiler, entering and exiting this routine may generate the required delay with no need for additional operations or NOPs.

5.2.2.11 user.c.5

The code section *user.c.5* contains the body of the routine *_EnableUserInterrupts()*, and is application specific. This routine, when executed, enables all the non-Bt8960 interrupts that are disabled by *_DisableUserInterrupts()* (see *user.c.6*). The Bt8960 interrupt is never disabled, so there is no need to enable it. The two routines *_DisableUserInterrupts()* and *_EnableUserInterrupts()* are called by the Bt8960 software in critical activation phases.

5.2.2.12 user.c.6

The code section *user.c.6* contains the body of the routine *_DisableUserInterrupts()*, and is application specific. The Activation sequence can tolerate 4 mS out of 6 mS of interrupts before performance degradation is noticed. This routine, when executed, should disable all the non-Bt8960 interrupts that cause this time criteria to fail. The Bt8960 interrupt is always enabled. The *_DisableUserInterrupts()* and *_EnableUserInterrupts()* routines are called by the Bt8960 software in critical activation phases.

5.2.2.13 api.c.1

5.2.2.14 api.c.2

5.2.2.15 api.c.3

5.2.2.16 api.h.1

In the API.C and API.H, the Channel Unit API hooks (found within the CHAN_UNIT compiler flags) can be modified to allow the user to define their own API commands. Please refer to the User Interface: Code Level Interface section for details.

6.0 User Interface

This chapter describes the User Interface (UI) to the Bt8960 HDSL bit-pump software. This interface allows the system designer to have full control over all of Bt8960 features, and get status and performance monitoring information from the bit-pump.

The interaction between the application and the Bt8960 may be achieved in one of two configurations, depending on whether the application software is implemented on the same microprocessor as the Bt8960 control software or on a different microprocessor. When only one microprocessor is used in a system, the application should interface the bit-pumps using a code-level Application-Program Interface (API). In a design in which the Bt8960 software runs on an 8032 microprocessor, and a second "host" microprocessor is used for the application software, a serial communication protocol between the two processors is used to control the bit-pumps.

Figure 6-1 and Figure 6-2 show the two possible control configurations.

Figure 6-1. Two Microprocessors Implementation

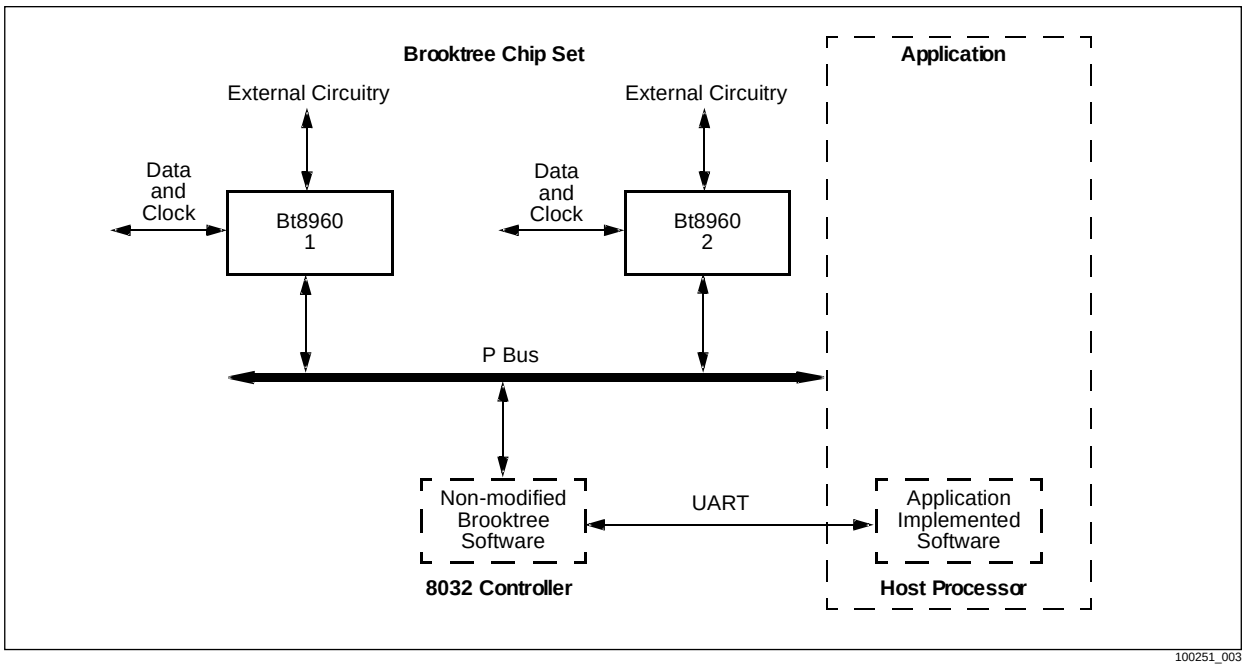
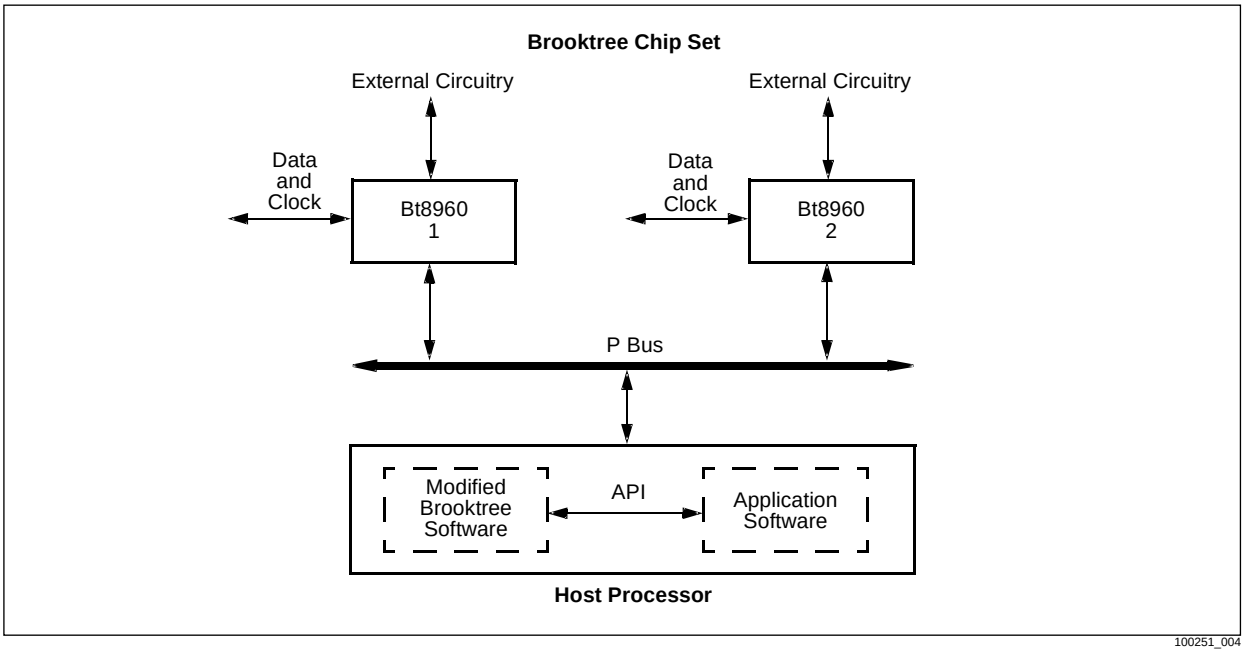


Figure 6-2. Single Microprocessor Implementation



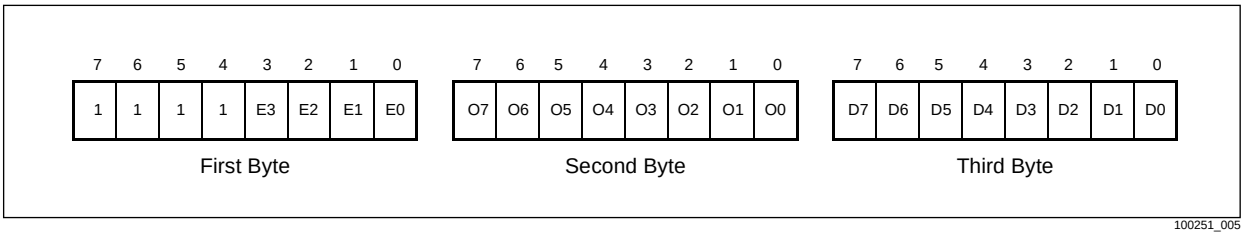
In both configurations described above, the control and monitoring operations are based on a set of commands sent by the application to the bit-pump control software and parameter status sent back to the application. The command set itself is identical in both system configurations.

The Command Structure section describes the structure of commands and status responses, which applies to both UI configurations. The Serial Communication Interface section describes the serial interface message structure and communication protocol which applies to a dual processor system architecture, and the Code Level Interface (API) section describes the API which applies to a single processor architecture. The Opcodes and Parameters section lists the available commands. Appendix A (Command Set Reference) supplies a detailed description of the command set including command syntax, operation, and application hints.

6.1 Command Structure

The structure of all commands is the same. A command is composed of three 1-byte fields: Destination, Opcode, and Data. Figure 6-3 shows the structure of a command.

Figure 6-3. Command Structure



The command fields are interpreted according to the following fields.

- Destination field (bits E3–E0)—This field selects the destination to which the command is targeted, according to Table 6-1.

Table 6-1. Destination Field Specification

E3	E2	E1	E0	Destination
0	0	0	0	Bt8960 #0
0	0	0	1	Bt8960 #1
0	0	1	0	Bt8960 #2
0	0	1	1	Bt8960 #3
0	1	0	0	Bt8960 #4
0	1	0	1	Bt8960 #5

- Opcode field (bits O7–O0)—The Opcode field selects the specific command or status request to be executed. The available commands and their opcodes are described in detail in Appendix A. The opcodes are also available in C source file api.h, which contains C constant definitions for all opcodes. The Opcodes and Parameters section contains a list of these opcode constants.
- Data (Parameter) field (bits D7–D0)—This field is used in some commands where additional data or parameter selection is required. In commands where there is no need for additional data, zeros should be placed as the data byte to ensure future compatibility. The data field options are also available in C source file *api.h*, which contains C constant definitions for the available parameters. The "Opcodes and Parameters" section contains a list of these data field constants.

The set of commands is divided into three logical groups: control commands, status request commands, and special messages. The first group includes all commands that control the operation of the bit-pump and set different parameters. The second group includes commands that request status values or monitoring information from the bit-pump. In response to a status request command, a 1-byte data word is returned to the application supplying the required information. The special messages include the Acknowledge message (applies only to serial communication UI).

6.2 Serial Communication Interface

In a dual processor system (see Figure 6-1), the application controls the bit-pumps using a serial communication message transfer protocol. The protocol is based on the command set described in Appendix A, and the command structure as described in the Command Structure section, with additional requirements for this type of interface.

6.2.1 Communication Protocol

The 8032 communicates with the host processor using a standard UART interface. The physical connection includes two lines: RXD (8032 pin 10) and TXD (8032 pin 11). The data is transferred in an asynchronous format: 9600 baud, 1 start bit, 8 data bits, 1 stop bit, no parity.

To select a separate baud rate, refer to the Compiling the Source Code" chapter.

6.2.2 Message Transfer Protocol

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

The 8032 is guaranteed to acknowledge a received message within 200 mS, except during activation where larger delays (up to 2 seconds) may be present. The host processor will usually retransmit a message that was not acknowledged within this time limit. No new message should be sent by the host processor before the previous one was acknowledged unless the time limit has been exceeded.

A status request message requiring information from the bit-pump will first be acknowledged, and only then will a response message containing the requested information be sent to the host processor.

6.2.3 Message Structure

All messages are 4 bytes long. Figure 6-4 shows the structure of a message sent by the host processor to the 8032. The first 3 bytes are the command bytes, as described in the Command Structure section. The fourth byte (the last transmitted byte) contains checksum information that is a function of the first 3 bytes (see the Checksum Function subsection for checksum function details). The checksum considerably reduces the probability of the 8032 misinterpreting an incoming message.

When the 8032 receives a status request command, it responds (after acknowledging the command) by sending a status message to the host processor. The structure of a status message is similar to the structure of a host-to-8032 message, and is shown in Figure 6-5.

The first 2 bytes are identical to the first 2 bytes of the corresponding status request command. Bits S3–S0 of the first byte are interpreted, according to Table 6-1, as the source bit-pump for the status response. The second opcode byte (bits O7–O0) contains the opcode of the command that requested the information.

The third byte (bits D7–D0) contains the requested information. The fourth byte (bits CS7–CS0) is the checksum value, calculated according to the formula described in the Checksum Function subsection.

Figure 6-4. Host Processor to 8032 Message Structure

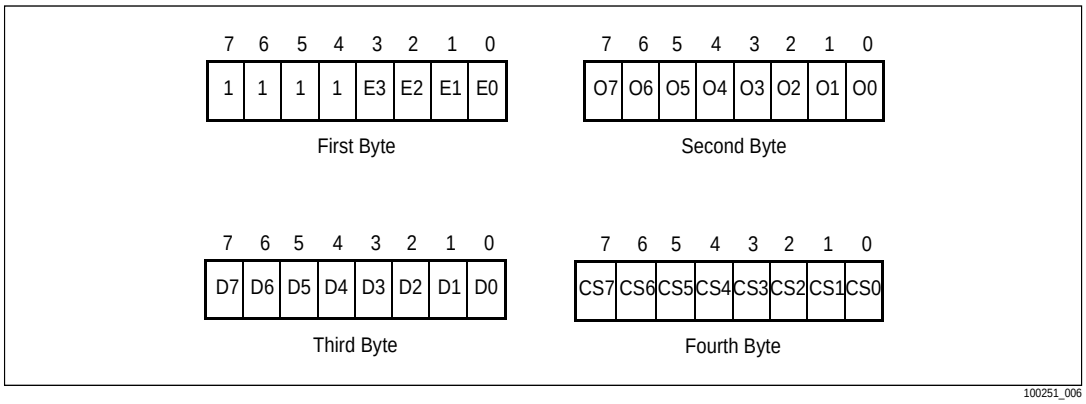
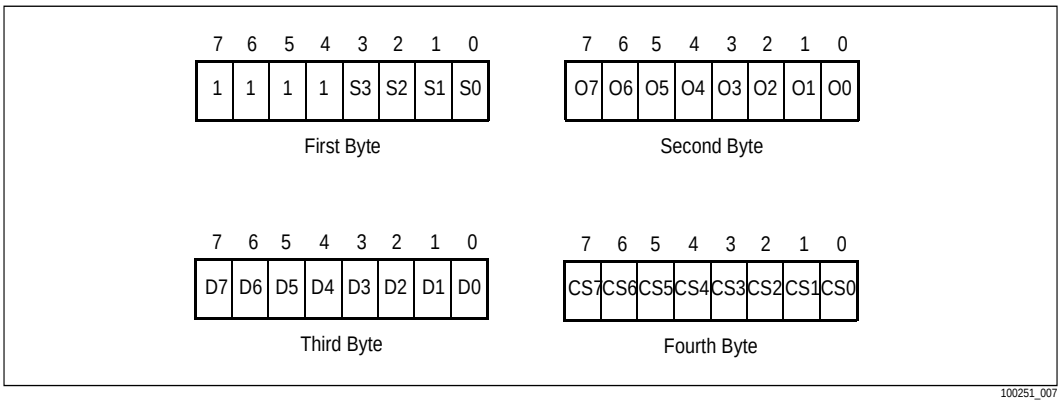


Figure 6-5. 8032 to Host Processor Message Structure



6.2.4 Checksum Function

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

```
CS = (Byte #1) | (Byte #2) | (Byte #3) | (0xAA);
```

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

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

6.3 Code Level Interface (API)

The code level API is intended for applications integrating the bit-pump control software in a single microprocessor system (see Figure 6-2). In such a configuration, the application issues a command and gets information from the bit-pumps by calling a specific routine.

To issue any one of the control commands (see the Control Commands section in Appendix A), the application should call the *_BtControl()* function. To issue a status request command (see the Serial Communication Interface section), the *_BtStatus()* function should be called.

6.3.1 _BtControl()

The *_BtControl()* routine is used to issue any one of the control commands specified in the Control Commands section in Appendix A.

Use	unsigned char _BtControl(unsigned char destination, unsigned char opcode, unsigned char parameter);
Description	The destination, opcode, and data parameters correspond to the three command bytes as described in the Command Structure section. The opcode field must contain a value corresponding to one of the control commands specified in the Control Commands section in Appendix A
Effect	The required command is executed on the bit-pump designated by the destination parameter (unless an error condition occurred).
Return Value	*_PASS (0x00)* indicates a successfully interpreted command. The command is executed. *_FAIL (0x01)* indicates an error condition. The command is not executed. An error condition may be caused by an illegal value in any one of the command fields.
Example	To configure bit-pump #2 as a remote terminal (HTU-R), execute the following routine call: _BtControl(_BIT_PUMP2, _TERMINAL_TYPE, _HTUR);

6.3.2 _BtStatus()

The *_BtStatus()* routine is used to issue any one of the status request commands specified in the Status Request Commands section in Appendix A.

Use	unsigned char _BtStatus(unsigned char destination, unsigned char opcode, unsigned char parameter, char indication);
Description	The destination, opcode, and data parameters correspond to the three command bytes as described in the Command Structure section. The opcode field must contain a value corresponding to one of the status request commands specified in the Special Messages section in Appendix A.
Effect	The status parameter is set to the required value (unless an error condition occurred).
Return Value	*_PASS (0x00)* indicates a successfully interpreted command. The value of the status parameter is correctly set.
Example	*_FAIL (0x01)* indicates an error condition. The value of the status parameter has no meaning and should be ignored. An error condition may be caused by an illegal value in any of the command fields.

6.3.3 User Defined API Commands - Channel Unit & Framer

The code level API structure provides hooks to allow the user to access their own user defined API commands. In the dual processor mode, this allows the user to use the Serial Communication Interface message protocol to handle the user defined API commands. The user then only needs to create (define) their own API commands.

The bit-pump code provides hooks to access the Channel Unit and Framer API commands, these are to support the Brooktree EVM Systems. However, the user may rename, add, or delete to these hooks.

```
_CuControl()
_CuStatus()
_FramerControl()
_FramerStatus()
```

These function descriptions are identical to the _BtControl() and _BtStatus() functions. The Channel Unit EVM has the following Destination field values (see also Table 3):

Table 6-2. Destination Field Values

E3	E2	E1	E0	Destination	#define
0	0	1	1	Channel Unit Common Sections	_CU_COMMON
0	1	0	0	Channel Unit HDSL Loop #1 Specific	_CU_CHAN1
0	1	0	1	Channel Unit HDSL Loop #2 Specific	_CU_CHAN2
0	1	1	0	Channel Unit HDSL Loop #3 Specific	_CU_CHAN3
0	1	1	1	E1/T1 Framer and LIU	_FRAMER

The channel Unit and Framer destination fields are numbered 3 to 7 to maintain compatibility with existing Brooktree EVM System and Support Programs. However, these destination fields will cause a conflict when the #FOUR_LOOPS or #SIX_LOOPS complier flags are specified, since the #FOUR_LOOPS or #SIX_LLPS flags specify the destination field between (0 and 3) or (0 to 5). If the user wishes to use the #CHAN_UNIT compiler flag with the #FOUR_LOOPS or #SIX_LOOPS flags, then the Channel Unit destination fields must be modified.

NOTE: The Brooktree EVMs are typically compiled with the #THREE_LOOPS and #CHAN_UNIT complier flags

6.4 Opcodes and Parameters

The source file *api.h* contains C constant definitions for the opcodes of all commands, and for the possible data field values of commands that use this field.

NOTE: These definitions should be employed in application code for the Bt8960 UI.

Table 6-3 lists the available commands, each with its corresponding C constant opcode and parameter definitions (where applicable). See Appendix A for details on command use, operation, and parameter setting.

Table 6-3. User Interface Command Summary *(1 of 3)*

Command Name	Opcode C Constants	Data Field Options	Data Field C Constants
Terminal Type	_TERMINAL_TYPE	HTU-R HTU-C	_HTUR _HTUC
Analog AGC Configuration	_ANALOG_AGC_CONFIG	No AGC 2-Level discrete AGC 4-Level discrete AGC 6-Level discrete AGC	_NO_AGC _TWO_LEVEL_AGC _FOUR_LEVEL_AGC _SIX_LEVEL_AGC
Start-up Sequence Source	_STARTUP_SEQ_ SOURCE	Internal start-up sequence External start-up sequence	_INTERNAL _EXTERNAL
Transmit Scrambler	_TRANSMIT_SCR	Scramble TX symbols Do not scramble TX symbols	_ACTIVATE_SCR _BYPASS
Receive Descrambler	_RECEIVE_DESCR	Descramble RX symbols Do not descramble RX symbols	_ACTIVATE_DESCR _BYPASS
Data Transfer Format	_FRAMER_FORMAT	Parallel data, clock outputs Parallel data, clock inputs Serial data Serial data, sign & magnitude swapped	_PARALLEL_MASTER _PARALLEL_SLAVE _SERIAL _SERIAL_SWAP
Other Side Bt8960	_BT_OTHER_SIDE	Bt8960 bit-pump used on other terminal Non-Bt8960 bit-pump on other terminal	_BT _NO_BT
LOST Time Period	_LOST_TIME_PERIOD	1-Byte unsigned integer	
Bit-pump ON/OFF	_SYSTEM_CONFIG	Bit-pump On Bit-pump OFF	_PRESENT _NOT_PRESENT
Transmit External Data	_TRANSMIT_EXT_DATA	0	
Activate	_ACTIVATE	0	
Deactivate	_DEACTIVATE	0	

Table 6-3. User Interface Command Summary *(2 of 3)*

Command Name	Opcode C Constants	Data Field Options	Data Field C Constants
Test Mode	_TEST_MODE		_EXIT_TEST_MODE _ANALOG_LOOPBACK _NEAR_LOOPBACK _FAR_LOOPBACK _ISOLATED_PULSE_PLUS3 _ISOLATED_PULSE_PLUS1 _ISOLATED_PULSE_MINUS1 _ISOLATED_PULSE_MINUS3 _FOUR_LEVEL_SCR _TWO_LEVEL_SCR _VCXO_NOMINAL _VCXO_MIN _VCXO_MAX _INTERNAL_ANALOG_ LOOPBACK _EXTERNAL_ANALOG_ LOOPBACK
Symbol Rate	_SYM_RATE	1-Byte unsigned integer	
Reset	_RESET_SYSTEM	0	
Enable/Disable Non-Linear EC	_OPERATE_NLEC	Enable NL EC Operation Disable NL EC Operation	_NLEC_ON _NLEC_OFF
Write Transmitter Gain register	_WRITE_TX_GAIN	1-Byte signed integer	
Input Signal Level	_SLM	0	
Input DC Offset	_DC_METER	0	
Far-end Signal Attenuation	_FELM	0	
Noise Margin	_NMR	0	
VCXO Control Voltage	_VCXO_CONTROL_ VOLTAGE	0	
Bit-pump Status	_STARTUP_STATUS	0	
AAGC Control Bits Value	_AAGC_VALUE	0	
Read Linear EC Coefficient	_LEC_COEFF	Coefficient index	
Read NL EC Coefficient	_NLEC_COEFF	Coefficient index	
Read EQ Coefficient	_EQ_COEFF	Coefficient index	
Read DFE Coefficient	_DFE_COEFF	Coefficient index	

Table 6-3. User Interface Command Summary *(3 of 3)*

Command Name	Opcode C Constants	Data Field Options	Data Field C Constants
Software / Chip Versions	_VERSION	Major SW & Bitpump Version Major SW Version Minor SW Version Bitpump Type & Version	_HW_SW_VERSIONS _MAJOR_SW_VERSION _MINOR_SW_VERSION _HW_TYPE_VERSIONS
Bit-pump Present	_BIT_PUMP_PRESENT	0	
Self-test	_SELF_TEST	0	
Read Bt8960 Register	_REGISTER	Bt8960 register address	
Bit-pump Configuration	_CONFIGURATION	Request low byte of user setup Request high byte of user setup Request LOST time period information Request symbol rate information	_USER_SETUP_LOW_BYTE _USER_SETUP_HIGH_BYTE _LOST _BIT_RATE
Stage Number	_STAGE_NUMBER	0	
Read Transmitter Gain register	_READ_TX_GAIN	Request Tx Calibration Value Request Tx Gain Value	_CALIBRATION _GAIN

Appendix A Command Set Reference

This appendix contains a reference guide to the Bt8960 User Interface command set. All available commands are listed, giving information on command use, application, syntax, and options.

The set of commands is divided into three groups:

1. Control commands—Control the operation modes of the bit-pump and set various parameters.
2. Status request commands—Inquire for status and monitoring information from the bit-pump. A 1-byte response containing the required information is transferred back.
3. Special Commands.

A.1Control Commands

For each command, a description of its operation, opcode, and options is given. Commands that select bit-pump operational options and set bit-pump parameters have their default value underlined. The default values are also given in the Setting Bit-pump Parameters section.

A.1.1 Terminal Type

Sets the bit-pump terminal type. In any operational HDSL system, a bit-pump on one side of the loop should be defined to be an HTU-C (HDSL Terminal Unit – Central office side) and the bit-pump on the other side should be defined to be an HTU-R (Remote terminal). The two terminal types differ in their activation procedure, timing recovery mechanism, scrambler/descrambler taps, and more. Terminal type must be properly defined prior to any activation operation.

In a multipair system, each bit-pump may be individually set as an HTU-C or HTU-R (although in most implementations all bit-pumps will be set to the same terminal type).

Opcode	
Numeric Value	**C Constant (defined in file api.h)**
0x01	_TERMINAL_TYPE

Options			
Option	**Description**	**Data Field**	**C Constant (api.h)**
HTU-C	Central Office Terminal	0x00	_HTUC
HTU-R	Remote Terminal	0x01	_HTUR

A.1.2 Analog AGC Configuration

Informs the bit-pump software of the analog AGC configuration implemented in the design.

Opcode	
Numeric Value	**C Constant (defined in file api.h)**
0x02	_ANALOG_AGC_CONFIG

Options			
Option	**Description**	**Data Field Value**	**C Constant (api.h)**
No AGC	No analog AGC is implemented, and constant receiver gain is used.	0x00	_NO_AGC
2-Step AGC	Discrete analog AGC is used, with two possible gain levels. The gain value is controlled by pin AGC[0].	0x01	_TWO_LEVEL_AGC
4-Step AGC	Discrete analog AGC is used, with four possible gain levels. The gain value is controlled by pins AGC[0] (LSB) and AGC[1].	0x02	_FOUR_LEVEL_AGC
6-Step AGC	Discrete analog AGC is used, with six possible gain levels. The gain value is controlled by pins AGC[0] (LSB), AGC[1], and AGC[2].	0x03	_SIX_LEVEL_AGC

The Bt8960 bit-pump internally supports six different analog AGC configurations as shown in Table A-1. Three internal gain control signals are supplied by the Bt8960. In a 2-level discrete AGC configuration, pin AGAIN[0] is used to select between two discrete gain values, where a 0 selects the lower gain. In a 4-level discrete AGC configuration, pin AGAIN[0] and AGAIN[1] are used to select between four discrete gain values, AGAIN[0] being the LSB. A '00' selects the lowest gain. In a 6-level discrete AGC configuration, pin AGAIN[0], AGAIN[1] and AGAIN[2] are used to select between six discrete gain values, AGAIN[0] being the LSB. A '000' selects the lowest gain.

In a 6-level AAGC implementation, the recommended relative gain settings are 0 dB, 3 dB, 6 dB, 9 dB, 12 dB, and 15 dB. In a 4-level AAGC implementation, the recommended relative gain settings are 0 dB, 3 dB, 6 dB, and 9 dB. In a 2-level AAGC implementation, the recommended relative gain settings are 0 dB and 6 dB. When no AGC is implemented, the fixed gain is referred to as the 0 dB level.

NOTE: The 6-Level AGC setting should be set for optimal performance.

Table A-1. Bt8960 AGAIN Settings

C Constants	Numeric Value	AGAIN[2]	AGAIN[1]	AGAIN[0]
_AGAIN0DB	0	0	0	0
_AGAIN3dB	1	0	0	1
_AGAIN6DB	2	0	1	0
_AGAIN9DB	3	0	1	1
_AGAIN12DB	4	1	0	0
_AGAIN15DB	5	1	0	1

A.1.3 Start-up Sequence Source

Selects the the source of activation sequences to be external or internal. The Bt8960 provides for internally generated activation sequences that may be used in a complete stand-alone transceiver implementation. The internally generated activation sequences are 2- or 4-level scrambled 1s, with no HDSL framing information. These sequences do not meet the standard activation requirements which require HDSL framing information to be included in the activation sequences.

If a framed activation sequence is required, it must be supplied prior to any activation operation and the start-up sequence source must be set to external. Only a 4-level sequence need be supplied. The bit-pump ignores the magnitude bit during the initial 2-level transmission.

NOTE: When the internal option is specified, the _TRANSMIT_EXT_DATA command must be called when the activation process is successfully completed.

Opcode	
Numeric Value	**C Constant (defined in file api.h)**
0x03	_STARTUP_SEQ_SOURCE

Options			
Option	**Description**	**Data Field**	**C Constant (api.h)**
External Start-up Sequence	Start-up sequences are supplied externally. A 4-level activation sequence must be supplied before activation is initiated.	0x00	_EXTERNAL
Internal Start-up Sequence	Start-up sequences are generated internally. These sequences are 2/4 level scrambled 1's, with no HDSL framing.	0x01	_INTERNAL

A.1.4 Transmit Scrambler

Activates or bypasses the internal transmit scrambler. The internal transmit scrambler (and receive descrambler) may be used for stand-alone operation of the bit-pump. When activated, all incoming data is scrambled and then converted to quaternary symbols for transmission.

Opcode	
Numeric Value	**C Constant (defined in file api.h)**
0x04	_TRANSMIT_SCR

Options			
Option	**Description**	**Data Field**	**C Constant (api.h)**
Bypass	The transmit scrambler is not used and the symbols supplied by the application are transmitted with no change.	0x00	_BYPASS
Active	The transmit scrambler is activated, and the bit stream supplied by the application is scrambled on chip before being transmitted. Standard scrambler taps are used (according to terminal type setting). The TX scrambler is operated (when turned ON) only during 4-level transmission, i.e., during 4-level activation transmission and during normal operation.	0x01	_ACTIVATE_SCR

A.1.5 Receive Descrambler

Activates or bypasses the internal receive descrambler. The internal receive descrambler may be used for stand-alone operation of the bit-pump. When activated, all detected symbols are converted to bits and descrambled, prior to being transferred to the data output pins. Note that all received bits are descrambled, including framing and overhead bits.

Opcode	
Numeric Value	**C Constant (defined in file api.h)**
0x05	_RECEIVE_DESCR

Options			
Option	**Description**	**Data Field**	**C Constant (api.h)**
Bypass	The receive descrambler is not used and the detected symbols are supplied with no change.	0x00	_BYPASS
Active	The receive descrambler is activated and the received bit stream is descrambled before being transferred.	0x01	_ACTIVATE_DESCR

A.1.6 Data Transfer Format

Selects the format in which data is transferred between the Bt8960 and the application. This option has no effect on data value, only the data transfer format is affected. Different data transfer formats allow for different schemes of framer bit-pump clock distribution. For more details, see the Bt8960 datasheet.

Opcode	
Numeric Value	**C Constant (defined in file api.h)**
0x06	_FRAMER_FORMAT

Options			
Option	**Description**	**Data Field**	**C Constant (api.h)**
Parallel data, clock outputs	The Bt8960 supplies a baud rate clock signal (QCLK) that times the data transfer in both the receive and transmit directions. Received quats are being transferred to the framer via the RQ[0], RQ[1] pins. Transmitted quats are being transferred to the bit-pump via the TQ[0], TQ[1] pins.	0x00	_PARALLEL_MASTER
Parallel data, clock inputs	A baud rate clock signal (RBCLK) is supplied that times the data transfer in the receive direction. Also supplied is a separate baud rate clock (TBCLK) that times the data transfer in the transmit direction. Received and transmitted quats are transferred via the TQ and RQ signals, as in "Parallel with clock outputs" mode. The RBCLK and TBCLK signals must be a derivative of the Bt8960's 16X clock at an arbitrary phase.	0x01	_PARALLEL_SLAVE
Serial data	The Bt8960 supplies a baud rate clock (on the QCLK pin) and a bit rate clock (on the RQ[0] pin) that time the data transfer in both receive and transmit directions. The received and transmitted quats are transferred serially, each on a single line, via the RQ[1] and TQ[1] pins. The 2B1Q magnitude bit is aligned to QCLK low and the 2B1Q sign bit is aligned to QCLK high.	0x02	_SERIAL
Serial Swap data	The Bt8960 supplies a baud rate clock (on the QCLK pin) and a bit rate clock (on the RQ[0] pin) that time the data transfer in both receive and transmit directions. The received and transmitted quats are transferred serially, each on a single line, via the RQ[1] and TQ[1] pins. The 2B1Q magnitude bit is aligned to QCLK high and the 2B1Q sign bit is aligned to QCLK low. This format satisfies the ETSI/ANSI requirements for output quat orientation	0x03	_SERIAL_SWAP

A.1.7 Other Side Bt8960

In a system where both terminals use Bt8960 bit-pumps, several activation operations can be performed more efficiently relative to the standard requirements. This command is used to inform the Bt8960 software that the other terminal uses a Bt8960 bit-pump.

Opcode	
Numeric Value	**C Constant (defined in file api.h)**
0x07	_BT_OTHER_SIDE

Options			
Option	**Description**	**Data Field**	**C Constant (api.h)**
Other side unknown	The terminal on the other loop end is not known to use the Bt8960.	0x00	_NO_BT
Other side Bt8960	The terminal on the other side of the loop employs a Bt8960.	0x01	_BT

A.1.8 LOST Time Period

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

The LOST mechanism is active only after a Deactivate command is issued. 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 0–31 seconds with a resolution of 1/10 second. The value of the LOST status bit may be checked using the Bit-pump Status command.

Opcode	
Numeric Value	**C Constant (defined in file api.h)**
0x08	_LOST_TIME_PERIOD

Options			
Option	**Description**	**Data Field**	**C Constant (api.h)**
LOST time period	1-byte unsigned integer X. The LOST time period is set to X/10 seconds. Default Value: 10 = 1 second.	X	

A.1.9 Bit-pump ON/OFF

Informs the Bt8960 software which bit-pumps are active in the system. Setting a bit-pump state to OFF causes the Bt8960 software to put the chip in a "power-down" mode and ignore any further control commands issued to this bit-pump, other than Bit-pump ON/OFF.

All bit-pumps that are intended to be activated should be turned ON prior to any other control operation. The bit-pumps are numbered from 0 to 5. Any of the bit-pumps may be used in a system (note that a hardware implementation may include, for example, three bit-pumps, only two of which are turned ON).

Opcode	
Numeric Value	**C Constant (defined in file api.h)**
0x09	_SYSTEM_CONFIG

Options			
Option	**Description**	**Data Field**	**C Constant (api.h)**
OFF	Turn bit-pump OFF	0x00	_NOT_PRESENT
ON	Turn bit-pump ON	0x01	_PRESENT

A.1.10 Transmit External Data

This command should be used only when the activation sequence source is internal. 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 and framer status responses. The transmitted data should be supplied to the bit-pump prior to issuing this command. Avoid situations where a long stream of constant value symbols is transmitted.

Opcode	
Numeric Value	**C Constant (defined in file api.h)**
0x0A	_TRANSMIT_EXT_DATA

Options			
Option	**Description**	**Data Field**	**C Constant (api.h)**
None		0x00	

A.1.11 Activate

Initiates the activation process. On an HTU-C terminal, this command causes 2-level activation sequence transmission to begin. On an HTU-R terminal, this command causes the bit-pump to wait for the detection of an incoming signal, and continue with the activation process when such a signal is detected. The activation process itself is fully automatic. An application may inquire about the activation status, SNR, etc., but otherwise no actions are required.

Opcode	
Numeric Value	**C Constant (defined in file api.h)**
0x0B	_ACTIVATE

Options			
Option	**Description**	**Data Field**	**C Constant (api.h)**
None		0x00	

A.1.12 Deactivate

Turns the transmitter off and stop all activation operations regardless of the current bit-pump status. The bit-pump goes to an IDLE state, where it awaits further commands. Issuing the Deactivate command enables the LOST mechanism (see the LOST Time Period command). If issued during normal operation, the receiver continues to function properly (useful for test purposes).

Opcode	
Numeric Value	**C Constant (defined in file api.h)**
0x0C	_DEACTIVATE

Options			
Option	**Description**	**Data Field**	**C Constant (api.h)**
None		0x00	

A.1.13 Test Mode

Operates the bit-pump in special test modes. These modes include maintenance operating modes and loopback configurations. To execute a bit-pump self-test procedure, use the Self-test status request command. To turn off any of the special test modes, use the Test Mode command with an _EXIT_TEST_MODE (0 value) parameter. All test modes except the two digital loopback modes require, after exiting the test mode, a complete activation procedure to be repeated (assuming normal operation is required). When exiting all these test modes (by issuing the Test Mode command with a 0 value parameter), the bit-pump is initialized to a reset state, and goes to the IDLE state, where it awaits further commands.

The two digital loopback test modes do not interfere with the operation of the bit-pump, and only affect the received/transmitted symbol stream. Therefore, operating the bit-pump in a digital (near/far) loopback during normal operation will not cause loss of synchronization. Exiting one of these two test modes will turn off the loopback operation, allowing for normal data transfer to continue with no need for a restart. Table A–A-2 summarizes the action taken when exiting different test modes.

NOTE: When exiting a "digital far loopback" mode, the transmitted symbol sequence source depends on the setting of the "internal/external activation sequence" flag.

Table A-2. Exiting Test Modes

Test Mode	Action Taken When Exiting Test Mode	How to Go Back to Normal Data Transfer(1)
Digital Near Loopback	RX symbol stream (bit-pump to framer) is sent back to be the detected symbols sequence.	Exit test mode.
Digital Far Loopback	External activation sequence mode: Transmit external data (TX scrambler is operated/bypassed according to the Transmit Scrambler mode setting).	Exit test mode.
	Internal activation sequence mode: Transmit internally generated 4-level scrambled 1's sequence.	Exit test mode. Issue Transmit External Data command to start transmitting the externally supplied (payload) data.
All Other Test Modes	Bit-pump initialized to a reset state, and goes to an IDLE state.	Complete activation operation is required.
(1) Assuming normal data transfer took place before issuing test mode		

Opcode	
Numeric Value	**C Constant (defined in file api.h)**
0x0D	_TEST_MODE

Options			
Option	**Description**	**Data Field**	**C Constant (api.h)**
Exit Test Mode	Cancel test mode. See Table A-1.	0x00	_EXIT_TEST_MODE
External Analog Loopback	Transmit the externally supplied TX symbols. Use the echo signal as a "received" signal and detect the symbols using the standard equalizer.	0x01	_ANALOG_LOOPBACK
Digital "Near" Loop Back	TX symbols supplied to the bit-pump by the framer are looped back as the RX symbols going from the bit-pump to the framer. Useful for testing the framer and the framer/bit-pump connection.	0x02	_NEAR_LOOPBACK
Digital "Far" Loop Back	Detected RX symbols are transmitted back on the loop. Useful for testing full 2-way transmission over a loop.(1)	0x03	_FAR_LOOPBACK
Transmit Isolated +3 Pulse	Transmit (repeatedly) an isolated +3 level pulse. Useful for testing the transmitted pulse shape.	0x04	_ISOLATED_PULSE_PLUS3
Transmit Isolated +1 Pulse	Transmit (repeatedly) an isolated +1 level pulse	0x05	_ISOLATED_PULSE_PLUS1
Transmit Isolated –1 Pulse	Transmit (repeatedly) an isolated –1 level pulse	0x06	_ISOLATED_PULSE_MINUS1
Transmit Isolated –3 Pulse	Transmit (repeatedly) an isolated –3 level pulse	0x07	_ISOLATED_PULSE_MINUS3
Continuous 4-level Transmission	Continuous transmission of a 4-level scrambled 1s sequence (internally generated). Useful for measuring transmitted power and spectral shape.	0x08	_FOUR_LEVEL_SCR
Continuous 2-level Transmission	Continuous transmission of a 2-level scrambled 1s sequence (internally generated).	0x09	_TWO_LEVEL_SCR
Set Nominal VCXO Frequency	Set VCXO control word to its nominal value. Useful for measuring VCXO center frequency.	0x0A	_VCXO_NOMINAL
Set Minimum VCXO Frequency	Set VCXO control word to its minimum value.	0x0B	_VCXO_MIN
Set Maximum VCXO Frequency	Set VCXO control word to its maximum value.	0x0C	_VCXO_MAX
Internal Analog Loop Back	Transmits the externally supplied Tx symbols out the TXP and TXN pins and detects the symbols on the hybrid inputs (RXBP, RXBN), the receive inputs (RXP and RXN) are bypassed.	0x0D	_INTERNAL_ANALOG_ LOOPBACK
Isolated Analog Loop Back	The externally supplied Tx symbols are internally looped back in the Bt8960. The transmitter (TXP and TXN) is turned off (silent).	0x0E	_ISOLATED_ANALOG_ LOOPBACK

(1) The operation of a digital "far" loopback requires activating the Bt8960 internal TX scrambler and RX descrambler.

A.1.14 Symbol Rate

Informs the Bt8960 software of the bit-pump symbol rate. This value is used to convert seconds to symbols when absolute time interval measurements are performed using the symbol rate on-chip timers. This value must be programmed for proper bit-pump operation.

NOTE: If the _SYM_RATE command is issued, then it must be called immediately after the _SYSTEM_CONFIG ON command. The _SYM_RATE command sets the PLL Clock Center Frequency. There is a potential for the bit-pump registers to get corrupted when the PLL Clock Frequency is changed. The _SYM_RATE command initializes the bit-pump registers to the default state after writing the PLL Clock Center Frequency. Therefore, the user must issue their application specific API Command Values after the _SYM_RATE command is issued.

Opcode	
Numeric Value	**C Constant (defined in file api.h)**
0x0E	_SYM_RATE

Options			
Option	**Description**	**Data Field**	**C Constant (api.h)**
Symbol rate	1-Byte unsigned integer X, calculated according to: X=Symbol Rate / 4096. Default Value: 35=144 Ks/sec.	X	

A.1.15 Reset

Loads bit-pump microcode, reset all bit-pump internal registers, and set all user-programmable options to their default values. After issuing a reset, all non-default user-programmable options should be reprogrammed using the appropriate commands. The Reset operation is automatically performed when turning a bit-pump ON using the Bit-Pump ON/OFF command. Therefore, under normal operating conditions, there should be no need for the Reset command.

Opcode	
Numeric Value	**C Constant (defined in file api.h)**
0x0F	_RESET_SYSTEM

Options			
Option	**Description**	**Data Field**	**C Constant (api.h)**
None		0x00	

A.1.16 Operate Non-Linear EC

Selects between operating or disabling of the Bt8960 non-linear echo canceler.

NOTE: The Non-Linear echo canceler is not functional in Bt8960 Rev A & B and should not be activated.

Opcode	
Numeric Value	**C Constant (defined in file api.h)**
0x11	_OPERATE_NLEC

Options			
Option	**Description**	**Data Field**	**C Constant (api.h)**
Disable NL EC	NL EC is not activated and not used	0x00	_NLEC_OFF
Enable NL EC	NL EC operates normally	0x01	_NLEC_ON

A.1.17 Write Transmitter Gain

Writes the Transmitter Gain Register. The Transmitter Gain register is a 4-bit, 2's complement value. The upper 4 bits of this field are ignored. The Tx Gain adjusts the nominal transmit power of the Bt8960. The Tx Gain ranges from –1.6 dBm (1000b) to +1.4 dBm (0111b) of the nominal transmit power level.

Opcode	
Numeric Value	**C Constant (defined in file api.h)**
0x13	_WRITE_TX_GAIN

Options			
Option	**Description**	**Data Field**	**C Constant (api.h)**
Tx Gain	4-bit, 2's complement integer.	Value	

A.1.18 Tip/Ring Reversal

Reverses the Tip/Ring polarity on the received signal. However, the Bt8960 does not reverse the Tip/Ring polarity on the transmitted signal. In addition, the Bt8960 does not provide the ability to detect when Tip/Ring reversal is necessary.

This command is useful in applications where a framer has the ability to detect Tip/Ring reversal but can not 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.

NOTE: This command is only supported by Versions 2.0+.

Opcode	
Numeric Value	**Opcode**
0x14	_REVERSE_TIP_RING

Options			
Option	**Description**	**Data Field**	**C Constant (defined in file api.h)**
Tip/Ring Normal	Sets the Tip/Ring polarity on the received signal to normal (not-reversed).	0x00	
Tip/Ring Reverse	Reverses the Tip/Ring polarity on the received signal.	0x01	

A.1.19 BER Meter Start

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

NOTE: This command is only supported by Versions 2.0+.

Opcode	
Numeric Value	**C Constant (defined in file api.h)**
0x15	_BER_METER_START

Options			
Option	**Description**	**Data Field**	**C Constant (defined in file api.h)**
None		0x00	

A.1.20 BER Meter Stop

Deactivates the BER Meter. The bit-pump is set to transmit external 4-Level data, the transmit scrambler is set based on the current _TRANSMIT_SCR API setting. The **enabled** bit is turned OFF. The **bit_errors** and **meter_intervals** variables are unmodified so they can be still read.

NOTE: This command is only supported by Versions 2.0+.

Opcode	
Numeric Value	**C Constant (defined in file api.h)**
0x16	_BER_METER_STOP

Options			
Option	**Description**	**Data Field**	**C Constant (defined in file api.h)**
None		0x00	

A.2Status Request Commands

Status request commands are used to get performance monitoring information from the bit-pump. These commands do not affect bit-pump operation in any way. The bit-pump response to all status request commands is a 1-byte response.

For each status request command, the type and format of information supplied, and the command opcode and the options are described.

A.2.1 Input Signal Level

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.

Opcode	
Numeric Value	**C Constant (defined in file api.h)**
0x80	_SLM

Options			
Option	**Description**	**Data Field**	**C Constant (api.h)**
None		0x00	

A.2.1.1 Bit-pump Response

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.

A.2.2 Input DC Offset

Requests the value of the average DC level at the ADC input.

NOTE: Any level of input DC offset is digitally canceled on-chip, but large DC offsets (>2% of full scale) may degrade performance because of the reduction in effective ADC dynamic range.

Opcode	
Numeric Value	**C Constant (defined in file api.h)**
0x81	_DC_METER

Options			
Option	**Description**	**Data Field**	**C Constant (api.h)**
None		0x00	

A.2.2.1 Bit-pump Response

Signed integer X, $-128 \leq X \leq 127$, relative to the average DC offset per ADC sample. The measurement scale is such that the actual DC offset in units of ADC LSB is 32X. In case the actual DC offset is outside the representable range (–4096 to 4095), the nearest representable value will be used.

A.2.3 Far-End Signal Attenuation

Requests a value of the far-end signal attenuation. This value is based on measuring the average far-end signal level after echo cancellation.

The result is calibrated to represent the overall signal power attenuation over the cable in dB. It is also calibrated for an analog gain value of 0 dB, with an absolute gain value as implemented in Brooktree's HDSL EVM system. When using a different analog gain value, or an AAGC gain selection other than 0 dB, the signal attenuation result must be scaled accordingly. For example, if an analog gain value of 6 dB is used, the actual signal attenuation is 6 dB larger than reported. Note also that this value depends on the transmitted power at the far-end side, and is calibrated for the nominal 13.5 dBm value.

Opcode	
Numeric Value	**C Constant (defined in file api.h)**
0x82	_FELM

Options			
Option	**Description**	**Data Field**	**C Constant (api.h)**
None		0x00	

A.2.3.1 Bit-pump Response

1-Byte unsigned integer X, $0 \leq X \leq 255$, indicating the overall signal power attenuation, in units of 0.5 dB. For example, a value of 60 means total cable attenuation of 30 dB.

A.2.4 Noise Margin

Requests a value of the noise margin (NMR) of the receiver. The noise margin is defined as the maximum tolerable increase in external noise power that still allows for BER of less than 1E–7.

The value is based on measuring the average absolute level of the noise at the input to the slicer. In order to get a stable response, it is recommended that the average of 10 noise margin readings is used to reduce the statistical measurement error.

Opcode	
Numeric Value	**C Constant (defined in file api.h)**
0x83	_NMR

Options			
Option	**Description**	**Data Field**	**C Constant (api.h)**
None		0x00	

A.2.4.1 Bit-pump Response

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 –8 means a noise margin of –4 dB.

A.2.5 Timing Recovery Control

Requests a 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 with the frequency offset in Hz depends on the timing recovery's slope.

NOTE: This replaces the _VCXO_CONTROL_VOLTAGE API command found in existing Brooktree HDSL products.

Opcode	
Numeric Value	**C Constant (defined in file api.h)**
0x84	_TIMING_RECOVERY_CONTROL

Options			
Option	**Description**	**Data Field**	**C Constant (api.h)**
None		0x00	

A.2.5.1 Bit-pump Response

Signed integer X, $-128 \leq X \leq 127$, indicating the 8 MSBs of the timing recovery control word.

A.2.6 Bit-Pump Status

Requests bit-pump status. The status bits are designated I0 (LSB) to I7 (MSB), and interpreted according to Table A-2.

Opcode	
Numeric Value	**C Constant (defined in file api.h)**
0x85	_STARTUP_STATUS

Options			
Option	**Description**	**Data Field**	**C Constant (api.h)**
None		0x00	

A.2.6.1 Bit-pump Response

Table A-3. Status Bits

Status Bit	Indicates	Value = 0	Value = 1
I0 (LSB)	LOS: A LOS condition is defined as the absence of a far-end signal at the receiver input.	No LOS condition	LOS condition
I1	LOST: A LOST condition is defined as a continuously active LOS condition for more than a period of LOST seconds. The LOST period is user programmable (see the LOST Time Period command).	No LOST condition	LOST condition
I2	Reserved		
I3	Activation timer: On an HTU-R, the Activation timer is activated when 2-level transmission begins. On an HTU-C, the Activation timer is activated when an Activate command is issued and 2-level transmission begins. The timer is restarted when a far-end (HTU-R) signal is detected. Note: The Activation Timer defaults to 50 seconds for a 288kbps data rate system	Timer not expired	Timer expired
I4	Noise Margin OK: Indicates that the noise margin is better than a threshold of –5 dB.	Noise margin ≤ –5dB	Noise margin > –5 dB
I5	Reserved		
I6	Transmitter 4-Level Indicator: indicates that a 4-Level signal is being transmitted.	Not transmitting 4-Level	Transmitting 4-Level
I7	Normal Operation Flag: This bit is set when the Bit-pump successfully completes startup. This bit is cleared when the startup is first Activated, when the Deactivate command is issued, and when Initialized.	Not Normal Operation	Normal Operation

A.2.7 Read Linear Echo Canceler Coefficient

Reads a specified Linear Echo Canceler coefficient.

Opcode	
Numeric Value	**C Constant (defined in file api.h)**
0x86	_LEC_COEFF

Options			
Option	**Description**	**Data Field**	**C Constant (api.h)**
Coefficient index	Unsigned integer X, 0 ≤ X ≤ 59, indicating the index of the linear EC coefficient to be read.	X	

A.2.7.1 Bit-pump Response

1-Byte signed integer X, containing the 8 MSBs of the requested Linear EC coefficient value.

A.2.8 Read Non-Linear Echo Canceler Coefficient

Reads a specified Non-Linear Echo Canceler Coefficient.

Opcode	
Numeric Value	**C Constant (defined in file api.h)**
0x87	_NLEC_COEFF

Options			
Option	**Description**	**Data Field**	**C Constant (api.h)**
Coefficient index	Unsigned integer X, $0 \leq X \leq 63$, indicating the index of the non-linear EC coefficient to be read.	X	

A.2.8.1 Bit-pump Response

Signed integer X, containing the 8 MSBs of the requested non-linear EC coefficient value.

A.2.9 Read EQ Coefficient

Reads a specified DAGC, FFE, or EP coefficient.

Opcode	
Numeric Value	**C Constant (defined in file api.h)**
0x88	_EQ_COEFF

Options			
Option	**Description**	**Data Field**	**C Constant (api.h)**
FFE coefficients	Request FFE coefficients 0–7	0–7	
FFE Data Taps	Request FFE Data Taps 0–7	8–15	
EP coefficients	Request EP coefficients 0–4	16–20	
EP Data Taps	Request EP Data Taps 0–4	21–25	
DAGC gain	Request DAGC gain value – LSB Request DAGC gain value – MSB	26 27	
DAGC Output	Request DAGC Output	28	
FFE Output	Request FFE Output	29	
DAGC Input	Request DAGC Input	30	
FFE Output, delayed 1 Symbol	Request FFE Output, delayed 1 Symbol	31	
DAGC Error Signal	Request DAGC Error Signal	32	
Equalizer Error Signal	Request Equalizer Error Signal	33	
Slicer Error Signal	Request Slicer Error Signal	34	
Reserved		35–47	

A.2.9.1 Bit-pump Response

Signed integer containing the 8 MSBs of the requested EQ coefficient value.

A.2.10 Read DFE Coefficient

Reads a specified DFE coefficient.

Opcode	
Numeric Value	**C Constant (defined in file api.h)**
0x89	_DFE_COEFF

Options			
Option	**Description**	**Data Field**	**C Constant (api.h)**
Coefficient index	Unsigned integer X, $0 \leq X \leq 57$, indicating the index of the DFE coefficient to be read.	X	

A.2.10.1 Bit-pump Response

Signed integer X, containing the 8 MSBs of the requested DFE coefficient value.

A.2.11 Software/Chip Version

Requests software and chip version numbers.

Opcode	
Numeric Value	**C Constant (defined in file api.h)**
0x8A	_VERSION

Options			
Option	**Description**	**Data Field**	**C Constant (api.h)**
Major SW Version Chip Version	The 4 LSBs of the status byte (bit 0–bit 3) contain the bit-pump major software version, bit 0 being the LSB. The 4 MSBs of the status byte (bit 4–bit 7) indicate the bit-pump chip version, bit 4 being the LSB	0x00	_HW_SW_VERSIONS
Major SW Version	1-Byte unsigned integer field returning the major software version.	0x01	_MAJOR_SW_VERSION
Minor SW Version	1-Byte unsigned integer field returning the minor software version.	0x02	_MINOR_SW_VERSION
Bit-pump Type Chip Version	The 3 LSBs of the status byte (bit 0–bit2) contain the bit-pump type (see table below). Bit 3 is undefined. The 4 MSBs of the status byte (bit 4–bit 7) indicate the bit-pump chip version, bit 4 being the LSB.	0x03	_HW_TYPE_VERSIONS

Bit-pump Types	
Bits 2-0	**Bit-pump Type**
000	Bt8952
001	Bt8960
010–111	undefined

A.2.12 Bit-pump Present

Requests a bit-pump presence status. A positive response means that the bit-pump is present in the system and connected to the microprocessor. The presence of a bit-pump is determined by performing a write/read operation to a single internal bit-pump RAM location. It is possible for hardware faults to masquerade as the presence of a bit-pump at a location. The presence of a bit-pump does not guarantee that all microprocessor bit-pump hardware connections are OK.

Opcode	
Numeric Value	**C Constant (defined in file api.h)**
0x8B	_BIT_PUMP_PRESENT

Options			
Option	**Description**	**Data Field**	**C Constant (api.h)**
None		0x00	

A.2.12.1 Bit-pump Response

- 0x00: Bit-pump not present
- 0x01: Bit-pump present

A.2.13 Self Test

Executes a bit-pump self-test. The self-test does not cover all aspects of the chip operation, but verifies read/write operations.

Opcode	
Numeric Value	**C Constant (defined in file api.h)**
0x8C	_SELF_TEST

Options			
Option	**Description**	**Data Field**	**C Constant (api.h)**
None		0x00	

A.2.13.1 Bit-pump Response

- 0x00: Bit-pump self-test pass
- 0x01: Bit-pump self-test fail

A.2.14 Read Bt8960 Register

Reads an internal Bt8960 register. Register address is required.

Opcode	
Numeric Value	**C Constant (defined in file api.h)**
0x8D	_REGISTER

Options			
Option	**Description**	**Data Field**	**C Constant (api.h)**
Bt8960 register address	Address of internal Bt8960 register to be read. See Bt8960 datasheet for register address map.	X	

A.2.14.1 Bit-pump Response

Value stored in specified Bt8960 internal register.

A.2.15 Bit-pump Configuration

Requests a bit-pump configuration. This includes user programmable options and parameters.

Opcode	
Numeric Value	**C Constant (defined in file api.h)**
0x8E	_CONFIGURATION

Option			
Option	**Description**	**Data Field**	**C Constant (api.h)**
Read low application setup byte	Read low byte of application setup word. See Table A-3 for structure of application setup word.	0x00	_USER_SETUP_LOW_BYTE
Read high application setup byte	Read high byte of application setup word. See Table A-4 for structure of application setup word.	0x01	_USER_SETUP_HIGH_BYTE
Read LOST time period	Read LOST time period. The format of this status byte is identical to the format in the LOST Time Period command.	0x02	_LOST
Read symbol rate	Read symbol rate. The format of this status byte is identical to the format in the Symbol Rate command.	0x03	_BIT_RATE

Table A-4. Low Application Setup Byte

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	Other side Bt8960	Receive descr.	Transmit scramb.	Start-up sequence source	Analog AGC Config 1	Analog AGC Config 0	Terminal type

Table A-5. High Application Setup Byte

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Test mode 3	Test mode 2	Test mode 1	Test mode 0	Reserved	Reserved	Data transfer format 1	Data transfer format 0

All fields of the application setup word are coded in exactly the same way as the data field byte in the corresponding control command. For example, the two Data Transfer Format bits (bits 0,1 of the high application setup byte) are interpreted exactly the same as the data field byte in the Data Transfer Format command.

The 4 test mode bits (Test mode 0–Test mode 3) show the current test mode index (numbered from 0 to 14) with Test Mode 0 being the LSB. A zero value indicates that no test mode is currently in effect. A non-zero value indicates that the bit-pump currently operates in the specified test mode. See the Test Mode command.

A.2.16 Stage Number

Gets the bit-pump software stage number.

Opcode	
Numeric Value	**C Constant (defined in file api.h)**
0x8F	_STAGE_NUMBER

Options			
Option	**Description**	**Data Field**	**C Constant (api.h)**
None		0x00	

A.2.16.1 Bit-pump Response

The Bt8960 software internal stage number. This stage number changes when the bit-pump goes through a activation procedure. File suc.h contains C definitions mapping the stage numbers to stage names. This command is only used for test and debug purposes.

A.2.17 AAGC Value

Gets the current value of the 3 AAGC control bits.

Opcode	
Numeric Value	**C Constant (defined in file api.h)**
0x90	_AAGC_VALUE

Options			
Option	**Description**	**Data Field**	**C Constant (api.h)**
None		0x00	

A.2.17.1 Bit-pump Response

The 3 LSBs of this status byte reflect the state of the AAGC control bits. This command is useful for calculating the true signal attenuation by compensating for the actual AAGC gain (see Far-End Signal Attenuation command).

A.2.18 Read Tx Gain

Reads the Transmitter Calibration/Gain registers. The Transmitter Gain registers are a 4-bit, 2's complement value. The upper 4 bits of this field are ignored.

Opcode	
Numeric Value	**C Constant (defined in file api.h)**
0x91	_READ_TX

Options			
Option	**Description**	**Data Field**	**C Constant (api.h)**
Tx Calibration	Transmitter Calibration Register	0x00	_CALIBRATION
Tx Gain	Transmitter Gain Register	0x01	_GAIN

A.2.18.1 Bit-pump Response

The Tx Calibration Register contains the nominal setting for the transmitter gain. The Tx Gain Register contains the current transmitter gain setting. On software initialization, the Tx Gain register is programmed to the Tx Calibration register.

A.2.19 BER Meter Status

Request the BER Meter Status. Reading the BER Meter status commands while the BER Meter is enabled does not effect the BER meter operation.

NOTE: This command is only supported by Versions 2.0+.

Opcode	
Numeric Value	**C Constant (defined in file api.h)**
0x92	_BER_METER_STATUS

Options			
Option	**Description**	**Data Field**	**C Constant (defined in file api.h)**
BER Status	Read the BER Meter Status, see the BER Status below for bit definitions.	0x00	_BER_STATUS
# Bit Errors Low Byte	Reads the low byte of the number of bit-errors.	0x01	_BER_BIT_ERRORS_LOW
# Bit Errors High Byte	Reads the high byte of the number of bit-errors.	0x02	_BER_BIT_ERRORS_HIGH
# Meter Intervals Low Byte	Reads the low byte of the number of meter intervals elapsed.	0x03	_BER_METER_INTERVALS_LOW
# Meter Intervals High Byte	Reads the high byte of the number of meter intervals elapsed.	0x04	_BER_METER_INTERVALS_HIGH

BER Status Bits			
Status Bit	**Indicates**	**Value = 0**	**Value = 1**
0	BER Meter Enabled	Not Active	Active
1-–7	Reserved		

A.3Special Messages

A.3.1 Acknowledge

This message is sent by the 8032 to acknowledge a correctly received message, in a serial communication UI configuration. The acknowledge message is 4 bytes long (like all other messages), with the first 3 being 0xFF, 0xFF, 0xFF. Byte 4 (last transmitted) is the standard checksum byte, which equals 0x55.

Appendix B Calibrating Noise Margin Table

B.1Introduction

The Noise Margin Table defined in module USER.C is calibrated to W & G ILS 200 ETSI Loop 2 with 1.5 MHz shaped noise with low crest factor. This document describes the steps needed to calibrate the noise margin table to a user's specific noise source. The Noise Margin Table has a 0.5 dB resolution.

B.1.1 Setup

Use a line simulator or true line and a noise generator. Select a simple standard loop like ISDN Loop 2 and perform startup. Connect a BER meter to the HDSL equipment.

B.1.2 Noise Margin Table Calibration

B.1.2.1 Calculate scale factor.

1. Read meter timer high byte address 0x19, using the API command :
   _BtStatus (loop_no, _REGISTER, 0x19, &scale);
2. Calculate scale factor.
   scale_factor = ((2^7) / scale);

B.1.2.2 Find 0 dB Noise Margin Reference Point.

1. Change the noise source until BER reading is 1.0e–7 (for 1 loop) or 5.0e–8 (for 2 loops). This point is the 0 dB Noise Margin reference point

B.1.2.3 Calculate reference point value.

1. Read NLM meter low byte address 0x50, using the API command:
   _BtStatus (loop_no, _REGISTER, 0x50, &low_byte);
2. Read NLM meter high byte address 0x51, using the API command:
   _BtStatus (loop_no, _REGISTER, 0x51, &high_byte);
3. Combine the two NLM bytes (low_byte and high_byte) to an integer:
   *nlm_value = 256 * high_byte + low_byte;*
4. Calculate the normalized value of noise level meter:
   *nlm_value = nlm_value * (2^scale factor);*
5. Repeat steps 1 to 4 ten times. Calculate the average *nlm_value* over the ten readings.

B.1.2.4 Set 0dB reference point.

1. Write the averaged value of noise level meter to cell 32 in the *_noise_margin[]* array, module USER.C.

B.1.2.5 Calculate +0.5dB Margin.

1. Decrement the noise source by 0.5 dB.
2. Repeat the Calculate Reference Point Value step B.1.2.3 (1 to 5).
3. Write the calculated value to next higher cell.

B.1.2.6 Calculate positive margin.

1. Repeat step B.1.2.5 (1 to 3) until the upper half of the table if full. The significant part of the table is 0 dB to 15 dB noise margin. Higher margins can not be measured reliably.

B.1.2.7 Set Noise Source to 0dB.

1. Set the noise source to the 0 dB noise margin reference point.

B.1.2.8 Calculate –0.5 dB Margin.

1. Increment the noise source by 0.5 dB.
2. Repeat the Calculate Reference Point Value step B.1.2.3 (1 to 5).
3. Write the calculated value to next lower cell.

B.1.2.9 Calculate negative margin.

1. Repeat step B.1.2.8 (1 to 3) until the lower half of the table is full.

Appendix C ERLE Diagnostic Code

C.1Introduction

The ERLE program is a diagnostic tool to verify the integrity of the hardware's analog front-end and echo canceler. The ERLE program is useful when modifying the hybrid for different data rates.

C.2What is ERLE?

ERLE = Echo Return Loss Enhancement.
Four results are given from the ERLE program.

1. Background Noise Test
2. ERLE
3. Analog ERLE
4. AAGC Check

The background test measures the SLM while the transmitter is turned off. This effectively gives the amount of noise present at the input of the A/D.

The ERLE result determines the amount of echo canceled. The echo is canceled initially by the external hybrid and then by the Linear Echo Canceler in the bit-pump. The ERLE measurement is determined by the following formula.

```
ERLE = 20 * log ( SLM / FELM )
SLM = Signal Level Meter
FELM = Far End Level Meter
```

The Analog ERLE determines how much echo is canceled by the hybrid. This is done by bypassing the hybrid and measuring the input signal level meter. The Analog ERLE measurement is determined by the following formula.

```
ERLE = 20 * log ( SLM2 / SLM )
SLM2 = Signal Level Meter with hybrid input bypassed
SLM = Signal Level Meter from ERLE result
```

The AAGC Check makes sure the gain settings are monatomic. The AAGC is the bit-pump's internal Analog Automatic Gain Control.

C.3ERLE Files

The ERLE.C and ERLE.H files found in the bit-pump directory contains the ERLE functions. Prior to calling these functions, the bit-pump must be configured using the appropriate API commands.

Routine	Description
_Erle()	Contains the ERLE diagnostic code. _Erle() is used to measure the Background Noise, ERLE, and Analog ERLE.
_MeasureAagc()	This function cycles through the AAGC settings and measures the SLM to make sure the gain settings are monatomic.

C.3.1 _ERLE()

The _ERLE() routine measures the Background Noise, ERLE, and Analog ERLE.
Parameters: no - which bit-pump, _BIT_PUMP0 - _BIT_PUMP6
mode - mode of operation
The mode bits are defined as follows:

Bit	Description
0	Adapt Non-linear Echo Canceler. A '0' does not adapt the NLEC. A '1' adapts the NLEC.
1	Reserved
2,3,4	Select AGAIN setting. These 3 bits correspond to the 3 Gain Control bits of the ADC Control Register (0x21). 000 = 0dB 001 = 3 dB 010 = 6 dB 011 = 9 dB 100 = 12 dB 101 = 15 dB
5,6,7	Reserved

C.3.2 _MeasureAagc()

The _MeasureAagc() routine verifies the gain settings are monatomic.
Parameters:no- which bit-pump, _BIT_PUMP0 - _BIT_PUMP6

C.4Invoking the Tests

C.4.1 Background Test

To perform the Background Noise Test, call _Erle() when the transmitter is OFF. In this mode, the program adapts the LEC (linear echo canceler). If the NLEC (non linear echo canceler) mode is on, then the NLEC is also adapted. After adapting the coefficients, the SLM and FELM are read and ERLE is calculated. The ERLE number is not important here, what is important are the SLM and FELM register values.

C.4.2 ERLE and Analog ERLE Test

To perform the ERLE and Analog ERLE test, call _Erle() when the transmitter is ON. In this mode, the program adapts the LEC (linear echo canceler). If the NLEC (non linear echo canceler) mode is on, then the NLEC is also adapted. After adapting the coefficients, the SLM and FELM are read and ERLE is calculated. After that, the hybrid bypass bit is set and Analog ERLE is calculated.

C.4.3 AAGC Check

To perform the AAGC Check test, call the _MeasureAagc() when the transmitter is ON.

C.5Compiling the ERLE Code

The ERLE code requires the compiler flag ERLE to be specified in the compiler flag define list.

C.5.1 TDEBUG Compiler Flag

The TDEBUG compiler is typically specified when generating the ERLE code. The TDEBUG provides a 'printf()' environment to display the results. In the TDEBUG environment, an RS232 cable must be connected to a terminal emulator.

NOTE: The SER_COM compiler flag can not be specified at the same time as the TDEBUG flag.

However, it is possible to compile without the TDEBUG flag. If this is the case, then the user would be responsible for extracting the results.

Appendix D Release Notes

Appendix D contains the previous release notes for the Bt8960 software. These notes have also been incorporated into their respective sections within this user's guide.

NOTE: Bt8960 Bit-pump Version 2.0, Release Notes—November 13, 1996

D.1Memory Requirements

The RAM and ROM requirements for four common build options with their associated compiler flags are given in Table D-1.

Table D-1. RAM and ROM Requirements

Build Option	RAM	ROM
BT8960CR w/o BER_METER	111	24.0 k
BT8960CR w/ BER_METER	117	24.4 k
BT8960C w/ BER_METER	117	20.0 k
BT8960R w/ BER_METER	117	20.6 k
NOTE(S): The RAM requirements do not include the stack size, an additional 24 bytes is required for the run-time stack.		

D.1.0.1 BT8960CR w/o BER_METER

C51, ADD_DELAY, PDATA_MODE, SER_COM, HTUC, HTUR, SINGLE_LOOP, CHAN_UNIT

D.1.0.2 BT8960CR w/ BER_METER

C51, ADD_DELAY, PDATA_MODE, SER_COM, HTUC, HTUR, SINGLE_LOOP, BER_METER, CHAN_UNIT

D.1.0.3 BT8960C w/ BER_METER

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

D.1.0.4 BT8960R w/ BER_METER

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

D.1.1 Bug Fixes

The following bugs were fixed in Version 2.0.

D.1.1.1 HTU-C Phase #0 = 0 vs AagcCheck() Bug

When the AagcCheck() incremented the gain setting, the Linear EC was not re-adapted so the first phase quality (phase #0) always reported 0. To solve this, in the START_OPEN_EYE stage, always set the next stage to PHASE_ADAPT_EC1 instead of OPEN_EYE1; this will force the Echo Canceler to be re-adapted. Fixing this bug improves startup reliability and performance.

D.1.1.2 External and Internal Analog Loopback

The External and Internal Analog loopback function did not work properly in Version 1.x. Please see the Analog Loopbacks section below for details.

D.1.2 Support new Bt8960 Rev C

Version 2.0 supports the new Bt8960 Rev C silicon. The Bt8960 Rev C is identified by the package marking PROTO3 or ES2, and the chip revision number 1 in the global modes register (Addr. 0x00) and JTAG ID register. Version 2.0 is backwards compatible with the existing Bt8960 Rev A & B parts. However, the existing Version 1.x Bt8960 Bit-pump Software is not forward compatible with the new Rev C parts.

D.1.2.1 _ReadAccessDataByte() function

When reading the indirect RAM (i.e. linear echo canceler, equalizer register, etc.) of the Bt8960 Rev C silicon, there can not be any other write operations to the Bt8960 after the indirect RAM read address register (i.e. Addr: 0x70, 0x72, etc.) is selected and before the Access Data Byte registers (Addr: 0x7C-0x7F) are read.

In the Bt8960 code, the only time this condition is violated is if an interrupt occurs in between selecting the read address and reading the access data bytes. The _ReadAccessDataByte() function disables the interrupts whenever reading the indirect RAM and re-enables the interrupts on exit. The four Access Data Byte registers are stored in the **global_access_data_byte[]** array. The calling function or routine then just accesses this global array without interfering with the bit-pump registers.

D.1.2.2 Analog Loopbacks

The Bt8960 Rev C silicon fixed the Isolated Analog loopback. Please see the Analog Loopbacks section below for details.

D.1.3 Improved Performance

The startup reliability and performance for ANSI Loops #1 & #2 and maximum reach were significantly improved.

D.1.3.1 Phase Quality Selection

In Bit-pump Version 1.x, a Bt8960 system would occasionally select the incorrect optimal phase on some of the more difficult loops. During this bad startup, the system would successfully pass data but the noise margin would be noticeably worse than a good startup. In addition, the Bt8960 could not startup on ANSI Loops #1 and #2. Version 2.0 modified the optimal phase selection to provide a more robust algorithm for the Bt8960 systems. This provides more reliable startups, the ability to startup on ANSI Loops #1 & #2, and gains more maximum reach.

NOTE: This improvement applies to both Rev B and Rev C silicon.

D.1.4 New / Modified Features

The following features were added or fixed in Version 2.0.

D.1.4.1 Implemented Internal BER Meter

Implemented the Bt8960 internal BER Meter. Please see the Internal BER Meter Operation section below for details.

D.1.4.2 READ_METER_REG() Macro

During the BER Meter operation, the bit-pump interrupt handler reads the BER Meter register every meter interval. To insure that the interrupt doesn't occur in between reading the low byte and high byte of another meter register, the READ_METER_REG() macro disables the bit-pump interrupt when accessing any of the meter registers.

D.1.4.3 Analog Loopbacks

The Bt8960 supports 3 types of analog loopbacks: External, Internal, and Isolated.

1. External—Transmitter ON, receiver uses both the receive (RXP, RXN) and hybrid (RXBP, RXBN) inputs.
2. Internal—Transmitter ON, receiver only uses the hybrid inputs (RXBP, RXBN).
3. Isolated—Transmitter silent, internally loopbacked.

Table D-2 shows the relationship between the Chip Rev, Software Version, and Analog Loopbacks supported.

Table D-2. Version Relationships

Version	External	Internal	Isolated
V 1.x	None	None	None
V 2.0	Rev B and C	Rev C only	Rev C only

D.1.5 New API Commands

The following API commands were added in Version 2.0.

D.1.5.1 Tip/Ring Reversal

This command reverses the Tip/Ring polarity on the received signal. However, the Bt8960 does not reverse the Tip/Ring polarity on the transmitted signal. In addition, the Bt8960 does not provide the ability to detect when Tip/Ring reversal is necessary.

This command is useful in applications where a framer has the ability to detect Tip/Ring reversal but can not 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.

Opcode	
Numeric Value	**C Constant (defined in file api.h)**
0x14	_REVERSE_TIP_RING

Options			
Option	**Description**	**Data Field**	**C Constant (defined in file api.h)**
Tip/Ring Normal	Sets the Tip/Ring polarity on the received signal to normal (not-reversed).	0x00	
Tip/Ring Reverse	Reverses the Tip/Ring polarity on the received signal.	0x01	

D.1.5.2 BER Meter Start

This command activates the BER Meter. The bit-pump is set to transmit an internal 4-Level scrambled ones pattern. The **enabled** bit is set and the **bit_errors** & **meter_intervals** variables are reset to 0. This command should only be called during the Bit-pump's normal operation.

Opcode	
Numeric Value	**C Constant (defined in file api.h)**
0x15	_BER_METER_START

Options			
Option	**Description**	**Data Field**	**C Constant (defined in file api.h)**
None		0x00	

D.1.5.3 BER Meter Stop

This command deactivates the BER Meter. The bit-pump is set to transmit external 4-Level data, the transmit scrambler is set based on the current _TRANSMIT_SCR API setting. The **enabled** bit is turned OFF. The **bit_errors** & **meter_intervals** variables are unmodified so they can be still read.

Opcode	
Numeric Value	**C Constant (defined in file api.h)**
0x16	_BER_METER_STOP

Options			
Option	**Description**	**Data Field**	**C Constant (defined in file api.h)**
None		0x00	

D.1.5.4 BER Meter Status

Request the BER Meter Status. Reading the BER Meter status commands while the BER Meter is enabled does not effect the BER meter operation.

Opcode	
Numeric Value	**C Constant (defined in file api.h)**
0x92	_BER_METER_STATUS

Options			
Option	**Description**	**Data Field**	**C Constant (defined in file api.h)**
BER Status	Read the BER Meter Status, see the BER Status below for bit definitions.	0x00	_BER_STATUS
# Bit Errors Low Byte	Reads the low byte of the number of bit-errors.	0x01	_BER_BIT_ERRORS_LOW
# Bit Errors High Byte	Reads the high byte of the number of bit-errors.	0x02	_BER_BIT_ERRORS_HIGH
# Meter Intervals Low Byte	Reads the low byte of the number of meter intervals elapsed.	0x03	_BER_METER_INTERVALS_LOW
# Meter Intervals High Byte	Reads the high byte of the number of meter intervals elapsed.	0x04	_BER_METER_INTERVALS_HIGH

BER Status Bits			
Status Bit	**Indicates**	**Value = 0**	**Value = 1**
0	BER Meter Enabled	Not Active	Active
1–7	Reserved		

D.1.6 Internal BER Meter Operation

This section describes how to use the Bt8960's Internal BER Meter. The BER Meter can be used to verify the integrity of the line. The BER Meter can be used as a diagnostic tool during production/field testing or hardware/software development.

The BER Meter uses its internal scrambled ones generator & de-scrambler to detect bit-errors. For the BER Meter to function properly, both the HTU-C and HTU-R must issue the _BER_METER_START API command. Since the BER Meter uses its own internal scrambled ones generator, the BER Meter can be not used when transporting real payload data. The BER Meter is only operational after the bit-pump has successfully completed startup.

D.1.6.1 BER_METER Compiler Flag

The BER_METER flag compiles in the BER Meter code. The BER Meter code adds ~400 bytes of ROM and 1 byte of RAM plus an additional 5 bytes of RAM for each bit-pump.

NOTE: The BER_METER flag can not be declared when the INT_BUG compiler flag is specified.

D.1.6.2 BER_METER API Commands

The BER Meter API commands are discussed in Section 5.

D.1.6.3 Calculating Avg BER and Elapsed Time

The following formulas are used to calculate the Avg BER and Elapsed Time.

$$AvgBER = \frac{\#ofBitErrors}{\#ofMeterIntervals * MeterIntervalLength * 2}$$

$$ElapsedTime = \frac{\#ofMeterIntervals * MeterIntervalLength * 2}{DataRate}$$

Variable	How derived
# of Bit-Errors	Read the # of Bit-Errors Low & High Byte API commands and build a 16-bit unsigned integer.
# of Meter Intervals	Read the # of Meter Intervals Low & High Byte API commands and build a 16-bit unsigned integer.
Meter Interval Length	Read the Bt8960 Meter Interval Register (Address 0x18, 0x19) and build a 16-bit unsigned integer. During normal operation, these registers should always read 0x8000 (32768).
Data Rate	Data Rate of the system, i.e. 288000 or 416000.
2	The '*** 2**' is necessary since there are 2 bits per symbol and the meter interval length is based on the number of symbols.
NOTE(S): 16-bit value = (high byte << 8) + (low byte)	

Figure D-1. Example

```
/*
 * Assuming 288kbps Data Rate, Normal Operation, and BER Meter Active
 * Also assumes using compiler/linker that supports floating point.
 */
void get_ber_meter_status (unsigned char no)
{
  unsigned char temp, temp1;
  unsigned int errors, intervals;
  float avg_ber, elapsed_time;

  _BtStatus(no, _BER_METER_STATUS, _BER_BIT_ERRORS_LOW, &temp);
  _BtStatus(no, _BER_METER_STATUS, _BER_BIT_ERRORS_HIGH, &temp1);
  errors = (unsigned)BYTE2WORD(temp1, temp);

  _BtStatus(no, _BER_METER_STATUS, _BER_METER_INTERVALS_LOW, &temp);
  _BtStatus(no, _BER_METER_STATUS, _BER_METER_INTERVALS_HIGH, &temp1)
  intervals = (unsigned)BYTE2WORD(temp1, temp);

  avg_ber = (errors) / (intervals * 0x8000 * 2)   /* equations don't show necessary type casting */
  elapsed_time = (intervals * 0x8000 * 2) / (288000)

#if TDEBUG
  printf("# Bit Errors = %u\n", errors);
  printf("# Meter Intervals = %u\n", intervals);
  printf("Avg Ber = %.2e\n", avg_ber);
  printf("Elapsed Time = %.1f seconds\n", elapsed_time);
#endif

  return;
}
```

100251_008

D.1.7 BER Meter Bit-pump Code Implementation

This section describes how the Bt8960's Internal BER Meter is implemented in the Bit-pump Software. The user does not need to be aware of the exact details since all access to the BER Meter status is done through the API.

D.1.7.1 BER_METER_STRUCT Structure

The user should not directly access the BER Meter structure.

The BER Meter Structure is defined as follows:

```
#ifdef BER_METER
typedef union
{
  unsigned char reg;
  struct
      {
      unsigned char enabled:1;
      unsigned char :7;
      } bits;
} ber_meter_status;

typedef struct
{
      ber_meter_status status;
      unsigned int bit_errors;
      unsigned int meter_intervals;
}BER_METER_STRUCT;
extern BER_METER_STRUCT idata ber_meter[_NO_OF_LOOPS];
#endif
```

The **ber_meter** array is stored in **idata** since it takes up 5 bytes of memory per bit-pump.

be_meter_status: struct

The ber_meter_status structure contains 1 field, **enabled**. The **enabled** flag is used to determine when the BER Meter is currently active. The **enabled** bit is deactivated whenever the part is reset, startup is activated or deactivated, or the _BER_METER_STOP API command is called.

bit_errors: int

This variable keeps track of the number of bit errors. This is updated every Meter Interval when the BER Meter is enabled. This variable is reset whenever the _BER_METER_START API command is called.

meter_intervals: int

This variable keeps track of the number of meter intervals elapsed. This is updated every Meter Interval when the BER Meter is enabled. This variable is reset whenever the _BER_METER_START API command is called.

D.1.7.2 Interrupt Handler

The interrupt handler reads the Bit Error Rate Meter Register (Address 0x4C, 0x4D) and updates the Number of Bit-Errors and Number of Meter Intervals counter after every meter interval. This causes the interrupt handler to be slightly longer, ~55uS per bit-pump on an 11.0592MHz Intel 8032.

D.2Bt8960 Bit-pump Version 1.1, Release Notes—September 2, 1996

D.2.1 PLL Modes CLK_FREQ Bits were Incorrect

The PLL Frequency #defines for 2 & 8 Channel in API.H (line 116) were incorrect. However, the 4 and 6 channel #defines were correct. This bug could potentially impact the performance on systems running below 220kbps or running above 483kbps since the wrong CLK_FREQ range setting was selected. The PLL Modes Register (Addr: 0x22) should be as follows:

clk_freq[1,0]	Number of Channels (N)	Resulting Data Rate	Nominal Crystal Center Freq
00	4	288 kbps	9.216 MHz
01	6	416 kbps	13.312 MHz
10	2	160 kbps	5.120 MHz
11	Reserved	-	-
NOTE(S): The 'Reserved' setting behaves identically to the 6 Channel setting and not as an 8 Channel setting as initially presumed. Brooktree reserves the right to change the definition of the 'Reserved' setting.			

D.2.2 Added ERLE Support

The ERLE code is a diagnostic tool to verify the integrity of the hardware's analog front-end and echo canceler. The files ERLE.C and ERLE.H contain the ERLE code. These 2 files are located in the bit-pump sub-directory. The SCRIPT.BLD file was modified to include the ERLE target.

Please see the Bt8960 Single-Chip 2B1Q Transceiver Bit-pump Software User's Guide Version 1.1 (UG8960-1A) for complete details of the ERLE code.

D.2.3 Bt8960EVM Single Board Support

Brooktree now offers a production Bt8960EVM System. These systems have integrated the Bt8960 Chip, Microprocessor, EPROM, etc. onto a single PCB. The Bt8960EVM System now has separate code and data space unlike previous EVM versions. The following changes were made to the bit-pump code to support the new systems:

D.2.3.1 USER.C - Different Bit-pump Address Mapping

The bit-pump address map is located at 0x0000, the _BIT_PUMP0_ADD #define was changed to reflect the new hardware.

D.2.3.2 SCRIPT.BLD - XDATA Mapping

The LFLAGS XDATA() option was changed to 0x0000 to reflect the new hardware. In addition, the CFLAGS is compiled with the SINGLE_LOOP bit-pump compiler option since the new hardware only supports one bit-pump.

D.2.3.3 Application Code

The high-level application code was modified to take advantage of the features of the new system. Please see the Bt8960 Single-Chip 2B1Q Transceiver EVM Software User's Guide Version 1.1 (UG8960-3A) and EVM Hardware User's Guide (UG8960-4A) for complete details on the new Bt8960EVMs.=

Appendix E Version 1.1 Release Notes

Bt8960 Bit-pump Version 1.1, Release Notes, September 2, 1996

E.1PLL Modes CLK_FREQ Bits were Incorrect

The PLL Frequency #defines for 2 & 8 Channel in API.H (line 116) were incorrect. However, the 4 and 6 channel #defines were correct. This bug could potentially impact the performance on systems running below 220kbps or running above 483kbps since the wrong CLK_FREQ range setting was selected. The PLL Modes Register (Addr: 0x22) should be as follows:

clk_freq[1,0]	Number of Channels (N)	Resulting Data Rate	Nominal Crystal Center Freq
00	4	288 kbps	9.216 MHz
01	6	416 kbps	13.312 MHz
10	2	160 kbps	5.120 MHz
11	Reserved	-	-
NOTE(S): The 'Reserved' setting behaves identically to the 6 Channel setting and not as an 8 Channel setting as initially presumed. Brooktree reserves the right to change the definition of the 'Reserved' setting.			

E.2Added ERLE Support

The ERLE code is a diagnostic tool to verify the integrity of the hardware's analog front-end and echo canceler. The files ERLE.C and ERLE.H contain the ERLE code. These 2 files are located in the bit-pump sub-directory. The SCRIPT.BLD file was modified to include the ERLE target.

Please see the Bt8960 Single-Chip 2B1Q Transceiver Bit-pump Software User's Guide Version 1.1 (UG8960-1A) for complete details of the ERLE code.

E.3Bt8960EVM Single Board Support

Brooktree now offers a production Bt8960EVM System. These systems have integrated the Bt8960 Chip, Microprocessor, EPROM, etc. onto a single PCB. The Bt8960EVM System now has separate code and data space unlike previous EVM versions. The following changes were made to the bit-pump code to support the new systems:

E.3.1 USER.C - Different Bit-pump Address Mapping

The bit-pump address map is located at 0x0000, the _BIT_PUMP0_ADD #define was changed to reflect the new hardware.

E.3.2 SCRIPT.BLD - XDATA Mapping

The LFLAGS XDATA() option was changed to 0x0000 to reflect the new hardware. In addition, the CFLAGS is compiled with the SINGLE_LOOP bit-pump compiler option since the new hardware only supports one bit-pump.

E.3.3 Application Code

The high-level application code was modified to take advantage of the features of the new system. Please see the Bt8960 Single-Chip 2B1Q Transceiver EVM Software User's Guide Version 1.1 (UG8960-3A) and EVM Hardware User's Guide (UG8960-4A) for complete details on the new Bt8960EVMs.

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