Odin TeleSystems Inc.

Application Programming Interface (API) Reference Guide for BALDER DOS Driver, Rev. 1.0

Doc. No. 1211-1-SDA-1000-1

Rev. 1.3 (Released)

June 26, 1997

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 Content

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-2 board contains two and the Balder-8 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 a full implementation of an ISDN digital phone.

The Balder boards are delivered with a DOS driver and an Application Programming Interface (API). This document contains a reference for the API. Figure 1 shows the relationship between the API and the OSI and ISDN protocol stacks.

.

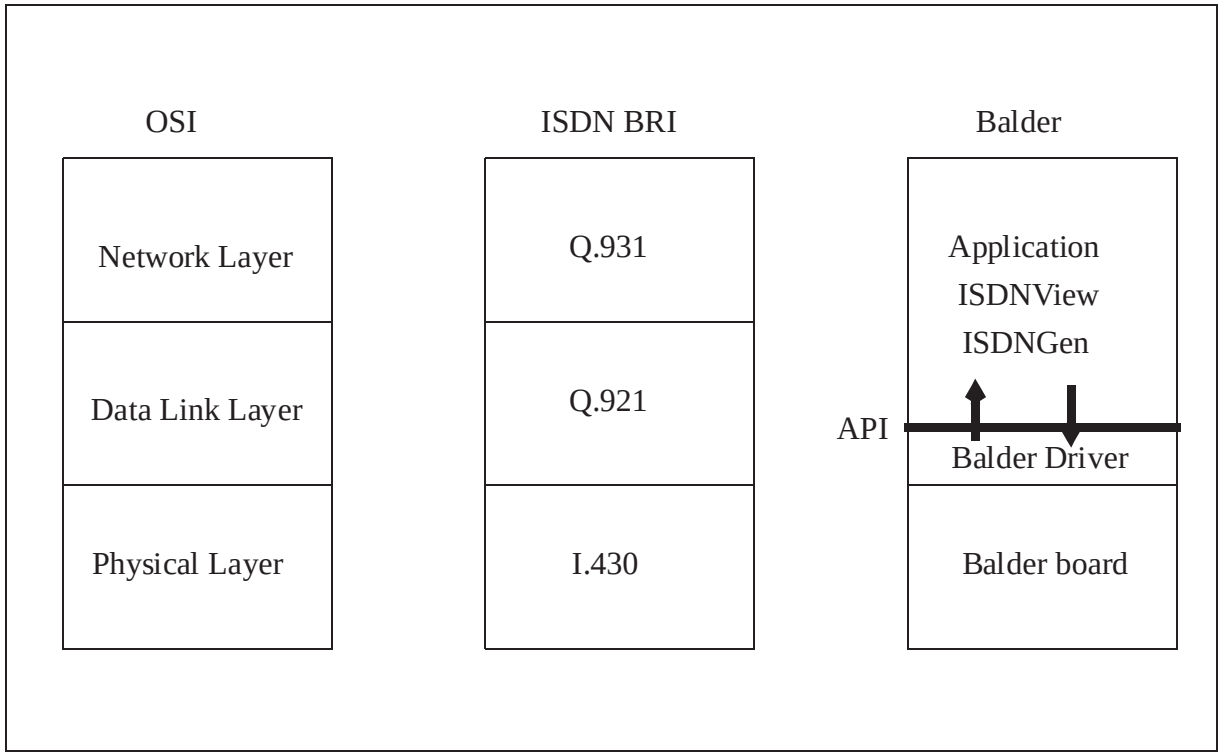

Figure 1. Layered Model

3. General

3.1 Files

The balder driver consists of two files:

- BALDER.H
- BLD.LIB

BLD.LIB is a C library file which can be linked to the C or C++ application. BALDER.H is an include header file containing the macro definitions, type definitions, and function declarations for the Balder Library.

3.2 Compilers

The library can be used with the following compilers:

- Borland C++ Compiler, Versions 3.1, 4.0, and 4.5
- Microsoft C Compiler, Versions 5.1, and 6.0

3.3 Naming Conventions

All functions and data types used within the API follow a naming convention. All the names have a prefix ‘Bld’ (for Balder). The function names consist of three parts: the header ‘Bld’, operation, and target. The operation word is a verb describing the action to be taken, e.g., construct, read. The target can be either ‘Driver’ or ‘Li’. ‘Driver’ functions have effect on the whole system, when ‘Li’ function only affect specified Line Interfaces.

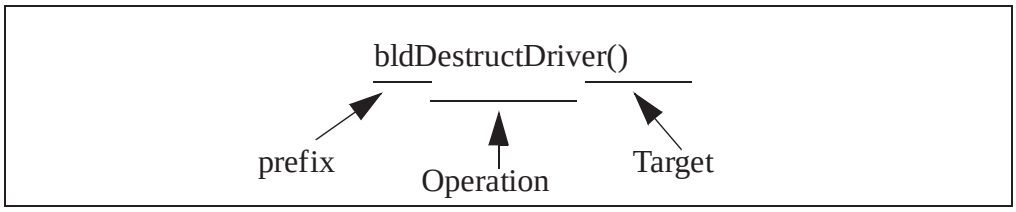

Figure 2. An Example of the Naming of the Functions

3.4 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-2 board contains 2 line interfaces (LIs) and each Balder-8 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-2 and Balder-8, respectively.

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

TABLE 1. Numbering of LIs

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

3.5 Exception Handling

Exception handling is provided via return codes. Each function returns 0 if successful (no errors) and an other non-zero value in the case of a problem. The return codes can be translated into strings containing the error message with the *bldGetErrMsg( )* function.

4. API Macro and Type Definitions

4.1 Frame End Codes

4.1.0.1 Synopsis

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

4.1.0.2 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                         */
```

4.2 Return Codes

4.2.0.1 Synopsis

Return Codes are used to return the execution result from the API functions.

4.2.0.2 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
                                .. 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 chip
                              .. functions.                              */
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 by L1 was
                              .. already activated                       */
BLD_ARCOFI_FAILED      = 23,  /* Arcofi did not respond to init
                              .. commands                                */
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           */
```

```
} BldRc;
```

4.3 Frame Type

4.3.0.1 Synopsis

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

4.3.0.2 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_UNDEF     /* Undefined frame type                                  */
} BldFrameType;
```

4.4 Line Interface Operating Modes

4.4.0.1 Synopsis

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

4.4.0.2 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;
```

4.5 Frame Info

4.5.0.1 Synopsis

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

4.5.0.2 Definition

```
typedef struct {
  int 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;
```

5. API Function Definitions

5.1 Driver Function

5.1.1 bldIdentDriver( )

5.1.1.1 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 where the application may want to query for the revision or name of a currently dynamically linked driver.

5.1.1.2 Definition

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

5.1.1.3 Returns

Pointer to a string containing the driver identification.

5.1.2 bldConstructDriver( )

5.1.2.1 Synopsis

The function *bldConstructDriver()* is called to initialize the driver. This function needs to be executed before the driver can be used. The function verifies that supplied IRQ numbers and I/O addresses are valid and that the boards are installed (or seems to be installed, rather. It also installs the Interrupt Service Routines (ISRs), but not until disabling the IRQ from all LI's on all boards.

5.1.2.2 Definition

```
BldRc bldConstructDriver(
  int  boardType,       /* Type of the board used. Boardtype for Balder-2 is
                        .. 102 and for Balder-8 is 108.                    */
  int  boardBaseAddr,   /* I/0-base address wrapped on the board. Base
                        .. address for Balder boards is valid if:
                        .. (0x200 < Addr <= 0x400) && (Addr % 0x20) == 0     */
  int  noOfBoards,      /* Number of Boards installed.                      */
  int  irqTable[]       /* Table containing IRQs for all the boards installed
                        .. For example: if two boards with IRQs 10 and 11
                        .. have been installed, then irqTable should contain
```

```
                                .. two items, 10 and 11. irqTable[] = { 10, 11 }     */
);
```

5.1.2.3 Returns

```
BLD_SUCCESS
BLD_WRONG_CONTEXT
BLD_INVALID_BOARD_TYPE
BLD_INVALID_BOARD_NO
BLD_INVALID_IRQ_NO
BLD_INVALID_ADDR
BLD_NO_BOARD
```

5.1.3 bldDestructDriver( )

5.1.3.1 Synopsis

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

5.1.3.2 Definition

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

5.1.3.3 Returns

```
BLD_SUCCESS
BLD_WRONG_CONTEXT
```

5.1.4 bldResetDriver( )

5.1.4.1 Synopsis

The *bldResetDriver()* function resets the driver software. Clears the receive and transmit memory buffers.

5.1.4.2 Definition

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

5.1.4.3 Returns

```
BLD_SUCCESS
```

5.1.4.4 See Also

```
BldResetLi()
```

5.2 Line Interface Functions

5.2.1 bldConfigureLi( )

5.2.1.1 Synopsis

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

5.2.1.2 Definition

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

5.2.1.3 Returns

```
BLD_INVALID_LI_NO
BLD_NO_BOARD
BLD_INVALID_MODE
```

5.2.2 bldGetStatusLi( )

5.2.2.1 Synopsis

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

5.2.2.2 Definition

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

5.2.2.3 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) */
```

5.2.3 bldActivate( )

5.2.3.1 Synopsis

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

5.2.3.2 Definition

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

5.2.3.3 Returns

```
BLD_SUCCESS
BLD_WRONG_CONTEXT
```

5.2.3.4 See Also

```
BldResetLi()
```

5.2.4 bldDeactivate( )

5.2.4.1 Synopsis

The function *bldDeactivate()* issues a layer 1 deactivation request.

5.2.4.2 Definition

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

5.2.4.3 Returns

```
BLD_SUCCESS
BLD_WRONG_CONTEXT
```

5.2.5 bldResetLi( )

5.2.5.1 Synopsis

The function *bldResetLi()* performs a hardware reset on the Line Interface.

5.2.5.2 Definition

```
BldRc bldResetLi(
  int liNo              /* Number of the Line Interface to reset             */
```


```
);
```

5.2.5.3 Returns

```
BLD_SUCCESS
BLD_L1_DOWN
BLD_RESET_FAILED
BLD_NO_CLOCKS
BLD_BAD_CHIP
```

5.2.5.4 See Also

```
BldResetDriver()
```

5.2.6 bldTermLi( )

5.2.6.1 Synopsis

The function *bldTermLi()* connects or disconnects the terminating resistor for a particular Li.

NOTE: This function works only if the board is equipped with analog switches. In the default configuration the analog switches are not included.

5.2.6.2 Definition

```
BldRc bldTermLi(
  int liNo,
  int    terminate  /* TRUE to connect and FALSE to disconnect */
);
```

5.2.6.3 Returns

```
BLD_SUCCESS
```

5.3 Message Sending and Receiving

5.3.1 bldRead( )

5.3.1.1 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.

5.3.1.2 Definition

```
BldRc bldRead(
 unsigned char   fmBuf[],   /* Buffer into which the received message will
                               .. be written. NOTE: The buffer must be
                               .. allocated by the application.              */
  int             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.       */
);
```

5.3.1.3 Returns

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

5.3.2 bldWriteLi( )

5.3.2.1 Synopsis

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

5.3.2.2 Definition

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

5.3.2.3 Returns

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

5.3.3 bldWriteIframeLi( )

5.3.3.1 Synopsis

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

5.3.3.2 Definition

```
BldRc bldWriteIframeLi(
  int             liNo,
  unsigned char   fmBuf[],
  int             fmLength
);
```

5.3.3.3 Returns

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

5.4 Phone Functions

5.4.1 bldPhoneOn( )

5.4.1.1 Synopsis

The function *bldPhoneOn()* connects a B-channel for a particular Li to the handset connector.

5.4.1.2 Definition

```
BldRc bldPhoneOn(
  int liNo,             /* Number of the line interface to be used      */
  int bChannelNo        /* Number of the BCH to be used                 */
);
```

5.4.1.3 Returns

```
BLD_SUCCESS
BLD_INVALID_LI_NO
BLD_INVALID_B_CHANNEL
BLD_ARCOFI_FAILED
```

5.4.2 bldPhoneOff( )

5.4.2.1 Synopsis

The function *bldPhoneOff()* disconnects the B-channel for the indicated Li from the handset connector

5.4.2.2 Definition

```
BldRc bldPhoneOff(
```

```
  int liNo,             /* Number of the line interface to be used      */
  int bChannelNo        /* Number of the BCH to be used                 */
);
```

5.4.2.3 Returns

```
BLD_SUCCESS
BLD_INVALID_LI_NO
BLD_INVALID_B_CHANNEL
```

5.4.3 bldDtmfOnBch( )

5.4.3.1 Synopsis

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

5.4.3.2 Definition

```
BldRc bldDtmfOnBch(
  int liNo,                 /* Number of the line interface to be used      */
  int bChannelNo,           /* Number of the BCH to be used                 */
  char *phoneNumber,        /* Phone number to dial                         */
  int duration,             /* Duration of DTMF tones in clock ticks (0 for
                            ..  (0 for infinity)                           */
  int silence               /* Silence time between of DTMF tones in clock
                            .. clock ticks                                 */
);
```

5.4.3.3 Returns

```
BLD_SUCCESS
BLD_INVALID_LI_NO
BLD_INVALID_B_CHANNEL
BLD_ARCOFI_FAILED
```

5.4.4 bldResetCodec( )

5.4.4.1 Synopsis

The function *bldResetCodec()* performs a software reset of the Codec.

5.4.4.2 Definition

```
BldRc bldResetCodec(
  int liNo                  /* Number of the line interface to be used      */
```

```
);
```

5.4.4.3 Returns

```
BLD_SUCCESS
BLD_ARCOFI_FAILED
```

5.5 Test Functions

5.5.1 bldByteOnBch( )

5.5.1.1 Synopsis

The function *bldByteOnBch()* starts sending a constant Byte on the specified B-channel and keeps sending it until turned off with the bldByteOffBch() function. The B-channel byte can have the value of 0 - 127. If a non-zero byte is set on one B-channel then the value of the other B-channel is restricted to 0.

5.5.1.2 Definition

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

5.5.1.3 Returns

```
BLD_SUCCESS
```

5.5.1.4 See Also

```
bldByteOffBch()
```

5.5.2 bldByteOffBch( )

5.5.2.1 Synopsis

The function *bldByteOffBch()* stops the sending of the constant byte value on the specified B-channel and turns the idle pattern (0xFF) on.

5.5.2.2 Definition

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

5.5.2.3 Returns

```
BLD_SUCCESS
```

5.5.2.4 See Also

```
bldByteOnBch()
```

5.5.3 bldByteReadBch( )

5.5.3.1 Synopsis

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

5.5.3.2 Definition

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

5.5.3.3 Returns

```
BLD_SUCCESS
```

5.5.3.4 See Also

```
BldResetLi()
```

5.5.4 bldBchLoopOn( )

5.5.4.1 Synopsis

The function *bldBchLoopOn()* loops a B-channel's receive pair to the transmit pair.

5.5.4.2 Definition

```
BldRc bldBchLoopOn(
  int liNo,                /* Number of the line interface to be used      */
  int bChannelNo           /* Number of the BCH to be used                 */
);
```

5.5.4.3 Returns

```
BLD_SUCCESS
BLD_INVALID_LI_NO
BLD_INVALID_B_CHANNEL
```

5.5.5 bldTestMode( )

5.5.5.1 Synopsis

The function *bldTestMode()* puts the Li into test mode during which a continuous square wave is sent out on the transmit pair. The frequency of the square wave is 2kHz

5.5.5.2 Definition

```
BldRc bldTestMode(
  int liNo              /* Number of the line interface to be used          */
);
```

5.5.5.3 Returns

```
BLD_INVALID_LI_NO
BLD_WRONG_CONTEXT
BLD_SUCCESS
```

5.6 Miscellaneous

5.6.1 bldGetErrMsg( )

5.6.1.1 Synopsis

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

5.6.1.2 Definition

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

5.6.1.3 Returns

Returns a pointer to test string describing the errors code in a general fashion. Can be used for quick and dirty solutions when the return code is not analyzed properly by the application but at least something needs to be printed.

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

Odin TeleSystems Inc.
800 E. Campbell Road, Suite 310
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/