Odin TeleSystems Inc.

Application Programming Interface (API) Reference Guide

for

BALDER-8S 2.0 Driver Rev. 1.0

Doc. No. 1211-1-SCA-1000-1

Doc. Rev. 1.0 P2 (Preliminary)

November 12, 1997

Changes

The material in this document is for information only and is subject to change without notice. While reasonable efforts have been made in the preparation of this document to assure its accuracy, Odin TeleSystems Inc., assumes no liability resulting from errors or omissions in this document, or from the use of the information contained herein.

Odin TeleSystems Inc. reserves the right to make changes in the product design without reservation and notification to its users.

Warranties

THE SOFTWARE AND ITS DOCUMENTATION ARE PROVIDED “AS IS” AND WITHOUT WARRANTY OF ANY KIND. ODIN TELESYSTEMS EXPRESSLY DISCLAIMS ALL THE WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR PARTICULAR PURPOSE. ODIN TELESYSTEMS DOES NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE SOFTWARE WILL MEET ANY REQUIREMENTS, OR THAT THE OPERATIONS OF THE SOFTWARE WILL BE UNINTERRUPTED OR ERROR-FREE, OR THAT DEFECTS WILL BE CORRECTED. FURTHERMORE, ODIN TELESYSTEMS DOES NOT WARRANT OR MAKE ANY REPRESENTATIONS REGARDING THE USE OR THE RESULTS OF THE SOFTWARE OR ITS DOCUMENTATION IN TERMS OF THEIR CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE. NO ORAL OR WRITTEN INFORMATION OR ADVISE GIVEN BY ODIN TELESYSTEMS OR ODIN TELESYSTEMS’ AUTHORIZED REPRESENTATIVE SHALL CREATE A WARRANTY. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO THE ABOVE EXCLUSION MAY NOT APPLY.

UNDER NO CIRCUMSTANCE SHALL ODIN TELESYSTEMS INC., ITS OFFICERS, EMPLOYEES, OR AGENTS BE LIABLE FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES (INCLUDING DAMAGES FOR LOSS OF BUSINESS, PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION) ARISING OUT OF THE USE OR INABILITY TO USE THE SOFTWARE AND ITS DOCUMENTATION, EVEN IF ODIN TELESYSTEMS HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN NO EVENT WILL ODIN TELESYSTEMS’ LIABILITY FOR ANY REASON EXCEED THE ACTUAL PRICE PAID FOR THE SOFTWARE AND ITS DOCUMENTATION. SOME JURISDICTIONS DO NOT ALLOW THE LIMITATION OR EXCLUSION OF LIABILITY FOR INCIDENTAL AND CONSEQUENTIAL DAMAGES, SO THE ABOVE LIMITATION OR EXCLUSION MAY NOT APPLY.

Odin TeleSystems Inc.

This document is published by:
Odin TeleSystems Inc.
P. O. Box 59686
Dallas, Texas 75229
U. S. A.

Printed in U. S. A.

1. Table of Contents

2. Introduction

The Balder boards are Integrated Services Digital Network (ISDN) interface cards for IBM PC compatible computers with the Industry Standard Architecture (ISA) bus. The Balder-2S board contains two and the Balder-8S board contains eight full duplex, ISDN 2B+D transceivers which comply with the ITU-T ISDN Layer 1 specifications (I.430). The board also contains two full implementations of an ISDN digital phone.

The Balder boards are delivered with a DOS, Windows'95, or Windows NT 4.0 device driver and an Application Programming Interface (API). The Balder driver provides the software needed by the PC host processor (CPU) to communicate with the board. The Driver Software is implemented in C language and it is provided in the form C wrapper libraries and device drivers. The driver also provides C-language header files (*.h) that declare the macros, types, and functions available in the wrapper libraries.

This document contains a reference for the API. Figure 1 shows the relationship between the API, the OSI 7-layer stack, and the ISDN protocol stack.

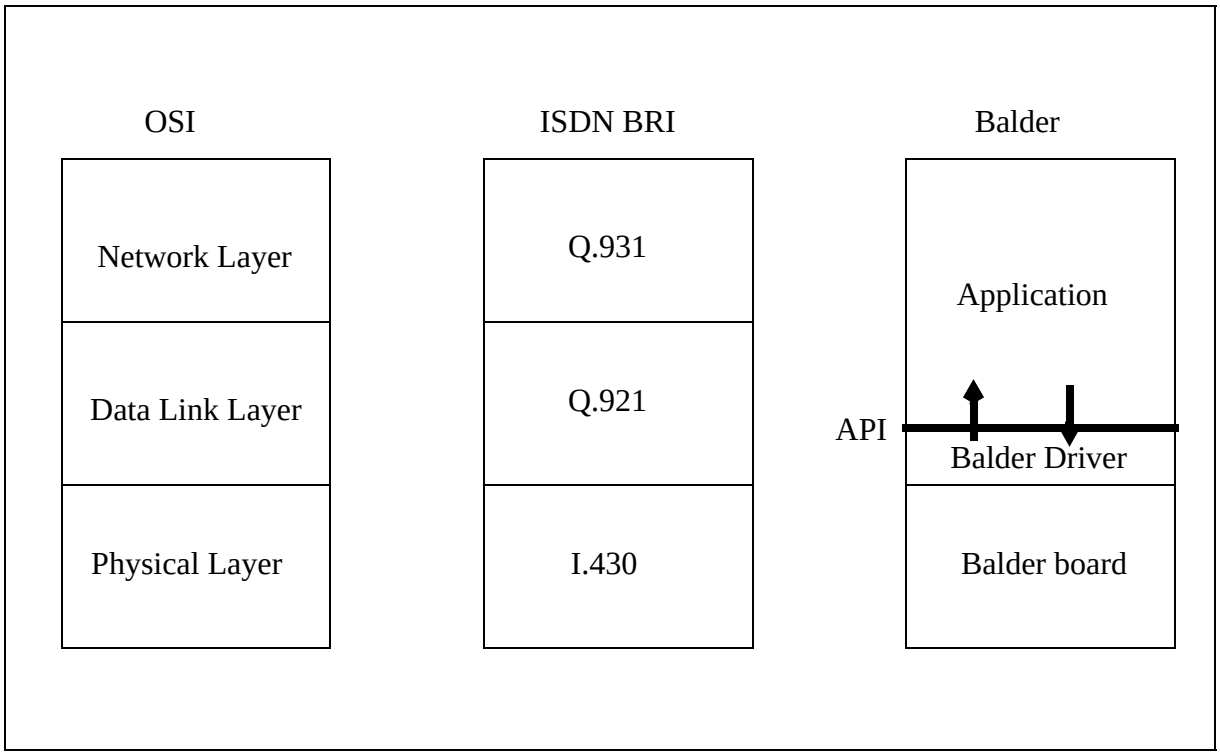

Figure 1. Relationship between Driver, OSI, and ISDN layers.

3. Applicable Products

The Balder drivers covered by this document are listed in Table 1.

TABLE 1. Balder Drivers covered by this document.

Odin Product Number	Description
SDA-1007-1	Balder-8S Rev. 2.0 DOS Driver and Application Programming Interface. Compiled with Borland C++ Version 4.53
SDA-1007-2	Balder-8S Rev. 2.0 DOS Driver and Application Programming Interface. Compiled with Microsoft C Version 5.1
SDA-1008-1	Balder-8S Rev. 2.0 Windows 95 Driver and Application Programming Interface. Compiled with Microsoft Visual C++ 5.0
SDA-1009-1	Balder-8S Rev. 2.0 Windows NT 4.0 Driver and Application Programming Interface. Compiled with Microsoft Visual C++ 5.0

4. Distribution

The Balder driver is distributed on 3.5” diskettes or it can be downloaded from the Odin TeleSystems Inc. Web-site: http://www.OdinTS.com/odin_sw.html . The Balder driver distribution is zipped and in the case of Windows 95 and Windows NT drivers contain a setup program (SETUP.EXE) which installs the driver distribution. After installation, the following files and directories are created:

TABLE 2. Balder driver distribution content

Directory	Files	Description
\	Readme	Release notes and changes made after printing of this documentation.
	revision.h	Header file containing the Driver Revision
Lib	Balder8s.lib	Balder-8S API Library compiled with the applicable compiler.
Inc	balder.h blddef.h	Balder API header files
Dsp	dtmffsk.out	Executable binary for the 8 on-board TI320C54 Digital Signal Processors. This program implements DTMF detection and generations as well as FSK (Caller-ID) detection.
Demos	Sample1.c Sample1.exe Makefile	Simple example application demonstrating some of the Balder-8S features.

TABLE 2. Balder driver distribution content

Directory	Files	Description
System	Balder8s.vxd (Windows 95 driver distribution only)	Balder-8S Windows 95 Virtual Device Driver
System	Balder8s.sys (Windows NT driver distribution only)	Balder-8S Windows NT 4.0 Kernel Mode Device Driver

The installation program also copies the Balder-8S Windows 95 Virtual device driver (Balder8s.vxd) or Windows NT Kernel Mode Driver (Balder8s.sys) into the Windows System Directory

4.1 Starting and Stopping the Windows NT Driver

In Windows NT the driver is installed in the mode where it needs to be started manually. To start the Balder-8S driver, in a command window give the command:

NET START BALDER8S

Note: You need to have administrative rights to the NT Workstation to start the driver. To stop the Balder-8S driver, give command:

NET STOP BALDER8S

5. API Overview

5.1 Naming Conventions

All functions and data types used within the API follow a naming convention. The names all have a *prefix* 'Bld' (for Balder). The function names consist of three parts: the header 'Bld', operation, and target. The *operation* is a verb or phrase describing the action to be taken (e.g., *construct*, *read*). The *target* can either be a physical subsystem block (e.g., *Li* or *Driver*) or a signal processing function (e.g., *Silence Detector* or *DtmfDetector*). Note that *Driver* target functions affect the whole system, whereas the *Li* function only affects specified line interfaces.

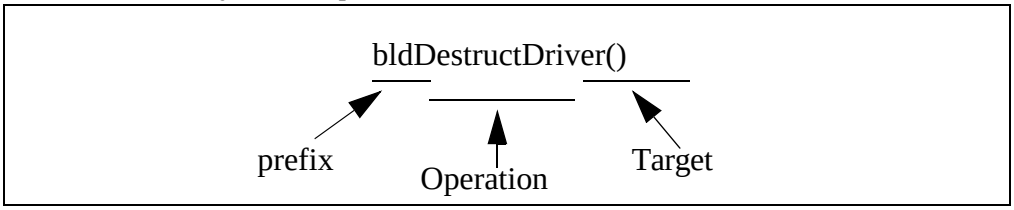

Figure 2. An Example of the Naming of the Functions

5.2 Numbering of Line Interfaces

If several Balder boards are installed into a PC, the boards should be numbered from 0 to *MAX_NO_OF_BOARDS*. Each Balder-2S board contains 2 line interfaces (LIs) and each Balder-8S board contains 8 LIs (ISAC-S chips). The LIs are numbered consecutively from 0 to *2*MAX_NO_OF_BOARDS-1* and from 0 to *8*MAX_NO_OF_BOARDS-1* for Balder-2S and Balder-8S, respectively.

For example, if we have three boards, the numbering is as follows:

TABLE 3. Numbering of LIs

Board #	Balder-2S LI #	Balder-8S LI #
0	0, 1	0 - 7
1	2, 3	8 - 15
2	4, 5	16 - 23
3	6, 7	24 - 31

5.3 Exception Handling

Exception handling is provided via return codes. Each function returns *BLD_SUCCESS* if it completes successfully (no errors). In the case of a problem, an error code of type *BldRc* is returned. The return codes can be translated into strings containing the error message with the *bldGetErrMsg()* function.

6. API Macro and Type Definitions

6.1 Frame End Codes

6.1.0.1 Synopsis

Frame End Codes (FECs) are used to indicate the status of received messages.

6.1.0.2 Definition

```
#define BLD_FEC_OK           0x00 /* Good message (no errors)              */
#define BLD_FEC_ABORTED      0x01 /* Set if the message was aborted by the
                                  .. remote side, i.e. a sequence of 7 1's
                                  .. was detected                          */
#define BLD_FEC_BAD_CRC      0x02 /* Set if CRC is bad                    */
#define BLD_FEC_OVERRUN      0x04 /* Set if at least one byte of the frame
                                  .. has been lost due hardware fifo
                                  .. overflow (PC too slow)                */
#define BLD_FEC_TRUNCATED    0x10 /* Set if message is longer than the
                                  .. buffer supplied by the application
                                  .. program                               */
#define BLD_FEC_OVERFLOW     0x20 /* Set if the application is too slow
                                  .. reading messages from the driver.
                                  .. Messages lost                         */
```

6.2 Return Codes

6.2.0.1 Synopsis

Return Codes are used to return the execution result from the API functions. If successful, a function returns *BLD_SUCCESS*. Otherwise an error code of type *BldRc* is returned. The return codes can be translated into strings containing the error message with the *bldGetErrMsg()* function.

6.2.0.2 Definition

```
typedef enum {
  BLD_UNDEFINED          =  0,  /* The driver does not know what to return
                                .. for this request                        */
  BLD_SUCCESS            =  1,  /* All OK, no errors.                       */
  BLD_INVALID_MODE       =  2,  /* Supplied Li mode is not allowed.         */
  BLD_INVALID_BOARD_TYPE =  3,  /* Supplied Board type is not supported by
                                .. by this driver.                         */
  BLD_INVALID_BOARD_NO   =  4,  /* Driver was supplied a Board number which
                                .. is not allowed (should be 0 <= NO <= 3).  */
```

```
BLD_INVALID_LI_NO      =  5,  /* Driver was supplied a Li number which
                              .. is not allowed (should be 0 <= NO <= 8).  */
BLD_INVALID_B_CHANNEL  =  6,  /* Driver was supplied an illegal  B channel
                              .. number (should be 1 <= NO <= 2).          */
BLD_INVALID_IRQ_NO     =  7,  /* Driver was supplied an IRQ number which
                              .. is not allowed (Not supported by
                              .. the board)                                */
BLD_INVALID_ADDR       =  8,  /* Driver was supplied an I/O-address which
                              .. is not allowed (Not supported by the
                              .. Board)                                    */
BLD_NOT_SETUP          =  9,  /* Functions has been called without a proper
                              .. configuration of the driver.              */
BLD_NO_CLOCKS          = 10,  /* Could not enable clocks for the ISAC
                              .. functions.  (??? is this used)            */
BLD_NO_FRAMES          = 11,  /* No full frames received and ready for
                              .. reading.                                  */
BLD_TX_BUSY            = 12,  /* Transmitter not ready. Transmission of the
                              .. previous frame has not been completed.    */
BLD_TIMEOUT            = 13,  /* Function timed out before the expected
                              .. response was obtained from ISR or ISAC    */
BLD_L1_OK              = 14,  /* Physical Layer (L1) is up.                   */
BLD_L1_DOWN            = 15,  /* Physical Layer (L1) is down,
                              .. no connection.                            */
BLD_BAD_CHIP           = 16,  /* Line Interface (ISAC-S chip) did not
                              .. respond correctly to a command            */
BLD_NO_BOARD           = 17,  /* No board, or IO base address mismatch
                              .. between board and config file             */
BLD_RESET_FAILED       = 18,  /* Driver could not find a board. I/O-address
                              .. or IRQ mismatch between board and values
                              .. supplied to the driver.                   */
BLD_WRONG_CONTEXT      = 21,  /* Requested action is not valid in current
                              .. context. Maybe the function call sequence
                              .. was wrong                                 */
BLD_ALREADY_ACTIVATED  = 22,  /* Activate request was issued but L1 was
                              .. already activated                         */
BLD_DSP_NOT_CONFIGURED = 23,  /* DSP has not yet been configured              */
BLD_DL_REL_IND         = 24,  /* Layer-2 DM message received                  */
BLD_MDL_ERR_IND        = 25,  /* Network does not respond                     */
BLD_MDL_ERR_RESP       = 26,  /* Wrong response from network                  */
BLD_AUTO_MODE          = 27,  /* Operating in Auto mode                       */
BLD_NON_AUTO_MODE      = 28,  /* NOT operating in Auto mode                   */
BLD_INVALID_KEY        = 29,  /* Invalid keypad key                           */
BLD_BYTE_GEN_OCCUPIED  = 30,  /* B-channel byte generator occupied. Use
                              .. value 0                                   */
```

```
  BLD_INVALID_BYTE_VAL   = 31,  /* Byte value must be less than 128          */
  BLD_DSP_LOAD_FAILED    = 32,  /* Could not load DSP Application            */
  BLD_UNDER_CONSTRUCTION = 33,  /* This function is under construction       */
  BLD_INVALID_AIC_NO     = 34,  /* AIC Number must be 0 or 1                 */
  BLD_DTMF_BUSY          = 35,  /* Still sending previous set of DTMF numbers*/
  BLD_DTMF_NO_TOO_LONG   = 36,  /* DTMF phone number too long                */
  BLD_INVALID_DTMF_VAL   = 37,  /* Invalid DTMF value                        */
  BLD_DSP_COMM_FAILURE   = 38,  /* DSP<->Host Communication Failure          */
  BLD_DSP_NO_KERNEL_RESP = 39,  /* No response from DSP Kernel               */
  BLD_FOPEN_ERROR        = 40,  /* Could not open file                       */
  BLD_DSP_INVALID_ENTRYPT= 41,  /* DSP Application Entry Point could not be
                                 .. located                                   */
  BLD_TONE               = 42,  /* Tone is detected                          */
  BLD_NO_TONE            = 43,  /* Tone is not detected                      */
  BLD_DSP_ADDR_MISMATCH  = 44,  /* HPI Address Mismatch. Internal Error      */
  BLD_DRV_CALL_FAILED    = 45,  /* Call to the Driver failed.                */
  BLD_DRV_NOT_INITIALIZED = 46, /* The Driver has not been initialized       */
  BLD_DATA_TOO_LARGE     = 47,  /* Data does not fit to the allocated buffer */
  BLD_MSG_TOO_LONG       = 48,  /* The supplied message is longer than
                                 .. internal buffers                          */
  BLD_DEVICE_OPEN_FAILED = 49,  /* Unable to connect to the driver           */
  BLD_INVALID_CALLBACK_FUNCTION = 50,   /* Provided callback function is not
                                        ..  is not valid                      */
  BLD_CALLBACK_ALREADY_SET = 51,       /* Callback already set                 */
  BLD_UNABLE_TO_CREATE_CALLBACK_THREAD = 52,  /* Unable to create a callback
                                              .. thread                       */
} BldRc;
```

6.3 Frame Type

6.3.0.1 Synopsis

The type of the frame to be transmitted or the type of the received frame is indicated by the Frame Type (FM).

6.3.0.2 Definition

```
typedef enum {
  BLD_FM_HDLC,    /* Normal HDLC frame (including Layer 2)                    */
  BLD_FM_I,       /* I-frame (Only Layer 3, No Layer 2)                       */
  BLD_FM_UI,      /* UI-frame (Only Layer 3, No Layer 2)                      */
  BLD_FM_STR,     /* Information message (text) used to pass status and
                  .. debug information. Note: This has nothing to do with
                  .. the ISDN Layer 2 I-frames.                               */
  BLD_FM_DTMF1,   /* DTMF digit string dectected on B-channel 1               */
```

```
  BLD_FM_DTMF2,   /* DTMF digit string dectected on B-channel 2          */
  BLD_FM_UNDEF    /* Undefined frame type                                */
} BldFrameType;
```

6.4 Companding Algorithms

6.4.0.1 Synopsis

The type of companding method to be used on the B-channel data is indicated by the Law Type (LT).

6.4.0.2 Definition

```
typedef enum {
  LT_ULAW,      /* 8-bit u255 PCM code (American standard)             */
  LT_ALAW       /* 8-bit A-law PCM code (European standard)            */
} BldLawType;
```

6.5 Tone Generation Types

6.5.0.1 Synopsis

The type of tone to be generated is indicated by the Tone Type (TT).

6.5.0.2 Definition

```
typedef enum {
  TT_SINEWAVE,      /* 1kHz sinusoidal wave                            */
  TT_SAWTOOTH,      /* Ramp wave                                       */
  TT_SWEEPING_SINE, /* Variable frequency sinusoidal wave              */
  TT_NOISE          /* Random noise                                    */
} BldDigGainType;
```

6.6 Digital Gain

6.6.0.1 Synopsis

The digital gain of the generated tones is indicated by Digital Gain (DG).

6.6.0.2 Definition

```
typedef enum {
  DG_0DB = 0,         /* Full Gain                                     */
  DG_MINUS_3DB = 1,   /* -3dB Gain                                     */
  DG_MINUS_6DB = 2,   /* -6dB Gain                                     */
```

```
  DG_MINUS_9DB = 3,   /* -9dB Gain                                    */
  DG_MINUS_12DB = 4,  /* -12dB Gain                                   */
  DG_MINUS_15DB = 5   /* -15dB Gain                                   */
} BldLawType;
```

6.7 Line Interface Operating Modes

6.7.0.1 Synopsis

The Line Interfaces (LIs) can be configured to operate in either Terminal (TE) or Line Termination (LT_S) mode.

6.7.0.2 Definition

```
typedef enum {
  BLD_TE,         /* The line interface is configured to operate in Terminal
                  .. mode (as an TE).                                  */
  BLD_LT_S,       /* The line interface is configured to operate in Line
                  .. Termination mode (as an NT).                      */
  BLD_NO_IMODE
} LiMode;
```

6.8 Frame Information

6.8.0.1 Synopsis

Each received message is supplied with a data structure containing information on the received frame.

6.8.0.2 Definition

```
typedef struct {
  short liNo;                /* Number of the Line Interface which received
                             .. the message.                          */
  unsigned char  fmSeqNo;    /* Sequence number. All the received messages
                             .. are assigned a sequence number by the driver.
                             .. After 256 frames the numbering is started
                             .. again from 0.                         */
  unsigned short fmLength;   /* Length of the received message.          */
  unsigned char  hour;       /* Time of reception: Hour                  */
  unsigned char  min;        /* Time of reception: Minute                */
  unsigned char  sec;        /* Time of reception: Second                */
  unsigned char  csec;       /* Time of reception: Hundreds of second    */
  BldFrameType   fmType;     /* Type of the received frame (HDLC or INFO)  */
  unsigned char  fmStatus;   /* Status of the received frame             */
```

```
} BldFrameHeader;
```

7. API Function Definitions

7.1 Driver and Device Configuration Functions

7.1.1 bldConfigureDsp( )

7.1.1.1 Synopsis

The function *bldConfigureDsp()* loads and starts a DSP application program. The application file must be in TI COFF file format (*.out or *.obj). This function must be called prior to using any of the functions which operate on the B-channels. At least one DSP program is supplied with the driver package.

7.1.1.2 Definition

```
BldRc bldConfigureDsp(
  short dspNo,           /* Number of the DSP to be used                    */
  char *appfile,         /* Application file to be loaded                   */
  char *entryPoint       /* Label for the entry point of the application    */
                         /* Usually "_c_int00" for a C-compiled application */
);
```

7.1.1.3 Returns

```
BLD_SUCCESS
BLD_DSP_COMM_FAILURE
BLD_DSP_NO_KERNEL_RESP
BLD_FOPEN_ERROR
BLD_DSP_INVALID_ENTRYPT
```

7.1.2 bldConfigureDtmfDetector

7.1.2.1 Synopsis

The function *bldConfigureDtmfDetector()* configures the Dual Tone Multiple Frequency (DTMF) Detector. The received digits can either be stored in the FIFO and be extracted with a call to *bldRead()*; or, if *storeInFifo* == 0 then only the most recently received digit can be retrieved with a call to *bldGetStatusDtmf()*.

7.1.2.2 Definition

```
BldRc bldConfigureDtmfDetector(
  short liNo,              /* Number of the line interface (DSP) to be used */
  short bChannelNo,        /* Number of the B-channel to be used (1 or 2)   */
  short detectorEnable,    /* 1 = Enable DTMF detector, 0 = Disable it      */
```

```
  short storeInFifo,        /* 1 = Store received digits in Fifo, 0 = Don't
                            .. store. Only remember the last received digit  */
  BldLawType law            /* u-law or a-law companding                     */
);
```

7.1.2.3 Returns

```
BLD_SUCCESS
BLD_INVALID_LI_NO
BLD_INVALID_B_CHANNEL
```

7.1.3 bldConfigureLi( )

7.1.3.1 Synopsis

The function *bldConfigureLi()* sets up a Line Interface (ISAC-S) chip in either Terminal or Line Termination mode.

7.1.3.2 Definition

```
BldRc bldConfigureLi(
  short liNo,           /* Number of the Li chip to be configured            */
  LiMode liMode         /* Mode to be configured to, BLD_TE or BLD_LT_S      */
);
```

7.1.3.3 Returns

```
BLD_INVALID_LI_NO
BLD_NO_BOARD
BLD_INVALID_MODE
```

7.1.4 bldConfigureSilenceDetector( )

7.1.4.1 Synopsis

The function *bldConfigureSilenceDetector()* configures the Silence Detector. A silence is detected when a certain number of received samples have a combined energy level of less than a specified threshold. The threshold can be modified to accommodate the desired noise level. A higher threshold allows more noise to pass as silence.

7.1.4.2 Definition

```
BldRc bldConfigureSilenceDetector(
  short liNo,                /* Number of the line interface (DSP) to be
                             .. used                                          */
  short bChannelNo,          /* Number of the B-channel to be used (1 or 2)  */
```

```
  short enableDetector,      /* 1 = Silence Detector Enabled, 0 = Silence
                             .. Detector Disabled                          */
  short silenceDuration,     /* Time (in 12.75ms increments) that silence
                             .. must be present to be reported             */
  short thresHold,           /* A low thresHold stipulates that very little
                             .. noise will be accepted for silence to be
                             .. reported; A higher thresHold accepts more
                             .. more noise in the silence                  */
  BldLawType law             /* u-law or a-law companding                 */
);
```

7.1.4.3 Returns

```
BLD_SUCCESS
BLD_INVALID_LI_NO
BLD_INVALID_B_CHANNEL
```

7.1.5 bldConstructDriver( )

7.1.5.1 Synopsis

Tells the driver the communications parameters to be used with the board(s). This function only needs to be called in DOS. In Windows 95 and Windows NT the information is available to the driver from the Windows Registry. The function verifies that supplied IRQ numbers and I/O addresses are valid and that the boards are installed (or seem to be installed, rather). It also installs the Interrupt Service Routines (ISRs).

7.1.5.2 Definition

```
BldRc bldConstructDriver(
  short  boardType,     /* Type of the board used. Boardtype for Balder-2S is
                        .. 122 and for Balder-8S is 128.                    */
  short  boardBaseAddr, /* I/0-base address wrapped on the board. Base
                        .. address for Balder boards is valid if:
                        .. (0x000 < Addr <= 0x400) && (Addr % 0x20) == 0    */
  short  noOfBoards,    /* Number of Boards installed.                     */
  short  irq            /* IRQ to be used with all the boards.             */
);
```

7.1.5.3 Returns

```
BLD_SUCCESS
BLD_WRONG_CONTEXT
BLD_INVALID_BOARD_TYPE
BLD_INVALID_BOARD_NO
BLD_INVALID_IRQ_NO
```

```
BLD_INVALID_ADDR
BLD_NO_BOARD
```

7.1.5.4 See Also

```
bldInitDriver()
bldDestructDriver()
```

7.1.6 bldDestructDriver( )

7.1.6.1 Synopsis

The *bldDestructDriver()* function releases the driver from memory. This function must be called before the application is exited. The function cleans up and clears the driver after use. It also uninstalls the Interrupt Service Routines installed by the *bldConstructDriver()*.

7.1.6.2 Definition

```
BldRc bldDestructDriver(
  void
);
```

7.1.6.3 Returns

```
BLD_SUCCESS
BLD_WRONG_CONTEXT
```

7.1.6.4 See Also

```
bldConstructDriver()
bldInitDriver()
```

7.1.7 bldDigGainOnBch( )

7.1.7.1 Synopsis

The function *bldDigGainOnBch()* is a digital volume control for the generated sound on a B-channel. It modifies the gain of the signal.

7.1.7.2 Definition

```
BldRc bldDigGainOnBch(
  short liNo,                /* Number of the line interface to be used      */
  short bChannelNo,          /* Number of the B-channel to be used (1 or 2)  */
  BldDigGainType digGain     /* Digital gain of generated tone               */
);
```

7.1.7.3 Returns

```
BLD_SUCCESS
BLD_INVALID_LI_NO
BLD_INVALID_B_CHANNEL
```

7.1.8 bldFifoUsage( )

7.1.8.1 Synopsis

The function *bldFifoUsage()* sets up the information used to provide a FIFO for the HDLC data of a certain LI. It returns pointers to both a receive and a transmit buffer.

7.1.8.2 Definition

```
BldRc bldFifoUsage(
  short liNo,                /* Number of the line interface to be used      */
  unsigned short *rxHdlcFifo,
  unsigned short *rxInfoFifo,
  unsigned short *txHdlcFifo
);
```

7.1.8.3 Returns

Pointer to a memory location for the FIFO.

7.1.9 bldIdentDriver( )

7.1.9.1 Synopsis

The function *bldIdentDriver()* returns a pointer to an identification string for the driver. This function is useful in situations where the driver and the applications are not statically linked, and an application may want to query for the revision or name of a currently dynamically linked driver.

7.1.9.2 Definition

```
char *bldIdentDriver(
  void
);
```

7.1.9.3 Returns

Pointer to a string containing the driver identification.

7.1.10 bldInitDriver( )

7.1.10.1 Synopsis

Connects to the Driver and initializes the Driver internal data structures. In Windows 95 and Windows NT this function must be called before any other functions are called. In DOS this function needs to be called after a call to *bldConstructDriver()*.

7.1.10.2 Definition

```
BldRc bldInitDriver(
  void
);
```

7.1.10.3 Returns

```
BLD_SUCCESS
```

7.1.10.4 See Also

```
bldConstructDriver()
```

7.1.11 bldRegisterCallback( )

7.1.11.1 Synopsis

Registers a callback function in the application to be called by the driver when it receives a message or a status change.

Note: Only available in Windows 95 and Windows NT.

7.1.11.2 Definition

```
BldRc bldConstructDriver(
void (*notificationCallback)(void) // function pointer to the callback function
                                   // in the application to be called by the driver
);
```

7.1.11.3 Returns

```
BLD_SUCCESS
BLD_INVALID_CALLBACK_FUNCTION
BLD_CALLBACK_ALREADY_SET
BLD_UNABLE_TO_CREATE_CALLBACK_THREAD
```

7.1.12 bldResetDriver( )

7.1.12.1 Synopsis

The *bldResetDriver()* function resets the driver software. This is accomplished by clearing the receive and transmit memory buffers.

7.1.12.2 Definition

```
BldRc bldResetDriver(
  void
);
```

7.1.12.3 Returns

```
BLD_SUCCESS
```

7.1.12.4 See Also

```
BldResetLi()
```

7.2 Line Interface Functions

7.2.1 bldActivate( )

7.2.1.1 Synopsis

The function *bldActivate()* issues a layer 1 activation request.

7.2.1.2 Definition

```
BldRc bldActivate(
  short liNo             /* Number of the Line Interface to activate         */
);
```

7.2.1.3 Returns

```
BLD_SUCCESS
BLD_WRONG_CONTEXT
```

7.2.1.4 See Also

```
BldResetLi()
```

7.2.2 bldDeactivate( )

7.2.2.1 Synopsis

The function *bldDeactivate()* issues a layer 1 deactivation request. This function is valid only if the line interface is operating in LT_S (NT) mode.

7.2.2.2 Definition

```
BldRc bldDeactivate(
  short liNo             /* Number of the Line Interface to deactivate       */
);
```

7.2.2.3 Returns

```
BLD_SUCCESS
BLD_WRONG_CONTEXT
```

7.2.3 bldGetStatusLi( )

7.2.3.1 Synopsis

The function *bldGetStatusLi()* reads and returns the status of an LI.

7.2.3.2 Definition

```
BldRc bldGetStatusLi(
  short liNo                  /* Number of the Line Interface to be read      */
);
```

7.2.3.3 Returns

```
BLD_L1_OK
BLD_L1_DOWN
BLD_UNDEFINED    /* If LI is in LT_S mode in which case the sister
                 .. LI holds the status. (Only happens in monitor mode) */
```

7.2.4 bldResetLi( )

7.2.4.1 Synopsis

The function *bldResetLi()* performs a hardware reset on an LI.

7.2.4.2 Definition

```
BldRc bldResetLi(
  short liNo    /* Number of the Line Interface to reset             */
);
```

7.2.4.3 Returns

```
BLD_SUCCESS
BLD_L1_DOWN
BLD_RESET_FAILED
BLD_NO_CLOCKS
BLD_BAD_CHIP
```

7.2.4.4 See Also

```
BldResetDriver()
```

7.2.5 bldTermLi( )

7.2.5.1 Synopsis

The function *bldTermLi()* connects or disconnects the 100-ohm terminating resistor for a particular LI.

7.2.5.2 Definition

```
BldRc bldTermLi(
  short liNo,      /* Line Interface Number                  */
```

```
  short terminate  /* TRUE to connect and FALSE to disconnect */
);
```

7.2.5.3 Returns

```
BLD_SUCCESS
```

7.3 D-Channel Functions

7.3.1 bldRead( )

7.3.1.1 Synopsis

The function *bldRead()* reads the next received message. The driver will automatically sort the received messages from different LIs according to the reception time. If several messages are waiting to be read, this function will read and return the oldest one.

7.3.1.2 Definition

```
BldRc bldRead(
 unsigned char   fmBuf[],   /* Buffer into which the received message will
                             .. be written. NOTE: The buffer must be
                             .. allocated by the application.               */
  short          fmBufSize,  /* Size of fmBuf[].                             */
  BldFrameHeader *fmHeader   /* Pointer to header structure that will be
                             .. fillied in by the function. NOTE: The struct
                             .. must be allocated by the application.       */
);
```

7.3.1.3 Returns

```
BLD_SUCCESS     /* Frame read successfully                                */
BLD_NO_FRAMES   /* if no complete frames have been received and ready to
                .. be read.                                            */
```

7.3.2 bldWriteIframeLi( )

7.3.2.1 Synopsis

The function *bldWriteIframeLi()* sends an ISDN I-frame message in auto mode excluding Layer2 (added automatically by the Balder board).

7.3.2.2 Definition

```
BldRc bldWriteIframeLi(
  short         liNo,      /* Number of the Line Interface to be used for
                           .. sending.                                     */
  unsigned char fmBuf[],   /* Buffer containing the message to send.       */
  short         fmLength   /* Length of the message to be sent.            */
);
```

7.3.2.3 Returns

```
BLD_SUCCESS      /* Message accepted and is being sent                  */
BLD_TX_BUSY      /* Previous message not yet sent completely
                 .. Try again later                                     */
```

7.3.3 bldWriteLi( )

7.3.3.1 Synopsis

The function *bldWriteLi()* sends an ISDN message including layer 2. Use *bldWriteIframeLi()* to send I-frames when operating in auto mode.

7.3.3.2 Definition

```
BldRc bldWriteLi(
  short         liNo,      /* Number of the Line Interface to be used for
                           .. sending.                                  */
  unsigned char fmBuf[],   /* Buffer containing the message to send.       */
  short         fmLength   /* Length of the message to be sent.            */
);
```

7.3.3.3 Returns

```
  BLD_SUCCESS      /* Message accepted and is being sent                  */
  BLD_TX_BUSY      /* Previous message not yet sent completely
                   .. Try again later                                     */
```

7.4 B-Channel Functions

7.4.1 bldByteOffBch( )

7.4.1.1 Synopsis

The function *bldByteOffBch()* sends constant silence on a B-channel.

7.4.1.2 Definition

```
BldRc bldByteOffBch(
  short liNo,               /* Number of the line interface to be used      */
  short bChannelNo          /* Number of the B-channel to be used (1 or 2)  */
);
```

7.4.1.3 Returns

```
BLD_SUCCESS
```

7.4.1.4 See Also

```
bldByteOnBch()
```

7.4.2 bldByteOnBch( )

7.4.2.1 Synopsis

The function *bldByteOnBch()* starts sending a constant byte on the specified B-channel and keeps sending it until either another call to this function is made or until a non-constant value is generated by calling *bldDtmfOnBch(), bldPhoneOn(,* or *bldToneOnBch()*. The B-channel byte can have the value of 0 - 127. If a non-zero byte is sent on one B-channel, then the value of the other B-channel is restricted to 0. Note that this is the only B-channel function which does not use a compounding law. The byte value is sent as specified (no compression is performed).

7.4.2.2 Definition

```
BldRc bldByteOnBch(
  short liNo,             /* Number of the line interface to be used      */
  short bChannelNo,       /* Number of the B-channel to be used (1 or 2)  */
  unsigned char aByte     /* Byte to be send                              */
);
```

7.4.2.3 Returns

```
BLD_SUCCESS
```

7.4.2.4 See Also

```
bldByteOffBch()
```

7.4.3 bldByteReadBch( )

7.4.3.1 Synopsis

The function *bldByteReadBch()* reads one byte from the specified B-channel and returns it in *recByte*.

7.4.3.2 Definition

```
BldRc bldByteReadBch(
  short liNo,              /* Number of the line interface to be used      */
  short bChannelNo,        /* Number of the B-channel to be used (1 or 2)  */
  unsigned char *recByte   /* Received byte from the B-channel             */
);
```

7.4.3.3 Returns

```
BLD_SUCCESS
```

7.4.3.4 See Also

```
BldResetLi()
```

7.4.4 bldDtmfOnBch( )

7.4.4.1 Synopsis

The function *bldDtmfOnBch()* dials a phone number using DTMF tones on a B-channel. The time between tones can be specified in the *silence* parameter and the duration of each tone is determined by the value of the *duration* parameter.

7.4.4.2 Definition

```
BldRc bldDtmfOnBch(
  short liNo,              /* Number of the line interface to be used      */
  short bChannelNo,        /* Number of the BCH to be used                 */
  char *phoneNumber,       /* Phone number to dial                         */
  short duration,          /* Duration of DTMF tones in 12.75 ms increments
                           ..  (0 for infinity)                            */
  short silence            /* Silence time between of DTMF tones in 12.75 ms
                           .. increments                                   */
  BldDigGainType digGain,  /* Digital gain of generated tones              */
  BldLawType law           /* u-law or a-law compounding                    */
```

```
);
```

7.4.4.3 Returns

```
BLD_SUCCESS
BLD_INVALID_LI_NO
BLD_INVALID_B_CHANNEL
```

7.4.5 bldGetStatusDtmf( )

7.4.5.1 Synopsis

The function *bldGetStatusDtmf()* returns the most recent DTMF digit that was detected on a specified B-channel in the *dtmfDigit* parameter. If there has been no detected DTMF tone since the last call to this function, then it returns *BLD_NO_TONE*.

7.4.5.2 Definition

```
BldRc bldGetStatusDtmf(
  short liNo,         /* Number of the line interface (DSP) to be used       */
  short bChannelNo,   /* Number of the B-channel to be used (1 or 2)         */
  char *dtmfDigit     /* Most recent DTMF digit detected (or \0 if none)     */
);
```

7.4.5.3 Returns

```
BLD_TONE
BLD_NO_TONE
```

7.4.6 bldGetStatusSilence( )

7.4.6.1 Synopsis

The function *bldGetStatusSilence()* returns the status of the Silence Detector for a specified B-channel. The function returns *BLD_NO_TONE* if silence is detected and *BLD_TONE* for non-silence.

7.4.6.2 Definition

```
BldRc bldGetStatusSilence(
  short liNo,       /* Number of the line interface (DSP) to be used       */
  short bChannelNo  /* Number of the B-channel to be used (1 or 2)         */
);
```

7.4.6.3 Returns

```
BLD_NO_TONE
BLD_TONE
```

7.4.7 bldPhoneOff( )

7.4.7.1 Synopsis

The function *bldPhoneOff()* disconnects the Analog Interface Circuit from a DSP if the indicated DSP (*liNo*) has an AIC connected.

7.4.7.2 Definition

```
BldRc bldPhoneOff(
  short liNo,         /* Number of the line interface to be used      */
);
```

7.4.7.3 Returns

```
BLD_SUCCESS
BLD_INVALID_LI_NO
BLD_INVALID_B_CHANNEL
```

7.4.8 bldPhoneOn( )

7.4.8.1 Synopsis

The function *bldPhoneOn()* connects one of the two Analog Interface Circuits (AIC0 or AIC1) to one of the two B-channels in one of the 8 DSPs. Note that only one AIC can be connected to one DSP at any time.

7.4.8.2 Definition

```
BldRc bldPhoneOn(
  short liNo,            /* Number of the line interface to be used         */
  short bChannelNo       /* Number of the BCH to be used                    */
  short aicNo,           /* Number of the Analog Interface Circuit be used  */
  short inputSrc,        /* 0 = Handset Mic, 1 = Line Input; of the AIC     */
  short spkrOnly,        /* 1 = AIC will not generate sound towards Line
                         .. Interface                                       */
  short spkrLoop,        /* 1 = Sound generated by this DSP is looped back
                         .. to the Handset speaker                          */
  BldDigGainType digGain,/* Digital gain of microphone/line input samples   */
  BldLawType law         /* u-law or a-law companding                       */
);
```

7.4.8.3 Returns

```
BLD_SUCCESS
BLD_INVALID_LI_NO
BLD_INVALID_B_CHANNEL
```

7.4.9 bldPlaybackBch( )

7.4.9.1 Synopsis

Plays (sends) signed short samples from a file on a B-channel. The 16-bit short values are companded to 8-bit value before they are sent out.

7.4.9.2 Definition

```
BldRc bldPlaybackBch(
  short dspNo,              /* Number of the line interface (DSP) to be used */
  short bChannelNo,         /* Number of the BCH to be used                  */
  char *fileName,           /* Source File Name for B-channel samples  */
  BldDigGainType digGain,   /* Digital gain of the sent samples */
  BldLawType law            /* u-law or a-law companding */
);
```

7.4.9.3 Returns

```
BLD_SUCCESS
BLD_FOPEN_ERROR
BLD_TIMEOUT
```

7.4.9.4 See Also

```
bldRecordBch()
```

7.4.10 bldRecordBch( )

7.4.10.1 Synopsis

Records a specified number of samples (duration) from a B-channel. The 8-bit B-channel samples are companded to signed int's according to the specified companding law. The companded data is then stored in a binary file (file name is a passed as a parameter).

7.4.10.2 Definition

```
BldRc bldRecordBch(
  short dspNo,            /* Number of the line interface (DSP) to be used */
  short bChannelNo,       /* Number of the BCH to be used                  */
```

```
  char *fileName,         /* Destination File Name for B-channel samples  */
  long duration,         /* Duration of B-channel recording in increments of ..
                          .. 12.75.ms  */
  BldLawType law          /* u-law or a-law companding */
);
```

7.4.10.3 Returns

```
BLD_SUCCESS
BLD_FOPEN_ERROR
BLD_TIMEOUT
```

7.4.10.4 See Also

```
bldPlaybackBch()
```

7.4.11 bldSinewaveOnBch( )

7.4.11.1 Synopsis

The *bldSinewaveOnBch()* function begins generating a custom sine wave on the specified B-channel.

7.4.11.2 Definition

```
BldRc bldSinewaveOnBch(
  short dspNo,         /* Number of the line interface (DSP) to be used */
  short bChannelNo,    /* Number of the B-channel to be used (1 or 2)   */
  short coeff,         /* Coefficient of sinewave (modifies frequency)  */
  short y1,            /* Initial y(n-1) value (modifies amplitude)     */
  short y2,            /* Initial y(n-2) value (modifies amplitude)     */
  BldDigGainType digGain,   /* Digital gain of generated tone                */
  BldLawType law            /* u-law or a-law compounding                    */
);
```

7.4.11.3 Returns

```
BLD_SUCCESS
```

7.4.11.4 See Also

```
BldToneOnBch()
```

7.4.12 bldToneOffBch( )

7.4.12.1 Synopsis

Stops generating a sound on a specified B-channel. Starts outputting a constant 0x7F (silence) on the B-channel. This function call has the same effect as a call to *bldByteOffBch( )*.

7.4.12.2 Definition

```
BldRc bldToneOffBch(
  short liNo,              /* Number of the line interface to be used      */
  short bChannelNo,        /* Number of the B-channel to be used (1 or 2)  */
);
```

7.4.12.3 Returns

```
BLD_SUCCESS
BLD_INVALID_LI_NO
BLD_INVALID_B_CHANNEL
```

7.4.12.4 See Also

```
bldToneOnBch( )
```

7.4.13 bldToneOnBch( )

7.4.13.1 Synopsis

The function *bldToneOnBch()* starts generating a desired sound on a specified B-channel. A call to this function overrides any prior B-channel related calls (e.g. *bldPhoneOn(), bldDtmfOnBch(), bldByteOnBch()*).

7.4.13.2 Definition

```
BldRc bldToneOnBch(
  short liNo,                /* Number of the line interface to be used      */
  short bChannelNo,          /* Number of the B-channel to be used (1 or 2)  */
  BldToneType toneType,      /* Type of "tone" to be generated on the Bch    */
  BldDigGainType digGain,    /* Digital gain of generated tone               */
  BldLawType law             /* u-law or a-law companding                    */
);
```

7.4.13.3 Returns

```
BLD_SUCCESS
BLD_INVALID_LI_NO
```

`BLD_INVALID_B_CHANNEL`

7.4.13.4 See Also

`bldToneOffBch( )`

7.5 Miscellaneous

7.5.1 bldGetErrMsg( )

7.5.1.1 Synopsis

The function *bldGetErrMsg()* converts a Bld-function return code into a string describing the error. The function returns a pointer to a static string owned by the function.

7.5.1.2 Definition

```
char *bldGetErrMsg(
  BldRc errCode         /* Error code to be converted                        */
);
```

7.5.1.3 Returns

This function returns a pointer to a test string describing the error code in a general fashion. This can be used for quick and dirty solutions when the return code is not analyzed properly by the application but something needs to be printed.

Doc. No. 1211-1-SCA-1000-1
For more information on this product, please contact:

Odin TeleSystems Inc.
800 E. Campbell Road, Suite 300
Richardson, Texas 75081
U. S. A.

Tel: +1-972-664-0100
Fax: +1-972-664-0855
Email: Info@OdinTS.com
URL: http://www.OdinTS.com/