MIL-STD-1553
PROGRAMMING MANUAL

for BTIDriver-Compliant Devices

May 12, 2016
Rev. H.0

Astronics Ballard Technology
Phone : +1.425.339.0281 800.829.1553
Email : support@ballardtech.com
Web : www.ballardtech.com

COPYRIGHT NOTICE

SAFETY WARNING

Ballard products are not intended, warranted, or certified for any particular use or application or for use in any application where failure of the products could lead directly to death, personal injury, or damage to property. Customers, licensees, and/or users are responsible for establishing and assuring suitability and proper use of Ballard products for their particular use or application.

TRADEMARKS

Astronics Ballard Technology
Phone : +1.425.339.0281 800.829.1553
Email : support@ballardtech.com
Web : www.ballardtech.com

ASTRONICS
BALLARD TECHNOLOGY

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

This page intentionally blank.

1. INTRODUCTION

This manual documents the general functions and MIL-STD-1553 functions of Astronics Ballard Technology's (Ballard) unified API library called **BTIDriver™**. You can use this manual to learn how to create custom applications for any of Ballard's MIL-STD-1553 BTIDriver-compliant interface products.

1.1 MIL-STD-1553 Overview

MIL-STD-1553 is a standard defining a local area network (LAN) originally developed for and widely used on military aircraft. This digital, command-response, time-division multiplexing network protocol is also used in many other military and commercial applications where fast and positive control is required. The standard defines the handshaking, data formats, and timing requirements of the protocol as well as the electrical characteristics of the bus and the terminals' interface electronics. This manual assumes that the user is familiar with the MIL-STD-1553 protocol.

1.2 BTIDriver for MIL-STD-1553

BTIDriver is a unified library of API (Application Program Interface) functions designed to control Ballard's BTIDriver-compliant hardware products. Ballard makes many hardware products that interface with various avionics databuses to facilitate avionics development, simulation, and testing. Software is used to operate these hardware Devices. Programmers can use BTIDriver to create custom software for Ballard hardware Devices.

BTIDriver supports different avionics databus protocols and different Ballard hardware Devices. As long as the Devices have similar hardware capability, BTIDriver applications written for one Device can run on another Device with little or no change. This manual only documents the principles and functions used for MIL-STD-1553 software applications. Other protocols are documented separately. Appendix C provides guidance for writing multi-protocol BTIDriver applications.

1.3 Ballard MIL-STD-1553 Devices

BTIDriver supports 1553 hardware Devices with different levels of capability. Each MIL-STD-1553 channel can be one of the ten levels summarized in Table 1.1 below. The examples and features described in this manual apply to all 1553 channels unless otherwise noted.

	Levels of Functionality									
Features	**A**	**B4**	**B32**	**C**	**D**	**S**	**M**	**P**	**B**	**R**
BC/RT/MON Operation										
BC or 1 RT or MON	✓									
BC and 3 RTs or 4 RTs or MON		✓								
BC and 32 RTs or MON			✓							
BC or 32 RTs or MON						✓				
BC and 32 RTs and MON				✓	✓		✓	✓		
32 RTs or MON										✓
MON Only									✓	
Number of Simultaneous Terminals	1	4	1 or 32	32	32	1 or 32	32	32	1	32
Monitor	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Filtering for terminal address	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Filtering for subaddress		✓	✓	✓	✓	✓	✓	✓	✓	✓
Concurrent terminal monitoring				✓	✓		✓	✓		
Protocol Error Injection				✓	✓		✓	✓		
Variable Transmit Amplitude					✓			✓		
Zero Crossing Distortion					✓			✓		

Table 1.1—Levels of Ballard MIL-STD-1553 Functionality

Note: Table A.3 shows the hardware Devices that operate directly with the BTIDriver/BTI1553 API.

1.4 CoPilot

As an alternative to writing your own custom programs, you can operate your MIL-STD-1553 interface Device with CoPilot®, Ballard's Windows®-based GUI (Graphical User Interface) software (see Figure 1.1). CoPilot greatly simplifies such tasks as defining and scheduling bus messages and capturing and analyzing data. Using CoPilot, you can transmit, monitor, and record databus messages with a few clicks of the mouse. CoPilot is user-friendly and has many timesaving features. For example, bus messages can be automatically detected, posted in the hardware tree, and associated with the appropriate attributes from the database of equipment, message, and engineering unit specifications.

CoPilot users can quickly configure, run, and display the activity of multiple databases in a unified view. Data can be observed and changed in engineering units while the bus is running. The Strip View graphically illustrates the history of the selected data values. Data can also be entered and viewed as virtual instruments (knobs, dials, gauges, etc.)—that can be created by the user or automatically generated by dragging and dropping an item into the Control View window.

CoPilot can host multiple channels and databus protocols in the same project, making it an ideal tool for operating multi-protocol Devices. CoPilot can be purchased separately or with a BTIDriver-compliant hardware Device. For more information on or for a free evaluation copy of CoPilot, contact Customer Support (see Section 1.6). In addition, you can learn more about the latest version of CoPilot at www.ballardtech.com.

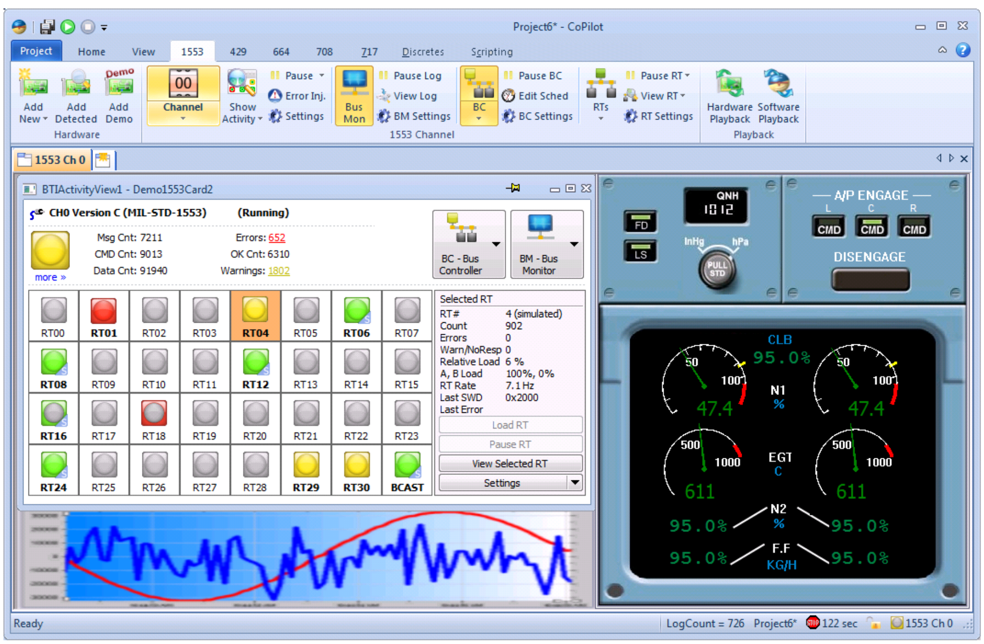


Figure 1.1—Sample CoPilot Screen

1.5 How to Use this Manual

This manual is designed to be both a tutorial and a reference guide. You can read only the sections that you need and refer to the rest as required. After reading Chapter 2 (Programming Basics), Chapter 3 (Single-Terminal Simulation), and referring to Appendix A (Function References), you should be able to write simple computer programs to operate your Ballard MIL-STD-1553 interface Device. Refer to Chapters 4 and 5 for applications that are more complex. This guide can be used in conjunction with the programming manuals for other avionics databus protocols (see Appendix C).

This manual assumes that you are familiar with the essentials of compiling, linking, and running programs in C. It is also assumed that you are familiar with the MIL-STD-1553 protocol. With minor exceptions, the content of this manual also applies to other programming languages.

The following conventions are observed throughout this manual:

- *Device* with a capital *D* is used generically to mean any Ballard BTIDriver-compliant interface device.
- Driver function names are in bold type and are all prefixed by `BTICard_` or `BTI1553_` (e.g., `BTI1553_RTConfig`).
- A lowercase *h* suffix indicates hexadecimal values (e.g., F01Ch).
- Constants defined in the driver software are written in all capital letters (e.g., `ERRDEF1553_PAR`).
- The symbol *??* is used in function names in this manual to indicate a category of functions with similar uses or attributes. These characters should be replaced by a category prefix or suffix in an actual function call (e.g., `BTI1553_MsgData??` to represent `BTI1553_MsgDataRd` and `BTI1553_MsgDataWr`).

1.6 Support and Service

Astronics Ballard Technology offers technical support before and after purchase of our products. Our hours of operation are 9:00 AM to 5:00 PM Pacific Time (PT). Support and sales engineers are often available outside these hours as well. We invite your questions and comments on any of our products.

Our contacts are:
Telephone: +1.425.339.0281
Fax: +1.425.339.0915
Web: www.ballardtech.com
Email: support@ballardtech.com

1.7 Updates

At Astronics Ballard Technology, we take pride in high quality, reliable products that meet the needs of our customers. Because we are continually improving our products, we may issue periodic updates to our documentation and software. Register your product at www.ballardtech.com for information on updates, customer service, and new products.

This page intentionally blank.

2. PROGRAMMING BASICS

Use of a Ballard Device requires software running on the host computer. BTIDriver is an Application Program Interface (API) that simplifies the development of software applications. The API is a library of functions your program calls to configure and control the Device. The remainder of this manual explains how to use the BTIDriver API to develop software for a Ballard 1553 Device.

This chapter explains the basic structure of any software application that uses BTIDriver. Helpful information is also covered on the internal data structures and associated BTIDriver functions. Chapter 3 introduces sample source code for single-terminal simulation, Chapter 4 discusses the use of advanced features, and Chapter 5 covers multi-terminal simulation and error injection.

2.1 Skeleton Application

BTIDriver was designed to be simple to use. Therefore, most programs can be modeled on the skeleton application shown in Figure 2.1 below.

```
HCARD hCard;
HCORE hCore;
INT cardnum = 0;       //Assumes only one Device has been installed
INT corenum = 0;             // Assumes only one core on the Device

BTICard_CardOpen(&hCard,cardnum);                       //Open the Device
BTICard_CoreOpen(&hCore,corenum,hCard);                  //Open each core
BTICard_CardReset(hCore);                               //Reset each core

BTI1553_??Config(...,hCore); //Configure channel as BC, RT, and/or Monitor

          //Create and initialize message structure(s) for the terminal(s)
Msg = BTI1553_??CreateMsg(...,hCore);
BTI1553_MsgDataWr(...,Msg,hCore);

BTICard_CardStart(hCore);                               //Start each core
                       //Handle data as required by the application
BTI1553_MsgDataRd(...,Msg,hCore);
BTI1553_MsgDataWr(...,Msg,hCore);

BTICard_CardStop(hCore);                                 //Stop each core
BTICard_CardClose(hCard);                              //Close the Device
```

Figure 2.1—Skeleton Application

The ellipses (...) in Figure 2.1 indicate parameters not shown, and the *??* prefixes should be replaced by either BC, RT, or Mon. As Figure 2.1 shows, using BTIDriver to control a 1553 Device involves eight steps:

1. Open the Device (card and cores)
2. Reset the Device (each core)—optional, but recommended
3. Configure terminal type(s)
4. Create and initialize message structure(s)
5. Start the Device (each core)
6. Handle message data according to application requirements
7. Stop the Device (each core)—optional
8. Close the Device (card)

There can be some variation to this layout. For example, no message creation (`BTI1553_??CreateMsg`) is required to run the Device as a monitor. Furthermore, between when it is opened and when it is closed, the Device may be started, stopped, reset, and/or reconfigured any number of times.

However, nearly all programs use the following functions:

- `BTICard_CardOpen`
- `BTICard_CoreOpen`
- `BTICard_CardReset`
- `BTI1553_??Config` (where *??* is either BC, RT, or Mon)
- `BTICard_CardStart`
- `BTICard_CardStop`
- `BTICard_CardClose`

The sample code in Figure 2.1 assumes that there is only one Device installed in the system, so the card number (`cardnum`) is given a value of zero, and that the Device has only one core, so the core number (`corenum`) is given a value of zero. The first driver functions in a program open the Device to obtain the handle(s) used by subsequent functions. The handle is necessary to identify the Device to which each function is to be applied. Section 2.2 contains a complete discussion on handles and opening the Device.

As discussed in Section 2.3, opening the Device is normally followed by `BTICard_CardReset`. Most programs end with `BTICard_CardStop`, which stops the Device from operating, and `BTICard_CardClose`, which releases the computer's associated hardware resources. If `BTICard_CardStop` is not called, the Device continues to transmit and receive 1553 bus traffic, even though the program may have terminated. This can be useful in some test situations. Always call `BTICard_CardClose` before exiting the program.

2.2 Handles, Cards, and Cores

As a means of communicating with the hardware, the software must be able to specify the hardware such that the hardware is differentiated from other similar/dissimilar hardware. In addition, the software will need to specify the correct section in the case of the hardware having multiple sections. Ballard hardware Devices are implemented with one or more sections called cores. Typically, each core has its own protocol processing circuitry. The BTIDriver functions specify the Device with a card handle and the core with a core handle.

Nearly all functions require a handle parameter. The handle is always the last parameter in any function in which it is required. `BTICard_CardOpen`, which is always the first function called, returns a card handle (`hCard`). The card handle uniquely identifies a Device when more than one is used in a single computer. The card handle is different from the card number assigned by the host computer's operating system (OS).

A multi-core Device can have cores that work with the same or different protocols. In a way, the software sees each core as a separate card, so a handle (`hCore`) is needed to differentiate the cores. When passed a card handle (`hCard`) and the core number (`corenum`), `BTICard_CoreOpen` returns a core handle (`hCore`), which uniquely identifies the core/card combination. Therefore, `BTICard_CardOpen` must be called first to provide the card handle for `BTICard_CoreOpen`.

The following are handle-related guidelines to use when developing programs for BTIDriver-compliant Devices:

1. Find the Devices and the number and configuration of their cores by running the test program provided on the distribution disk with the Device (e.g., on Windows run `BTITST32.EXE`). When there is only one Device in the system, its `cardnum` is zero (0). When a Device has only one core, its `corenum` is zero (0). At any time, you can use this test program or the Windows Device Manager (for Windows operating systems) to reassign the card number.
2. At the beginning of the program, get the card handle (`hCard`) for each Device by calling `BTICard_CardOpen`.
3. Get the core handle (`hCore`) for each core on each Device by calling `BTICard_CoreOpen`.
4. Use `hCore` as the handle in all subsequent functions except `BTICard_CardClose.`
5. At the end of your program, call `BTICard_CardClose` once for each Device using its `hCard` handle to release the card and all of its core resources.
6. It is advisable to use the above procedure (using both `hCard` and `hCore`) even when the Device has only one core. This makes the code more easily adapted for use with multicore Devices.

7. Use of `hCard` in place of `hCore` addresses the first core (i.e., Core A with `corenum = 0`). This allows software written for legacy BTIDriver-compliant Devices to operate on the first core of multi-core Devices.
8. Legacy BTIDriver-compliant Devices that had not referred to cores in the documentation have only one core and should use core number (`corenum`) zero (0) in `BTICard_CoreOpen`.

2.3 Configuration Overview

A Device with MIL-STD-1553 capability can be configured to simulate a Bus Controller (BC), Remote Terminal (RT), and/or Bus Monitor. Depending on the Device model, each 1553 core may have one or more 1553 channels, and each channel may have different capabilities. On some models, a channel can only operate as a single terminal, while on others it can simultaneously simulate multiple terminals and inject errors (see Section 1.3).

A set of internal data structures constitutes a terminal configuration. BTIDriver functions set up and fill in these data structures. The `BTI1553_??Config` functions initialize terminal configurations. On Devices with multi-terminal simulation, `BTICard_CardReset` should precede `BTI1553_??Config` to clear the existing terminal configurations. Because the configuration stays resident on the Device, even after closing and exiting a program, most applications should call `BTICard_CardReset` to be assured that residual configurations are cleared.

As described in the following three sections, the primary goal of configuring the Device is creating message structures in which message parameters and data are stored. Both BC and RT configurations require message structures. Message structures are not shared between terminal configurations. Message structures may contain:

- Data words
- Command words
- Status words
- Time-tags
- Error flags
- Control flags

The content of a message structure is accessed and manipulated using various functions.

2.3.1 Bus Controller Configuration

As shown in Figure 2.2, a BC terminal configuration consists of two main components: a set of message structures and the schedule.

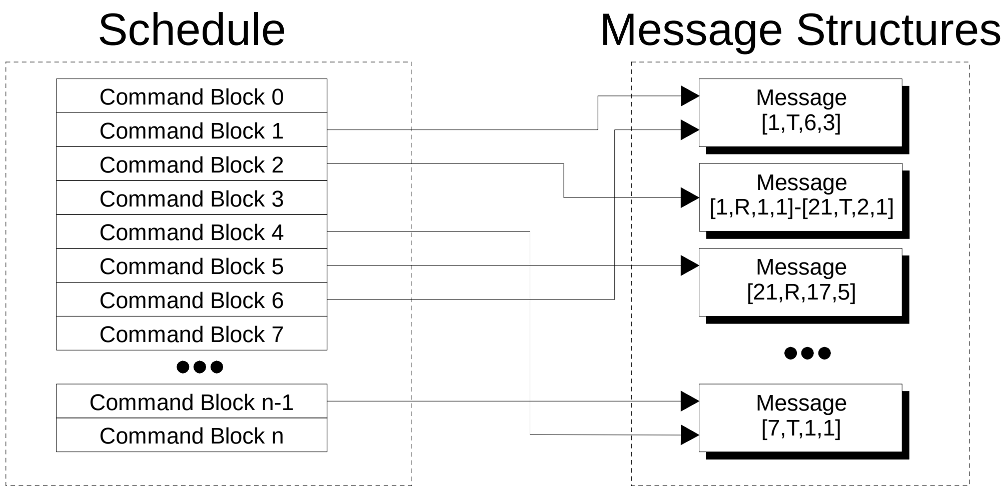


Figure 2.2—Bus Controller Terminal Configuration

Message structures for a BC configuration must be created with the `BTI1553_BCCreateMsg` function before they can be scheduled. These message structures contain space for command, status, and data words, as well as time-tags and other information.

The *schedule* is a program composed of an array of Command Blocks executed by a processor on the Device. Each Command Block contains an opcode. The opcodes control timing and transmission of messages. After the BC configuration is initialized with `BTI1553_BCConfig`, a schedule can be constructed with the scheduling functions (all of which are prefixed by `BTI1553_BCSched??`). Each of the scheduling functions appends one or more Command Blocks to the current end of the schedule.

Most schedules consist of only *Frame* and *Message* Command Blocks as illustrated in Section 3.2. Much more complex schedules can be created using the other available Scheduling functions shown in Table 2.1. Information on Scheduling functions not documented in Appendix A are available on the software distribution disk and from Customer Support.

Opcode Name	Function	Description
FRAME	BTI1553_BCSchedFrame	Creates a frame with a specified time
LOG	BTI1553_BCSchedLog	Generates Event Log List entry (see Section 4.5.1)
MESSAGE	BTI1553_BCSchedMsg	Inserts the 1553 message into a frame
RETRY	BTI1553_BCSchedRetry	Retries transmission of the last message
GAP	BTI1553_BCSchedGap	Inserts a timed gap
ENTRY	BTI1553_BCSchedEntry	Indicates the starting point for the schedule
HALT	BTI1553_BCSchedHalt	Halts the schedule until restarted with BTI1553_ChStart
PAUSE	BTI1553_BCSchedPause	Pauses the schedule until BTI1553_BCResume is called
RESTART	BTI1553_BCSchedRestart	Restarts the schedule at the beginning
USERCODE	BTI1553_BCSchedUser	Calls custom processor code
BRANCH	BTI1553_BCSchedBranch	Jumps to specified Command Block and resumes execution.
CALL	BTI1553_BCSchedCall	Jumps to specified Command Block, saving the return address on stack.
RETURN	BTI1553_BCSchedReturn	Returns to address following last call.
NOP	BTI1553_BCSchedNop	No operation.
PULSE	BTI1553_BCSchedPulse	Pulse an external discrete signal

Table 2.1—Bus Controller Scheduling Functions

Messages can also be transmitted singly using the `BTI1553_BCTransmitMsg` function. Messages transmitted in this way do not need to be created with `BTI1553_BCCreateMsg`.

2.3.2 Remote Terminal Configuration

As shown in Figure 2.3, a Remote Terminal configuration consists of a collection of message structures. Each message structure in an RT configuration is associated with a particular T/R (transmit/receive) bit and subaddress/mode-code combination. The `BTI1553_RTConfig` function can automatically create message structures for every T/R bit and subaddress/mode-code combination. Alternatively, RT message structures may be created individually with `BTI1553_RTCreateMsg`. The command words for which no message structures have been created are considered illegalized.

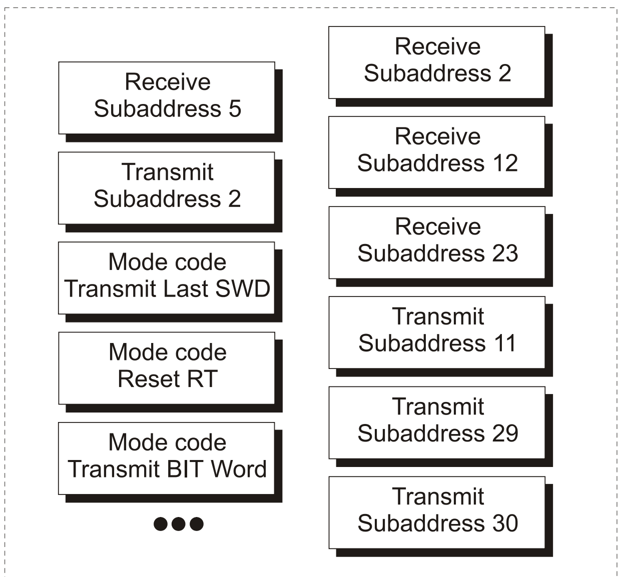


Figure 2.3—Message Structure for Remote Terminal Configuration

2.3.3 Bus Monitor Configuration

A Bus Monitor is a terminal that listens to, filters, and saves 1553 messages along with other useful information (e.g., a time-tag). `BTI1553_MonConfig` creates the Bus Monitor terminal and `BTICard_SeqConfig` sets up the Sequential Monitor and allocates Device memory for the data. Both of these functions must be used to configure the Device for 1553 monitoring. For those Devices that support it, the `BTI1553_MonFilter??` functions enable terminal address and subaddress filtering. See Section 3.5 for a further discussion on monitoring.

2.4 Runtime Overview

The Device does not interact in any way with the MIL-STD-1553 databus until `BTICard_CardStart` is called. At that time, all terminal configurations become active and the Device responds to the databus independently of the host computer until `BTICard_CardStop` halts it. A simulated BC processes its schedule (if present); simulated RTs respond automatically to commands from the BC.

As seen in Figure 2.4, the Device transmits data from and stores received data in its Random Access Memory (RAM). Various BTIDriver functions transfer message data between message structures in this RAM and the software application. None of these functions directly affects the MIL-STD-1553 databus. For example, updating data to be transmitted with the `BTI1553_MsgDataWr` function updates a specific message structure in RAM. The BC schedule or bus activity, not the `BTI1553_MsgDataWr` function, determines when this data is actually transmitted on the databus.

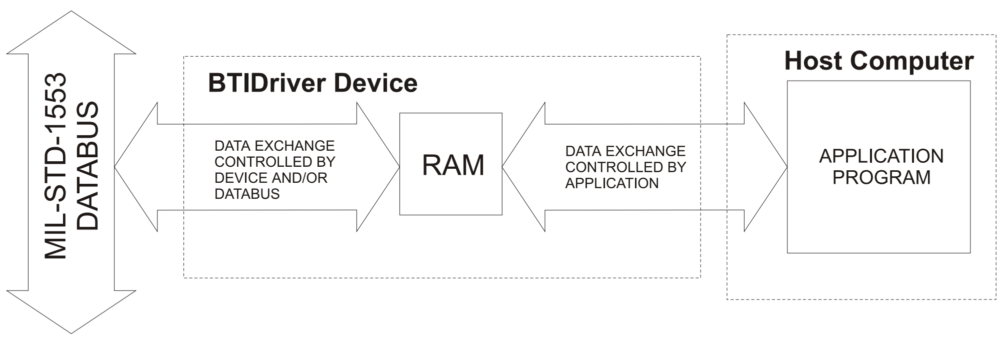


Figure 2.4—Control of Data Transfer while the Device is Active

An application program can coordinate its accesses to the Device with the operation of the Device by either polling status information on the Device or responding to interrupts from the Device.

This may be useful for timing data exchanges between the application program and the Device or controlling error injection in messages. Interrupt generation is Device-dependent. See Section 4.5 for more information on polling and interrupts.

This page intentionally blank.

3. SINGLE-TERMINAL SIMULATION

Using sample programs, this chapter illustrates single-terminal simulation that applies to all Ballard 1553 BTIDriver-compliant Devices. This chapter demonstrates how to

- Configure the channel as a BC, an RT, or a Bus Monitor
- Initialize and access data in each mode

After finishing this chapter, you will be familiar with the programming essentials of a single-terminal application, and you will be able to recognize the important functions. Chapter 4 discusses the use of advanced features such as list buffers, the Sequential Record, and special software events. Chapter 5 covers multi-terminal simulation and error injection.

Examples are presented as sequences of C function calls, not complete programs. To simplify and focus the presentation on BTIDriver, the examples omit standard program elements (such as data declarations) except where they are crucial to understanding the example. Complete source code and executable versions of all sample code are provided on the MIL-STD-1553 BTIDriver software distribution disk. These examples can serve as starting points for custom applications. Some of these samples may be useful as they are.

3.1 Simulating the BC: Unscheduled Messages

The simplest BC application transmits a single 1553 message, waits for its completion, and then processes and/or displays the status word and/or data response. Figure 3.1 shows all the function calls necessary to accomplish this.

`BTICard_CardOpen` and `BTICard_CoreOpen` are always the first functions called since they obtain the handles used by subsequent functions. `BTICard_CardReset` clears the core memory, removing any existing configuration data. `BTI1553_BCConfig` configures the channel to be a Bus Controller. Even though messages are not being scheduled as discussed in the next example, a dummy `BTI1553_BCSchedFrame` is still required when using `BTI1553_BCTransmitMsg` in default mode.

`BTI1553_BCTransmitMsg` commands the configured BC to transmit the message defined in the `XMITFIELDS1553` structure. The `XMITFIELDS1553` structure (declared in the header file) contains many fields that pass message parameters to `BTI1553_BCTransmitMsg`, which also passes information back through this structure to your program. The code initializes only the fields relevant to this example before passing the structure to `BTI1553_BCTransmitMsg`. Here the message type, command word, and three data words are initialized. The command word could have been defined as `0843h`, but instead the utility function `BTI1553_ValPackCWD` is used to create the command word from the separate fields (terminal address one, receive, subaddress two, and word count three).

`BTI1553_BCTransmitMsg` is a blocking function and returns after the message was transmitted. If the transmission was successful, it returns the status word in the `Msg.swd1` field. If the message transmitted was an RT-to-BC or RT-to-RT transfer, it would also return data values in the `Msg.data[]` array. The program ends with `BTICard_CardStop` and `BTICard_CardClose`.

```
HCARD hCard;
HCORE hCore;
XMITFIELDS1553 Msg;
INT cardnum = 0;      //Assumes only one Device has been installed
INT corenum = 0;             //Assumes only one core on the Device

BTICard_CardOpen(&hCard,cardnum);                   //Open the Device
BTICard_CoreOpen(&hCore,corenum,hCard);              //Open each core
BTICard_CardReset(hCore);                           //Reset each core

BTI1553_BCConfig(BCCFG1553_DEFAULT,CH0,hCore);

BTI1553_BCSchedFrame(1000,channum,hCore); //Schedule an empty frame

BTICard_CardStart(hCore);                            //Start the core

         //Initialize transfer mode and 1553 command and data words
Msg.ctrlflags = MSGCRT1553_BCRT;
Msg.cwd1      = BTI1553_ValPackCWD(1,0,2,3);      //Receive command
Msg.data[0]   = some_value0;
Msg.data[1]   = some_value1;
Msg.data[2]   = some_value2;

BTI1553_BCTransmitMsg(&Msg,CH0,hCore);      //Transmit the message

if (Msg.errflags & (MSGERR1553_NORESP | MSGERR1553_ANYERR))
{
     //The RT did not respond or had an error in its response
}
else
{
     //The message was completed successfully
     //Check Msg.swd1 for the RT's status word response
}

BTICard_CardStop(hCore);                             //Stop each core
BTICard_CardClose(hCard)                           //Close the Device
```

Figure 3.1—Example 1: Unscheduled Message Transmission

`BTI1553_BCTransmitMsg` requires a dummy `BTI1553_BCSchedFrame` in default operation only because the message is transmitted last in the frame. On some Devices, `BTI1553_BCTransmitMsg` is capable of transmitting high-priority messages. On these Devices, the high-priority message is transmitted at the next opportunity without need for scheduled frames.

To transmit messages at given intervals, `BTI1553_BCTransmitMsg` could be placed in a timed loop and passed different `XMITFIELDS1553` structures. However, this is unnecessary work for the host computer since

BTIDriver-compliant Devices can time message transmissions internally, as demonstrated in the next section.

3.2 Simulating the BC: Scheduled Messages

Messages on a MIL-STD-1553 databus are typically transmitted according to periodic schedules. These schedules are usually composed of timed groups of messages called frames. Frames are sequenced so that each message is transmitted at the desired frequency.

Suppose, for example, that we would like to transmit three messages at the following rates:

- msg0 every 10 milliseconds
- msg1 every 20 milliseconds
- msg2 every 40 milliseconds

Figure 3.2 shows a schedule to achieve these rates. Notice in Figure 3.2 that by repeating a message in different frames and possibly repeating frames in the cycle, we are able to meet the timing requirements for each message.

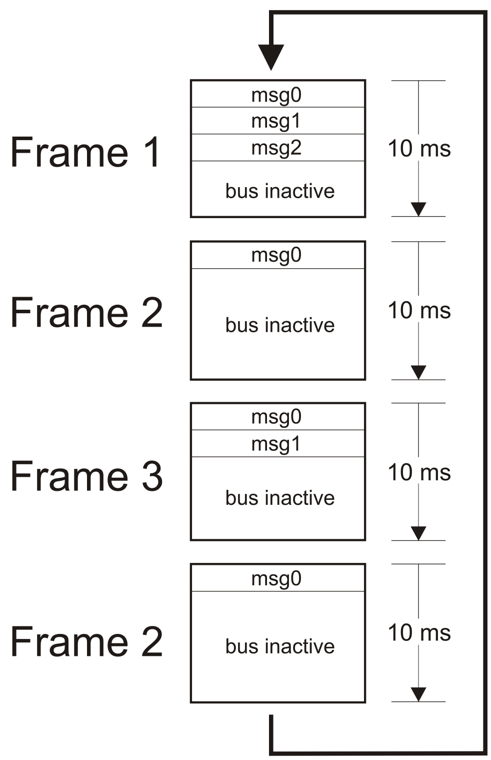


Figure 3.2—Typical BC Message Schedule

The core for each channel can execute such a schedule autonomously, maintaining all associated timing onboard. The code in Figure 3.3 demonstrates how to configure the channel with the schedule shown in Figure 3.2 above.

After obtaining the Device handles with BTICard_CardOpen and BTICard_CoreOpen and resetting the core with BTICard_CardReset, BTI1553_BCConfig configures the channel as a Bus Controller. If BTICard_CardReset is omitted, any previous terminal configuration remains intact and may corrupt the current configuration when the core is started. Flags can replace BCCFG1553_DEFAULT to enable or disable options such as the Event Log List or mode codes. See the Function Reference (Appendix A).

BTI1553_BCCreateMsg creates a message structure and initializes it with the command words and, for BC-to-RT messages, the data to be transmitted. This function returns the address of the message structure. Other functions such as BTI1553_MsgDataWr require this address to access the message. The BCMsgs array holds the returned message addresses.

The first BTI1553_BCCreateMsg function creates a BC-to-RT message. The 0x0843 parameter is the 1553 command word (in hex) which commands terminal address one (1) to receive three (3) data words into its receive subaddress two (2). Notice that the third BTI1553_BCCreateMsg function creates an RT-to-RT message. The MSGCRT1553_DEFAULT parameter is replaced by MSGCRT1553_RTRT, and a second command word is provided.

```
HCARD hCard;
HCORE hCore;
MSGADDR BCMsgs[3];
INT cardnum = 0;       //Assumes only one Device has been installed
INT corenum = 0;             // Assumes only one core on the Device

BTICard_CardOpen(&hCard,cardnum);                  //Open the Device
BTICard_CoreOpen(&hCore,corenum,hCard);             //Open each core
BTICard_CardReset(hCore);                          //Reset each core


BTI1553_BCConfig(BCCFG1553_DEFAULT,CH0,hCore);
                                                //Create the messages
BCMsgs[0] =
BTI1553_BCCreateMsg(MSGCRT1553_DEFAULT,0x843,0,&InitalData,hCore);
BCMsgs[1] =
BTI1553_BCCreateMsg(MSGCRT1553_DEFAULT,CMD1,0,NULL,hCore);
BCMsgs[2] =
BTI1553_BCCreateMsg(MSGCRT1553_RTRT,RCV_CMD2,XMT_CMD2,NULL,hCore);

BTI1553_BCSchedFrame(10000,CH0,hCore);          //Create a Schedule
BTI1553_BCSchedMsg(BCMsgs[0],CH0,hCore);
BTI1553_BCSchedMsg(BCMsgs[1],CH0,hCore);
BTI1553_BCSchedMsg(BCMsgs[2],CH0,hCore);

BTI1553_BCSchedFrame(10000,CH0,hCore);
BTI1553_BCSchedMsg(BCMsgs[0],CH0,hCore);

BTI1553_BCSchedFrame(10000,CH0,hCore);
BTI1553_BCSchedMsg(BCMsgs[0],CH0,hCore);
BTI1553_BCSchedMsg(BCMsgs[1],CH0,hCore);

BTI1553_BCSchedFrame(10000,CH0,hCore);
BTI1553_BCSchedMsg(BCMsgs[0],CH0,hCore);

BTICard_CardStart(hCore);

while (!done)     //Process the data as required by the application
{
   error_field =
   BTI1553_MsgFieldRd(FIELD1553_ERROR,BCMsgs[0],hCore);

   if (error_field & MSGERR1553_HIT)
   {
      //The current data values have been transmitted, so update them
      BTI1553_MsgDataWr(&NewData,3,BCMsgs[0],hCore);

                             //Clear all errors, including the hit bit
      BTI1553_MsgFieldWr(0,FIELD1553_ERROR,BCMsgs[0],hCore);
   }
}

BTICard_CardStop(hCore);
BTICard_CardClose(hCard);
```

Figure 3.3—Example 2: Scheduled Message Transmission

After the messages are created, they can be scheduled. Notice how the blocks of `BTI1553_BCSched`?? functions exactly correspond with the frames illustrated in Figure 3.2. `BTI1553_BCSchedFrame` marks the start of a frame and includes the frame time (in µs) as one of its parameters. Frames can be up to 1.3 seconds in length. `BTI1553_BCSchedMsg` inserts the specified message into the schedule. The schedule contains references to the messages structures, not copies of structures. This means message data exists in only one place, so updating message data updates all instances of it in the schedule.

The scheduling functions (BTI1553_BCSched??) actually create a program in core memory. After BTICard_CardStart, the core operates the channel as a BC by executing this program. The program is cyclic by default. That is, the sequence of frames described in the code is transmitted repeatedly according to the schedule.

The program can read or update message data at any time with the BTI1553_MsgDataRd or BTI1553_MsgDataWr functions. The code in this example only deals with the BC-to-RT message. The BTI1553_MsgField?? functions access message parameters other than data. This example gets the value of the error field, which it tests to see if the message has processed (the hit bit). If so, it writes a new data value and clears the error field.

3.3 Simulating an RT

The code in Figure 3.4 complements the preceding example. It configures the Device as a Remote Terminal with terminal address one to receive the BC-to-RT message transmitted by the BC in the preceding example (i.e., command word 0843h). It polls receive subaddress two where it expects to be receiving three data words.

```
HCARD hCard;
HCORE hCore;

BTICard_CardOpen(&hCard,cardnum);                 //Open the Device
BTICard_CoreOpen(&hCore,corenum,hCard);            //Open each core
BTICard_CardReset(hCore);                         //Reset each core

BTI1553_RTConfig(RTCFG1553_DEFAULT,1,CH0,hCore);

RTMsg = BTI1553_RTGetMsg(SUBADDRESS,1,RCV,2,CH0,hCore);

BTICard_CardStart(hCore);

while (!done)
{
   BTI1553_MsgDataRd(&Data,3,RTMsg,hCore);//Read the message from the BC
}

BTICard_CardStop(hCore);
BTICard_CardClose(hCard);
```

Figure 3.4—Example 3: RT Simulation

The most significant configuration parameter, its terminal address, is assigned in the BTI1553_RTConfig function. The RTCFG1553_DEFAULT parameter causes message structures to be set up for all subaddresses and mode codes. Since message access requires a message address, you must follow BTI1553_RTConfig with a call to BTI1553_RTGetMsg for each message of interest. In this example, BTI1553_RTGetMsg gets the message address associated with the 0843h command word from Figure 3.3 (i.e., terminal address one (1) to receive data for subaddress two (2)).

Since this example only anticipates receiving data from the BC, the code does not initialize any of the Device's RT transmit message structures. You could use BTI1553_MsgDataWr after BTI1553_RTGetMsg to initialize data for transmission. Within the while loop BTI1553_MsgDataRd reads received message data using the address returned by BTI1553_RTGetMsg.

Since BTI1553_RTConfig was called with RTCFG1553_DEFAULT, all subaddresses and mode codes are legal. After BTICard_CardStart activates the core, it responds to any valid command to terminal address one (1), although this example only anticipates command word 0843h.

Alternatively, you could call BTI1553_RTConfig with RTCFG1553_NOBUILD in place of RTCFG1553_DEFAULT and then call BTI1553_RTCreateMsg for each message of interest. In this case, only subaddresses and mode codes specified in a BTI1553_RTCreateMsg function are mapped into memory; others are illegalized. This conserves memory for multi-terminal simulation.

3.4 RT Address 31 and the Broadcast Option

An RT can be configured to ignore or accept broadcast message data. If it is configured to accept broadcast data, the RT always follows Notice 2 to MIL-STD-1553B regarding the storage of this data. That is, broadcast data are always stored separately from non-broadcast data.

A common terminal, in which all broadcast data is stored, is automatically created at address 31 if any RT address is configured to accept broadcast data (using RTCFG1553_BCAST in BTI1553_RTConfig). Access to this data is obtained by calling BTI1553_RTGetMsg with the terminal address 31.

You may also enable RT address 31 as a genuine RT by calling BTI1553_RTConfig (..., 31,...). This completely disables the use of broadcast messages.

3.5 Monitoring the Bus

In a sense, any receiving terminal is a monitor that holds the most recent value of the received data. However, Ballard MIL-STD-1553 Devices have two types of specialized monitors: a Shadow Monitor and a Sequential Monitor.

3.5.1 Shadow Monitor

The Shadow Monitor is just a variation of a Remote Terminal. When an RT is configured using the `RTCFG1553_MONITOR` control flag in the `BTI1553_RTConfig` function, it becomes a Shadow Monitor and does not transmit on the 1553 bus. Instead, it receives the data sent to and transmitted by the real RT with the specified Remote Terminal address. The most recent value of the data may be read at any time using `BTI1553_MsgDataRd`.

3.5.2 Sequential Monitor

A Sequential Monitor records a time-tagged history of user-selected bus activity in what is called a Sequential Record. A Sequential Record is useful for analyzing and reconstructing all or selected bus activity. On MIL-STD-1553 Devices, a Bus Monitor terminal is the source of data for the Sequential Monitor. All monitor constants and functions, indicated by the `MON` or `Mon` prefix, apply only to the 1553 Bus Monitor and Sequential Monitor, and not to the Shadow Monitor. Additional information on the Sequential Record may be found in Section 4.2.

A Sequential Monitor saves messages sequentially in its memory in the order they are received from the bus. Each message structure contains:

- Command word(s)
- Data word(s)
- Status word(s)
- Time-tag
- Error flags

The code in Figure 3.5 demonstrates a Sequential Monitor that captures 1553 bus traffic and prints a simple report including time-tag, command word, and if appropriate, a *No Response* message.

```
HCARD hCard;
HCORE hCore;
USHORT seqbuf[2048];
ULONG seqcount;
LPSEQRECORD1553 pRec1553;
SEQFINDINFO sfinfo;

BTICard_CardOpen(&hCard,cardnum);                    //Open the Device
BTICard_CoreOpen(&hCore,corenum,hCard);              //Open each core
BTICard_CardReset(hCore);                            //Reset each core

//Create Bus Monitor terminal
BTI1553_MonConfig(MONCFG1553_DEFAULT,CH0,hCore);
BTICard_SeqConfig(SEQCFG_DEFAULT,hCore);//Configure Sequential Monitor

BTICard_CardStart(hCore);

while(!done)
{
   //Read the Sequential Record
   seqcount = BTICard_SeqBlkRd(seqbuf,bufcount,blockcount,hCore);
   //Initialize Find functions
   BTICard_SeqFindInit(seqbuf,seqcount,&sfinfo);
   //Find/process 1553 records
   while(!BTICard_SeqFindNext1553(&pRec1553,&sfinfo))
   {
      //Write to disk, display data, etc. as desired. For example:
      printf("\n Time:%lu",pRec1553->timestamp);

      if (pRec1553->activity & MSGACT1553_RCVCWD1)
         printf(" Cwd1:%04X", pRec1553->cwd1);
      if (pRec1553->error & MSGERR1553_NORESP)
         printf(" No response!");
   }
}

BTICard_CardStop(hCore);
BTICard_CardClose(hCard);
```

Figure 3.5—Example 4: Bus Monitor Simulation

The first few lines of code declare variables used by the Sequential Monitor functions. After opening and resetting the core, the BTI1553_MonConfig function sets up the Bus Monitor, and then BTICard_SeqConfig configures the Sequential Monitor and allocates onboard memory for the Sequential Record. After the core is started, it monitors the bus and saves the information in the onboard Sequential Record.

The while loop in the code polls and processes the Sequential Record. BTICard_SeqBlkRd copies a block of available records from the onboard Sequential Record to the user-supplied buffer (seqbuf) and returns the number of words copied (seqcount). To process records in seqbuf, it is necessary to find the start and type of each record, which may be done with different BTICard_SeqFindNext?? functions. Here, BTICard_SeqFindInit initializes a structure (sfinfo). Then, repeated calls to BTICard_SeqFindNext1553 find and point pRec1553 to the next occurrence of a 1553-type record. The desired values are then printed from the record. As usual, the program ends with BTICard_CardStop and BTICard_CardClose.

MIL-STD-1553 channels (level B4 and above) can filter data so that only the 1553 messages of interest are entered into the Sequential Record. (See Section 1.3 for a table showing which Ballard 1553 channels support Bus Monitor filtering.) Filtering by terminal address and subaddress is established through the `BTI1553_MonFilterTA` and `BTI1553_MonFilterSA` functions.

3.6 Summary

The preceding examples illustrate the most important BTIDriver functions. Programs for your Device can be modeled on the code in the preceding examples, for which complete source code is provided on the distribution disk. Detailed function descriptions are found in the Function Reference (Appendix A). Check the `README` files on the software distribution disk for updated function information.

4. SPECIAL FEATURES

This chapter covers special features supported by 1553 BTIDriver-compliant Devices. These features allow the user to create custom message structures, record selected bus activity for later analysis, and receive notification of special events on the bus through polling or (in some cases) hardware interrupts.

4.1 List Buffering

Normally, there is a single buffer per message. Alternatively, list buffering allocates multiple memory locations for each message and increases the versatility of the Device. These multiple memory locations operate in a variety of ways, depending on the List Buffer mode.

List buffers are created using functions that specify the mode and size of the list. These functions are similar to the functions used for creating single messages and automatically keep track of the message locations in the list.

4.1.1 List Buffer Modes

The three different modes of list buffering outlined below control the behavior and configuration of the memory blocks. The Circular mode can be used only for transmission, while the other two list buffering modes can be used for either transmission or reception. In all modes, the internal indices are automatically maintained by the functions that read from and write to the List Buffer. As with single buffers, it is a good idea to initialize the data values in List Buffers before transmission.

Circular List Buffer

The Circular List Buffer is used for repeatedly transmitting patterns of data. The circular list only operates as a transmit buffer. When this mode is specified, each time a message is transmitted onto the databus, the core automatically advances the index to the next list entry. When the end of the list is reached, the index returns to the beginning of the list and continues. The Circular List Buffer could be used to transmit a pattern of data repeatedly, like a ramp or sine wave.

FIFO List Buffer

The *first-in first-out* (FIFO) List Buffer is used for transmitting or receiving a sequence of data, as in a file transfer. This type of List Buffer operates like a queue on a first-in first-out basis. Operation of the FIFO is the same for both BC and RT configurations, but differs between transmission and reception.

When the transmitting FIFO buffer is full and the host tries to write more data to it, the list write function returns an error. If the transmitting FIFO is empty, the last value is used for transmissions until the host writes a new value into the FIFO.

When receiving, if the FIFO becomes full, newly received messages are written to the FIFO and the oldest messages are lost. If the receiving FIFO becomes empty and the host tries to read the FIFO, the list read function returns a zero to indicate that there was no data to be read. This process may be used for polling the status of the FIFO buffer.

Ping-Pong List Buffer

The Ping-Pong List Buffer guarantees data integrity by preventing a problem that can occur when only a single buffer is used. If the host computer and the core processor simultaneously access the same message in a single buffer memory, the data being read by the host or transmitted on the bus could contain part of the old message and part of the new message. The Ping-Pong List Buffer uses multiple memory locations, so the host always reads the most recent, complete copy of a received message. In addition, the most recent, complete copy of a message is automatically used for transmissions.

4.1.2 List Buffer Operation

When a List Buffer is needed in place of a single buffer, the `BTI1553_BCCreateList` or `BTI1553_RTCreateList` functions are used in place of the `BTI1553_BCCreateMsg` or `BTI1553_RTCreateMsg` functions. The `BTI1553_??CreateList` functions have two additional parameters, which specify the mode of the List Buffer and the number of records. For a Ping-Pong List Buffer, the number of records parameter is ignored. List buffer messages are transferred to and from the host computer using `BTI1553_ListDataWr` and `BTI1553_ListDataRd` functions, much like the single-buffer accesses using `BTI1553_MsgDataWr` and `BTI1553_MsgDataRd`.

Examples of the List Buffer modes are found on the software distribution disk, as explained in the `README.TXT` file.

4.1.3 Unscheduled List Transmission

Some Devices are capable of transmitting a list of unscheduled messages without the need of calling `BTI1553_BCTransmitMsg` for each individual message. `BTI1553_BCTransmitList` commands the configured BC to transmit a list of messages defined by an array of `XMITFIELDS1553` structures. This improves performance by reducing host intervention, while still allowing the flexibility of unscheduled messages. The entire list will either be transmitted after all scheduled messages in a frame (`MSGCRT1553_EOF`) or transmitted high-priority (`MSGCRT1553_HIPRI`). Transmission mode for the entire list is configured with the first `XMITFIELDS1553` in the list.

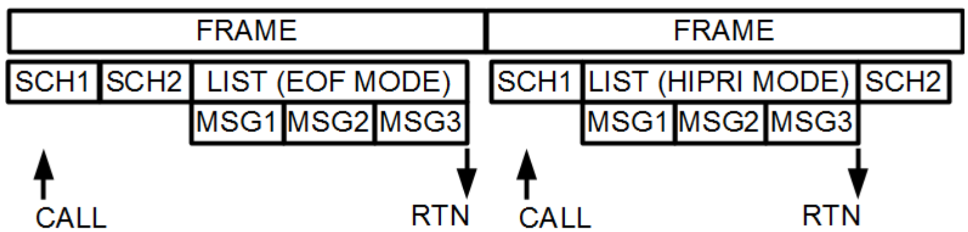


Figure 4.1—Various Asynchronous Lists in a Schedule

`BTI1553_BCTransmitList` is a blocking function and returns after the entire list was transmitted. Each `XMITFIELDS1553` structure will be processed in the same method as individual `BTI1553_BCTransmitMsg` calls, therefore `BTI1553_BCTransmitList` also requires a dummy `BTI1553_BCSchedFrame` in default (`MSGCRT1553_EOF`) mode.

`BTI1553_BCTransmitList` is capable of transmitting a high-priority list. High-priority lists are transmitted at the next opportunity without need for scheduled frames, so `BTI1553_BCSchedFrame` is not needed with empty schedules. Use caution if messages are scheduled because high-priority lists will interrupt and affect the timing relationship of the schedule. The following example demonstrates unscheduled high-priority list transmission with an empty schedule.

```
HCARD hCard;
HCORE hCore;
XMITFIELDS1553 msgs[8];
INT j = 0;
INT cardnum = 0;       //Assumes only one Device has been installed
INT corenum = 0;              //Assumes only one core on the Device

BTICard_CardOpen(&hCard,cardnum);                //Open the Device
BTICard_CoreOpen(&hCore,corenum,hCard);           //Open each core
BTICard_CardReset(hCore);                        //Reset each core

BTI1553_BCConfig(BCCFG1553_DEFAULT,CH0,hCore);

//BCTransmitList will be HIPRI, so no need to schedule an empty frame

BTICard_CardStart(hCore);                         //Start the core

//Initialize first message in the list and set HIPRI mode
msgs[0].ctrlflags = (MSGCRT1553_BCRT | MSGCRT1553_HIPRI);

msgs[0].cwd1      = BTI1553_ValPackCWD(1,0,2,3);  //Receive command
msgs[0].data[0]   = some_value0;
msgs[0].data[1]   = some_value1;
msgs[0].data[2]   = some_value2;

//Initialize second message in the list and set HIPRI mode
msgs[1].ctrlflags = (MSGCRT1553_BCRT | MSGCRT1553_HIPRI);

msgs[1].cwd1      = BTI1553_ValPackCWD(1,1,2,3); //Transmit command

//Initialize msgs[2] through msgs[7] not shown

BTI1553_BCTransmitList(msgs,8,CH0,hCore);      //Transmit the list

for (j=0;j<8;j++)
{
   if (msgs[j].errflags & (MSGERR1553_NORESP | MSGERR1553_ANYERR))
   {
        //The RT did not respond or had an error in its response
   }
   else
   {
        //The message was completed successfully
        //Check msgs[j].swd1 for the RT’s status word response
   }
}

BTICard_CardStop(hCore);                          //Stop each core
BTICard_CardClose(hCard)                        //Close the Device
```

Figure 4.2—Unscheduled High-Priority List Transmission

4.2 Sequential Record

The Sequential Monitor records a time-tagged history of selected MIL-STD-1553 transmitted and received messages. This history is stored in a buffer called the Sequential Record. There is one Sequential Record per core, so even if the core supports multiple channels of the same or different protocols, all messages associated with the core are stored in the one Sequential Record (see Appendix C for more information). Individual channels and/or individual messages within a channel may be selectively recorded. The filtering of desired messages is illustrated in the code in Figure 5.1 and described in Section 5.1.

On 1553 channels, the Sequential Monitor receives its information from a Bus Monitor terminal. To configure a terminal as a Bus Monitor, call `BTI1553_MonConfig` and then call `BTICard_SeqConfig` that configures the Sequential Monitor and allocates internal memory for the Sequential Record.

By default, recording halts when the onboard Sequential Record is full. This happens to prevent unread data from being overwritten when the host gets behind in reading data from Sequential Record. However, if the Sequential Record is configured with the `SEQCFG_CONTINUOUS` flag in `BTICard_SeqConfig`, recording is not halted. In this mode, it automatically wraps around and continues recording, overwriting old messages. The core can log an entry in the Event Log List when the Sequential Record either halts or wraps around (depending on the selected option). Alternatively, the core can log an entry on every n^{th} message recorded into the Sequential Record. (See Section 4.5.1 for more information on enabling entries into the Event Log List.)

The Sequential Monitor records data to the Sequential Record only while the core is running (`BTICard_CardStart`). However, while the core is running, the Sequential Monitor can be stopped and restarted without affecting other core functions. `BTICard_SeqStart` is used to start the Sequential Monitor; it also stops and initializes it if necessary before restarting it. `BTICard_SeqStop` stops data from being added to the Sequential Record. If `BTICard_SeqResume` is called after `BTICard_SeqStop`, data recording continues and the original data is not lost.

Head and tail pointers are used to keep track of the location of the most recently entered data and the oldest data that needs to be read. When the core adds a message to the Sequential Record, it updates the head pointer; when the host reads the contents of the Sequential Record, the tail pointer is updated. The Sequential Record may be read using any one of a family of functions: `BTICard_SeqRd` reads a single record at a time, while `BTICard_SeqBlkRd` and `BTICard_SeqCommRd` read as many records as they can (except for BUSBox BB1xxx series Devices). Note that in the special case of BUSBox BB1xxx series Devices, all three functions read as many records as they can and produce similar results. As long as one of these functions is used to read the Sequential Record, the head and tail pointers are automatically maintained.

4.3 Time-Tagging

The time-tag is a 64-bit value derived from an internal clock. The resolution and range of the time-tag is dependent on the Device and its capabilities. For example, some Devices can be configured to use binary time values or BCD IRIG based time values (see `BTICard_TimerStatus` for more information). In addition, on some Devices the resolution of time-tag values may be adjusted with the `BTICard_TimerResolution` function. Please see the `BTICard_Timer??` and `BTICard_IRIG??` function synopses in Appendix B for more information on Device dependencies.

Time-tag location is generation-dependent (refer to Table A.3). For 3G and 4G Devices (when using the binary timer), the time-tag of a transmitted word represents when the word is loaded into the encoder, *not* when the word is actually transmitted. Similarly, the time-tag of a received word represents when the word was read from the decoder. Thus, the time-tag can deviate slightly from the actual time of the bus activity. For 5G and 6G Devices, the time-tag represents when the word is actually transmitted and received. Thus, these time-tags represent the actual time of the bus activity.

While both 5G and 6G time-tags represent bus activity, there are slight differences in the location of the time-tag capture (timestamp). 5G Devices will timestamp either at the end of the command word in BC-RT commands or at the end of the second command word in RT-RT commands. 6G Devices will always timestamp at the zero-crossing of the command word sync regardless of command. For this reason, 6G Devices time-tags are more consistent across all command types.

4.4 TSM: Theory of Operation

Systems with high reliability and fidelity requirements need system-level time synchronization to reduce the effort involved to compare event timing across interconnected devices. In a distributed system, each device generally contains a local oscillator to clock digital logic circuitry, including that of the local timing source. Crystal oscillators naturally ‘drift’ from their fundamental frequencies over time and are subject to aging, temperature, and radiation effects. Each crystal oscillator will drift from its fundamental frequency uniquely, leading to situations for example, where two oscillators in a system having the *same* frequency may be slightly off in different directions from the fundamental: one fast and one slow. In this event, one device will increment its timer too quickly, and the other will increment its timer too slowly, relative to the norm. Left unmanaged over several seconds this may lead to measurable difference in the time between the two devices of the system introducing uncertainty in event and error timestamping.

To counter this effect, system designers typically use a single source to periodically broadcast system time. Each device of the system will load the received time at a timing mark and then increment until the next time broadcast is received. This approach reduces the possible system time difference to a function of the time broadcast period.

A drawback to this approach, however, is that non-monotonically increasing timestamps can be generated by devices receiving broadcast time, especially in systems with high-resolution timestamping. If the local oscillator is *fast* compared to the system-time source, the device time will be ahead of the system time. When the broadcast time is received, the device time will *jump* back to the slower system time. If two timestamps are generated by the device, occurring on each side of the timing mark, the later timestamp can actually show a lower time value than the earlier timestamp. Inconsistent timestamps can increase debug and validation efforts, as additional energy is needed to understand event sequences.

To solve this problem, Ballard introduces the Timing Synchronization Manager (TSM) in the OmniBus II (OB2) family products. The TSM provides the ability to synchronize time between an external reference (IRIG PCM, IRIG AM, PPS) and the protocol Cores on the OB2 family product. Additionally, using one of the above references or a 10 MHz input, the TSM can adjust the frequency of the local clock oscillator up to 425 ppm to reduce clock drift between the Device and the external reference. Use of the drift control functionality can help increase synchronization between the OB2 family Core Timer and the external reference during the gap between timing samples. The TSM can adjust for not only the clock frequency differences (drift), but also can adjust the frequency of the local clock to account for the time differences. Instead of *jumping* to the new broadcast time, introducing discontinuities in timestamps, the TSM can adjust the local clock frequency such that the time delta is minimized at the next timing interval.

The TSM can be the source of system time by calling `BTICard_TSMSourceConfig` and passing the appropriate flags for IRIG PCM, IRIG AM, PPS or 10 MHz. The TSM can also be a sync for system time by calling `BTICard_TSMSyncConfig`, which takes arguments for both time and drift. The TSM can synchronize time and drift to some combinations of different sources.

The TSM also provides numerous configuration options. See `BTICard_TSM*` functions in Appendix B for details.

4.5 Special Events

In some software programs, it may be necessary to know when a special event has happened. Examples of special events are when a specific message is received, when the transmit schedule reaches a certain point, when an error occurs, or when the Sequential Record or a List Buffer needs service. An Event Log List is used to record these special events for each core. Notification that the special event has occurred can happen through polling or interrupts. If your core(s) supports multiple protocols, the Event Log List can contain events from more than just MIL-STD-1553 related activity. See the description of `BTICard_EventLogRd` in Appendix B for a table of event types for different protocols.

4.5.1 Event Log List

To use an Event Log List, it is necessary to create the Event Log List and to enable the specific events that are to be recorded. The `BTICard_EventLogConfig` function creates the Event Log List. To enable a specific event, use the corresponding enabling constant in the enabling function. Many BTIDriver functions can be configured to enable Event Log List entries. See the description of `BTICard_EventLogRd` in Appendix B for a table that lists the enabling function(s) for each event.

The Event Log List is a circular buffer, which records all events in order of occurrence. An entry is added to this list each time an enabled event occurs. An event is identified by reading and evaluating its entry from the Event Log List. Each Event Log List entry contains three fields. The description of `BTICard_EventLogRd` in Appendix B summarizes the meanings and values of these parameters.

4.5.2 Polling

When the program is running, the Event Log List may be polled using `BTICard_EventLogRd`. This function returns a zero if the Event Log List is empty. Otherwise, it may be evaluated to determine the source of the event. See the description in Appendix B for a table that describes the Event Log List fields. Each entry in the Event Log List may be read only once, since `BTICard_EventLogRd` automatically increments the list pointers each time it is called. It returns the oldest entry from this list and updates the tail pointer.

4.5.3 Interrupts

If your Device supports hardware interrupts, you can configure each core individually to issue a hardware interrupt each time an entry is made in the Event Log List, which virtually becomes an interrupt log list.

Using hardware interrupts requires an interrupt service routine and an understanding of the host computer's operating system. The `BTICard_IntInstall` function is used to enable the hardware interrupt and to associate the interrupt service routine with the interrupts from a core. To identify the source of the interrupt, the interrupt service routine calls and analyzes the response of the `BTICard_EventLogRd` function, just as it did when polling in the previous section. Before returning, the interrupt service routine must call `BTICard_IntClear` to clear the hardware interrupt. More discussion may be found under `BTICard_IntInstall` in Appendix B. Also, see the interrupt examples on the software distribution disk.

This page intentionally blank.

5. ADVANCED OPERATION

Ballard 1553 Devices, with level B4 or M and above capability, can simultaneously simulate a Bus Controller and multiple Remote Terminals. Those with level C or M and above functionality are capable of concurrent monitoring and protocol error injection, and those with level D or P have zero-crossing distortion and variable transmit amplitude. This chapter demonstrates how to program these advanced capabilities. To determine which channels on your Device support these advanced features, please see the documentation for your Device or the run the test program provided with your Device. Channels for Ballard MIL-STD-1553 Devices are available in the multiple levels (summarized in Section 1.3).

5.1 Multi-Terminal Simulation

To simulate more than one terminal simultaneously, simply combine the configuration code for each terminal. The example in Figure 5.1 illustrates this by combining terminal configurations similar to Examples 2, 3, and 4 from Chapter 3. This example assumes the Device is capable of concurrently monitoring and simulating multiple terminals.

```
HCARD hCard;
HCORE hCore;
MSGADDR BCMsg, RTMsg;

BTICard_CardOpen(&hCard,cardnum);             //Open the Device
BTICard_CoreOpen(&hCore,corenum,hCard);        //Open each core
BTICard_CardReset(hCore);                     //Reset each core

//Bus Controller configuration from Example 2 (Figure 3.3)
BTI1553_BCConfig(BCCFG1553_DEFAULT,CH0,hCore);

BCMsg =
BTI1553_BCCreateMsg(MSGCRT1553_DEFAULT,0x0843,0,&InitialData,hCore);

BTI1553_BCSchedFrame(10000,CH0,hCore);
BTI1553_BCSchedMsg(BCMsg,CH0,hCore);

//Remote Terminal configuration from Example 3 (Figure 3.4)
BTI1553_RTConfig(RTCFG1553_DEFAULT,1,CH0,hCore);

RTMsg = BTI1553_RTGetMsg(SUBADDRESS,1,RCV,2,CH0,hCore);

//Monitor configuration from Example 4 (Figure 3.5)
BTI1553_MonConfig(MONCFG1553_DEFAULT,CH0,hCore);
BTI1553_MonFilterTA(0x00000002,CH0,hCore);
BTICard_SeqConfig(SEQCFG_DEFAULT,hCore);

BTICard_CardStart(hCore);

while (!done)
{
   //Handle data as in Examples 2 through 4 in Chapter 3
}

BTICard_CardStop(hCore);
BTICard_CardClose(hCard);
```

Figure 5.1—Example 5: Multi-Terminal Simulation

BTICard_CardOpen, BTICard_CoreOpen, and BTICard_CardReset only need to be called once. Note that BTICard_CardReset is especially important in cores that support multiple terminals. If BTICard_CardReset is omitted, terminal configurations from a previous program may be intact. This could result in simulation of unintended terminals.

This example, like the single-terminal examples, follows the program model discussed in Section 2.1. In this example, the message configuration functions for a given terminal immediately follow the corresponding BTI1553_??Config function. This grouping is arbitrary. BTI1553_??Config functions could be grouped and all other configuration functions follow. The order in which the BTI1553_??Config functions are called is also arbitrary.

The BTI1553_BCSchedFrame and BTI1553_BCSchedMsg functions create a schedule containing only one frame. This frame contains only the 0843h command, so this one message is transmitted every 10 ms.

There is one additional function not found in the previous examples: BTI1553_MonFilterTA. It takes a 32-bit bitmask parameter that specifies which terminal addresses to monitor. Only messages to or from the specified terminals appear in the Sequential Record. The 00000002h parameter in the example specifies that only Remote Terminal address one (1) will be monitored. Not all Devices with monitor capability support Bus Monitor filtering (see Section 1.3).

As described in Section 2.4, the BTICard_CardStart function activates all configured terminals. Message data, parameters, and the Sequential Record can be accessed as in previous examples.

5.2 Error Injection

This section applies only to those channels that have the capability to generate specific errors in its transmissions. This capability is useful for testing the response of hardware or systems to bus faults. This section explains how to use BTIDriver to control the generation of errors. Three examples demonstrate how to specify the type, location, and frequency of an error.

5.2.1 Error Injection Overview

Error injection is the act of transmitting a signal that does not comply with the MIL-STD-1553 specification. Except as noted, any of the following types of errors may be injected on channels with a level C or M and above:

1. Parity: inverts the parity bit
2. Sync error: inverts the sync pulse
3. Bit count: transmits a word with fewer or greater than the normal 20 bit times
4. Manchester error: inverts either the first or second half-bit in a given bit position
5. Word count: transmits a specified number of data words, ignoring the word count field of the command word
6. Gap errors: insert a gap of specified length preceding a word
7. Zero-crossing distortion (level D or P only): move a specified zero-crossing from its normal position

Error injection is a three or four-step process. An application program:

1. Defines an error in the Device.
2. Tags the message or messages that are candidates to contain the error. (This step is optional.)
3. Enables error injection.
4. Associates error injection with an external trigger signal (optional).

The following subsections describe these steps in detail.

5.2.1.1 Defining the Error

Errors are defined using the `BTI1553_ErrorDefine` function. Only one error may be defined on the channel at any given time, and that error is used in all subsequent error injections. The error may be changed by repeated calls to `BTI1553_ErrorDefine`. Table 5.1 shows the parameters required for complete definition of different types of errors.

Error Type	Word Position	Bit Position	Count	Value
Parity	✓			
Sync	✓			
Manchester	✓	✓		
Word count			✓	
Bit count	✓		✓	✓
Gap	✓			✓
Zero crossing distortion	✓	✓	✓	

Table 5.1—BTI1553_ErrorDefine Parameters According to Error Type

5.2.1.2 Tagging Messages

Error injection is most useful when the error is associated with a specific message(s). For example, when simulating the BC, you may want to inject errors only in messages to a specific RT. In this case, you would tag those messages in the Bus Controller schedule for that RT. Then when error injection is enabled, only those messages are candidates to contain errors.

Errors can be injected into the transmissions of both simulated RTs and a simulated BC. Since different parts of a MIL-STD-1553 message are transmitted by different terminals, you must tag the message in the terminal that transmits the part of the message with the error. For example, if the error definition specifies an error in the status word of a message, the message must be tagged in the simulated RT (see Section 5.2.4).

Messages can be tagged in two ways. They can be tagged when they are created by using the `MSGCRT1553_ERR` flag in the `BTI1553_??CreateMsg` function. In addition, they can be tagged and untagged thereafter by calling one of the two `BTI1553_ErrorTag??` functions.

> *Note: Tagging a message does not guarantee that it will contain an error the next time it is transmitted. A tagged message is only a candidate.*

Two factors could prevent a tagged message from containing an error:

1. The error definition does not make sense in the given message. For example, the error definition specifies the sixth data word in a message, but the message tagged only transmits three data words.
2. Several messages have been tagged, but the channel is configured to only transmit one error after being enabled (see Section 5.2.1.3 below).

5.2.1.3 Enabling Error Injection

After an error has been defined and messages have been tagged, error injection can be enabled with the `BTI1553_ErrorCtrl` function. This function arms the channel's error-injection mechanism. Depending on the options selected by its flag parameter, one of four things happens following a call to `BTI1553_ErrorCtrl`:

1. OFF: Error injection is completely disabled.
2. ON: Every applicable message transmission contains the error until the condition is changed by another `BTI1553_ErrorCtrl` call.
3. ONCE: The next applicable message transmission contains an error. After one transmission of the error, error injection is automatically disabled.
4. EXTERNAL: Error injection occurs after an external trigger input. External trigger capability is Device-dependent. If the Device has multiple trigger lines, then `BTI1553_ErrorTriggerDefine` must also be used.

By default, only tagged messages are candidates to contain errors. However, this may be overridden by `BTI1553_ErrorCtrl`. If the `ERRCTRL1553_ANYMSG` flag is used in this function, any transmitted message can contain the error and tagging is ignored.

As the following examples show, the above steps can be performed at any time, including while the core is running. For example, you could inject intermittent, random errors in Bus Controller transmissions without stopping the BC schedule.

The following examples demonstrate how to define errors, tag messages, and enable error injection. Note that the examples in this section are similar to those in earlier sections, except that they ignore data to focus on the process of error injection.

5.2.2 Errors in BC Transmissions: Unscheduled Messages

The following code (Figure 5.2) modifies Example 1 (Figure 3.1) so that the transmitted message has a parity error in its command word.

Only the `BTI1553_ErrorDefine` and `BTI1553_ErrorCtrl` functions are added to Example 1. `BTI1553_ErrorDefine` specifies that the first command word in a message should contain a parity error. `BTI1553_ErrorCtrl` arms the error-injection mechanism. This example can safely use the `ERRCTRL1553_ANYMSG` flag since only one message is to be transmitted.

When `BTI1553_BCTransmitMsg` is executed, the transmitted message contains incorrect parity in its command word as specified by the parameters of `BTI1553_ErrorDefine`. According to MIL-STD-1553, an RT should never respond to invalid commands. Thus, in this example, no response is expected from the RT, and the first block of the *if…else* statement should be executed.

```
HCARD hCard;
HCORE hCore;
XMITFIELDS1553 Msg;

BTICard_CardOpen(&hCard,cardnum);
BTICard_CoreOpen(&hCore,corenum,hCard);
BTICard_CardReset(hCore);

BTI1553_BCConfig(BCCFG1553_DEFAULT,CH0,hCore);

BTI1553_BCSchedFrame(1000,CH0,hCore);

Msg.ctrlflags = MSGCRT1553_BCRT;
Msg.cwd1 = 0x0843;
Msg.data[0] = some_value0;
Msg.data[1] = some_value1;
Msg.data[2] = some_value2;

BTI1553_ErrorDefine(ERRDEF1553_PAR,0,0,ERRDEF1553_CWD1,0,CH0,hCore);
BTI1553_ErrorCtrl(ERRCTRL1553_ONCE | ERRCTRL1553_ANYMSG,CH0,hCore);

BTICard_CardStart(hCore);

BTI1553_BCTransmitMsg(&Msg,CH0,hCore);

if (Msg.errflags & (MSGERR1553_NORESP | MSGERR1553_ANYERR))
{
     //The RT did not respond, or it had an error in its response
}
else
{
     //The message was completed successfully
     //Check Msg.status1 for the RT’s status word response
}

BTICard_CardStop(hCore);
BTICard_CardClose(hCard);
```

Figure 5.2—Example 6: Errors in Unscheduled BC Transmissions

5.2.3 Errors in BC Transmissions: Scheduled Messages

The example in Figure 5.3 on the following page demonstrates how to

- Inject a single type of error into a Bus Controller's schedule
- Control which message contains an error

The code creates a schedule with two messages, defines a bit count error, and alternates which message contains the error while the schedule is running.

This example first creates a simple message schedule consisting of a single 10-ms frame containing two messages. Then `BTI1553_ErrorDefine` specifies a bit count error in which the first data word contains two extra bits, and the value of those bits is three (11 binary). `BTI1553_ErrorTagBC` marks the `0842h` message as a candidate to receive the error. Since `BTI1553_ErrorCtrl` is called before `BTICard_CardStart`, the very first transmission of `0842h` will contain the error. Since one occurrence was specified (by `ERRCTRL1553_ONCE`), subsequent transmissions will be error-free until the body of the if statement is executed.

```
HCARD hCard;
HCORE hCore;
MSGADDR BCMsgs[2];
INT ErrorCount = 0;

BTICard_CardOpen(&hCard,cardnum);
BTICard_CoreOpen(&hCore,corenum,hCard);
BTICard_CardReset(hCore);

BTI1553_BCConfig(BCCFG1553_DEFAULT,CH0,hCore);

                                                    //Create the messages
BCMsgs[0] =
BTI1553_BCCreateMsg(MSGCRT1553_DEFAULT,0x0842,0,&InitialData1,hCore);
BCMsgs[1] =
BTI1553_BCCreateMsg(MSGCRT1553_DEFAULT,0x1042,0,&InitialData2,hCore);

BTI1553_BCSchedFrame(10000,CH0,hCore);              //Create the schedule
BTI1553_BCSchedMsg(BCMsgs[0],CH0,hCore);
BTI1553_BCSchedMsg(BCMsgs[1],CH0,hCore);

                                      //Initialize error injection controls
BTI1553_ErrorDefine(ERRDEF1553_CNTBIT,3,2,0,0,CH0,hCore);
BTI1553_ErrorTagBC(TRUE,BCMsgs[0],CH0,hCore);
BTI1553_ErrorCtrl(ERRCTRL1553_ONCE |
ERRCTRL1553_TAGMSG,CH0,hCore);

BTICard_CardStart(hCore);

while (!done)
{
   if (BTI1553_ErrorSent(CH0,Card))
   {                                       //Switch error to another message
      if (++ErrorCount & 1)
      {
         BTI1553_ErrorTagBC(FALSE,BCMsgs[0],CH0,hCore);
         BTI1553_ErrorTagBC(TRUE,BCMsgs[1],CH0,hCore);
      }
      else
      {
         BTI1553_ErrorTagBC(TRUE,BCMsg[0],CH0,hCore);
         BTI1553_ErrorTagBC(FALSE,BCMsg[1],CH0,hCore);
      }
                                           //Reenable injection of one error
      BTI1553_ErrorCtrl(ERRCTRL1553_ONCE | ERRCTRL1553_TAGMSG,CH0,hCore);
   }
}

BTICard_CardStop(hCore);
BTICard_CardClose(hCard);
```

Figure 5.3—Example 7: Errors in Scheduled BC Transmissions

The *if (BTI1553_ErrorSent...)* conditional statement guarantees that the currently tagged message has been transmitted (at least once) with an error before the code reassigns the error to the other message. The code counts the number of errors that have been transmitted. The BTI1553_ErrorTagBC functions untag one of the two messages and tag the other according to the state of *ErrorCount*. Error injection is then re-enabled for a single occurrence by BTI1553_ErrorCtrl.

5.2.4 Errors in RT Transmissions

The following example (Figure 5.4) configures the channel to simulate a single RT and inject a variety of errors.

```
HCARD hCard;
HCORE hCore;

BTICard_CardOpen(&hCard,cardnum);
BTICard_CoreOpen(&hCore,corenum,hCard);
BTICard_CardReset(hCore);

BTI1553_RTConfig(RTCFG1553_DEFAULT,1,CH0,hCore);

BTI1553_ErrorDefine(ERRDEF1553_GAP,     //a gap of...
                  200,             //20 microseconds
                  0,               //(don't care)
                  ERRDEF_SWD1,     //...preceding the status word
                  0,               //(don't care)
                  CH0,
                  hCore);

BTI1553_ErrorTagRT(TRUE,SUBADDRESS,1,XMT,2,CH0,hCore);
BTI1553_ErrorCtrl(ERRCTRL1553_ONCE |
ERRCTRL1553_TAGMSG,CH0,hCore);

BTICard_CardStart(hCore);

while (!done)
{
   if (BTI1553_ErrorSent(CH0,Card)
   {                                      //Change error as desired
      BTI1553_ErrorDefine(new_error_type,
                       new_value,
                       new_count,
                       new_word_position,
                       new_bit_position,
                       CH0,
                       hCore);

      BTI1553_ErrorCtrl(ERRCTRL1553_ONCE |
      ERRCTRL1553_TAGMSG,CH0,hCore);
   }
}

BTICard_CardStop(hCore);
BTICard_CardClose(hCard);
```

Figure 5.4—Example 8: Errors in RT Transmission

After opening the card and configuring an RT with terminal address one (1), `BTI1553_ErrorDefine` defines a gap error of 20 μs to precede the status word in the RT's response. The value parameter of `BTI1553_ErrorDefine` for gap errors is in units of 0.1 μs. `BTI1553_ErrorTagRT` specifies that the error can occur when the RT is commanded to transmit from its second subaddress. `BTI1553_ErrorCtrl` enables one occurrence of the defined error.

As in the preceding example, `BTI1553_ErrorSent` is used to guarantee that the currently defined error is sent at least once before the code changes it. Inside the *if statement*, `BTI1553_ErrorDefine` changes the error definition. This function could merely change the length of the inserted gap error, or it could change the type of error altogether.

5.3 Variable Transmit Amplitude

On level D or P channels, the amplitude of the transmitted signal can be set using `BTI1553_ParamAmplitudeConfig`. This is a global setting for the channel, so all messages are transmitted with the same amplitude. The default is full amplitude, so `BTI1553_ParamAmplitudeConfig` only needs to be called to set a lower value or to restore the full amplitude later. For the relationship between the `dacvalue` setting and the amplitude, please refer to the hardware manual for your Device.

This page intentionally blank.

APPENDIX A: FUNCTION REFERENCES

This appendix provides detailed information on the primary MIL-STD-1553 BTIDriver functions for Ballard Devices. The descriptions and examples discussed here are intended for use with programs written in the C language. Users of other languages should contact Customer Support (see Section 1.6) for assistance.

> *Note: Each API function in this manual has a device dependency section. See Table A.3 below for these device dependency details.*

Overview of the BTIDriver API

The general naming convention for BTIDriver functions consists of a prefix/category/action format. The functions that make up the BTIDriver library are either specific to a particular avionics databus protocol, or are protocol-independent. The MIL-STD-1553–specific functions are prefixed by `BTI1553_` (see Table A.1) and the protocol-independent functions are prefixed by `BTICard_` (see Table B.1). Functions for other protocols are documented in separate manuals. The functions fall into several operational categories. The initial letters of a function name, after the prefix, indicate the category to which it belongs (e.g., BC, RT, Msg, Ch, etc.). These initial characters are followed by an action. For example, the function `BTI1553_BCSchedMsg` is a member of the *BC* category and causes a *1553* message to be scheduled for transmission. In this appendix, the function descriptions are listed alphabetically without regard to prefix.

"handle" Parameters

Nearly all functions require a `handle` parameter. The handle is always the last parameter in any function in which it is required. The first function called in a program is `BTICard_CardOpen`, which returns a card handle (`hCard`). This card handle is passed to `BTICard_CoreOpen`, which returns a core handle (`hCore`). Most functions then take this core handle; the only functions that require a card handle are `BTICard_CoreOpen` and `BTICard_CardClose`. Please refer to Section 2.2 for a complete discussion of handles, cards, and cores.

"ctrlflags" Parameters

Many functions have a `ctrlflags` parameter. Each bit controls an option in this bitmask parameter. Constants are defined in the header file for these parameters. The name of a constant reflects the function in which it is used (e.g., `RTCFG1553_DEFAULT` is used in the `BTI1553_RTConfig` function). Option parameters are always first in the parameter list of a function that accepts them. The default options can always be selected by using the `??_DEFAULT` constant where `??` depends on the function in which it is used (e.g., `BCCFG1553_DEFAULT`). When multiple options are selected, the constants should be bitwise OR-ed together. The default options are shown in bold throughout the function details in this appendix. Since the default constants are defined as zero, only nondefault constants actually need to be included in the OR-ing. The constants defined in the header file should be used by name (not value) in your code since the values are subject to change.

Schedule Indices (SCHNDX)

All of the scheduling functions (`BTI1553_BCSched??`) return a value of type `SCHNDX` (Schedule index). This is the index of the Command Block that the function appends to the schedule. This index is a parameter of some of the advanced scheduling functions (e.g., `BTI1553_BCSchedCall`, `BTI1553_BCSchedBranch`).

"channel" Parameters

Some functions take a `channel` parameter to specify which MIL-STD-1553 channel applies to the function. The header file defines the constants `CH0`, `CH1`, etc., which may be used for this purpose.

"message" Parameters

Message data and related information such as the time-tag are stored in individual message structures on the Device core. All of the message manipulation functions (e.g., `BTI1553_MsgDataRd`) require a message address parameter that uniquely identifies a message structure. Several different functions (e.g., `BTI1553_??CreateMsg` and `BTI1553_RTGetMsg`) return the message address.

Error Values

Type `ERRVAL` functions return a negative value if an error occurs or zero if successful. This value can be interpreted by `BTICard_ErrDescStr`, which returns a string describing the specified error. This same error information may also be obtained from the header file.

MIL-STD-1553 protocol errors are passed through the error field of the associated structures (declared in the header file). For example, the `errflags` field of the `XMITFIELDS1553` structure can be tested by AND-ing the values returned with the corresponding error constants (see `BTI1553_BCTransmitMsg`).

UINT16/UINT32

BTIDriver functions can be called by applications that run on both 32-bit systems and 64-bit systems, but BTIDriver functions always assume *short* integers are 16-bits long and *long* integers are 32 bits long. An application running on some 64-bit operating systems (for example, Linux) assumes a *short* integer is 16-bits long, but a *long* integer is 64-bits long. To resolve this discrepancy and to keep types consistent, applications that call BTIDriver functions from 64-bit operating systems supported by BTIDriver should use the `UINT16` type instead of `USHORT`, and the `UINT32` type instead of `ULONG`.

Function Summaries

A summary of the MIL-STD-1553_ functions are summarized in the Table A.1 (in the page below). As illustrated in the body of this manual, most applications can be implemented using just a few functions.

BUS CONTROLLER Functions	
BTI1553_BCConfig	Initializes a BC terminal configuration
BTI1553_BCConfigEx	Initializes a BC terminal configuration, allowing variable BC opcodes
BTI1553_BCConfigMsg	Reconfigures the message with additional configuration information
BTI1553_BCCreateList	Creates and initializes a message list for the BC
BTI1553_BCCreateMsg	Creates and initializes a message structure for the BC
BTI1553_BCPause	Pauses operation of the BC Schedule
BTI1553_BCPauseCheck	Checks to see if the BC Schedule is paused
BTI1553_BCResume	Resumes operation of the BC Schedule
BTI1553_BCSchedAgain	Copies BC command block and adds it to the end of the BC Schedule
BTI1553_BCSchedBranch	Inserts a branch opcode into the BC Schedule
BTI1553_BCSchedBranchUpdate	Updates a branch opcode in the BC Schedule
BTI1553_BCSchedBuild	Automatically sequences messages and frames according to given frequencies
BTI1553_BCSchedCall	Inserts a call opcode into the BC Schedule
BTI1553_BCSchedCallUpdate	Updates the destination of a call opcode
BTI1553_BCSchedEntry	Resets entry of BC Schedule to current location
BTI1553_BCSchedFrame	Inserts a FRAME Command Block into the Schedule
BTI1553_BCSchedFrameEnd	Schedules a frame end opcode in the BC Schedule
BTI1553_BCSchedFrameStart	Schedules a frame start opcode in the BC Schedule
BTI1553_BCSchedGap	Schedules a gap in the BC Schedule
BTI1553_BCSchedHalt	Inserts a HALT opcode into the BC Schedule
BTI1553_BCSchedLog	Inserts a LOG Command Block into the BC Schedule
BTI1553_BCSchedMsg	Inserts a MESSAGE Command Block into the BC Schedule
BTI1553_BCSchedPause	Inserts a PAUSE opcode into the BC Schedule
BTI1553_BCSchedPulse	Inserts opcodes to pulse a discrete I/O into the BC Schedule
BTI1553_BCSchedPulse0	Inserts opcode to turn off a discrete I/O into the BC Schedule
BTI1553_BCSchedPulse1	Inserts opcode to turn on a discrete I/O into the BC Schedule
BTI1553_BCSchedRestart	Inserts a RESTART opcode into the BC schedule
BTI1553_BCSchedRetry	Inserts a conditional RETRY of the last message transmission
BTI1553_BCSchedReturn	Inserts a RETURN opcode into the BC schedule
BTI1553_BCSetDefaultGap	Sets the intermessage default gap time used by the BC
BTI1553_BCSetTimeout	Sets the maximum time for declaring an RT non-responsive
BTI1553_BCSyncDefine	Defines the sync settings and configures all BC messages to output a sync
BTI1553_BCTransmitList	Transmits a list of messages
BTI1553_BCTransmitMsg	Transmits a single message one time
BTI1553_BCTriggerDefine	Defines the trigger settings and configures all BC messages for a trigger
BTI1553_CmdMaxLoopRd	Reads the maximum loop count for the BC Schedule
BTI1553_CmdMaxLoopWr	Writes the maximum loop count for the BC Schedule
BTI1553_CmdShotRd	Reads the single-shot bit for the specified BC schedule entry
BTI1553_CmdShotWr	Sets the single-shot bit for the specified BC schedule entry
BTI1553_CmdSkipRd	Reads the skip bit for the specified BC schedule entry
BTI1553_CmdSkipWr	Sets the skip bit for the specified BC schedule entry
BTI1553_CmdStepRd	Reads the step bit for the specified BC schedule entry
BTI1553_CmdStepWr	Sets the step bit for the specified BC schedule entry
BTI1553_PlayConfig	Configures the BC for hardware playback mode
BTI1553_PlayStatus	Determines the status of the hardware playback buffer
BTI1553_PlayWr	Writes data to the hardware playback buffer

Table A.1—MIL-STD-1553 (BTI1553_) Functions (continued in next page)

CHANNEL Functions	
BTI1553_ChGetInfo	Gets level, mode, terminal, and other information of the specified channel
BTI1553_ChIs1553	Checks to see if the specified channel is a 1553 bus
BTI1553_ChIsA	Checks if the specified 1553 channel is a level A
BTI1553_ChIsB4	Checks if the specified 1553 channel is a level B4
BTI1553_ChIsB32	Checks if the specified 1553 channel is a level B32
BTI1553_ChIsBM	Checks if the specified 1553 channel is a level BM
BTI1553_ChIsC	Checks if the specified 1553 channel is a level C
BTI1553_ChIsD	Checks if the specified 1553 channel is a level D
BTI1553_ChIsM	Checks if the specified 1553 channel is a level M
BTI1553_ChIsP	Checks if the specified 1553 channel is a level P
BTI1553_ChIsRM	Checks if the specified 1553 channel is a level RM
BTI1553_ChIsS	Checks if the specified 1553 channel is a level S
BTI1553_ChStart	Starts operation of a previously stopped 1553 channel
BTI1553_ChStop	Stops operation of a 1553 channel
ERROR INJECTION Functions	
BTI1553_ErrorCtrl	Enables error injection
BTI1553_ErrorDefine	Defines the error to be injected
BTI1553_ErrorSent	Determines whether an error was sent since the last BTI1553_ErrorCtrl call
BTI1553_ErrorTagBC	Tags the specified message as a candidate to receive an error (BC side)
BTI1553_ErrorTagRT	Tags the specified message as a candidate to receive an error (RT side)
BTI1553_ErrorTriggerDefine	Defines the trigger settings and associates error injection with a trigger signal
LIST Functions	
BTI1553_ListAddr	Determines address of message record contained in List Buffer
BTI1553_ListBlockRd	Reads the next message structure in the list
BTI1553_ListBlockWr	Writes the next message structure associated with a list
BTI1553_ListDataRd	Reads the next data associated with a list
BTI1553_ListDataWr	Writes the next data associated with a list
BTI1553_ListMultiBlockRd	Reads multiple structures from the list
BTI1553_ListMultiBlockWr	Writes multiple structures to the list
BTI1553_ListStatus	Checks the status of the list
MONITOR Functions	
BTI1553_MonConfig	Initializes and configures a Bus Monitor terminal
BTI1553_MonFilterSA	Configures subaddress filtering
BTI1553_MonFilterTA	Configures terminal address filtering

Table A.1—MIL-STD-1553 (BTI1553_) Functions (continued from previous page)

MESSAGE Functions	
BTI1553_MsgBlockRd	Reads an entire message structure from the core
BTI1553_MsgBlockWr	Writes an entire message structure on the core
BTI1553_MsgCommRd	Reads an entire message structure from the core (non-contending)
BTI1553_MsgCommWr	Writes an entire message structure on the core (non-contending)
BTI1553_MsgDataRd	Reads the data associated with a message
BTI1553_MsgDataWr	Writes the data associated with a message
BTI1553_MsgFieldRd	Reads a field from a message structure
BTI1553_MsgFieldWr	Writes a field to a message structure
BTI1553_MsgGroupDataRd	Reads a group of message data from the core
BTI1553_MsgGroupDataWr	Writes a group of message data to the core
BTI1553_MsgGroupRd	Reads message structures from the core in a single operation
BTI1553_MsgGroupWr	Writes message structures to the core in a single operation
BTI1553_MsgSkipRd	Reads the state of the skip bit for the Message Record
BTI1553_MsgSkipWr	Writes the state of the skip bit for the Message Record
BTI1553_MsgSyncDefine	Defines the sync settings and configures the message to output a sync
BTI1553_MsgTriggerDefine	Defines the trigger setting and configures the message for a trigger signal
PARAMETRIC Functions	
BTI1553_ParamAmplitudeConfig	Sets the transmitter amplitude on the specified channel
REMOTE TERMINAL Functions	
BTI1553_RTConfig	Initializes an RT terminal configuration
BTI1553_RTCreateList	Creates and initializes an RT message list
BTI1553_RTCreateMsg	Creates and initializes a message structure for an RT
BTI1553_RTGetMsg	Gets the address of an existing message structure
BTI1553_RTReset	Performs a logical reset of the RT (similar to Reset RT mode code)
BTI1553_RTResponseTimeSet	Sets the response time of the specified Remote Terminal
BTI1553_RTSetMode	Changes the mode of a configured Remote Terminal
BTI1553_RTSWDRd	Reads the status word for the RT
BTI1553_RTSWDWr	Sets the status word for the RT
BTI1553_RTSyncDefine	Defines the sync settings and configures the specified SAs/MCs to output a sync
DATA CONVERSION/UTILITY Functions	
BTI1553_ValPackCWD	Creates a MIL-STD-1553 command word from its four fields
BTI1553_ValUnpackCWD	Extracts the four bit fields from a MIL-STD-1553 command word

Table A.1—MIL-STD-1553 (BTI1553_) Functions (continued from previous page)

Levels of Functionality

A few functions only apply to MIL-STD-1553 channels with specific levels of functionality. The acceptable parameters for some functions may also be Device-dependent. For example, the error injection functions only apply to channels with level C, level D, level M, or level P capability. A MIL-STD-1553 channel can be one of the ten levels summarized in Table A.2. The functions in this appendix apply to all 1553 channels unless noted as limited to a certain level(s).

	Levels of Functionality									
Features	**A**	**B4**	**B32**	**C**	**D**	**S**	**M**	**P**	**B**	**RM**
BC/RT/MON Operation										
BC or 1 RT or MON	✓									
BC and 3 RTs or 4 RTs or MON		✓								
BC and 32 RTs or MON			✓							
BC or 32 RTs or MON						✓				
BC and 32 RTs and MON				✓	✓		✓	✓		
32 RTs or MON										✓
MON Only									✓	
Number of Simultaneous Terminals	1	4	1 or 32	32	32	1 or 32	32	32	1	32
Monitor	✓	✓	✓	✓	✓	✓	✓	✓	✓	
Filtering for terminal address	✓	✓	✓	✓	✓	✓	✓	✓	✓	
Filtering for subaddress		✓	✓	✓	✓	✓	✓	✓	✓	
Concurrent terminal monitoring				✓	✓		✓	✓		
Protocol Error Injection				✓	✓		✓	✓		
Variable Transmit Amplitude					✓			✓		
Zero Crossing Distortion					✓			✓		

Table A.2—Levels of Ballard MIL-STD-1553 Functionality

Device Dependence

The BTIDriver unified API supports many generations of Ballard hardware Devices. This cross-compatibility allows for application reuse when migrating from one Device to another. Each successive generation of Ballard hardware Devices tries to build upon the feature set of the previous one. Therefore, not all features supported by this API apply to all hardware Devices. Functions that depend upon a particular hardware Device will reference the products listed in Table A.3 by generation or by other functionality.

	Generation/Group				
Product	**3G**	**4G**	**5G**	**6G**	**RPC**
BUSBox (BB1xxx)	✓				
OmniBus PCI (111-xxx, 112-xxx-xxx)		✓			
OmniBus cPCI (121-xxx, 122-xxx-xxx)		✓			
OmniBus PMC (141-xxx)		✓			
OmniBus VME (152-xxx-xxx, 154-xxx-xxx-xxx-xxx)		✓			✓
OmniBusBox (162-xxx-xxx)		✓			✓
Avionics BusBox 1000 (AB1xxx)			✓		✓
Avionics BusBox 2000 (AB2xxx)			✓		✓
AB3000 Series (AB3xxx)			✓		✓
Lx1553-5, Lx429-5, PM1553-5, PM429-2, PE1000			✓		
USB 1553, USB 429/717, USB 708, USB Multi (UA1xxx)			✓		
Mx5 (Mx5x-xx-xx)			✓		
OmniBus II PCIe (212-xxx-xxx)				✓	
OmniBus II PXIe (222-xxx-xxx)				✓	
OmniBusBox II (262-xxx-xxx, 264-xxx-xxx-xxx-xxx)				✓	✓

Table A.3—Devices Grouped by Generation and Functionality

Function Detail

The following pages contain descriptions of the BTIDriver functions (in alphabetical order without regard to prefix). The constants in bold in the tables are the default options. Note that the "`BTICard_`" and "`BTI1553_`" prefixes have been omitted from the headings for easier reading, but all BTIDriver functions must begin with the appropriate prefix in source code.

BCConfig

```
ERRVAL BTI1553_BCConfig
(
   ULONG ctrlflags,          //Selects configuration options
   INT channel,              //Channel number
   HCORE hCore               //Core handle
)
```

RETURNS

A negative value if an error occurs or zero if successful.

DESCRIPTION

Configures a Bus Controller (BC) for the specified channel with the options defined by `ctrlflags`. This function allocates memory for a schedule of 512 Command Blocks and initializes internal variables associated with the schedule. Therefore, any previously created schedules are cleared by calling this function.

ctrlflags	
Constant	**Description**
BCCFG1553_DEFAULT	Select all default settings (**bold** below)
BCCFG1553_ENABLE	Enable the Bus Controller
BCCFG1553_DISABLE	Disable the Bus Controller
BCCFG1553_TRIGNONE	BC will not use external triggering
BCCFG1553_TRIGEXT	Sets BC trigger mode such that each message tagged for triggering will need an external trigger
BCCFG1553_TRIGSTART	Sets BC trigger mode such that only the first message tagged for triggering will need an external trigger (trigger condition is latched)
BCCFG1553_NOLOGHALT	No entry will be made in the Event Log List on a HALT command
BCCFG1553_LOGHALT	An entry will be made in the Event Log List on a HALT command
BCCFG1553_NOLOGPAUSE	No entry will be made in the Event Log List on a PAUSE command
BCCFG1553_LOGPAUSE	An entry will be made in the Event Log List on a PAUSE command
BCCFG1553_NOLOOPMAX	Disable maximum loop count
BCCFG1553_LOOPMAX	Enable maximum loop count
BCCFG1553_NOSTEP	Disables single-stepping of the Schedule
BCCFG1553_STEP	Enables single-stepping (use `BTI1553_BCResume` to step)
BCCFG1553_MC01	Select SA=00000 or SA=11111 for mode codes
BCCFG1553_MC1	Select SA=11111 for mode codes
BCCFG1553_MC0	Select SA=00000 for mode codes
BCCFG1553_MCNONE	Disable mode codes
BCCFG1553_SYNCSEL	Sync out selected at message level
BCCFG1553_SYNCALL	Sync out driven for all messages
BCCFG1553_UNPAUSE	Mark Bus Controller as unpaused
BCCFG1553_PAUSE	Mark Bus Controller as paused
BCCFG1553_TERMOFF	Direct coupled termination resistance off for Bus A and B (default)
BCCFG1553_TERMONA	Direct coupled termination resistance on for Bus A
BCCFG1553_TERMONB	Direct coupled termination resistance on for Bus B
BCCFG1553_TERMONAB	Direct coupled termination resistance on for Bus A and B

DEVICE DEPENDENCY

Termination flags are only supported on 4G and 6G Devices. 3G Devices have only one trigger and sync line while 4G, 5G, and 6G Devices support 3 sync and 3 trigger lines per core.

WARNINGS

None.

SEE ALSO

`BTI1553_BCConfigEx`, `BTI1553_RTConfig`, `BTI1553_MonConfig`, `BTI1553_BCTriggerDefine`, `BTI1553_MsgTriggerDefine`

BCConfigEx

```
ERRVAL BTI1553_BCConfigEx
(
  ULONG ctrlflags,          //Selects configuration options
  USHORT count,             //BC schedule entry count
  INT channel,              //Channel number
  HCORE hCore               //Core handle
)
```

RETURNS

A negative value if an error occurs or zero if successful.

DESCRIPTION

Configures the BC similar to `BTI1553_BCConfig`, but with the addition of `count`. The `count` parameter is used to specify the number of BC schedule entries to allocate for the schedule. `BTI1553_BCConfig` defaults to 512 entries, and `count` can be used to allocate a higher number. Each `BTI1553_BCSched???` function will use one or more schedule entry. For example, `BTI1553_BCSchedMsg` will use one schedule entry for each message scheduled.

DEVICE DEPENDENCY

3G and 4G Devices support up to 4089 schedule entries while 5G and 6G Devices support up to 8187 schedule entries.

WARNINGS

None.

SEE ALSO

`BTI1553_BCConfig`

BCConfigMsg

```
MSGADDR BTI1553_BCConfigMsg
(
   ULONG msgctrlflags,       //Selects message options
   MSGADDR message,          //Address of message
   HCORE hCore               //Core handle
)
```

RETURNS

The message address of the message that was updated.

DESCRIPTION

Reconfigures a `message` with the options specified by the `msgctrlflags`. The `msgctrlflags` are listed in the table under `BTI1553_BCCreateMsg`.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

This redefines the configuration options for a message.

SEE ALSO

`BTI1553_BCCreateMsg`

BCCreateList

```
LISTADDR BTI1553_BCCreateList
(
  ULONG listctrlflags,      //Selects List Buffer options
  INT count,                //One more than the number of entries in list
  ULONG msgctrlflags,       //Selects message options
  USHORT cwd1,              //Command word 1 to transmit
  USHORT cwd2,              //Command word 2 (RT-RT only)
  LPUSHORT data,            //Buffer containing data words
  HCORE hCore               //Core handle
)
```

RETURNS

The address of the List Buffer if successful, otherwise zero.

DESCRIPTION

Creates and initializes a message List Buffer for the BC. Similar to `BTI1553_BCCreateMsg` except creates a List Buffer. This function allocates memory for a list of message structures and initializes each entry with the command and data words provided. If the `data` pointer is NULL, data initialization is skipped. The maximum number of entries that may be stored in the list is `count-1`.

If the host is reading data from the List Buffer, the `LISTCRT1553_LOG` flag will generate an entry in the Event Log List when the list is full. If the host is writing data to the List Buffer and the `LISTCRT1553_LOG` flag is set, an entry in the Event Log List is generated when the list is empty.

When the `LISTCRT1553_LOGHALF` flag is specified, the List Buffer generates an entry in the Event Log List when the middle or the last entry is processed. This flag is not used if the List Buffer is configured for ping-pong mode.

listctrlflags	
Constant	**Description**
LISTCRT1553_DEFAULT	Select all default settings (**bold** below)
LISTCRT1553_PINGPONG	Selects ping-pong mode
LISTCRT1553_FIFO	Selects FIFO mode
LISTCRT1553_CIRCULAR	Selects circular mode
LISTCRT1553_LOG	Generates an EVENTYPE_1553LIST entry in the Event Log when the list is empty/full
LISTCRT1553_NOLOG	Does not generate an entry in the Event Log when the list is empty/full
LISTCRT1553_LOGHALF	Generates an EVENTYPE_1553MSG entry in the Event Log when either the middle or last entry is processed. To determine which message generated the Event Log entry, use BTI1553_ListAddr to get the address of the message.
LISTCRT1553_NOLOGHALF	Does not generate an entry in the Event Log when the middle or last entry is processed
LISTCRT1553_TRBIT	The host will read or write data depending upon the T/R bit of *cwd1* if a BC-RT transfer, or read data if an RT-RT transfer
LISTCRT1553_READ	The host will read receive data from the list
LISTCRT1553_WRITE	The host will write transmit data to the list

msgctrlflags	
Constant	**Description**
MSGCRT1553_DEFAULT	Select all default settings (**bold** below)
MSGCRT1553_NOLOG	This message will not generate an entry in the Event Log List
MSGCRT1553_LOG	This message will generate an entry in the Event Log List
MSGCRT1553_NOERR	This message is not a candidate for errors
MSGCRT1553_ERR	This message is a candidate for errors
MSGCRT1553_BUSA	Message will be transmitted on bus A (BC only)
MSGCRT1553_BUSB	Message will be transmitted on bus B (BC only)
MSGCRT1553_BCRT	Message is a BC-RT transfer (BC only)
MSGCRT1553_RTRT	Message is an RT-RT transfer (BC only)
MSGCRT1553_NOMON	Message will not be monitored (BC only, level C, D, P, and M channels only)
MSGCRT1553_MON	Message will be sent to the Sequential Monitor (BC only, level C, D, P, and M channels only)
MSGCRT1553_NOTIMETAG	Message will not record time-tag
MSGCRT1553_TIMETAG	Message will record time-tag
MSGCRT1553_NOELAPSE	Message will not record elapsed time
MSGCRT1553_ELAPSE	Message will record elapsed time
MSGCRT1553_NOMIN	Message will not record min time
MSGCRT1553_MIN	Message will record min time
MSGCRT1553_NOMAX	Message will not record max time
MSGCRT1553_MAX	Message will record max time
MSGCRT1553_NOHIT	Message will not record hit count
MSGCRT1553_HIT	Message will record hit count
MSGCRT1553_NOSYNC	Sync out will not be driven for this message
MSGCRT1553_SYNC	Sync out will be driven for this message
MSGCRT1553_WIPE	Enable the wipe options (below)
MSGCRT1553_NOWIPE	Disable the wipe options (data is random values)
MSGCRT1553_WIPE123	Initialize the data words with incrementing values
MSGCRT1553_WIPECWD	Initialize the data words with command word values
MSGCRT1553_WRAP	Rcv SA Data will be automatically wrapped to corresponding Xmt SA Data (RT only)
MSGCRT1553_NOWRAP	SA Data buffers operate normally (RT only)

DEVICE DEPENDENCY

LISTCRT1553_PINGPONG is not supported by 5G or 6G Devices since they have inherent protection for data coherency (use BTI1553_MsgCommRd and BTI1553_MsgCommWr instead).

3G and 4G Devices support up to 511 list entries while 5G and 6G Devices support up to 1023 list entries.

WARNINGS

None.

SEE ALSO

BTI1553_RTCreateList, BTI1553_ListDataWr, BTI1553_ListDataRd, BTI1553_BCCreateMsg, BTI1553_BCSchedMsg

BCCreateMsg

```
MSGADDR BTI1553_BCCreateMsg
(
  ULONG msgctrlflags,        //Selects message options
  USHORT cwd1,               //Command word 1 to transmit
  USHORT cwd2,               //Command word 2 (RT-RT only)
  LPUSHORT data,             //Buffer containing data words
  HCORE hCore                //Core handle
)
```

RETURNS

The address of the message structure if successful, otherwise zero.

DESCRIPTION

Allocates memory for a BC message structure and initializes that structure with the command and data words provided. If the `data` pointer is NULL, data initialization is skipped.

msgctrlflags	
Constant	**Description**
MSGCRT1553_DEFAULT	Select all default settings (**bold** below)
MSGCRT1553_NOLOG	This message will not generate an entry in the Event Log List
MSGCRT1553_LOG	This message will generate an entry in the Event Log List
MSGCRT1553_NOERR	This message is not a candidate for errors
MSGCRT1553_ERR	This message is a candidate for errors
MSGCRT1553_BUSA	Message will be transmitted on bus A (BC only)
MSGCRT1553_BUSB	Message will be transmitted on bus B (BC only)
MSGCRT1553_BCRT	Message is a BC-RT transfer (BC only)
MSGCRT1553_RTRT	Message is an RT-RT transfer (BC only)
MSGCRT1553_NOMON	Message will not be monitored (BC only, level C, D, P, and M channels only)
MSGCRT1553_MON	Message will be sent to the Sequential Monitor (BC only, level C, D, P, and M channels only)
MSGCRT1553_NOTIMETAG	Message will not record time-tag
MSGCRT1553_TIMETAG	Message will record time-tag
MSGCRT1553_NOELAPSE	Message will not record elapsed time
MSGCRT1553_ELAPSE	Message will record elapsed time
MSGCRT1553_NOMIN	Message will not record min time
MSGCRT1553_MIN	Message will record min time
MSGCRT1553_NOMAX	Message will not record max time
MSGCRT1553_MAX	Message will record max time
MSGCRT1553_NOHIT	Message will not record hit count
MSGCRT1553_HIT	Message will record hit count
MSGCRT1553_NOSYNC	Sync out will not be driven for this message
MSGCRT1553_SYNC	Sync out will be driven for this message
MSGCRT1553_WIPE	Enable the wipe options (below)
MSGCRT1553_NOWIPE	Disable the wipe options (data is random values)
MSGCRT1553_WIPE123	Initialize the data words with incrementing values
MSGCRT1553_WIPECWD	Initialize the data words with command word values
MSGCRT1553_WRAP	Rcv SA Data will be automatically wrapped to corresponding Xmt SA Data (RT only)
MSGCRT1553_NOWRAP	SA Data buffers operate normally (RT only)

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

None.

SEE ALSO

`BTI1553_RTCreateMsg`, `BTI1553_BCCreateList`, `BTI1553_BCSchedMsg`, `BTI1553_MsgDataWr`, `BTI1553_MsgDataRd`

BCPause

```
VOID BTI1553_BCPause
(
   INT channel,                    //Channel number
   HCORE hCore                     //Core handle
)
```

RETURNS

None.

DESCRIPTION

Pauses the operation of the Bus Controller Schedule on the `channel`. The Bus Controller remains paused until resumed by `BTI1553_BCResume`. `BTI1553_BCConfig` initializes the Bus Controller Schedule as either unpaused (default) or paused.

Note: A Bus Controller Schedule can also be paused when it encounters a `PAUSE` Command Block.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

Do not confuse this Bus Controller pause/resume with either channel enable or the core-level controls. Channel enable is controlled by `BTI1553_ChStart`, and `BTI1553_ChStop`. Core-level controls are activated through `BTICard_CardStart`, `BTICard_CardStop`, and `BTICard_CardResume`.

SEE ALSO

`BTI1553_BCResume`, `BTI1553_ChStart`, `BTI1553_ChStop`, `BTICard_CardStart`, `BTICard_CardStop`, and `BTICard_CardResume`

BCPauseCheck

```
INT BTI1553_BCPauseCheck
(
   INT channel,                //Channel number
   HCORE hCore                 //Core handle
)
```

RETURNS

A non-zero value if the channel is paused or zero if the channel is not paused.

DESCRIPTION

Determines whether the BC Schedule on the `channel` is paused.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

None.

SEE ALSO

`BTI1553_BCPause, BTI1553_BCResume`

BCResume

```
VOID BTI1553_BCResume
(
  INT channel,            //Channel number
  HCORE hCore             //Core handle
)
```

RETURNS

None.

DESCRIPTION

Resumes the operation of the Bus Controller Schedule on `channel` after it has been paused by `BTI1553_BCPause` or the BC Schedule has encountered a `PAUSE` Command Block. If the channel is running, the Bus Controller Schedule continues to be processed from where it left off prior to being paused. If the channel is stopped, the schedule will continue to be processed when the channel is started.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

Do not confuse this Bus Controller pause/resume with either channel enable or the core-level controls. Channel enable is controlled by `BTI1553_ChStart`, and `BTI1553_ChStop`. Core-level controls are activated through `BTICard_CardStart`, `BTICard_CardStop`, and `BTICard_CardResume`.

SEE ALSO

`BTI1553_BCPause`, `BTI1553_ChStart`, `BTI1553_ChStop`, `BTICard_CardStart`, `BTICard_CardStop`, and `BTICard_CardResume`

BCSchedAgain

```
SCHNDX BTI1553_BCSchedAgain
(
   SCHNDX index,            //Previous schedule index
   INT channel,             //Channel number
   HCORE hCore              //Core handle
)
```

RETURNS

Schedule index of the newly created BC Schedule entry or a negative value if an error occurred.

DESCRIPTION

Copies the BC command block specified by index and inserts it at the end of the BC Schedule.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

None.

SEE ALSO

None.

BCSchedBranch

```
SCHNDX BTI1553_BCSchedBranch
(
   ULONG condition,           //Condition flags
   SCHNDX index,              //Destination schedule index
   INT channel,               //Channel number
   HCORE hCore                //Core handle
)
```

RETURNS

Schedule index of the newly created BC Schedule entry or a negative value if an error occurred.

DESCRIPTION

Inserts a branch opcode into the BC Schedule. The destination of the branch is specified by `index` and the conditions of the branch are specified by `condition`.

Condition	
Constant	**Description**
COND1553_ALWAYS	Unconditional
COND1553_FAIL	All retries have failed
COND1553_SRQ	Service request bit set in status word
COND1553_INS	Instrumentation bit set in status word
COND1553_SSF	Subsystem fail bit set in status word
COND1553_TF	Terminal flag bit set in status word
COND1553_BUSY	Busy bit set in status word
COND1553_ME	Message error bit set in status word
COND1553_RESPERR	An error in the status word response
COND1553_NORESP	No status word response received
COND1553_ALTBUS	Retries are performed on alternate bus (applies to BTI1553_BCSchedRetry only)
COND1553_DIO1ACT	DIO1 is active
COND1553_DIO1NACT	DIO1 is inactive
COND1553_DIO2ACT	DIO2 is active
COND1553_DIO2NACT	DIO2 is inactive
COND1553_DIO3ACT	DIO3 is active
COND1553_DIO3NACT	DIO3 is inactive
COND1553_DIO4ACT	DIO4 is active
COND1553_DIO4NACT	DIO4 is inactive

DEVICE DEPENDENCY

The mapping of DIO numbers to physical discrete I/O is hardware dependent. Please consult the hardware manual for the Device.

WARNINGS

None.

SEE ALSO

`BTI1553_BCSchedBranchUpdate, BTI1553_BCSchedCall`

BCSchedBranchUpdate

```
ERRVAL BTI1553_BCSchedBranchUpdate
(
   SCHNDX destindex,            //Destination schedule index
   SCHNDX opcodeindex,          //Branch opcode schedule index
   INT channel,                 //Channel number
   HCORE hCore                  //Core handle
)
```

RETURNS

A negative value if an error occurs or zero if successful.

DESCRIPTION

Updates the destination of the branch opcode specified by `opcodeindex`. The destination of the branch is specified by `destindex`.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

None.

SEE ALSO

`BTI1553_BCSchedBranch`

BCSchedBuild

```
ERRVAL BTI1553_BCSchedBuild
(
  USHORT nummsgs,          //Number of messages to schedule
  MSGADDR msgs[],          //Array of message addresses on the core
  USHORT freqs[],          //Array of message frequencies
  INT channel,             //Channel number
  HCORE hCore              //Core handle
)
```

RETURNS

A negative value if an error occurs or zero if successful.

DESCRIPTION

Creates a new BC Schedule that sequences messages and frames by given frequencies. This function clears any previously created BC Schedule from the specified channel. The new BC Schedule will consist of `nummsgs` messages, each transmitted at a specific frequency. `Msgs` points to an array of message addresses, each previously generated by a call to `BTI1553_BCCreateMsg`.*`freqs`* points to an array of frequencies in units of hertz (Hz).

The function schedules messages and frames to generate the specified frequencies. If the schedule can not be generated, an error is returned.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

A call to `BTI1553_BCConfig` must precede this function as well as a call to `BTI1553_BCCreateMsg` for each message to be scheduled.

SEE ALSO

`BTI1553_BCSchedFrame`, `BTI1553_BCSchedMsg`

BCSchedCall

```
SCHNDX BTI1553_BCSchedCall
(
  ULONG condition,          //Condition flags
  SCHNDX index,             //Destination schedule index
  INT channel,              //Channel number
  HCORE hCore               //Core handle
)
```

RETURNS

Schedule index of the newly created BC schedule entry or a negative value if an error occurs.

DESCRIPTION

Inserts a call opcode into the BC Schedule. The destination of the call is specified by `index`, and the conditions of the call are specified by `condition`. To return from the call, use `BTI1553_BCSchedReturn`.

Condition	
Constant	**Description**
COND1553_ALWAYS	Unconditional
COND1553_FAIL	All retries have failed
COND1553_SRQ	Service request bit set in status word
COND1553_INS	Instrumentation bit set in status word
COND1553_SSF	Subsystem fail bit set in status word
COND1553_TF	Terminal flag bit set in status word
COND1553_BUSY	Busy bit set in status word
COND1553_ME	Message error bit set in status word
COND1553_RESPERR	An error in the status word response
COND1553_NORESP	No status word response received
COND1553_ALTBUS	Retries are performed on alternate bus (applies to `BTI1553_BCSchedRetry` only)
COND1553_DIO1ACT	DIO1 is active
COND1553_DIO1NACT	DIO1 is inactive
COND1553_DIO2ACT	DIO2 is active
COND1553_DIO2NACT	DIO2 is inactive
COND1553_DIO3ACT	DIO3 is active
COND1553_DIO3NACT	DIO3 is inactive
COND1553_DIO4ACT	DIO4 is active
COND1553_DIO4NACT	DIO4 is inactive

DEVICE DEPENDENCY

The mapping of DIO numbers to physical discrete I/O is hardware dependent. Please consult the hardware manual for the Device.

WARNINGS

None.

SEE ALSO

`BTI1553_BCSchedReturn`, `BTI1553_BCSchedCallUpdate`, `BTI1553_BCSchedBranch`

BCSchedCallUpdate

```
ERRVAL BTI1553_BCSchedCallUpdate
(
   SCHNDX destindex,         //Destination schedule index
   SCHNDX opcodeindex,       //Branch opcode schedule index
   INT channel,              //Channel number
   HCORE hCore               //Core handle
)
```

RETURNS

A negative value if an error occurs or zero if successful.

DESCRIPTION

Updates the destination of the call opcode specified by the `opcodeindex`. The destination of the call is specified by the `destindex`.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

None.

SEE ALSO

`BTI1553_BCSchedCall`

BCSchedEntry

```
SCHNDX BTI1553_BCSchedEntry
(
   INT channel,                //Channel number
   HCORE hCore                 //Core handle
)
```

RETURNS

Schedule index of the newly created BC Schedule entry or a negative value if an error occurred.

DESCRIPTION

Resets the entry point of the BC Schedule to the current location. Note that this is not needed by default since the BC Schedule entry point is automatically set to the first opcode scheduled.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

None.

SEE ALSO

None.

BCSchedFrame

```
SCHNDX BTI1553_BCSchedFrame
(
  ULONG timeval,          //Length of frame in microseconds
  INT channel,            //Channel number
  HCORE hCore             //Core handle
)
```

RETURNS

Schedule index of the newly created BC Schedule entry or a negative value if an error occurred.

DESCRIPTION

Appends a `FRAME` Command Block to the current end of the Schedule. A `FRAME` Command Block marks the start of a new frame. When the onboard processor encounters a `FRAME` Command Block, it waits until the timer set by a previous `FRAME` Command Block reaches zero. It then sets the timer to the *`timeval`* specified by the current `FRAME` Command Block and continues processing the schedule.

This function supports `timeval` lengths between 20 µs and 1,310,700 µs.

DEVICE DEPENDENCY

3G, 4G, and 5G Devices utilize a 20 µs resolution frame timer; therefore, `timeval` lengths are internally rounded down to the nearest 20 µs step size. 6G Devices utilize a 1 µs resolution frame timer; therefore, `timeval` lengths are not internally rounded.

WARNINGS

A call to `BTI1553_BCConfig` must precede this function.

SEE ALSO

`BTI1553_BCSchedMsg`

BCSchedFrameEnd

```
SCHNDX BTI1553_BCSchedFrameEnd
(
   INT channel,              //Channel number
   HCORE hCore               //Core handle
)
```

RETURNS

Schedule index of the newly created BC Schedule entry or a negative value if an error occurred.

DESCRIPTION

Schedules a frame end opcode in the BC Schedule. The frame end causes the BC to wait while the frame timer is non-zero. After the frame timer expires, the BC continues with the schedule.

If using `BTI1553_BCSchedFrame`, it is not necessary to use this function as internally `BTI1553_BCSchedFrame` calls `BTI1553_BCSchedFrameEnd` and `BTI1553_BCSchedFrameStart`.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

None.

SEE ALSO

`BTI1553_BCSchedFrameStart, BTI1553_BCSchedFrame`

BCSchedFrameStart

```
SCHNDX BTI1553_BCSchedFrameStart
(
  ULONG timeval,          //Frame timer value
  INT channel,            //Channel number
  HCORE hCore             //Core handle
)
```

RETURNS

Schedule index of the newly created BC schedule entry or a negative value if an error occurred.

DESCRIPTION

Schedules a frame start opcode in the BC schedule. The frame start opcode causes the BC to load the frame timer with the value specified by the `timeval` parameter.

This function supports `timeval` lengths between 20 µs and 1,310,700 µs.

If using `BTI1553_BCSchedFrame`, it is not necessary to use this function as internally `BTI1553_BCSchedFrame` calls `BTI1553_BCSchedFrameEnd` and `BTI1553_BCSchedFrameStart`.

DEVICE DEPENDENCY

3G, 4G, and 5G Devices utilize a 20 µs resolution frame timer; therefore, `timeval` lengths are internally rounded down to the nearest 20 µs step size. 6G Devices utilize a 1 µs resolution frame timer; therefore, `timeval` lengths are not internally rounded.

WARNINGS

None.

SEE ALSO

`BTI1553_BCSchedFrame`, `BTI1553_BCSchedFrameEnd`

BCSchedGap

```
SCHNDX BTI1553_BCSchedGap
(
   INT gapval,          //Gap value in tenths of a bit time (100 ns)
   INT channel,         //Channel number
   HCORE hCore          //Core handle
)
```

RETURNS

Schedule index of the newly created BC Schedule entry or a negative value if an error occurred.

DESCRIPTION

Schedules a gap in the BC Schedule. Valid gap times are between 40 and 8191, inclusively. Gaps are specified in tenths of a bit time (100 ns) from the mid-bit zero-crossing of the previous parity bit to the mid-bit zero-crossing of the next command word sync. Multiple gaps can be scheduled back to back to create larger gaps.

The intended use of the gap counter is to create gap times in between messages. The frame timer should be used to create the timing for groups of messages, and the gap timer should be used to create gaps in between the messages if needed.

DEVICE DEPENDENCY

The minimum gap times of 3G and 4G Devices are limited by bus loading. Deterministic gap times down to 4 µs can be created on 5G and 6G Devices, but gap times less than 80 (8 µs) are not supported unless `BTI1553_BCSetDefaultGap` has been called with a value less than 80. Schedule entries that are dependent upon the previous status word response (eg. `BTI1553_BCSchedRetry`) will often violate the specified gap time.

WARNINGS

None.

SEE ALSO

`BTI1553_BCSchedFrame`, `BTI1553_BCSchedFrameEnd`, `BTI1553_BCSchedFrameStart`, `BTI1553_BCSetDefaultGap`

BCSchedHalt

```
SCHNDX BTI1553_BCSchedHalt
(
   INT channel,              //Channel number
   HCORE hCore               //Core handle
)
```

RETURNS

Schedule index of the newly created BC Schedule entry or a negative value if an error occurred.

DESCRIPTION

Inserts a HALT opcode into the BC Schedule. A HALT opcode will stop the BC Schedule from running. Alternatively, to pause the BC Schedule instead of halting it, use `BTI1553_BCSchedPause`.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

None.

SEE ALSO

`BTI1553_BCSchedPause`

BCSchedLog

```
SCHNDX BTI1553_BCSchedLog
(
   ULONG condition,          //Value to test
   USHORT tagval,            //Event tag value
   INT channel,              //Channel number
   HCORE hCore               //Core handle
)
```

RETURNS

Schedule index of the newly created BC Schedule entry or a negative value if an error occurred.

DESCRIPTION

Appends a conditional LOG Command Block to the current end of the Schedule. A conditional LOG Command Block causes the core to generate an Event Log List entry if `condition` evaluates as TRUE. The event type placed in the Event Log List is `EVENTTYPE_1553OPCODE` and the user-specified value `tagval` is used as the info value. Entries are read out of the Event Log List using `BTICard_EventLogRd.`

The `condition` flags listed below may be used to specify the Event condition.

Condition	
Constant	**Description**
COND1553_ALWAYS	Unconditional
COND1553_FAIL	All retries have failed
COND1553_SRQ	Service request bit set in status word
COND1553_INS	Instrumentation bit set in status word
COND1553_SSF	Subsystem fail bit set in status word
COND1553_TF	Terminal flag bit set in status word
COND1553_BUSY	Busy bit set in status word
COND1553_ME	Message error bit set in status word
COND1553_RESPERR	An error in the status word response
COND1553_NORESP	No status word response received
COND1553_ALTBUS	Retries are performed on alternate bus (applies to `BTI1553_BCSchedRetry` only)
COND1553_DIO1ACT	DIO1 is active
COND1553_DIO1NACT	DIO1 is inactive
COND1553_DIO2ACT	DIO2 is active
COND1553_DIO2NACT	DIO2 is inactive
COND1553_DIO3ACT	DIO3 is active
COND1553_DIO3NACT	DIO3 is inactive
COND1553_DIO4ACT	DIO4 is active
COND1553_DIO4NACT	DIO4 is inactive

DEVICE DEPENDENCY

The mapping of DIO numbers to physical discrete I/O is hardware dependent. Please consult the hardware manual for the Device.

WARNINGS

A call to `BTI1553_BCConfig` must precede this function.

SEE ALSO

`BTICard_EventLogRd`

BCSchedMsg

```
SCHNDX BTI1553_BCSchedMsg
(
   MSGADDR message,          //Address of message or list
   INT channel,              //Channel number
   HCORE hCore               //Core handle
)
```

RETURNS

Schedule index of the newly created BC Schedule entry or a negative value if an error occurred.

DESCRIPTION

Appends a MESSAGE Command Block to the current end of the schedule. When a MESSAGE Command Block is encountered in the schedule, the message or the next message from the associated list specified by message is transmitted.

Note: Execution of this function does NOT transmit the message. The message is transmitted only when the resulting schedule is executed after BTICard_CardStart starts the core.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

A call to BTI1553_BCConfig must precede this function. In addition, the message must have been created with BTI1553_BCCreateMsg or BTI1553_BCCreateList.

SEE ALSO

BTI1553_BCCreateMsg, BTI1553_BCCreateList, BTI1553_BCSchedFrame, BTI1553_BCSchedRetry

BCSchedPause

```
SCHNDX BTI1553_BCSchedPause
(
  INT channel,              //Channel number
  HCORE hCore               //Core handle
)
```

RETURNS

Schedule index of the newly created BC Schedule entry or a negative value if an error occurred.

DESCRIPTION

Inserts a PAUSE opcode into the BC Schedule. A PAUSE opcode will pause operation of the BC Schedule. To resume the schedule, use BTI1553_BCResume. Alternatively, to halt the BC Schedule instead of pausing it, use BTI1553_BCSchedHalt.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

Using BCSchedPause with frames (BCSchedFrame, BCSchedFrameStart) causes the frame timer to act as if the frame time has expired when schedule/channel is resumed.

Schedules internally use prefetching to minimize intermessage gap time. When scheduling a BCSchedPause entry, two very small preceding gaps (BCSchedGap with a gapval of 100=10 μs) should be inserted because of schedule prefetching. If these additional gaps are not inserted, the schedule could pause before a message immediately preceding the BCSchedPause entry is transmitted.

SEE ALSO

BTI1553_BCSchedHalt, BTI1553_BCPauseCheck,
BTI1553_BCResume

BCSchedPulse

```
SCHNDX BTI1553_BCSchedPulse
(
  INT dionum,           //DIO number to pulse
  INT channel,          //Channel number
  HCORE hCore           //Core handle
)
```

RETURNS

Schedule index of the newly created BC Schedule entry or a negative value if an error occurred.

DESCRIPTION

Schedules a pair of opcodes that pulse the specified `dionum` to the *On* state followed by the *Off* state. Use `BTICard_ExtDIOWr` for normal updating of the discrete I/O signals.

DEVICE DEPENDENCY

The level of the `dionum` *On* and *Off* states, as well as the mapping of `dionum` to physical discrete I/O is hardware dependent. Please consult the hardware manual for the Device.

WARNINGS

The `SCHNDX` returned is for the second of the pair of opcodes that are scheduled. The opcodes are scheduled sequentially, so the `SCHNDX` of the first pulse opcode can be derived from the return value by subtracting one.

Schedules internally use prefetching to minimize intermessage gap time. Because of schedule prefetching and asynchronous discrete operation, the absolute timing of the pulse from a `BCSchedPulse` entry can vary by a gap/message transmission. When comparing the occurrence of scheduled discrete transitions with adjacent 1553 messages, the discrete can appear to transition early.

For discrete output pulses synchronous to 1553 databus activity, use `BTI1553_BCSyncDefine`, `BTI1553_MsgSyncDefine`, or `BTI1553_RTSyncDefine`.

SEE ALSO

`BTI1553_BCSchedPulse0`, `BTI1553_BCSchedPulse1`, `BTICard_ExtDIORd`, `BTICard_ExtDIOWr`

BCSchedPulse0

```
SCHNDX BTI1553_BCSchedPulse0
(
  INT dionum,              //DIO number to pulse
  INT channel,             //Channel number
  HCORE hCore              //Core handle
)
```

RETURNS

Schedule index of the newly created BC Schedule entry, or a negative value if an error occurred.

DESCRIPTION

Sets the state of the specified `dionum` to the *Off* state. Use `BTICard_ExtDIOWr` for normal updating of the discrete I/O signals.

Note that by using this function in combination with either `BTI1553_BCSchedGap`, `BTI1553_BCSchedFrameStart`, and/or `BTI1553_BCSchedFrameEnd` the I/O signals can be used to frame messages, groups of messages, or to create arbitrary pulse width signals.

DEVICE DEPENDENCY

The level of the `dionum` *On* and *Off* states, as well as the mapping of `dionum` to physical discrete I/O is hardware dependent. Please consult the hardware manual for the Device.

WARNINGS

Schedules internally use prefetching to minimize intermessage gap time. Because of schedule prefetching and asynchronous discrete operation, the absolute timing of the pulse from a `BCSchedPulse` entry can vary by a gap/message transmission. When comparing the occurrence of scheduled discrete transitions with adjacent 1553 messages, the discrete can appear to transition early.

For discrete output pulses synchronous to 1553 databus activity, use `BTI1553_BCSyncDefine`, `BTI1553_MsgSyncDefine`, or `BTI1553_RTSyncDefine`.

SEE ALSO

`BTI1553_BCSchedFrameEnd`, `BTI1553_BCSchedFrameStart`, `BTI1553_BCSchedGap`, `BTI1553_BCSchedPulse`, `BTI1553_BCSchedPulse1`, `BTICard_ExtDIORd`, `BTICard_ExtDIOWr`

BCSchedPulse1

```
SCHNDX BTI1553_BCSchedPulse1
(
   INT dionum,                //DIO number to pulse
   INT channel,               //Channel number
   HCORE hCore                //Core handle
)
```

RETURNS

Schedule index of the newly created BC Schedule entry or a negative value if an error occurred.

DESCRIPTION

Sets the state of the specified `dionum` to the *On* state. Use `BTICard_ExtDIOWr` for normal updating of the discrete I/O signals.

Note that by using this function in combination with either `BTI1553_BCSchedGap`, `BTI1553_BCSchedFrameStart`, and/or `BTI1553_BCSchedFrameEnd` the I/O signals can be used to frame messages, groups of messages, or to create arbitrary pulse width signals.

DEVICE DEPENDENCY

The level of the `dionum` *On* and *Off* states, as well as the mapping of `dionum` to physical discrete I/O is hardware dependent. Please consult the hardware manual for the Device.

WARNINGS

Schedules internally use prefetching to minimize intermessage gap time. Because of schedule prefetching and asynchronous discrete operation, the absolute timing of the pulse from a `BCSchedPulse` entry can vary by a gap/message transmission. When comparing the occurrence of scheduled discrete transitions with adjacent 1553 messages, the discrete can appear to transition early.

For discrete output pulses synchronous to 1553 databus activity, use `BTI1553_BCSyncDefine`, `BTI1553_MsgSyncDefine`, or `BTI1553_RTSyncDefine`.

SEE ALSO

`BTI1553_BCSchedFrameEnd`, `BTI1553_BCSchedFrameStart`, `BTI1553_BCSchedGap`, `BTI1553_BCSchedPulse`, `BTI1553_BCSchedPulse0`, `BTICard_ExtDIORd`, `BTICard_ExtDIOWr`

BCSchedRestart

```
SCHNDX BTI1553_BCSchedRestart
(
  INT channel,                //Channel number
  HCORE hCore                 //Core handle
)
```

RETURNS

Schedule index of the newly created BC Schedule entry or a negative value if an error occurred.

DESCRIPTION

Inserts a RESTART opcode into the BC Schedule. When this opcode is executed, the schedule will continue execution at the schedule entry point.

Note that the BC is automatically configured to restart the schedule by default. To deviate from this it is necessary to schedule a HALT opcode by using BTI1553_BCSchedHalt.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

None.

SEE ALSO

BTI1553_BCSchedHalt

BCSchedRetry

```
SCHNDX BTI1553_BCSchedRetry
(
   ULONG condition,          //Value to test
   USHORT retries,           //Number of retries (0–255)
   INT channel,              //Channel number
   HCORE hCore               //Core handle
)
```

RETURNS

Schedule index of the newly created BC Schedule entry or a negative value if an error occurred.

DESCRIPTION

Appends a conditional `RETRY` Command Block to the current end of the schedule. When a conditional `RETRY` Command Block is encountered in the schedule, it causes the Bus Controller to retransmit the most recently transmitted message, subject to the conditions specified by `condition`. The `condition` flags listed below may be OR-ed together. The BC retransmits the message up to `retries` times or until all conditions fail.

Condition	
Constant	**Description**
COND1553_ALWAYS	Unconditional
COND1553_FAIL	All retries have failed
COND1553_SRQ	Service request bit set in status word
COND1553_INS	Instrumentation bit set in status word
COND1553_SSF	Subsystem fail bit set in status word
COND1553_TF	Terminal flag bit set in status word
COND1553_BUSY	Busy bit set in status word
COND1553_ME	Message error bit set in status word
COND1553_RESPERR	An error in the status word response
COND1553_NORESP	No status word response received
COND1553_ALTBUS	Retries are performed on alternate bus (applies to `BTI1553_BCSchedRetry` only)
COND1553_DIO1ACT	DIO1 is active
COND1553_DIO1NACT	DIO1 is inactive
COND1553_DIO2ACT	DIO2 is active
COND1553_DIO2NACT	DIO2 is inactive
COND1553_DIO3ACT	DIO3 is active
COND1553_DIO3NACT	DIO3 is inactive
COND1553_DIO4ACT	DIO4 is active
COND1553_DIO4NACT	DIO4 is inactive

DEVICE DEPENDENCY

The mapping of `dionum` to physical discrete I/O is hardware dependent. Please consult the hardware manual for the Device.

WARNINGS

A call to `BTI1553_BCConfig` must precede this function.

SEE ALSO

`BTI1553_BCSchedFrame`, `BTI1553_BCSchedMsg`

BCSchedReturn

```
SCHNDX BTI1553_BCSchedReturn
(
   INT channel,               //Channel number
   HCORE hCore                //Core handle
)
```

RETURNS

Schedule index of the newly created BC Schedule entry, or a negative value if an error occurred.

DESCRIPTION

Inserts a `RETURN` opcode into the BC Schedule. A `RETURN` opcode returns from a previous `CALL` opcode and continues `BC` Schedule execution after the previous call.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

None.

SEE ALSO

`BTI1553_BCSchedCall`

BCSetDefaultGap

```
ERRVAL BTI1553_BCSetDefaultGap
(
  INT gapval,                //Gap value in tenths of a bit time (100 ns)
  INT channel,               //Channel number
  HCORE hCore                //Core handle
)
```

RETURNS

A negative value if an error occurs or zero if successful.

DESCRIPTION

Sets the default gap time between scheduled messages. This can be replaced by a unique gap value at any point in the schedule by using `BTI1553_BCSchedGap`. Valid gap times are between 40 and 8191, inclusively. *`Gapval`* is measured in tenths of a bit time (100 ns) from the mid-bit zero-crossing of the previous parity bit to the mid-bit zero-crossing of the next command word sync. The default intermessage gap time is 80 (8 µs).

DEVICE DEPENDENCY

Only supported on 5G and 6G Devices. Gap times less than 80 (8 µs) cause the BC to enter a mode where it will begin a new word without verifying that the current RT is not transmitting unexpected data words. Schedule entries that are dependent upon the previous status word response (eg. `BTI1553_BCSchedRetry`) will often insert a gap that is larger than the specificed gap time.

WARNINGS

None.

SEE ALSO

`BTI1553_BCSchedGap`

BCSetTimeout

```
ERRVAL BTI1553_BCSetTimeout
(
  USHORT timeoutval,        //Timeout value in tenths of a bit time (100 ns)
  INT channel,              //Channel number
  HCORE hCore               //Core handle
)
```

RETURNS

A negative value if an error occurs or zero if successful.

DESCRIPTION

When simulating the BC, sets the maximum time the BC will wait before declaring that an RT is non-responsive. `Timeoutval` is measured in tenths of a bit time (100 ns) from the mid-bit zero crossing of the previous parity bit to the mid-bit zero crossing of the status word sync. `Timeoutval` is the *No Response* timeout value plus 3.0 µs. The default value for `timeoutval` is 190, making the default no-response timeout 16 µs. The maximum value for `timeoutval` is 1023, making the maximum no-response timeout 99.3 µs.

The no-response timeout value set by this funciton is also used by the receiving RT of an RT-RT transfer (when simulated by the Device). The *No Response* timeout value should be set higher than the longest expected RT response time of the bus. Use the function `BTI1553_RTResponseTimeSet` to control the RT response time of a simulated RT.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

This function should not be called while the channel is running. `BTI1553_BCSetTimeout` is a Protocol Error Injection feature (see Table A.2).

SEE ALSO

`BTI1553_BCConfig`

BCSyncDefine

```
ERRVAL BTI1553_BCSyncDefine
(
   BOOL enable,               //Enable/disable external sync output
   USHORT syncmask,           //Line(s) used for sync output
   USHORT pinpolarity,        //Active pin polarity (high/low)
   INT channel,               //Channel number
   HCORE hCore                //Core handle
)
```

RETURNS

A negative value if an error occurs or zero if successful.

DESCRIPTION

Defines the sync output settings for `channel` on `hCore` and configures all Bus Controller messages to output a sync signal. This sync signal appears on all lines specified by `syncmask` with the polarity specified by `pinpolarity` (see tables below). `BTI1553_BCSyncDefine` may be called during run time to redefine the sync output settings.

When enabled, the sync output line(s) is driven active with every BC transmission. The sync signal is active from just before the start of the BC transmission to just after the end of the BC transmission.

The constants in the tables below may be bitwise OR-ed together to configure multiple lines. Sync outputs are both generation dependent and device dependent. Refer to Table A.3 and the following tables for more information.

syncmask				
Constant	**Description**	**4G**	**5G**	**6G**
`SYNCMASK_SYNCA`	Selects discrete sync line A	✓	✓	✓
`SYNCMASK_PXITRIGA`	Selects PXIe TRIG sync line A			*
`SYNCMASK_SYNCB`	Selects discrete sync line B	✓	✓	✓
`SYNCMASK_PXITRIGB`	Selects PXIe TRIG sync line B			*
`SYNCMASK_SYNCC`	Selects discrete sync line C	✓	✓	✓
`SYNCMASK_PXITRIGC`	Selects PXIe TRIG sync line C			*
`SYNCMASK_PXISTARC`	Selects PXIe STAR sync line C			*

** PXIe Syncs are only available on PXIe Devices. Refer to `BTICard_CardSyncValid` for device dependent support*

pinpolarity	
Constant	**Description**
`SYNCPOL_SYNCAL`	Sets active low polarity for sync line A
`SYNCPOL_SYNCAH`	Sets active high polarity for sync line A
`SYNCPOL_SYNCBL`	Sets active low polarity for sync line B
`SYNCPOL_SYNCBH`	Sets active high polarity for sync line B
`SYNCPOL_SYNCCL`	Sets active low polarity for sync line C
`SYNCPOL_SYNCCH`	Sets active high polarity for sync line C

Alternatively, to configure selected message(s) to output a sync pulse, use `BTI1553_MsgSyncDefine`.

Additionally, to configure an RT to output a sync signal during its transmissions, use `BTI1553_RTSyncDefine`.

DEVICE DEPENDENCY

Applies to 4G, 5G, and 6G Devices. 3G Devices, which have only a single sync line, can use `BTI1553_BCConfig` with the `BCCFG1553_SYNCALL` flag. The mapping of sync lines is hardware dependent. Please consult the hardware manual for the Device.

WARNINGS

None.

SEE ALSO

`BTI1553_MsgSyncDefine`, `BTI1553_RTSyncDefine`, `BTICard_CardSyncValid`

BCTransmitList

```
ERRVAL BTI1553_BCTransmitList
(
  LPXMITFIELDS1553 msgs,    //Pointer to an array of transmit structures
  INT count,                //Number of messages in the array
  INT channel,              //Channel number
  HCORE hCore               //Core handle
)
```

RETURNS

A negative value if an error occurs or zero if successful.

DESCRIPTION

Transmits a list of `count` messages (*`msgs`*) on the specified `channel`. Messages in the list are transmitted sequentially without host intervention. By default, the entire list is transmitted after the last scheduled message of a frame (`MSGCRT1553_EOF`). However, the entire list can interrupt scheduled messages by transmitting in *high-priority* (`MSGCRT1553_HIPRI`). Transmission mode is device dependent and can be selected using the `ctrlflags` of the first message structure (in the list).

When transmitting with `MSGCRT1553_EOF`, a schedule with at least one frame must have been created using `BTI1553_BCSchedFrame` prior to calling `BTI1553_BCTransmitList`. When transmitting with `MSGCRT1553_HIPRI`, no scheduled frames are required prior to calling `BTI1553_BCTransmitList`. This function does not return until the entire list transmission is complete. The `XMITFIELDS1553` structure is declared in the header file and shown below. All of its members, except for `ctrlflags`, correspond with those of internal message structures created by `BTI1553_BCCreateMsg`. The `ctrlflags` member is a bit field, the bits of which correspond to the `MSGCRT1553_ constants` (see table on next page).

XMITFIELDS1553 Structure		
Field	**Size**	**Description**
ctrlflags	ULONG	User writes message configuration options (see table on next page)
flag1	USHORT	Device writes message flag 1
flag2	USHORT	Device writes message flag 2
errflags	USHORT	Device writes error flags (see table below)
actflags	USHORT	Device writes activity flags
resptime1	USHORT	Device writes response time +3.0 µs (in units of 0.1 µs) of status word 1
resptime2	USHORT	Device writes response time +3.0 µs (in units of 0.1 µs) of status word 2
datacount	USHORT	Device writes the number of valid data words associated with this message
timetag	ULONG	Device writes the lower 32 bits of the time-tag value
elapsetime	ULONG	Device writes the value of elapsed time
timetagh	ULONG	Device writes the upper 32 bits of the time-tag value
cwd1	USHORT	User writes command word 1
cwd2	USHORT	User writes command word 2 (RT-RT only)
swd1	USHORT	Device writes status word 1
swd2	USHORT	Device writes status word 2 (RT-RT only)
data[32]	USHORT	Device or user writes data words
extra[8]	USHORT	User writes additional data (when generating a word count error)

ctrlflags	
Constant	**Description**
MSGCRT1553_DEFAULT	Select all default settings (**bold** below)
MSGCRT1553_EOF	Message will be transmitted after the last message in the frame
MSGCRT1553_HIPRI	Message transmission will interrupt scheduled messages (Device dependent)
MSGCRT1553_NOLOG	This message will not generate an entry in the Event Log List
MSGCRT1553_LOG	This message will generate an entry in the Event Log List
MSGCRT1553_NOERR	This message is not a candidate for errors
MSGCRT1553_ERR	This message is a candidate for errors
MSGCRT1553_BUSA	Message will be transmitted on bus A
MSGCRT1553_BUSB	Message will be transmitted on bus B
MSGCRT1553_BCRT	Message is a BC-RT transfer
MSGCRT1553_RTRT	Message is an RT-RT transfer
MSGCRT1553_NOMON	Message will not be monitored (BC only, level C channels only)
MSGCRT1553_MON	Message will be sent to the Sequential Monitor (BC only, level C channels only)
MSGCRT1553_NOTIMETAG	Message will not record time-tag
MSGCRT1553_TIMETAG	Message will record time-tag
MSGCRT1553_NOELAPSE	Message will not record elapsed time
MSGCRT1553_ELAPSE	Message will record elapsed time
MSGCRT1553_NOMIN	Message will not record min time
MSGCRT1553_MIN	Message will record min time
MSGCRT1553_NOMAX	Message will not record max time
MSGCRT1553_MAX	Message will record max time
MSGCRT1553_NOHIT	Message will not record hit count
MSGCRT1553_HIT	Message will record hit count
MSGCRT1553_NOSYNC	Sync out will not be driven for this message
MSGCRT1553_SYNC	Sync out will be driven for this message
MSGCRT1553_WIPE	Enable the wipe options (below)
MSGCRT1553_NOWIPE	Disable the wipe options (data is random values)
MSGCRT1553_WIPE0	Initialize the data words with zeros
MSGCRT1553_WIPE123	Initialize the data words with incrementing values
MSGCRT1553_WIPECWD	Initialize the data words with command word values

errflags	
Constant	**Description**
MSGERR1553_NORESP	No response was received from the RT
MSGERR1553_ANYERR	Set if any other error bits are set
MSGERR1553_PROTOCOL	A protocol error occurred
MSGERR1553_SYNC	Wrong polarity of the sync pulse
MSGERR1553_DATACOUNT	Too many/too few data words
MSGERR1553_MANCH	Manchester error
MSGERR1553_PARITY	Parity error
MSGERR1553_WORD	Reserved
MSGERR1553_RETRY	All attempts to retry transmission of this message failed
MSGERR1553_SYSTEM	Internal Device error
MSGERR1553_HIT	Indicates that this message was transmitted since this bit was last cleared

DEVICE DEPENDENCY

Both BCTransmitList and high-priority message transmission (MSGCRT1553_HIPRI) are device dependent.

Applies to all 6G Devices and some 5G Devices. Applications can check for support for this feature using BTI1553_ChGetInfo.

WARNINGS

Calls to BTI1553_BCConfig and BTI1553_BCSchedFrame must precede this function if using MSGCRT1553_EOF.

Use caution when using BCTransmitList in conjunction with a schedule of messages. BCTransmitList will interrupt and impact the timing relationship of the schedule.

SEE ALSO

BTI1553_BCConfig, BTI1553_BCSchedFrame, BTI1553_ChGetInfo

BCTransmitMsg

```
ERRVAL BTI1553_BCTransmitMsg
(
  LPXMITFIELDS1553 msg,      //Pointer to a structure
  INT channel,               //Channel number
  HCORE hCore                //Core handle
)
```

RETURNS

A negative value if an error occurs or zero if successful.

DESCRIPTION

Transmits a single message (`msg`) one time on the specified `channel.` By default, the message is transmitted after the last scheduled message of a frame (`MSGCRT1553_EOF`). However, on some devices the message can interrupt scheduled messages by transmitting high-priority (`MSGCRT1553_HIPRI`). Transmission mode is device dependent and can be selected using `ctrlflags`.

When transmitting with `MSGCRT1553_EOF`, a schedule with at least one frame must have been created using `BTI1553_BCSchedFrame` prior to calling `BTI1553_BCTransmitMsg`. When transmitting with `MSGCRT1553_HIPRI`, no scheduled frames are required prior to calling `BTI1553_BCTransmitMsg`.

This function does not return until transmission is complete.

The `XMITFIELDS1553` structure is declared in the header file and shown below. All of its members, except for `ctrlflags`, correspond with those of internal message structures created by `BTI1553_BCCreateMsg`. The `ctrlflags` member is a bit field, the bits of which correspond to the `MSGCRT1553_` constants (see table on next page).

XMITFIELDS1553 Structure		
Field	**Size**	**Description**
ctrlflags	ULONG	User writes message configuration options (see table on next page)
flag1	USHORT	Device writes message flag 1
flag2	USHORT	Device writes message flag 2
errflags	USHORT	Device writes error flags (see table below)
actflags	USHORT	Device writes activity flags
resptime1	USHORT	Device writes response time + 3.0 µs (in units of 0.1 µs) of status word 1
resptime2	USHORT	Device writes response time + 3.0 µs (in units of 0.1 µs) of status word 2
datacount	USHORT	Device writes the number of valid data words associated with this message
timetag	ULONG	Device writes the lower 32 bits of the time-tag value
elapsetime	ULONG	Device writes the value of elapsed time
timetagh	ULONG	Device writes the upper 32 bits of the time-tag value
cwd1	USHORT	User writes command word 1
cwd2	USHORT	User writes command word 2 (RT-RT only)
swd1	USHORT	Device writes status word 1
swd2	USHORT	Device writes status word 2 (RT-RT only)
data[32]	USHORT	Device or user writes data words
extra[8]	USHORT	User writes additional data (when generating a word count error)

ctrlflags	
Constant	**Description**
MSGCRT1553_DEFAULT	Select all default settings (**bold** below)
MSGCRT1553_EOF	Message will be transmitted after the last message in the frame
MSGCRT1553_HIPRI	Message transmission will interrupt scheduled messages (Device dependent)
MSGCRT1553_NOLOG	This message will not generate an entry in the Event Log List
MSGCRT1553_LOG	This message will generate an entry in the Event Log List
MSGCRT1553_NOERR	This message is not a candidate for errors
MSGCRT1553_ERR	This message is a candidate for errors
MSGCRT1553_BUSA	Message will be transmitted on bus A
MSGCRT1553_BUSB	Message will be transmitted on bus B
MSGCRT1553_BCRT	Message is a BC-RT transfer
MSGCRT1553_RTRT	Message is an RT-RT transfer
MSGCRT1553_NOMON	Message will not be monitored (BC only, level C channels only)
MSGCRT1553_MON	Message will be sent to the Sequential Monitor (BC only, level C channels only)
MSGCRT1553_NOTIMETAG	Message will not record time-tag
MSGCRT1553_TIMETAG	Message will record time-tag
MSGCRT1553_NOELAPSE	Message will not record elapsed time
MSGCRT1553_ELAPSE	Message will record elapsed time
MSGCRT1553_NOMIN	Message will not record min time
MSGCRT1553_MIN	Message will record min time
MSGCRT1553_NOMAX	Message will not record max time
MSGCRT1553_MAX	Message will record max time
MSGCRT1553_NOHIT	Message will not record hit count
MSGCRT1553_HIT	Message will record hit count
MSGCRT1553_NOSYNC	Sync out will not be driven for this message
MSGCRT1553_SYNC	Sync out will be driven for this message
MSGCRT1553_WIPE	Enable the wipe options (below)
MSGCRT1553_NOWIPE	Disable the wipe options (data is random values)
MSGCRT1553_WIPE0	Initialize the data words with zeros
MSGCRT1553_WIPE123	Initialize the data words with incrementing values
MSGCRT1553_WIPECWD	Initialize the data words with command word values

errflags	
Constant	**Description**
MSGERR1553_NORESP	No response was received from the RT
MSGERR1553_ANYERR	Set if any other error bits are set
MSGERR1553_PROTOCOL	A protocol error occurred
MSGERR1553_SYNC	Wrong polarity of the sync pulse
MSGERR1553_DATACOUNT	Too many/too few data words
MSGERR1553_MANCH	Manchester error
MSGERR1553_PARITY	Parity error
MSGERR1553_WORD	Reserved
MSGERR1553_RETRY	All attempts to retry transmission of this message failed
MSGERR1553_SYSTEM	Internal Device error
MSGERR1553_HIT	Indicates that this message was transmitted since this bit was last cleared

DEVICE DEPENDENCY

When IRIG is enabled on a 4G Device, time-tags in message structures and Sequential Records will be in BCD format (see `BTICard_TimerStatus`).

High-priority message transmission (`MSGCRT1553_HIPRI`) is device dependent. Applications can check for support for this feature using `BTI1553_ChGetInfo`.

WARNINGS

Calls to `BTI1553_BCConfig` and `BTI1553_BCSchedFrame` (EOF mode only) must precede this function.

SEE ALSO

`BTI1553_BCConfig`, `BTI1553_BCSchedFrame`,
`BTI1553_ChGetInfo`

BCTriggerDefine

```
ERRVAL BTI1553_BCTriggerDefine
(
   BOOL enable,             //Enable/disable external trigger
   USHORT trigmask,         //Line(s) used for trigger signal
   USHORT trigval,          //Active/inactive condition for trigger line(s)
   USHORT pinpolarity,      //Active pin polarity (high/low)
   INT channel,             //Channel number
   HCORE hCore              //Core handle
)
```

RETURNS

A negative value if an error occurs or zero if successful.

DESCRIPTION

Defines the trigger input settings for `channel` on `hCore` and associates all Bus Controller messages with a trigger signal. The input line(s) are specified by `trigmask` with an active trigger state being the combination of `trigval` and `pinpolarity`, as defined in the tables below. `BTI1553_BCTriggerDefine` may be called during run time to redefine the trigger input settings.

Before calling `BTI1553_BCTriggerDefine`, the trigger mode must first be set by calling `BTI1553_BCConfig` with the `BCCFG1553_TRIGSTART` or `BCCFG1553_TRIGEXT` flag. When start mode is selected (`BCCFG1553_TRIGSTART`), the Bus Controller waits for the trigger signal, then starts transmitting the schedule and runs normally. However, if external trigger mode is selected (`BCCFG1553_TRIGEXT`), the Bus Controller waits for the trigger signal before transmitting each message.

The constants in the tables below may be bitwise OR-ed together to configure multiple lines. Only one `trigmask` may be selected per trigger line (A, B, and C), and all combined states must be true for the trigger to occur. Triggers are both generation dependent and device dependent. Refer to Table A.3 and the following tables for more information.

trigmask				
Constant	**Description**	**4G**	**5G**	**6G**
TRIGMASK_TRIGA	Selects discrete trigger line A	✓	✓	✓
TRIGMASK_PXITRIGA	Selects PXIe TRIG trigger line A			*
TRIGMASK_PXISTARA	Selects PXIe STAR trigger line A			*
TRIGMASK_TRIGB	Selects discrete trigger line B	✓	✓	✓
TRIGMASK_PXITRIGB	Selects PXIe TRIG trigger line B			*
TRIGMASK_PXISTARB	Selects PXIe STAR trigger line B			*
TRIGMASK_TRIGC	Selects discrete trigger line C	✓	✓	✓
TRIGMASK_PXITRIGC	Selects PXIe TRIG trigger line C			*
TRIGMASK_PXISTARC	Selects PXIe STAR trigger line C			*

** PXIe Syncs are only available on PXIe Devices. Refer to* `BTICard_CardTriggerValid` *for device dependent support*

trigval	
Constant	**Description**
TRIGVAL_TRIGAOFF	Trigger on line A inactive
TRIGVAL_TRIGAON	Trigger on line A active
TRIGVAL_TRIGBOFF	Trigger on line B inactive
TRIGVAL_TRIGBON	Trigger on line B active
TRIGVAL_TRIGCOFF	Trigger on line C inactive
TRIGVAL_TRIGCON	Trigger on line C active

pinpolarity				
Constant	**Description**	**4G**	**5G**	**6G**
TRIGPOL_TRIGAL	Sets active low polarity for trigger line A	✓	✓	✓
TRIGPOL_TRIGAH	Sets active high polarity for trigger line A	✓	✓	✓
TRIGPOL_TRIGAF	Sets active on falling edge of trigger line A			✓
TRIGPOL_TRIGAR	Sets active on rising edge of trigger line A			✓
TRIGPOL_TRIGBL	Sets active low polarity for trigger line B	✓	✓	✓
TRIGPOL_TRIGBH	Sets active high polarity for trigger line B	✓	✓	✓
TRIGPOL_TRIGBF	Sets active on falling edge of trigger line B			✓
TRIGPOL_TRIGBR	Sets active on rising edge of trigger line B			✓
TRIGPOL_TRIGCL	Sets active low polarity for trigger line C	✓	✓	✓
TRIGPOL_TRIGCH	Sets active high polarity for trigger line C	✓	✓	✓
TRIGPOL_TRIGCF	Sets active on falling edge of trigger line C			✓
TRIGPOL_TRIGCR	Sets active on rising edge of trigger line C			✓

Alternatively, to associate selected message(s) to a trigger signal, use `BTI1553_MsgTriggerDefine`.

Additionally, to associate error injection with a trigger signal, use `BTI1553_ErrorTriggerDefine`.

DEVICE DEPENDENCY

Applies to 4G, 5G, and 6G Devices. 3G Devices, which have only a single trigger line, can use `BTI1553_BCConfig` with the `BCCFG1553_TRIGSTART` or `BCCFG1553_TRIGEXT` flag, as described below. The mapping of trigger lines is hardware dependent. Please consult the hardware manual for the Device.

WARNINGS

`BTI1553_BCConfig` must be called and configured with the `BCCFG1553_TRIGSTART` or `BCCFG1553_TRIGEXT` flag before calling `BTI1553_BCTriggerDefine.`

SEE ALSO

`BTI1553_BCConfig, BTI1553_MsgTriggerDefine, BTI1553_ErrorTriggerDefine, BTICard_CardTrigger, BTICard_CardTriggerEx, BTICard_CardTriggerValid`

ChGetInfo

```
ULONG BTI1553_ChGetInfo
(
   USHORT infotype,        //Type of information to be returned
   INT channel,            //Channel number
   HCORE hCore             //Core handle
)
```

RETURNS

The requested information about the specified channel.

DESCRIPTION

Provides information about the functionality of the specified *channel* (see Table A.2). The `infotype` constants listed in the table below may be used to specify the requested information.

Infotype		
Constant	**Value**	**Description**
INFO1553_MODEL	See table below	Returns the channel model type
INFO1553_MAXCOUNT	1, 4, 0 (see table below)	Returns the number of terminals on the channel (1, 4, 32)
INFO1553_MULTIMODE	1=TRUE 0=FALSE	Returns whether the channel can simulate multiple modes (Bus Controller, Remote Terminal, and/or Bus Monitor).
INFO1553_ERRORGEN	1=TRUE 0=FALSE	Returns whether the channel can generate protocol errors (Device-dependent)
INFO1553_CONCURMON	1=TRUE 0=FALSE	Returns whether the channel has a concurrent Bus Monitor (Device-dependent)
INFO1553_MONFILTER	1=TRUE 0=FALSE	Returns whether the channel has Bus Monitor filtering for subaddresses (Device-dependent)
INFO1553_PARAM	1=TRUE 0=FALSE	Returns whether the channel supports variable transmit amplitude control and zero crossing distortion (Device-dependent)
INFO1553_RTRESPTIME	1=TRUE 0=FALSE	Returns whether the channel supports programmable RT response time (Device-dependent)
INFO1553_BCDFLTGAP	1=TRUE 0=FALSE	Returns whether the channel supports programmable BC intermessage gap time (Device-dependent)
INFO1553_BCXMTLIST	1=TRUE 0=FALSE	Returns whether the Channel supports `BCTransmitList` (Device-dependent)
INFO1553_BCXMTHIPRI	1=TRUE 0=FALSE	Returns whether the Channel supports high priority asynchronous messages (Device-dependent)

Level	**MAXCOUNT**
A	1
B4	4
B32	0
C	0
D	0
M	0
S	0
BM	1
RM	0
P	0

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

None.

SEE ALSO

BTI1553_ChIsA, BTI1553_ChIsB4, BTI1553_ChIsB32, BTI1553_ChIsBM, BTI1553_ChIsC, BTI1553_ChIsM, BTI1553_ChIsP, BTI1553_ChIsRM, BTI1553_ChIsS

ChIs1553

```
BOOL BTI1553_ChIs1553
(
   INT channel,                //Channel number to test
   HCORE hCore                 //Core handle
)
```

RETURNS

TRUE if the channel is 1553, otherwise FALSE.

DESCRIPTION

Checks to see if the channel number specified by `channel` is a 1553 channel.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

None.

SEE ALSO

BTI1553_ChGetInfo, BTI1553_ChIsA, BTI1553_ChIsB4, BTI1553_ChIsB32, BTI1553_ChIsBM, BTI1553_ChIsC, BTI1553_ChIsD, BTI1553_ChIsM, BTI1553_ChIsP, BTI1553_ChIsRM, BTI1553_ChIsS

ChIsA

```
BOOL BTI1553_ChIsA
(
   INT channel,                //Channel number to test
   HCORE hCore                 //Core handle
)
```

RETURNS

TRUE if the channel is a level A, otherwise FALSE.

DESCRIPTION

Checks to see if the channel number specified by `channel` has level A capability. See Table A.2 for information on the levels of capability for MIL-STD-1553 channels.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

None.

SEE ALSO

`BTI1553_ChGetInfo`, `BTI1553_ChIsB4`, `BTI1553_ChIsB32`, `BTI1553_ChIsBM`, `BTI1553_ChIsC`, `BTI1553_ChIsD`, `BTI1553_ChIsM`, `BTI1553_ChIsP`, `BTI1553_ChIsRM`, `BTI1553_ChIsS`

ChIsB4

```
BOOL BTI1553_ChIsB4
(
   INT channel,                 //Channel number to test
   HCORE hCore                  //Core handle
)
```

RETURNS

TRUE if the channel is a level B4, otherwise FALSE.

DESCRIPTION

Checks to see if the channel number specified by `channel` has level B4 capability. See Table A.2 for information on the levels of capability for MIL-STD-1553 channels.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

None.

SEE ALSO

`BTI1553_ChGetInfo`, `BTI1553_ChIsA`, `BTI1553_ChIsB32`, `BTI1553_ChIsBM`, `BTI1553_ChIsC`, `BTI1553_ChIsD`, `BTI1553_ChIsM`, `BTI1553_ChIsP`, `BTI1553_ChIsRM`, `BTI1553_ChIsS`

ChIsB32

```
BOOL BTI1553_ChIsB32
(
   INT channel,                //Channel number to test
   HCORE hCore                 //Core handle
)
```

RETURNS

TRUE if the channel is a level B32, otherwise FALSE.

DESCRIPTION

Checks to see if the channel number specified by `channel` has level B32 capability. See Table A.2 for information on the levels of capability for MIL-STD-1553 channels.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

None.

SEE ALSO

`BTI1553_ChGetInfo`, `BTI1553_ChIsA`, `BTI1553_ChIsB4`, `BTI1553_ChIsBM`, `BTI1553_ChIsC`, `BTI1553_ChIsD`, `BTI1553_ChIsM`, `BTI1553_ChIsP`, `BTI1553_ChIsRM`, `BTI1553_ChIsS`

ChIsBM

```
BOOL BTI1553_ChIsBM
(
   INT channel,                  //Channel number to test
   HCORE hCore                   //Core handle
)
```

RETURNS

TRUE if the channel is a level BM, otherwise FALSE.

DESCRIPTION

Checks to see if the channel number specified by `channel` has level BM capability. See Table A.2 for information on the levels of capability for MIL-STD-1553 channels.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

None.

SEE ALSO

`BTI1553_ChGetInfo, BTI1553_ChIsA, BTI1553_ChIsB4, BTI1553_ChIsB32, BTI1553_ChIsC, BTI1553_ChIsD, BTI1553_ChIsM, BTI1553_ChIsP, BTI1553_ChIsRM, BTI1553_ChIsS`

ChIsC

```
BOOL BTI1553_ChIsC
(
   INT channel,                 //Channel number to test
   HCORE hCore                  //Core handle
)
```

RETURNS

TRUE if the channel is a level C, otherwise FALSE.

DESCRIPTION

Checks to see if the channel number specified by `channel` has level C capability. See Table A.2 for information on the levels of capability for MIL-STD-1553 channels.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

None.

SEE ALSO

`BTI1553_ChGetInfo, BTI1553_ChIsA, BTI1553_ChIsB4, BTI1553_ChIsB32, BTI1553_ChIsBM, BTI1553_ChIsD, BTI1553_ChIsM, BTI1553_ChIsP, BTI1553_ChIsRM, BTI1553_ChIsS`

ChIsD

```
ULONG BTI1553_ChIsD
(
   INT channel,                //Channel number to test
   HCORE hCore                 //Core handle
)
```

RETURNS

TRUE if the channel is a level D, otherwise FALSE.

DESCRIPTION

Checks to see if the channel number specified by `channel` has level D capability. See Table A.2 for information on the levels of capability for MIL-STD-1553 channels.

The availability of level D MIL-STD-1553 channels is Device-dependent.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

None.

SEE ALSO

`BTI1553_ChGetInfo`, `BTI1553_ChIsA`, `BTI1553_ChIsB4`, `BTI1553_ChIsB32`, `BTI1553_ChIsBM`, `BTI1553_ChIsC`, `BTI1553_ChIsM`, `BTI1553_ChIsP`, `BTI1553_ChIsRM`, `BTI1553_ChIsS`

ChIsM

```
BOOL BTI1553_ChIsM
(
   INT channel,            //Channel number to test
   HCORE hCore             //Core handle
)
```

RETURNS

TRUE if the channel is a level M channel, otherwise FALSE.

DESCRIPTION

Checks to see if the channel number specified by `channel` has level M (Multi-Function) capabilities. See Table A.2 for information on the levels of capability for MIL-STD-1553 channels.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

None.

SEE ALSO

`BTI1553_ChIs1553`, `BTI1553_ChIsA`, `BTI1553_ChIsB32`, `BTI1553_ChIsB4`, `BTI1553_ChIsBM`, `BTI1553_ChIsC`, `BTI1553_ChIsD`, `BTI1553_ChIsP`, `BTI1553_ChIsRM`, `BTI1553_ChIsS`

ChIsP

```
BOOL BTI1553_ChIsP
(
   INT channel,           //Channel number to test
   HCORE hCore            //Core handle
)
```

RETURNS

TRUE if the channel is a level P channel, otherwise FALSE.

DESCRIPTION

Checks to see if the channel number specified by `channel` has level P (Multi-Function with Parametric) capabilities. See Table A.2 for information on the levels of capability for MIL-STD-1553 channels.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

None.

SEE ALSO

`BTI1553_ChIs1553, BTI1553_ChIsA, BTI1553_ChIsB32, BTI1553_ChIsB4, BTI1553_ChIsBM, BTI1553_ChIsC, BTI1553_ChIsD, BTI1553_ChIsM, BTI1553_ChIsRM, BTI1553_ChIsS`

ChIsRM

```
BOOL BTI1553_ChIsRM
(
   INT channel,              //Channel number to test
   HCORE hCore               //Core handle
)
```

RETURNS

TRUE if the channel is a level RM channel, otherwise FALSE.

DESCRIPTION

Checks to see if the channel number specified by `channel` has level RM (RT or Monitor) capabilities. See Table A.2 for information on the levels of capability for MIL-STD-1553 channels.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

None.

SEE ALSO

`BTI1553_ChIs1553, BTI1553_ChIsA, BTI1553_ChIsB32, BTI1553_ChIsB4, BTI1553_ChIsBM, BTI1553_ChIsC, BTI1553_ChIsD, BTI1553_ChIsM, BTI1553_ChIsP, BTI1553_ChIsS`

ChIsS

```
BOOL BTI1553_ChIsS
(
   INT channel,                 //Channel number to test
   HCORE hCore                  //Core handle
)
```

RETURNS

TRUE if the channel is a level S channel, otherwise FALSE.

DESCRIPTION

Checks to see if the channel number specified by `channel` has level S (Single-Function) capabilities. See Table A.2 for information on the levels of capability for MIL-STD-1553 channels.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

None.

SEE ALSO

`BTI1553_ChIs1553, BTI1553_ChIsA, BTI1553_ChIsB32, BTI1553_ChIsB4, BTI1553_ChIsBM, BTI1553_ChIsC, BTI1553_ChIsD, BTI1553_ChIsM, BTI1553_ChIsP`

ChStart

```
BOOL BTI1553_ChStart
(
   INT channel,                  //Channel number
   HCORE hCore                   //Core handle
)
```

RETURNS

TRUE if the channel was previously enabled, otherwise FALSE.

DESCRIPTION

Enables the operation of the channel specified by `channel`. If there is a BC Schedule, it restarts at the beginning. The channel remains enabled until `BTI-1553_ChStop` is called or a HALT Command Block is encountered in the BC Schedule. If the core is stopped, then channel activity begins when the core is started with `BTICard_CardStart`.

`BTI1553_ChStart` and `BTI1553_ChStop` enable and disable a channel. These functions allow the channel to be stopped and reconfigured with different settings while other channels are running.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

Do not confuse this channel enable with either Bus Controller pause or the core-level controls. Bus Controller pause is controlled by `BTI1553_BCConfig`, `BTI1553_BCPause`, and `BTI1553_BCResume`. Core-level controls are activated through `BTICard_CardStart`, `BTICard_CardStop`, and `BTI-Card_CardResume`.

SEE ALSO

`BTI1553_ChStop`, `BTICard_CardStart`, `BTICard_CardStop`

ChStop

```
BOOL BTI1553_ChStop
(
   INT channel,                  //Channel number
   HCORE hCore                   //Core handle
)
```

RETURNS

TRUE if the channel was previously enabled, otherwise FALSE.

DESCRIPTION

Disables operation of the channel specified by `channel`. If a message is being sent or received, the processing is allowed to finish before the channel is halted. Use `BTI1553_ChStart` to re-enable the channel.

`BTI1553_ChStart` and `BTI1553_ChStop` enable and disable a channel. These functions allow the channel to be stopped and reconfigured with different settings while other channels are running.

Note: A transmit channel can also be stopped when the Bus Controller encounters a `HALT` Command Block in the BC Schedule.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

Do not confuse this channel enable with either Bus Controller pause or the core-level controls. Bus Controller pause is controlled by `BTI1553_BCConfig`, `BTI1553_BCPause`, and `BTI1553_BCResume`. Core-level controls are activated through `BTICard_CardStart`, `BTICard_CardStop`, and `BTICard_CardResume`.

SEE ALSO

`BTI1553_ChStart`, `BTICard_CardStart`, `BTICard_CardStop`

CmdMaxLoopRd

```
USHORT BTI1553_CmdMaxLoopRd
(
  INT channel,            //Channel number
  HCORE hCore             //Core handle
)
```

RETURNS

The maximum loop count for the BC Schedule.

DESCRIPTION

Reads the maximum loop count for the BC Schedule. When enabled, the BC Schedule will automatically stop when it reaches the maximum loop count. A schedule loop counter is incremented for each `RESTART` opcode executed. Maximum loop counting is enabled via `BTI1553_BCConfig` and the `BCCFG1553_LOOPMAX` flag.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

None.

SEE ALSO

`BTI1553_BCConfig`, `BTI1553_BCSchedRestart`, `BTI1553_CmdMaxLoopWr`

CmdMaxLoopWr

```
VOID BTI1553_CmdMaxLoopWr
(
   USHORT countval,          //Maximum BC loop count
   INT channel,              //Channel number
   HCORE hCore               //Core handle
)
```

RETURNS

None.

DESCRIPTION

Writes the maximum loop count for the BC Schedule. When enabled, the BC Schedule will automatically stop when it reaches the maximum loop count. A schedule loop counter is incremented for each `RESTART` opcode executed. Maximum loop counting is enabled via `BTI1553_BCConfig` and the `BCCFG1553_LOOPMAX` flag.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

None.

SEE ALSO

`BTI1553_BCConfig`, `BTI1553_BCSchedRestart`, `BTI1553_CmdMaxLoopRd`

CmdShotRd

```
BOOL BTI1553_CmdShotRd
(
  SCHNDX index,          //Schedule index of item to read
  INT channel,           //Channel number
  HCORE hCore            //Core handle
)
```

RETURNS

TRUE if the single-shot bit is set, otherwise FALSE if not set.

DESCRIPTION

Reads the value of the single-shot bit for the BC Schedule opcode specified by `index`.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

Some BC Scheduling functions insert more than one opcode into the schedule. Therefore, it may be necessary to call this function on multiple schedule index values to get the desired effect.

SEE ALSO

`BTI1553_CmdShotWr, BTI1553_CmdSkipWr`

CmdShotWr

```
ERRVAL BTI1553_CmdShotWr
(
   BOOL value,              //Value of single-shot bit
   SCHNDX index,            //Schedule index of item to single-shot
   INT channel,             //Channel number
   HCORE hCore              //Core handle
)
```

RETURNS

A negative value if an error occurs, or zero if successful.

DESCRIPTION

Sets the single-shot bit to `value` for the schedule entry specified by `index`. When set to TRUE, the single-shot bit instructs the BC Schedule to process the specified opcode one time, and then to set the skip bit after processing is complete. The single-shot bit is FALSE (disabled) by default.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

Some BC Scheduling functions insert more than one opcode into the schedule. Therefore, it may be necessary to call this function on multiple schedule index values to get the desired effect.

SEE ALSO

`BTI1553_CmdShotRd, BTI1553_CmdSkipWr`

CmdSkipRd

```
BOOL BTI1553_CmdSkipRd
(
  SCHNDX index,                //Schedule index of item to read
  INT channel,                 //Channel number
  HCORE hCore                  //Core handle
)
```

RETURNS

TRUE if the skip bit is set, otherwise FALSE if not set.

DESCRIPTION

Reads the value of the skip bit for the BC Schedule opcode specified by `index`.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

Some BC Scheduling functions insert more than one opcode into the schedule. Therefore, it may be necessary to call this function on multiple schedule index values to get the desired effect.

SEE ALSO

`BTI1553_CmdSkipWr`, `BTI1553_CmdShotWr`

CmdSkipWr

```
ERRVAL BTI1553_CmdSkipWr
(
   BOOL value,              //Value of skip bit
   SCHNDX index,            //Schedule index of item to skip
   INT channel,             //Channel number
   HCORE hCore              //Core handle
)
```

RETURNS

A negative value if an error occurs or zero if successful.

DESCRIPTION

Sets the skip bit to `value` for the schedule entry specified by `index`. When set to TRUE, the skip bit instructs the BC Schedule to skip over processing the specified opcode. The skip bit is FALSE (disabled) by default.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

Some BC Scheduling functions insert more than one opcode into the schedule. Therefore, it may be necessary to call this function on multiple schedule index values to get the desired effect.

SEE ALSO

`BTI1553_CmdShotWr`, `BTI1553_CmdSkipRd`

CmdStepRd

```
BOOL BTI1553_CmdStepRd
(
    SCHNDX index,              //Schedule index of item to read
    INT channel,               //Channel number
    HCORE hCore                //Core handle
)
```

RETURNS

TRUE if the step bit is set, otherwise FALSE if not set.

DESCRIPTION

Reads the value of the step bit for the BC Schedule opcode specified by `index`.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

Some BC Scheduling functions insert more than one opcode into the schedule. Therefore, it may be necessary to call this function on multiple schedule index values to get the desired effect.

The user is especially cautioned to consider the effects of stepping through opcodes that depend upon or generate hardware timing.

SEE ALSO

`BTI1553_BCConfig, BTI1553_BCResume, BTI1553_CmdStepWr`

CmdStepWr

```
ERRVAL BTI1553_CmdStepWr
(
   BOOL value,                   //Value of step bit
   SCHNDX index,                 //Schedule index of item to step
   INT channel,                  //Channel number
   HCORE hCore                   //Core handle
)
```

RETURNS

A negative value if an error occurs or zero if successful.

DESCRIPTION

Sets the step bit to `value` for the schedule entry specified by `index`. When set to TRUE, the step bit instructs the BC Schedule to pause after processing the specified opcode. The step bit is FALSE (disabled) by default. To resume operation of the schedule, call `BTI1553_BCResume`.

Note that the entire BC Schedule can be configured for single-step operation by using the `BCCFG1553_STEP` flag when calling `BTI1553_BCConfig`.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

Some BC Scheduling functions insert more than one opcode into the schedule. Therefore, it may be necessary to call this function on multiple schedule index values to get the desired effect.

The user is especially cautioned to consider the effects of stepping through opcodes that depend upon or generate hardware timing.

SEE ALSO

`BTI1553_BCConfig`, `BTI1553_BCResume`, `BTI1553_CmdStepRd`

ErrorCtrl

```
ERRVAL BTI1553_ErrorCtrl
(
  ULONG ctrlflags,          //Select control options
  INT channel,              //Channel number
  HCORE hCore               //Core handle
)
```

RETURNS

A negative value if an error occurs or zero if successful.

DESCRIPTION

Enables error injection after an error has been defined. The `ctrlflags` control both how errors are triggered and whether all or only tagged messages can contain the error.

Typically, either `ERRCTRL1553_ON`, `ERRCTRL1553_ONCE`, or `ERRCTRL1553_EXTERNAL` is bitwise OR-ed with either `ERRCTRL1553_TAGMSG` or `ERRCTRL1553_ANYMSG` to produce the *`ctrlflags`* value.

Ctrlflags	
Constant	**Description**
ERRCTRL1553_OFF	Completely disables error injection
ERRCTRL1553_ON	Errors are injected in all candidate messages until error injection is disabled.
ERRCTRL1553_ONCE	An error is injected only in the next candidate message.
ERRCTRL1553_EXTERNAL	An error is injected in the next candidate message following an external trigger pulse.
ERRCTRL1553_TAGMSG	Only messages which have been tagged (with BTI1553_ErrorTag?? Or the MSGCRT1553_ERR flag when creating a message) are candidates
ERRCTRL1553_ANYMSG	All messages are candidates

DEVICE DEPENDENCY

Applies only to channels that support Protocol Error Injection (see Table A.2).

WARNINGS

The `ERRCTRL1553_ANYMSG` flag should be used with extra care, especially when multiple terminals are being simulated. When `ERRCTRL1553_ANYMSG` is specified, any transmission from any simulated terminal (BC or RT) may contain the error.

SEE ALSO

`BTI1553_ErrorTagBC`, `BTI1553_ErrorTagRT`, `BTI1553_ErrorDefine`, `BTI1553_??CreateMsg`

ErrorDefine

```
ERRVAL BTI1553_ErrorDefine
(
   ULONG ctrlflags,          //Selects the type of error
   USHORT value,             //See below
   INT count,                //See below
   USHORT wordpos,           //See below
   USHORT bitpos,            //See below
   INT channel,              //Channel number
   HCORE hCore               //Core handle
)
```

RETURNS

A negative value if an error occurs or zero if successful.

DESCRIPTION

Defines the error to be injected into any subsequent message transmission. Note that both protocol error injection functions and zero crossing distortion only apply to certain channel levels (see Table A.2). `BTI1553_ErrorDefine` may be called during run time to redefine the error.

Since only one error may be defined at a time, the `ctrlflags` constants in the following table may NOT be bitwise OR-ed together.

ctrlflags	
Constant	**Description**
ERRDEF1553_NONE	Disable error generation (default)
ERRDEF1553_GAP	Generate a gap error preceding a word
ERRDEF1553_CNTWRDABS	Generate an absolute word count error
ERRDEF1553_CNTWRDREL	Generate a word count error interpreting count as relative to the word count in the command word
ERRDEF1553_CNTBIT	Generate a bit count error
ERRDEF1553_MAN1	Generate Manchester error on first half of a specified bit
ERRDEF1553_MAN2	Generate Manchester error on second half of a specified bit
ERRDEF1553_SYNC	Generate a sync pulse error (invert the sync)
ERRDEF1553_PAR	Generate a parity error (invert the parity)
ERRDEF1553_ZEROX1	Generate zero crossing distortion on the leading zero-crossing (if exists)
ERRDEF1553_ZEROX2	Generate zero crossing distortion on the mid-bit zero-crossing

For any given type of error, some of the parameters of this function are unused. The following table shows which parameters apply to each error type.

PARAMETERS USED				
ctrlflags	**value**	**count**	**wordpos**	**bitpos**
ERRDEF1553_PAR			✓	
ERRDEF1553_SYNC			✓	
ERRDEF1553_MAN1			✓	✓
ERRDEF1553_MAN2			✓	✓
ERRDEF1553_CNTWRDABS		✓		
ERRDEF1553_CNTWRDREL		✓		
ERRDEF1553_CNTBIT	✓	✓	✓	
ERRDEF1553_GAP	✓		✓	
ERRDEF1553_ZEROX1		✓	✓	✓
ERRDEF1553_ZEROX2	✓	✓	✓	

Value

Specifies a time for gap errors or the values of any extra bits in a bit count error. The `value` may be 0–8191 for gap errors, and the units of time are 0.1 μs. The least significant bits of `value` determine the values of extra bits in a bit count error.

Count

Specifies the size of a bit or word count error. For bit count errors, `count` may be in the range of –2 to +3. For absolute word count errors, `count` may be from 0 to 40. For relative word count errors, the sum of `count` and the word count field in the command word must be less than or equal to 40.

For zero-crossing distortion, `count` specifies the amount in nanoseconds that the edge is shifted. A positive value delays the edge (moves it to the right on an oscilloscope) and a negative value advances it (moves it to the left on an oscilloscope).

Wordpos

Specifies the location within a message of a word error. The `wordpos` may be a value from 0 to 31 indicating a data word, or it may be one of the constants from the following table.

Wordpos **PARAMETER**	
Constant	**Description**
ERRDEF1553_CWD1	Specifies the first command word
ERRDEF1553_CWD2	Specifies the second command word
ERRDEF1553_SWD1	Specifies the first status word
ERRDEF1553_SWD2	Specifies the second status word

Bitpos

The `bitpos` specifies the bit containing a Manchester or zero crossing error. It may range from 0 to 19.

DEVICE DEPENDENCY

Both protocol error injection functions and zero crossing distortion only applies to certain channel levels (see Table A.2). Consult the hardware manual for your Device to determine the appropriate value to pass for `count`.

WARNINGS

None.

SEE ALSO

`BTI1553_ErrorCtrl`, `BTI1553_ErrorSent`, `BTI1553_ErrorTriggerDefine`

ErrorSent

```
BOOL BTI1553_ErrorSent
(
   INT channel,                //Channel number
   HCORE hCore                 //Core handle
)
```

RETURNS

TRUE if an error has been sent since the last call to `BTI1553_ErrorCtrl`, otherwise FALSE.

DESCRIPTION

Checks an internal flag that indicates whether at least one message with an error has been transmitted. This flag is set when an error is sent. It remains set until cleared by `BTI1553_ErrorCtrl`. This function is used to verify that the currently defined error has been sent before redefining it.

DEVICE DEPENDENCY

Applies only to channels that support Protocol Error Injection (see Table A.2).

WARNINGS

This function cannot detect the occurrence of errors triggered by the external trigger signal.

SEE ALSO

`BTI1553_ErrorCtrl`, `BTI1553_ErrorDefine`, `BTI1553_ErrorTriggerDefine`

ErrorTagBC

```
ERRVAL BTI1553_ErrorTagBC
(
   BOOL tagval,                //Tag if TRUE, untag if FALSE
   MSGADDR message,            //Message to receive error
   INT channel,                //Channel number
   HCORE hCore                 //Core handle
)
```

RETURNS

A negative value if an error occurs or zero if successful.

DESCRIPTION

Makes the specified message a candidate to contain errors when error injection is enabled.

DEVICE DEPENDENCY

Applies only to channels that support Protocol Error Injection (see Table A.2).

WARNINGS

None.

SEE ALSO

BTI1553_ErrorTagRT, BTI1553_ErrorDefine, BTI1553_BCCreateMsg, BTI1553_ErrorTriggerDefine

ErrorTagRT

```
ERRVAL BTI1553_ErrorTagRT
(
   BOOL tagval,              //Tag if TRUE, untag if FALSE
   BOOL mcflag,              //TRUE if mode code, FALSE if subaddress
   INT taval,                //Terminal address
   BOOL trval,               //T/R bit
   INT saval,                //Subaddress or mode-code number
   INT channel,              //Channel number
   HCORE hCore               //Core handle
)
```

RETURNS

A negative value if an error occurs or zero if successful.

DESCRIPTION

Makes the specified message a candidate to contain errors when error injection is enabled.

DEVICE DEPENDENCY

Applies only to channels that support Protocol Error Injection (see Table A.2).

WARNINGS

None.

SEE ALSO

`BTI1553_ErrorTagBC`, `BTI1553_ErrorDefine`, `BTI1553_ErrorTriggerDefine`, `BTI1553_RTCreateMsg`

ErrorTriggerDefine

```
ERRVAL BTI1553_ErrorTriggerDefine
(
   BOOL enable,              //Enable/disable external trigger
   USHORT trigmask,          //Line(s) used for trigger signal
   USHORT trigval,           //Active/inactive condition for trigger line(s)
   USHORT pinpolarity,       //Active pin polarity (high/low)
   INT channel,              //Channel number
   HCORE hCore               //Core handle
)
```

RETURNS

A negative value if an error occurs or zero if successful.

DESCRIPTION

Defines the trigger input settings for `channel` on `hCore` and associates error injection with a trigger signal. The input line(s) are specified by `trigmask` with an active trigger state being the combination of `trigval` and `pinpolarity`, as defined in the tables below. During run time, the trigger signal is latched when it goes active, and then it is cleared when the error is injected into the next candidate message. Consequently, if the trigger stays active after the error is injected, it is relatched and the subsequent candidate message will also contain an error. `BTI1553_ErrorTriggerDefine` may be called during run time to redefine the trigger input settings.

Before calling `BTI1553_ErrorTriggerDefine`, the error must be defined with `BTI1553_ErrorDefine` and the error injection mode must be set for external triggering by calling `BTI1553_ErrorCtrl` with the `_EXTERNAL` flag.

The constants in the tables below may be bitwise OR-ed together to configure multiple lines. Only one `trigmask` may be selected per trigger line (A, B, and C), and all combined states must be true for the trigger to occur. Triggers are both generation dependent and device dependent. Refer to Table A.3 and the following tables for more information.

trigmask				
Constant	**Description**	**4G**	**5G**	**6G**
TRIGMASK_TRIGA	Selects discrete trigger line A	✓	✓	✓
TRIGMASK_PXITRIGA	Selects PXIe TRIG trigger line A			*
TRIGMASK_PXISTARA	Selects PXIe STAR trigger line A			*
TRIGMASK_TRIGB	Selects discrete trigger line B	✓	✓	✓
TRIGMASK_PXITRIGB	Selects PXIe TRIG trigger line B			*
TRIGMASK_PXISTARB	Selects PXIe STAR trigger line B			*
TRIGMASK_TRIGC	Selects discrete trigger line C	✓	✓	✓
TRIGMASK_PXITRIGC	Selects PXIe TRIG trigger line C			*
TRIGMASK_PXISTARC	Selects PXIe STAR trigger line C			*

** PXIe Syncs are only available on PXIe Devices. Refer to `BTICard_CardTriggerValid` for device dependent support*

trigval	
Constant	**Description**
TRIGVAL_TRIGAOFF	Trigger on line A inactive
TRIGVAL_TRIGAON	Trigger on line A active
TRIGVAL_TRIGBOFF	Trigger on line B inactive
TRIGVAL_TRIGBON	Trigger on line B active
TRIGVAL_TRIGCOFF	Trigger on line C inactive
TRIGVAL_TRIGCON	Trigger on line C active

pinpolarity				
Constant	**Description**	**4G**	**5G**	**6G**
TRIGPOL_TRIGAL	Sets active low polarity for trigger line A	✓	✓	✓
TRIGPOL_TRIGAH	Sets active high polarity for trigger line A	✓	✓	✓
TRIGPOL_TRIGAF	Sets active on falling edge of trigger line A			✓
TRIGPOL_TRIGAR	Sets active on rising edge of trigger line A			✓
TRIGPOL_TRIGBL	Sets active low polarity for trigger line B	✓	✓	✓
TRIGPOL_TRIGBH	Sets active high polarity for trigger line B	✓	✓	✓
TRIGPOL_TRIGBF	Sets active on falling edge of trigger line B			✓
TRIGPOL_TRIGBR	Sets active on rising edge of trigger line B			✓
TRIGPOL_TRIGCL	Sets active low polarity for trigger line C	✓	✓	✓
TRIGPOL_TRIGCH	Sets active high polarity for trigger line C	✓	✓	✓
TRIGPOL_TRIGCF	Sets active on falling edge of trigger line C			✓
TRIGPOL_TRIGCR	Sets active on rising edge of trigger line C			✓

Additionally, to associate messages to a trigger signal, use `BTI1553_BCTriggerDefine` (all messages) or `BTI1553_MsgTriggerDefine` (selected messages).

DEVICE DEPENDENCY

On 4G, 5G, and 6G Devices, which have multiple external trigger lines, this function is required to inject errors in response to an external trigger signal. For 3G Devices, with a single trigger line, use only `BTI1553_ErrorDefine` and `BTI1553_ErrorCtrl` with the `ERRCTRL1553_EXTERNAL` flag. The mapping of trigger lines is hardware dependent. Please consult the hardware manual for the Device.

WARNINGS

None.

SEE ALSO

`BTI1553_ErrorDefine`, `BTI1553_ErrorCtrl`, `BTICard_CardTrigger`, `BTICard_CardTriggerEx`, `BTICard_CardTriggerValid`

ListAddr

```
MSGADDR BTI1553_ListAddr
(
   INT index,          //Index of entry in List Buffer
   LISTADDR list,      //List buffer from which to get the message address
   HCORE hCore         //Core handle
)
```

RETURNS

The message address of a message contained in the List Buffer.

DESCRIPTION

Returns the message address of the message contained in the List Buffer specified by `list`. The specific message within the list is determined by `index`.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

None.

SEE ALSO

`BTI1553_BCCreateList`, `BTI1553_RTCreateList`, `BTI1553_MsgBlockRd`, `BTI1553_MsgCommRd`, `BTI1553_MsgBlockWr`

ListBlockRd

```
MSGADDR BTI1553_ListBlockRd
(
  LPUSHORT buf,        //Pointer to destination
  LISTADDR list,       //List buffer from which to read
  HCORE hCore          //Core handle
)
```

RETURNS

The address within the list of the message structure that was read, or zero if an error occurred or unable to read from the list.

DESCRIPTION

Reads the next message structure in the List Buffer specified by `list.` Similar to `BTI1553_MsgBlockRd` except it reads from a List Buffer. This function copies the `MSGFIELDS1553` formatted message structure to `buf.` The parameter `list` is the value returned when the List Buffer was created using `BTI1553_BCCreateList` or `BTI1553_RTCreateList.` See `BTI1553_MsgBlockRd` for a table of message structure fields.

The position of the message to be read is determined by the mode of the List Buffer as follows:

Circular mode	: Not valid for this function.
FIFO mode	: Reads the oldest complete messages received.
Ping-Pong mode	: Reads the newest complete message received.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

The `list` parameter must be configured as a read (receive) List Buffer.

SEE ALSO

`BTI1553_ListBlockWr`, `BTI1553_ListMultiBlockRd`, `BTI1553_MsgBlockRd`, `BTI1553_MsgCommRd`

ListBlockWr

```
MSGADDR BTI1553_ListBlockWr
(
   LPUSHORT buf,       //Pointer to source message structure
   LISTADDR list,      //List buffer to write to
   HCORE hCore         //Core handle
)
```

RETURNS

The address within the List Buffer of the message structure that was written to, or zero if an error occurred or unable to write to the list.

DESCRIPTION

Writes to the next message structure associated with the List Buffer specified by *list*. Similar to BTI1553_MsgBlockWr except it writes to a List Buffer. This function copies *buf* to the MSGFIELDS1553 message structure in the List Buffer. The parameter *list* is the value returned when the List Buffer was created using BTI1553_BCCreateList or BTI1553_RTCreateList.

The position to which the message is written in the List Buffer is determined by the mode of the list as follows:

Circular mode: If the number of writes exceeds the number of entries in the list, the message structure at the top of the list is overwritten.

FIFO mode: The message structure is written to one end of the list and is transmitted and removed from the other end of the list. This function returns zero if the list is full.

Ping-Pong mode: When writing is complete, the message structure will be used for the next message transmission.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

The *list* parameter must be configured as a write (transmit) List Buffer.

SEE ALSO

BTI1553_ListBlockRd, BTI1553_ListMultiBlockWr, BTI1553_MsgBlockWr

ListDataRd

```
INT BTI1553_ListDataRd
(
  LPUSHORT buf,             //Pointer to destination
  INT count,                //One more than the number of words to read
  LISTADDR list,            //List buffer from which to read data
  HCORE hCore               //Core handle
)
```

RETURNS

The number of data words read from the List Buffer, or zero if an error occurred or unable to read from the list.

DESCRIPTION

Reads the next data associated with a List Buffer. Similar to `BTI1553_MsgDataRd` except `BTI1553_ListDataRd` reads from a List Buffer. This function copies *`count`* number of data words to *`buf`* from the message structure in the List Buffer specified by the *`list`* parameter. The *`list`* parameter is the value returned when the List Buffer was created using `BTI1553_BCCreateList` or `BTI1553_RTCreateList`.

The position of the message to be read is determined by the mode of the List Buffer as follows:

Circular mode: Not valid for this function.

FIFO mode: Reads the oldest complete message received.

Ping-Pong mode: Reads the newest complete message received.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

The *`list`* parameter must be configured as a read (receive) List Buffer. The maximum number of entries that may be read from the list per read is limited to the smaller of the *`buf`* size or the size of the list (the value of `count-1` used in the CreateList function).

SEE ALSO

`BTI1553_MsgDataRd`, `BTI1553_ListDataWr`,
`BTI1553_BCCreateList`, `BTI1553_RTCreateList`

ListDataWr

```
INT BTI1553_ListDataWr
(
   LPUSHORT buf,            //Pointer to data words
   INT count,               //One more than the number of words to write
   LISTADDR list,           //List buffer to write new data
   HCORE hCore              //Core handle
)
```

RETURNS

The number of data words written to the List Buffer or zero if an error occurred or unable to write to the list.

DESCRIPTION

Writes the next data associated with a List Buffer. Similar to `BTI1553_MsgDataWr` except `BTI1553_ListDataWr` writes to a List Buffer. This function copies *`count`* data words from *`buf`* to the message structure in the List Buffer specified by the *`list`* parameter. The *`list`* parameter is the value returned when the List Buffer was created using `BTI1553_BCCreateList` or `BTI1553_RTCreateList`.

The position to which the message is written in the List Buffer is determined by the mode of the list as follows:

Circular mode: If the number of writes exceeds the number of entries in the list, data is overwritten starting at the top of the list.

FIFO mode: Data is written to one end of the list and is transmitted and removed from the other end of list. This function returns a zero if the list is full.

Ping-Pong mode: When writing is complete, the data will be used for the next message transmission.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

The *`list`* parameter must be configured as a write (transmit) List Buffer. The maximum number of entries that may be written to the list per write is limited to the smaller of the *`buf`* size or the remaining size of the list (the value of `count-1` used in the `CreateList` function).

SEE ALSO

`BTI1553_MsgDataWr`, `BTI1553_ListDataRd`, `BTI1553_BCCreateList`, `BTI1553_RTCreateList`

ListMultiBlockRd

```
BOOL BTI1553_ListMultiBlockRd
(
  LPUSHORT buf,        //Pointer to destination
  LPINT blkcount,      //Number of blocks in buf, number of blocks to read
  LISTADDR list,       //List buffer from which to read
  HCORE hCore          //Core handle
)
```

RETURNS

A non-zero value if the function succeeded, or zero if unable to read the List Buffer.

DESCRIPTION

Reads multiple message structures from a List Buffer. Similar to `BTI1553_ListBlockRd` except it reads multiple message structures from the List Buffer instead of just one. This function copies the `MSGFIELDS1553` formatted message structures to *`buf`*. The *`list`* parameter is the value returned when the List Buffer was created using `BTI1553_BCCreateList` or `BTI1553_RTCreateList`.

When called, *`blkcount`* must point to a variable that contains the maximum number of message structures that *`buf`* can hold. On a successful return, *`blkcount`* will contain the number of message structures written to *`buf`*.

The position of the messages to read is determined by the mode of the List Buffer as follows:

Circular mode: Not valid for this function.

FIFO mode: Reads the oldest complete messages received.

Ping-Pong mode: Not valid for this function.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

The *`list`* parameter must be configured as a read (receive) List Buffer.

SEE ALSO

`BTI1553_ListBlockWr`, `BTI1553_ListMultiBlockWr`, `BTI1553_MsgBlockRd`

ListMultiBlockWr

```
BOOL BTI1553_ListMultiBlockWr
(
   LPUSHORT buf,        //Pointer to source
   INT blkcount,        //Count of messages to write
   LISTADDR list,       //List buffer to write to
   HCORE hCore          //Core handle
)
```

RETURNS

A non-zero value if the function succeeded, or zero if unable to read the List Buffer.

DESCRIPTION

Writes multiple message structures to a List Buffer. Similar to `BTI1553_ListBlockWr` except it writes multiple message structures to the List Buffer instead of just one. This function copies `blkcount` `MSGFIELDS1553` formatted message structures from `buf` to the list buffer. The parameter *`list`* is the value returned when the list was created using `BTI1553_BCCreateList` or `BTI1553_RTCreateList`.

The position to which the messages are written in the List Buffer is determined by the mode of the list as follows:

Circular mode: Not valid for this function.

FIFO mode: The message structures are written to one end of the list and transmitted and removed from the other end of the list. This function returns zero if the list is full.

Ping-Pong mode: Not valid for this function.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

The `list` parameter must be configured as a write (transmit) List Buffer.

SEE ALSO

`BTI1553_ListBlockRd, BTI1553_ListMultiBlockWr, BTI1553_MsgBlockWr`

ListStatus

```
INT BTI1553_ListStatus
(
  LISTADDR list,          //List buffer address
  HCORE hCore             //Core handle
)
```

RETURNS

The status value of the List buffer.

DESCRIPTION

Checks the status of the List buffer `list`, without removing an entry. The status value can be tested using the predefined constants below:

Constant	Description
`STAT_EMPTY`	List is empty
`STAT_PARTIAL`	List is partially filled
`STAT_FULL`	List is full
`STAT_OFF`	List is off

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

When the buffer is full it wraps around and overwrites previous entries.

SEE ALSO

`BTI1553_ListDataRd, BTI1553_ListDataWr`

MonConfig

```
ERRVAL BTI1553_MonConfig
(
  ULONG ctrlflags,          //Selects configuration options
  INT channel,              //Channel number
  HCORE hCore               //Core handle
)
```

RETURNS

A negative value if an error occurs or zero if successful.

DESCRIPTION

Creates a 1553 Bus Monitor for the specified channel and configures it with the options defined by `ctrlflags`. The Bus Monitor captures 1553 messages and feeds them to the Sequential Monitor, which must be configured separately using `BTICard_SeqConfig`.

ctrlflags	
Constant	**Description**
MONCFG1553_DEFAULT	Select all default settings (**bold** below)
MONCFG1553_BCAST	Enable broadcast reception
MONCFG1553_NOBCAST	Disables broadcast
MONCFG1553_ENABLE	Enable the Bus Monitor
MONCFG1553_DISABLE	Disable the Bus Monitor
MONCFG1553_MC01	Select SA=00000 or SA=11111 for mode codes
MONCFG1553_MC1	Select SA=11111 for mode codes
MONCFG1553_MC0	Select SA=00000 for mode codes
MONCFG1553_MCNONE	Disable mode codes
MONCFG1553_TERMOFF	Direct coupled termination resistance off for Bus A and B
MONCFG1553_TERMONA	Direct coupled termination resistance on for Bus A
MONCFG1553_TERMONB	Direct coupled termination resistance on for Bus B
MONCFG1553_TERMONAB	Direct coupled termination resistance on for Bus A and B

DEVICE DEPENDENCY

Termination flags are only supported on 4G Devices.

WARNINGS

`BTICard_SeqConfig` must also be called to configure the Sequential Monitor.

If the Device is configured to monitor with broadcast (*MONCFG1553_BCAST*) enabled and mode codes are enabled (*MONCFG1553_MC01*, *MONCFG1553_MC1*, or *MONCFG1553_MC0*) and RT's are not configured, then the side effect of receiving a Synchronize Mode Code (transmit receive bit = 1, mode code number = 1) resets the Device's timer. See the `RTConfig` function description for additional information.

SEE ALSO

`BTI1553_MonFilterTA`, `BTI1553_MonFilterSA`, `BTICard_SeqConfig`

MonFilterSA

```
ERRVAL BTI1553_MonFilterSA
(
  INT ta,                  //Terminal address of filtered messages
  ULONG rsa,               //Bitmask of receive subaddresses
  ULONG tsa,               //Bitmask of transmit subaddresses
  ULONG rmc,               //Bitmask of receive mode codes
  ULONG tmc,               //Bitmask of transmit mode codes
  INT channel,             //Channel number
  HCORE hCore              //Core handle
)
```

RETURNS

A negative value if an error occurs or zero if successful.

DESCRIPTION

Enables and configures subaddress and mode code filtering on the Bus Monitor. Through bitmask parameters, this function specifies which subaddress and mode code messages to monitor for the terminal address specified by `ta`. If this function is not used, all messages to the terminal address are monitored.

In each bitmask parameter, the least significant bit corresponds to subaddress or mode code number zero. The most significant bit corresponds to subaddress or mode code number 31.

Note: Subaddress filtering applies to all channel levels, except level A (see Table A.2).

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

`BTI1553_MonConfig` must be called prior to using this function. `BTI1553_MonFilterTA` must be called prior to using this function.

SEE ALSO

`BTI1553_MonFilterTA, BTI1553_MonConfig`

MonFilterTA

```
ERRVAL BTI1553_MonFilterTA
(
  ULONG termaddresses,       //Bitmask specifying Tas to monitor
  INT channel,               //Channel number
  HCORE hCore                //Core handle
)
```

RETURNS

A negative value if an error occurs or zero if successful.

DESCRIPTION

Enables terminal address filtering on the Bus Monitor previously enabled with `BTI1553_MonConfig`. The bitmask `termaddresses` specifies which terminal addresses to monitor. If this function is not used, all terminal addresses are monitored.

The least significant bit in `termaddresses` corresponds to terminal address 0. The most significant bit corresponds to terminal address 31.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

`BTI1553_MonConfig` must be called prior to using this function.

SEE ALSO

`BTI1553_MonFilterSA`, `BTI1553_MonConfig`

MsgBlockRd

```
MSGADDR BTI1553_MsgBlockRd
(
   LPMSGFIELDS1553 msgfields,   //Pointer to destination structure
   MSGADDR message,             //Message from which to read
   HCORE hCore                  //Core handle
)
```

RETURNS

The address of the message structure that was read.

DESCRIPTION

Reads an entire message structure from the core.

MSGFIELDS1553 Structure		
Field	**Size**	**Description**
msgflag	USHORT	BTIDriver writes message flags (do not modify this field)
msgopt	USHORT	BTIDriver writes message options (do not modify this field)
msgerr	USHORT	Error field. Errors associated with this message (see table below)
msgact	USHORT	Activity field. Bits denoting the bus activity associated with this message (see table on the following page)
resptime1	USHORT	Response time + 3.0 µs (in units of 0.1 µs) of status word 1
resptime2	USHORT	Response time + 3.0 µs (in units of 0.1 µs) of status word 2 (RT-RT only)
datacount	USHORT	Data count field. Number of valid data words associated with this message
timetag	ULONG	Lower 32 bits of the time-tag value
hitcount	ULONG	Hit counter value. Used instead of time-tag when in hit counter mode
elapsetime	ULONG	Elapse time field. Value of elapsed time
mintime	ULONG	Minimum time. Used instead of elapsed time when in min time mode
maxtime	ULONG	Maximum time. Used instead of elapsed time when in max time mode
timetagh	ULONG	Upper 32 bits of the time-tag value
cwd1	USHORT	Command word 1
cwd2	USHORT	Command word 2 (RT-RT only)
swd1	USHORT	Status word 1
swd2	USHORT	Status word 2 (RT-RT only)
data[32]	USHORT	Data words
extra[8]	USHORT	Additional data when generating a word count error

The error and activity fields always contain the result of the last transmission or reception of the message. These fields may be tested by AND-ing the values returned with constants from the following tables:

Error field	
Constant	**Description**
MSGERR1553_NORESP	No response was received from the RT
MSGERR1553_ANYERR	Set if any other error bits are set
MSGERR1553_PROTOCOL	A protocol error occurred
MSGERR1553_SYNC	Wrong polarity of the sync pulse
MSGERR1553_DATACOUNT	Too many/too few data words
MSGERR1553_MANCH	Manchester error
MSGERR1553_PARITY	Parity error
MSGERR1553_WORD	Reserved
MSGERR1553_RETRY	All attempts to retry transmission of this message failed (BC only)
MSGERR1553_SYSTEM	Internal Device error
MSGERR1553_HIT	Indicates that this message was transmitted or received since this bit was last cleared

Activity field	
Constant	**Description**
MSGACT1553_CHMASK	The channel number mask value. Shift the result right with MSGACT1553_CHSHIFT.
MSGACT1553_CHSHIFT	Channel number shift value. See CHMASK above.
MSGACT1553_XMTCWD1	The first command word was transmitted.
MSGACT1553_XMTCWD2	The second command word was transmitted.
MSGACT1553_XMTSWD1	The first status word was transmitted.
MSGACT1553_XMTSWD2	The second status word was transmitted.
MSGACT1553_RCVCWD1	The first command word was received.
MSGACT1553_RCVCWD2	The second command word was received.
MSGACT1553_RCVSWD1	The first status word was received.
MSGACT1553_RCVSWD2	The second status word was received.
MSGACT1553_XMTDWD	Data words were transmitted.
MSGACT1553_RCVDWD	Data words were received.
MSGACT1553_BUS	Message was transmitted/received on bus A (0) or B (1).

Extract the channel number from the activity word by AND-ing the activity field with MSGACT1553_CHMASK and right-shifting the result by MSGACT1553_CHSHIFT. The resulting value is the channel number associated with the MIL-STD-1553 record.

```
Channel = (activity & MSGACT1553_CHMASK) >> MSGACT1553_CHSHIFT;
```

DEVICE DEPENDENCY

When IRIG is enabled on a 4G Device, time-tags in message structures and Sequential Records will be in BCD format (see BTICard_TimerStatus).

WARNINGS

When data is read from the message the host accesses might contend with the onboard 1553 message processing of the same message. If this happens, then the data read from the message may contain partially updated data. To prevent contended accesses, either make sure to read from a message structure when that particular message is not being received on the 1553 bus (i.e. synchronize message accesses to just after they are received) or use BTI1553_MsgCommRd.

SEE ALSO

BTI1553_MsgDataRd, BTI1553_MsgFieldRd,
BTI1553_MsgBlockWr, BTI1553_MsgCommRd

MsgBlockWr

```
MSGADDR BTI1553_MsgBlockWr
(
  LPMSGFIELDS1553 msgfields,  //Pointer to source structure
  MSGADDR message,            //Message to receive new data
  HCORE hCore                 //Core handle
)
```

RETURNS

The address of the message structure that was written.

DESCRIPTION

Writes an entire message structure to the core. This function is used to modify certain fields in a message structure after it has been read using `BTI1553_MsgBlockRd`. The user can clear the hit bit (`msgerr`), time-tag, hit count, elapsetime, mintime, and maxtime fields and update the cwd1, cwd2, and data fields. All other fields should be restored to the value read. See `BTI1553_MsgBlockRd` for a table of message structure fields.

DEVICE DEPENDENCY

When IRIG is enabled on a 4G Device, time-tags in message structures and Sequential Records will be in BCD format (see `BTICard_TimerStatus`).

WARNINGS

Do not modify any fields other than those listed above.

When data is written to the message the host accesses might contend with the onboard 1553 message processing of the same message. If this happens, then the data transmitted on the 1553 bus may contain partially updated data. To prevent contended accesses, either make sure to write to a message structure when that particular message is not being transmitted on the 1553 bus (i.e. synchronize message accesses to just after they are transmitted) or use `BTI1553_MsgCommWr`.

SEE ALSO

`BTI1553_MsgDataWr`, `BTI1553_MsgFieldWr`,
`BTI1553_MsgBlockRd`, `BTI1553_MsgCommWr`

MsgCommRd

```
MSGADDR BTI1553_MsgCommRd
(
  LPMSGFIELDS1553 msgfields,   //Pointer to destination structure
  MSGADDR message,             //Message from which to read
  HCORE hCore                  //Core handle
)
```

RETURNS

The address of the message structure that was read.

DESCRIPTION

Reads an entire message structure from the core. Similar to `BTI1553_MsgBlockRd`, except it uses non-contended accesses of Device memory. See `BTI1553_MsgBlockRd` for a table of message structure fields.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

None.

SEE ALSO

`BTI1553_MsgDataRd`, `BTI1553_MsgBlockRd`, `BTI1553_MsgCommWr`

MsgCommWr

```
MSGADDR BTI1553_MsgBlockWr
(
   LPMSGFIELDS1553 msgfields,   //Pointer to source structure
   MSGADDR message,             //Message to receive new data
   HCORE hCore                  //Core handle
)
```

RETURNS

The address of the message structure that was written.

DESCRIPTION

Writes an entire message structure to the core. Similar to `BTI1553_MsgBlkWr`, except it uses non-contended accesses of Device memory. This function is used to modify certain fields in a message structure after it has been read using `BTI1553_MsgCommRd`. The user can clear the hit bit (`msgerr`), time-tag, hit count, elapsetime, mintime, and maxtime fields and update the `cwd1`, `cwd2`, and data fields. All other fields should be restored to the value read. See `BTI1553_MsgBlockRd` for a table of message structure fields.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

Do not modify any fields other than those listed above.

SEE ALSO

`BTI1553_MsgDataWr`, `BTI1553_MsgBlockWr`, `BTI1553_MsgCommRd`

MsgDataRd

```
VOID BTI1553_MsgDataRd
(
   LPUSHORT buf,                //Pointer to data words
   INT count,                   //Number of words to read
   MSGADDR message,             //Message from which to read
   HCORE hCore                  //Core handle
)
```

RETURNS

None.

DESCRIPTION

Reads the data associated with a message. This function copies `count` data words to `buf` from the message structure specified by `message`.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

When data is read from the message the host accesses might contend with the onboard 1553 message processing of the same message. If this happens, then the data read from the message may contain partially updated data. To prevent contended accesses, either make sure to read from a message structure when that particular message is not being received on the 1553 bus (i.e. synchronize message accesses to just after they are received) or use `BTI1553_MsgCommRd`.

SEE ALSO

`BTI1553_MsgBlockRd`, `BTI1553_MsgFieldRd`, `BTI1553_BCCreateMsg`, `BTI1553_MsgCommRd`, `BTI1553_RTCreateMsg`

MsgDataWr

```
VOID BTI1553_MsgDataWr
(
  LPUSHORT buf,          //Pointer to data words
  INT count,             //Number of words to write
  MSGADDR message,       //Message to receive new data
  HCORE hCore            //Core handle
)
```

RETURNS

None.

DESCRIPTION

Writes the data associated with a message. This function copies `count` data words from `buf` to the message structure specified by `message`.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

When data is written to the message the host accesses might contend with the onboard 1553 message processing of the same message. If this happens, then the data transmitted on the 1553 bus may contain partially updated data. To prevent contended accesses, either make sure to write to a message structure when that particular message is not being transmitted on the 1553 bus (i.e. synchronize message accesses to just after they are transmitted) or use `BTI1553_MsgCommWr`.

SEE ALSO

`BTI1553_BCCreateMsg`, `BTI1553_MsgBlockWr`, `BTI1553_MsgCommWr`, `BTI1553_MsgFieldWr`, `BTI1553_RTCreateMsg`

MsgFieldRd

```
ULONG BTI1553_MsgFieldRd
(
  USHORT fieldtype,          //Field to read
  MSGADDR message,           //Message to read
  HCORE hCore                //Core handle
)
```

RETURNS

The value of the specified field in the specified message structure.

DESCRIPTION

Reads the value of the field specified by `fieldtype` from the message structure at address `message`. It is typically used to read the status words and error fields in a BC message structure after message transmission.

fieldtype	
Constant	**Description**
FIELD1553_CWD1	Command word 1 field
FIELD1553_CWD2	Command word 2 field (RT-RT only)
FIELD1553_SWD1	Status word 1 field
FIELD1553_SWD2	Status word 2 field (RT-RT only)
FIELD1553_TTAG	32-bit time-tag field
FIELD1553_ELAPSE	Elapsed time field
FIELD1553_ERROR	Error field (see below)
FIELD1553_ACT	Activity field (see below)
FIELD1553_RESP1	Response time 1 field
FIELD1553_RESP2	Response time 2 field (RT-RT only)
FIELD1553_COUNT	Data count field
FIELD1553_TTAGH	Upper 32 bits of the 64-bit time-tag

All of the fields are numeric values except for `FIELD1553_ERROR` and `FIELD1553_ACT`, which are bit fields.

The constants for the `FIELD1553_ERROR` and `FIELD1553_ACT` fields are shown below. These fields always contain the results of the last transmission or reception of the message. They may be tested by AND-ing the values returned by `BTI1553_MsgFieldRd` with constants from the following tables.

FIELD1553_ERROR	
Constant	**Description**
MSGERR1553_HIT	Indicates that this message was transmitted or received since this bit was last cleared
MSGERR1553_SYSTEM	Internal Device error
MSGERR1553_RETRY	All attempts to retry transmission of this message failed (BC only)
MSGERR1553_WORD	Reserved
MSGERR1553_PARITY	Parity error
MSGERR1553_MANCH	Manchester error
MSGERR1553_DATACOUNT	Too many/too few data words
MSGERR1553_SYNC	Wrong polarity of the sync pulse
MSGERR1553_PROTOCOL	A protocol error occurred
MSGERR1553_ANYERR	Set if any other error bits are set
MSGERR1553_NORESP	No response was received from the RT

The core always sets bit 0 (MSGERR1553_HIT) in the FIELD1553_ERROR field after transmission or reception of the message. The user should explicitly clear this bit to detect subsequent transmissions/receptions of the message.

Activity field	
Constant	**Description**
MSGACT1553_CHMASK	The channel number mask value. Shift the result right with MSGACT1553_CHSHIFT.
MSGACT1553_CHSHIFT	Channel number shift value. See CHMASK above.
MSGACT1553_XMTCWD1	The first command word was transmitted.
MSGACT1553_XMTCWD2	The second command word was transmitted.
MSGACT1553_XMTSWD1	The first status word was transmitted.
MSGACT1553_XMTSWD2	The second status word was transmitted.
MSGACT1553_RCVCWD1	The first command word was received.
MSGACT1553_RCVCWD2	The second command word was received.
MSGACT1553_RCVSWD1	The first status word was received.
MSGACT1553_RCVSWD2	The second status word was received.
MSGACT1553_XMTDWD	Data words were transmitted.
MSGACT1553_RCVDWD	Data words were received.
MSGACT1553_BUS	Message was transmitted/received on bus A (0) or B (1).

Extract the channel number from the activity word by AND-ing the activity field with MSGACT1553_CHMASK and right-shifting the result by MSGACT1553_CHSHIFT. The resulting value is the channel number associated with the MIL-STD-1553 record.

```
Channel = (activity & MSGACT1553_CHMASK) >> MSGACT1553_CHSHIFT;
```

DEVICE DEPENDENCY

When IRIG is enabled on a 4G Device, time-tags in message structures and Sequential Records will be in BCD format (see BTICard_TimerStatus).

WARNINGS

The _ERROR field indicates detected errors. It does NOT relate to error injection on level C channels.

SEE ALSO

BTI1553_MsgBlockRd, BTI1553_MsgCommRd, BTI1553_MsgDataRd, BTI1553_BCCreateMsg, BTI1553_RTCreateMsg

MsgFieldWr

```
ULONG BTI1553_MsgFieldWr
(
  ULONG fieldvalue,          //New field value
  USHORT fieldtype,          //Field to rewrite
  MSGADDR message,           //Message to rewrite
  HCORE hCore                //Core handle
)
```

RETURNS

None.

DESCRIPTION

Writes `fieldvalue` to the specified field of the specified message structure. This function can be used to reconfigure an existing message.

Although all fields are writable, this function is typically only used to clear the hit bit in the `FIELD1553_ERROR` field. The hit bit indicates that the message has been transmitted or received.

See `BTI1553_MsgFieldRd` for tables describing the fields.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

None.

SEE ALSO

`BTI1553_MsgBlockWr`, `BTI1553_MsgCommWr`, `BTI1553_MsgDataWr`, `BTI1553_BCCreateMsg`, `BTI1553_RTCreateMsg`

MsgGroupDataRd

```
VOID BTI1553_MsgGroupDataRd
(
   INT count,                   //Number of messages to read
   USHORT databufs[][32],       //Array of 32-word data buffers
   MSGADDR msgs[],              //Array of source message addresses
   HCORE hCore                  //Core handle
)
```

RETURNS

None.

DESCRIPTION

Reads all 32 data words of `count` messages on a core in a single operation. The `msgs` parameter points to the array of message addresses from which to read. The `databufs` parameter points to an array of 32 word buffers to which the data words from each message are written to.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

None.

SEE ALSO

`BTI1553_MsgBlockRd`, `BTI1553_MsgGroupDataWr`, `BTI1553_MsgGroupRd`

MsgGroupDataWr

```
VOID BTI1553_MsgGroupDataWr
(
   INT count,                   //Number of messages to write
   USHORT databufs[][32],       //Array of 32-word data buffers
   MSGADDR msgs[],              //Array of destination message addresses
   HCORE hCore                  //Core handle
)
```

RETURNS

None.

DESCRIPTION

Writes all 32 data words to `count` messages on a core in a single operation. The `databufs` parameter points to an array of 32 word buffers containing the data words to write. The `msgs` parameter points to the array of message addresses to write to.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

None.

SEE ALSO

`BTI1553_MsgGroupDataRd`, `BTI1553_MsgGroupWr`, `BTI1553_MsgBlockWr`

MsgGroupRd

```
VOID BTI1553_MsgGroupRd
(
   INT count,                     //Number of messages to read
   MSGFIELDS1553 msgflds[],       //Array of destination structures
   MSGADDR msgs[],                //Array of source message addresses
   HCORE hCore                    //Core handle
)
```

RETURNS

None.

DESCRIPTION

Reads `count` message structures from a core in a single operation. The `msgs` parameter points to the array of message addresses from which to read. The `msgflds` parameter points to the array to which the contents of each message structures are written to. See `BTI1553_MsgBlockRd` for a table describing the message structure fields.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

None.

SEE ALSO

`BTI1553_MsgGroupWr`, `BTI1553_MsgGroupDataRd`, `BTI1553_MsgBlockRd`

MsgGroupWr

```
VOID BTI1553_MsgGroupWr
(
   INT count,                   //Number of messages to write
   MSGFIELDS1553 msgflds[],     //Array of source structures
   MSGADDR msgs[],              //Array of destination message addresses
   HCORE hCore                  //Core handle
)
```

RETURNS

None.

DESCRIPTION

Writes `count` message structures to a core in a single operation. The `msgflds` parameter points to the array of source message structures to write. The `msgs` parameter points to the array of destination message addresses to write to.

This function is typically used to modify certain fields in a message structure after it has been read using `BTI1553_MsgGroupRd`. See `BTI1553_MsgBlockRd` for a description of fields which can be modified.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

None.

SEE ALSO

`BTI1553_MsgBlockRd`, `BTI1553_MsgGroupRd`,
`BTI1553_MsgGroupDataWr`, `BTI1553_MsgBlockWr`

MsgSkipRd

```
BOOL BTI1553_MsgSkipRd
(
  MSGADDR message,          //Message to read
  HCORE hCore               //Core handle
)
```

RETURNS

The state of the skip bit for a message. The return value will be TRUE if previously set, or FALSE if previously clear.

DESCRIPTION

Reads the state of the skip bit for the Message Record specified by `message.` If the bit is zero, the `message` will be processed normally by the BC schedule. If the bit is non-zero, the `message` will be skipped by the BC schedule and will not be processed.

DEVICE DEPENDENCY

Applies to 4G, 5G, and 6G Devices.

WARNINGS

None.

SEE ALSO

`BTI429_MsgSkipWr, CmdSkipRd, CmdSkipWr`

MsgSkipWr

```
VOID BTI1553_MsgSkipWr
(
   BOOL skip,                 //Value for setting the skip bit
   MSGADDR message,           //Message to control
   HCORE hCore                //Core handle
)
```

RETURNS

None.

DESCRIPTION

Writes the state of the skip bit for the Message Record specified by `message`. If the bit is zero, the message will be processed normally by the BC Schedule. If the bit is non-zero, the message will be skipped by the BC Schedule and will not be processed.

DEVICE DEPENDENCY

Applies to 4G, 5G, and 6G Devices.

WARNINGS

None.

SEE ALSO

`BTI429_MsgSkipRd`

MsgSyncDefine

```
ERRVAL BTI1553_MsgSyncDefine
(
   BOOL enable,                //Enable/disable external sync pulse
   USHORT syncmask,            //Line(s) used for sync output
   USHORT pinpolarity,         //Active pin polarity (high/low)
   MSGADDR message,            //Message address
   HCORE hCore                 //Core handle
)
```

RETURNS

A negative value if an error occurs or zero if successful.

DESCRIPTION

Defines the sync output conditions for a BC `message` and configures it to output a sync signal. This sync signal appears on all lines specified by `syncmask` with the polarity specified by `pinpolarity` (see tables below). `BTI1553_MsgSyncDefine` may be called during run time to redefine the sync output settings. When enabled, the sync output line(s) is driven active with the transmission of `message`. The sync signal is active from just before the start of the BC transmission to just after the end of the BC transmission.

The constants in the tables below may be bitwise OR-ed together to configure multiple lines. Sync outputs are both generation dependent and device dependent. Refer to Table A.3 and the following tables for more information.

syncmask				
Constant	**Description**	**4G**	**5G**	**6G**
SYNCMASK_SYNCA	Selects discrete sync line A	✓	✓	✓
SYNCMASK_PXITRIGA	Selects PXIe TRIG sync line A			*
SYNCMASK_SYNCB	Selects discrete sync line B	✓	✓	✓
SYNCMASK_PXITRIGB	Selects PXIe TRIG sync line B			*
SYNCMASK_SYNCC	Selects discrete sync line C	✓	✓	✓
SYNCMASK_PXITRIGC	Selects PXIe TRIG sync line C			*
SYNCMASK_PXISTARC	Selects PXIe STAR sync line C			*

** PXIe Syncs are only available on PXIe Devices. Refer to `BTICard_CardSyncValid` for device dependent support*

pinpolarity	
Constant	**Description**
SYNCPOL_SYNCAL	Sets active low polarity for sync line A
SYNCPOL_SYNCAH	Sets active high polarity for sync line A
SYNCPOL_SYNCBL	Sets active low polarity for sync line B
SYNCPOL_SYNCBH	Sets active high polarity for sync line B
SYNCPOL_SYNCCL	Sets active low polarity for sync line C
SYNCPOL_SYNCCH	Sets active high polarity for sync line C

Alternatively, to configure all messages to output a sync pulse, use `BTI1553_BCSyncDefine`. Additionally, to configure an RT to output a sync signal during its transmissions, use `BTI1553_RTSyncDefine`.

DEVICE DEPENDENCY

Applies only to 4G, 5G, and 6G Devices. 3G Devices, which have only a single sync line, can use `BTI1553_BCConfig` or `BTI1553_RTConfig` with the `_SYNCSEL` flag and either `BTI1553_BCCreateMsg`, `BTI1553_RTCreateMsg`, `BTI1553_BCCreateList`, or `BTI1553_RTCreateList` with the `MSGCRT1553_SYNC` flag. The mapping of sync lines is hardware dependent. Please consult the hardware manual for the Device.

WARNINGS

None.

SEE ALSO

`BTI1553_BCSyncDefine`, `BTI1553_RTSyncDefine`, `BTICard_CardSyncValid`

MsgTriggerDefine

```
ERRVAL BTI1553_MsgTriggerDefine
(
   BOOL enable,                 //Enable/disable external trigger
   USHORT trigmask,             //Line(s) used for trigger signal
   USHORT trigval,              //Active/inactive condition for trigger line(s)
   USHORT pinpolarity,          //Active pin polarity (high/low)
   MSGADDR message,             //Message address
   HCORE hCore                  //Core handle
)
```

RETURNS

A negative value if an error occurs or zero if successful.

DESCRIPTION

Defines the trigger input settings for a BC `message` and associates this message with a trigger signal. The input line(s) are specified by `trigmask` with an active trigger state being the combination of `trigval` and `pinpolarity`, as defined in the tables below. `BTI1553_MsgTriggerDefine` may be called during run time to redefine the trigger input settings.

Before calling this function, the trigger mode must first be set by calling `BTI1553_BCConfig` with the `BCCFG1553_TRIGSTART` or `BCCFG1553_TRIGEXT` flag. When start mode is selected (`BCCFG1553_TRIGSTART`), the Bus Controller schedule runs until it encounters `message`, waits for the trigger signal, then runs normally. However, if external trigger mode is selected (`BCCFG1553_TRIGEXT`), the schedule halts and waits for the trigger signal before transmitting each tagged message.

The constants in the tables below may be bitwise OR-ed together to configure multiple lines. Only one `trigmask` may be selected per trigger line (A, B, and C), and all combined states must be true for the trigger to occur. Triggers are both generation dependent and device dependent. Refer to Table A.3 and the following tables for more information.

trigmask				
Constant	**Description**	**4G**	**5G**	**6G**
TRIGMASK_TRIGA	Selects discrete trigger line A	✓	✓	✓
TRIGMASK_PXITRIGA	Selects PXIe TRIG trigger line A			*
TRIGMASK_PXISTARA	Selects PXIe STAR trigger line A			*
TRIGMASK_TRIGB	Selects discrete trigger line B	✓	✓	✓
TRIGMASK_PXITRIGB	Selects PXIe TRIG trigger line B			*
TRIGMASK_PXISTARB	Selects PXIe STAR trigger line B			*
TRIGMASK_TRIGC	Selects discrete trigger line C	✓	✓	✓
TRIGMASK_PXITRIGC	Selects PXIe TRIG trigger line C			*
TRIGMASK_PXISTARC	Selects PXIe STAR trigger line C			*

** PXIe Syncs are only available on PXIe Devices. Refer to `BTICard_CardTriggerValid` for device dependent support*

trigval	
Constant	**Description**
TRIGVAL_TRIGAOFF	Trigger on line A inactive
TRIGVAL_TRIGAON	Trigger on line A active
TRIGVAL_TRIGBOFF	Trigger on line B inactive
TRIGVAL_TRIGBON	Trigger on line B active
TRIGVAL_TRIGCOFF	Trigger on line C inactive
TRIGVAL_TRIGCON	Trigger on line C active

pinpolarity				
Constant	**Description**	**4G**	**5G**	**6G**
TRIGPOL_TRIGAL	Sets active low polarity for trigger line A	✓	✓	✓
TRIGPOL_TRIGAH	Sets active high polarity for trigger line A	✓	✓	✓
TRIGPOL_TRIGAF	Sets active on falling edge of trigger line A			✓
TRIGPOL_TRIGAR	Sets active on rising edge of trigger line A			✓
TRIGPOL_TRIGBL	Sets active low polarity for trigger line B	✓	✓	✓
TRIGPOL_TRIGBH	Sets active high polarity for trigger line B	✓	✓	✓
TRIGPOL_TRIGBF	Sets active on falling edge of trigger line B			✓
TRIGPOL_TRIGBR	Sets active on rising edge of trigger line B			✓
TRIGPOL_TRIGCL	Sets active low polarity for trigger line C	✓	✓	✓
TRIGPOL_TRIGCH	Sets active high polarity for trigger line C	✓	✓	✓
TRIGPOL_TRIGCF	Sets active on falling edge of trigger line C			✓
TRIGPOL_TRIGCR	Sets active on rising edge of trigger line C			✓

Alternatively, to associate all messages to a trigger signal, use `BTI1553_BCTriggerDefine`.

Additionally, to associate error injection with a trigger signal, use `BTI1553_ErrorTriggerDefine`.

DEVICE DEPENDENCY

Applies only to 4G, 5G, and 6G Devices. The mapping of trigger lines is hardware dependent. Please consult the hardware manual for the Device.

WARNINGS

None.

SEE ALSO

`BTI1553_BCConfig`, `BTI1553_BCTriggerDefine`, `BTICard_CardTrigger`, `BTICard_CardTriggerEx`, `BTICard_CardTriggerValid`

ParamAmplitudeConfig

```
ERRVAL BTI1553_ParamAmplitudeConfig
(
   ULONG configval,          //Configuration options to set
   USHORT dacval,            //12-bit digital-analog converter value
   INT channel,              //Channel number
   HCORE hCore               //Core handle
)
```

RETURNS

A negative value if an error occurs or zero if successful.

DESCRIPTION

Enables variable transmit amplitude control as defined by `configval` on the channel specified by `channel` and sets the digital-to-analog converter to `dacval`. If this parametric control is not used or is disabled, then the amplitude reverts to default (full) amplitude. Some `configval` constants are level dependent (see table below).

Configval			
Constant	**Description**	**Level D**	**Level P**
PARAMCFG1553_DEFAULT	Select all default settings (**bold** below)	✓	✓
PARAMCFG1553_AMPLON	Enables parametric amplitude control	✓	✓
PARAMCFG1553_AMPLOFF	Disables parametric amplitude control	✓	✓
PARAMCFG1553_AMPLHI	Enables parametric amplitude control high range		✓
PARAMCFG1553_AMPLLO	Enables parametric amplitude control low range		✓

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

This function may only be used with 1553 channels with level D or level P capability.

SEE ALSO

`BTI1553_ChIsD`, `BTI1553_ChIsP`

PlayConfig

```
ERRVAL BTI1553_PlayConfig
(
   ULONG configval,           //Configuration options to set
   ULONG tamask,              //RT address mask
   INT channel,               //Channel number
   HCORE hCore                //Core handle
)
```

RETURNS

A negative value if an error occurs or zero if successful.

DESCRIPTION

Configures the BC for hardware playback mode. In hardware playback mode, the BC will use previously recorded monitor records to regenerate the 1553 bus traffic. Additionally, the Device can simulate the RT responses for the enabled terminals specified by `tamask`. The number of terminals that the Device can simultaneously simulate is dependent on the level of MIL-STD-1553 functionality for the specified channel (see Table A.2). The `configval` flags are shown in the table below.

Configval	
Constant	**Description**
PLAYCFG1553_DEFAULT	Select all default settings (**bold** below)
PLAYCFG1553_ENABLE	Enable Playback mode
PLAYCFG1553_DISABLE	Disable Playback mode
PLAYCFG1553_ALLAVAIL	Allocate all available memory to playback buffer
PLAYCFG1553_16K	Allocate 16K to playback buffer
PLAYCFG1553_32K	Allocate 32K to playback buffer
PLAYCFG1553_64K	Allocate 64K to playback buffer
PLAYCFG1553_128K	Allocate 128K to playback buffer
PLAYCFG1553_MC01	Select SA=00000 or SA=11111 for mode codes
PLAYCFG1553_MC1	Select SA=11111 for mode codes
PLAYCFG1553_MC0	Select SA=00000 for mode codes
PLAYCFG1553_MCNONE	Disable mode codes
PLAYCFG1553_TERMOFF	Direct coupled termination resistance off (Bus A & B)
PLAYCFG1553_TERMONA	Direct coupled termination resistance on (Bus A)
PLAYCFG1553_TERMONB	Direct coupled termination resistance on (Bus B)
PLAYCFG1553_TERMONAB	Direct coupled termination resistance on (Bus A & B)

DEVICE DEPENDENCY

Applies only to 4G, 5G, and 6G Devices. Termination flags are only supported on 4G Devices.

WARNINGS

None.

SEE ALSO

`BTI1553_PlayStatus`, `BTI1553_PlayWr`

PlayStatus

```
INT BTI1553_PlayStatus
(
   INT channel,                 //Channel number
   HCORE hCore                  //Core handle
)
```

RETURNS

Status of the hardware playback buffer.

DESCRIPTION

Determines the status of the hardware playback buffer. The status values are shown below.

Status	
Constant	**Description**
STAT_EMPTY	Buffer is empty
STAT_PARTIAL	Buffer is partially filled
STAT_FULL	Buffer is full
STAT_OFF	Buffer is off

DEVICE DEPENDENCY

Applies only to 4G, 5G, and 6G Devices.

WARNINGS

None.

SEE ALSO

BTI1553_PlayConfig, BTI1553_PlayWr

PlayWr

```
USHORT BTI1553_PlayWr
(
   LPUSHORT seqbuf,          //Pointer to sequential record buffer
   USHORT bufcount,          //Number of 16-bit words to write
   INT channel,              //Channel number
   HCORE hCore               //Core handle
)
```

RETURNS

Number of words written to the playback buffer.

DESCRIPTION

Writes data to the hardware playback buffer. The data should be in the same format as the Sequential Monitor records.

DEVICE DEPENDENCY

Applies only to 4G, 5G, and 6G Devices.

WARNINGS

None.

SEE ALSO

`BTI1553_PlayConfig`, `BTI1553_PlayStatus`

RTConfig

```
ERRVAL BTI1553_RTConfig
(
  ULONG ctrlflags,        //Selects configuration options
  INT taval,              //Terminal address
  INT channel,            //Channel number
  HCORE hCore             //Core handle
)
```

RETURNS

A negative value if an error occurs or zero if successful.

DESCRIPTION

Configures a Remote Terminal for the specified channel with the options defined by `ctrlflags.` For all channel levels, except level A, this function can be called repeatedly with different terminal addresses to configure the channel to simulate multiple terminals.

By default, `BTI1553_RTConfig` creates message structures for all subaddresses and mode codes. These message structures are allocated using the default `BTI1553_RTCreateMsg` options. The addresses of these structures are obtained as needed by calling `BTI1553_RTGetMsg`.

If `BTI1553_RTConfig` is called with the `RTCFG1553_NOBUILD` flag, all command words are illegalized until message structures are created with `BTI1553_RTCreateMsg.`

ctrlflags	
Constant	**Description**
RTCFG1553_DEFAULT	Select all default settings (**bold** below)
RTCFG1553_DISABLE	Disable the Remote Terminal
RTCFG1553_SIMULATE	Enable RT simulation
RTCFG1553_MONITOR	Enable this RT as a Shadow Monitor
RTCFG1553_NOBCAST	Disable broadcast
RTCFG1553_BCAST	Enable broadcast receptions
RTCFG1553_NOAUTOBUSY	Disable auto busy
RTCFG1553_AUTOBUSY	Set busy bit after the next command
RTCFG1553_BUILD	All Sas and MCs initially enabled
RTCFG1553_NOBUILD	All Sas and MCs initially disabled
RTCFG1553_NODYNBC	Ignore dynamic BC mode code
RTCFG1553_DYNBC	Respond to dynamic BC mode code
RTCFG1553_NOIMMCLR	Do not clear status word bits
RTCFG1553_IMMCLR	Clear status word bits after transmitted
RTCFG1553_NOB-CASTADDR	Disable broadcast for specified address
RTCFG1553_BCASTADDR	Enable broadcast receptions for specified address
RTCFG1553_CHANAB	Respond to bus A and bus B
RTCFG1553_CHANA	Respond to bus A only
RTCFG1553_CHANB	Respond to bus B only
RTCFG1553_CHANNONE	Respond to neither bus A or B

(Continued on next page)

(Continued from previous page)

ctrlflags	
Constant	**Description**
RTCFG1553_MC01	00000 & 11111 indicates mode codes
RTCFG1553_MC1	Select SA=11111 for mode codes
RTCFG1553_MC0	Select SA=00000 for mode codes
RTCFG1553_MCNONE	Disable mode codes
RTCFG1553_SYNCSEL	Sync out selected at message level
RTCFG1553_SYNCALL	Sync out driven for all messages
RTCFG1553_WIPE	Enable the wipe options (below)
RTCFG1553_NOWIPE	Disable the wipe options (data is random values)
RTCFG1553_WIPE0	Initialize the data words with zeros
RTCFG1553_WIPE123	Initialize the data words with incrementing values
RTCFG1553_WIPECWD	Initialize the data words with command word values
RTCFG1553_RESPONSEB	RT responds using MIL-STD-1553B response timing. The RT will validate that there is no extra contiguous data following reception of a command word
RTCFG1553_RESPONSEA	RT responds using MIL-STD-1553A response timing. The RT will assume that there are no extra contiguous data words
RTCFG1553_TERMOFF	Direct coupled termination resistance off for Bus A and B (default)
RTCFG1553_TERMONA	Direct coupled termination resistance on for Bus A
RTCFG1553_TERMONB	Direct coupled termination resistance on for Bus B
RTCFG1553_TERMONAB	Direct coupled termination resistance on for Bus A and B

When simulating an RT (RTCFG1553_SIMULATE) and mode codes are enabled (RTCFG1553_MC01, RTCFG1553_MC0, or RTCFG1553_MC1), the following side effects occur in response to these mode codes commands:

Mode Code effects		
T/R bit	**Mode Code**	**Description**
1	0 – Dynamic bus control	Sets the DBC bit in the status word when RTCFG1553_DYNBC is set
1	1 – Synchronize	Simulated RTs reset the timer/time-tag for the entire Device. Contact support for info on disabling this operation.
1	2 – Transmit last status	Transmits the last Status Word (SWD)
1	4 – Transmitter shutdown	Simulated RTs are changed to non-responding 'Monitored' (as if RTCFG1553_MONITOR had been configured) RT's. Does not have any effect on Bus Monitor (BM)
1	5 – Override transmitter shutdown	An RT that had been previously shutdown (i.e. from MC 4) is restored back to a simulated RT. Does not have any effect on BM
1	6 – Inhibit terminal flag bit	Inhibits TF bit for the terminal – masks off the TF bit to set the TF bit of SWD to zero
1	7 – Override inhibit terminal flag bit	Un-inhibits TF bit for the terminal – value of the TF bit is set in SWD
1	8 –Reset remote terminal	Clears the RT SWD and un-inhibits any shutdown bus for the terminal. Does not affect timer (time-tag) or have any effect on BM
1	18 – Transmit last command	Transmits the last Command Word (CWD)
1/0	<all other mode codes>	No additional action taken by the hardware

Other mode codes have no other additional action taken.

DEVICE DEPENDENCY

Termination flags are only supported on 4G Devices.

WARNINGS

None.

SEE ALSO

`BTI1553_RTGetMsg`, `BTI1553_RTCreateMsg`

RTCreateList

```
LISTADDR BTI1553_RTCreateList
(
   ULONG listctrlflags,      //Selects List Buffer options
   INT count,                //One more than the number of entries in list
   ULONG msgctrlflags,       //Selects configuration options.
   BOOL mcflag,              //TRUE if mode code, FALSE if subaddress
   INT taval,                //Terminal address containing message
   BOOL trval,               //Transmit/receive bit
   INT saval,                //Subaddress or mode-code number
   INT channel,              //Channel number
   HCORE hCore               //Core handle
)
```

RETURNS

The address of the List Buffer if successful, otherwise zero.

DESCRIPTION

Creates a message List Buffer and associates it with a specified Remote Terminal message. Similar to `BTI1553_RTCreateMsg` except creates a List Buffer. This function allocates memory for a list of message structures. The flags `SUBADDRESS` or `MODECODE` may be used for the `mcflag`. `XMT` or `RCV` may be used for the `trval.` The maximum number of entries that may be stored in the list is `count-1`.If the host is reading data from the List Buffer, the `LISTCRT1553_LOG` flag will generate an entry in the Event Log List when the list is full. If the host is writing data to the List Buffer, the `LISTCRT1553_LOG` flag will generate an entry in the Event Log List when the list is empty.

When the `LISTCRT1553_LOGHALF` flag is specified, the List Buffer generates an entry in the Event Log List when the middle or the last entry is processed. This flag is not used if the List Buffer is configured for ping pong mode.

Listctrlflags	
Constant	**Description**
LISTCRT1553_DEFAULT	Select all default settings (**bold** below)
LISTCRT1553_PINGPONG	Selects ping-pong mode
LISTCRT1553_FIFO	Selects FIFO mode
LISTCRT1553_CIRCULAR	Selects circular mode
LISTCRT1553_LOG	Generates an `EVENTYPE_1553LIST` entry in the Event Log when the list is empty/full
LISTCRT1553_NOLOG	Does not generate an entry in the Event Log when the list is empty/full
LISTCRT1553_LOGHALF	Generates an `EVENTYPE_1553MSG` entry in the Event Log when either the middle or last entry is processed. To determine which message generated the Event Log entry, use `BTI1553_ListAddr` to get the address of the message.
LISTCRT1553_NOLOGHALF	Does not generate an entry in the Event Log when the middle or last entry is processed
LISTCRT1553_TRBIT	The host will read if `trval` is zero or write if `trval` is one
LISTCRT1553_READ	The host will read receive data from the list
LISTCRT1553_WRITE	The host will write transmit data to the list

msgctrlflags	
Constant	**Description**
MSGCRT1553_DEFAULT	Select all default settings (**bold** below)
MSGCRT1553_NOLOG	This message will not generate an entry in the Event Log List
MSGCRT1553_LOG	This message will generate an entry in the Event Log List
MSGCRT1553_NOERR	This message is not a candidate for errors
MSGCRT1553_ERR	This message is a candidate for errors
MSGCRT1553_BUSA	Message will be transmitted on bus A (BC only)
MSGCRT1553_BUSB	Message will be transmitted on bus B (BC only)
MSGCRT1553_BCRT	Message is a BC-RT transfer (BC only)
MSGCRT1553_RTRT	Message is an RT-RT transfer (BC only)
MSGCRT1553_NOMON	Message will not be monitored (BC only, level C, D, P, and M channels only)
MSGCRT1553_MON	Message will be sent to the Sequential Monitor (BC only, level C, D, P, and M channels only)
MSGCRT1553_NOTIMETAG	Message will not record time-tag
MSGCRT1553_TIMETAG	Message will record time-tag
MSGCRT1553_NOELAPSE	Message will not record elapsed time
MSGCRT1553_ELAPSE	Message will record elapsed time
MSGCRT1553_NOMIN	Message will not record min time
MSGCRT1553_MIN	Message will record min time
MSGCRT1553_NOMAX	Message will not record max time
MSGCRT1553_MAX	Message will record max time
MSGCRT1553_NOHIT	Message will not record hit count
MSGCRT1553_HIT	Message will record hit count
MSGCRT1553_NOSYNC	Sync out will not be driven for this message
MSGCRT1553_SYNC	Sync out will be driven for this message
MSGCRT1553_WIPE	Enable the wipe options (below)
MSGCRT1553_NOWIPE	Disable the wipe options (data is random values)
MSGCRT1553_WIPE123	Initialize the data words with incrementing values
MSGCRT1553_WIPECWD	Initialize the data words with command word values
MSGCRT1553_WRAP	Rcv SA Data will be automatically wrapped to corresponding Xmt SA Data (RT only)
MSGCRT1553_NOWRAP	SA Data buffers operate normally (RT only)

DEVICE DEPENDENCY

LISTCRT1553_PINGPONG is not supported by 5G or 6G Devices since they have inherent protection for data coherency (use BTI1553_MsgCommRd and BTI1553_MsgCommWr instead).

3G and 4G Devices support up to 511 list entries while 5G and 6G Devices support up to 1023 list entries.

WARNINGS

A call to BTI1553_RTConfig must precede the use of this function for a given terminal address.

SEE ALSO

BTI1553_BCCreateList, BTI1553_ListDataRd, BTI1553_ListDataWr, BTI1553_RTCreateMsg

RTCreateMsg

```
MSGADDR BTI1553_RTCreateMsg
(
  ULONG msgctrlflags,        //Selects configuration options.
  BOOL mcflag,               //TRUE if mode code, FALSE if subaddress
  INT taval,                 //Terminal address containing message
  BOOL trval,                //transmit/receive bit
  INT saval,                 //Subaddress or mode-code number
  INT channel,               //Channel number
  HCORE hCore                //Core handle
)
```

RETURNS

The address of the message structure if successful, otherwise zero.

DESCRIPTION

Creates a message structure for the indicated command word in the specified terminal address, if one does not already exist. If a message structure already exists for the specified parameters, `BTI1553_RTCreateMsg` reconfigures it with the new `ctrlflags`. This function is usually only needed if the `RTCFG1553_NOBUILD` flag was used in `BTI1553_RTConfig`. The flags `SUBADDRESS` or `MODECODE` may be used for the `mcflag`. `XMT` or `RCV` may be used for the `trval`.

msgctrlflags	
Constant	**Description**
MSGCRT1553_DEFAULT	Select all default settings (**bold** below)
MSGCRT1553_NOLOG	This message will not generate an entry in the Event Log List
MSGCRT1553_LOG	This message will generate an entry in the Event Log List
MSGCRT1553_NOERR	This message is not a candidate for errors
MSGCRT1553_ERR	This message is a candidate for errors
MSGCRT1553_BUSA	Message will be transmitted on bus A (BC only)
MSGCRT1553_BUSB	Message will be transmitted on bus B (BC only)
MSGCRT1553_BCRT	Message is a BC-RT transfer (BC only)
MSGCRT1553_RTRT	Message is an RT-RT transfer (BC only)
MSGCRT1553_NOMON	Message will not be monitored (BC only, level C, D, P, and M channels only)
MSGCRT1553_MON	Message will be sent to the Sequential Monitor (BC only, level C, D, P, and M channels only)
MSGCRT1553_NOTIMETAG	Message will not record time-tag
MSGCRT1553_TIMETAG	Message will record time-tag
MSGCRT1553_NOELAPSE	Message will not record elapsed time
MSGCRT1553_ELAPSE	Message will record elapsed time
MSGCRT1553_NOMIN	Message will not record min time
MSGCRT1553_MIN	Message will record min time
MSGCRT1553_NOMAX	Message will not record max time
MSGCRT1553_MAX	Message will record max time
MSGCRT1553_NOHIT	Message will not record hit count
MSGCRT1553_HIT	Message will record hit count

(Continued on next page)

(Continued from previous page)

msgctrlflags	
Constant	**Description**
MSGCRT1553_NOSYNC	Sync out will not be driven for this message
MSGCRT1553_SYNC	Sync out will be driven for this message
MSGCRT1553_WIPE	Enable the wipe options (below)
MSGCRT1553_NOWIPE	Disable the wipe options (data is random values)
MSGCRT1553_WIPE0	Initialize the data words with zeros
MSGCRT1553_WIPE123	Initialize the data words with incrementing values
MSGCRT1553_WIPECWD	Initialize the data words with command word values
MSGCRT1553_WRAP	Rcv SA Data will be automatically wrapped to corresponding Xmt SA Data (RT only)
MSGCRT1553_NOWRAP	SA Data buffers operate normally (RT only)

DEVICE DEPENDENCY

When IRIG is enabled on a 4G Device, time-tags in message structures and Sequential Records will be in BCD format (see `BTICard_TimerStatus`).

WARNINGS

A call to `BTI1553_RTConfig` must precede the use of this function for a given terminal address.

SEE ALSO

`BTI1553_RTCreateList`, `BTI1553_BCCreateMsg`,
`BTI1553_RTConfig`, `BTI1553_RTGetMsg`

RTGetMsg

```
MSGADDR BTI1553_RTGetMsg
(
  BOOL mcflag            //TRUE if mode code, FALSE if subaddress
  INT taval              //Terminal address containing message
  BOOL trval             //transmit/receive bit
  INT saval              //Subaddress or mode-code number
  INT channel,           //Channel number
  HCORE hCore            //Core handle
)
```

RETURNS

The address of the message structure if successful, otherwise zero.

DESCRIPTION

Obtains the message address for an existing message structure. This function is typically used when `BTI1553_RTConfig` is allowed to automatically create all possible message structures for the specified terminal address instead of explicitly creating them with `BTI1553_RTCreateMsg`. The flags `SUBADDRESS` or `MODECODE` may be used for the `mcflag` parameter. `XMT` or `RCV` may be used for the *`trval`* parameter.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

None.

SEE ALSO

`BTI1553_RTCreateMsg`, `BTI1553_RTConfig`, `BTI1553_MsgFieldRd`, `BTI1553_MsgFieldWr`, `BTI1553_MsgDataRd`, `BTI1553_MsgDataWr`

RTReset

```
ERRVAL BTI1553_RTReset
(
  INT taval,                //Terminal address
  INT channel,              //Channel number
  HCORE hCore               //Core handle
)
```

RETURNS

A negative value if an error occurs or zero if successful.

DESCRIPTION

Perform a logical reset of the RT with address `taval.` This function clears all bits of the status word (except the terminal address) and enables both bus A and bus B transmitters. It disables override of the TF bit of the status word. This is the same as receiving the Reset RT mode code.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

None.

SEE ALSO

`BTI1553_RTConfig`

RTResponseTimeSet

```
ERRVAL BTI1553_RTResponseTimeSet
(
   INT resptime,               //New response time to set
   INT taval,                  //Terminal address of RT
   INT channel,                //Channel number
   HCORE hCore                 //Core handle
)
```

RETURNS

A negative value if an error occurs or zero if successful.

DESCRIPTION

Defines the response time for the specified Remote Terminal (`taval`) on `channel`. All Devices have a default RT response time of 9 µs which is consistent with the MIL-STD-1553 protocol. This function is used to alter the default RT response time. `BTI1553_RTResponseTimeSet` may be called during run time to redefine the RT response time.

The response time of the specified RT is specified by `resptime` in tenths of a bit time (100 ns), and is measured from the mid-bit zero-crossing of the previous parity bit to the mid-bit zero-crossing of the status word sync. The values for `resptime` depend on the selection of the `RTCFG1553_RESPONSEA` and `RTCFG1553_RESPONSEB` control flags used when the RT was configured by calling `BTI1553_RTConfig`. The table below shows the valid ranges for *`resptime`*.

resptime Ranges				
	RTCFG1553_RESPONSEA		**RTCFG1553_RESPONSEB**	
	MIN	**MAX**	**MIN**	**MAX**
4G	36 (3.6 µs)	980 (98.0 µs)	76 (7.6 µs)	980 (98.0 µs)
5G 6G	37 (3.7 µs)	8191 (819.1 µs)	77 (7.7 µs)	8191 (819.1 µs)

DEVICE DEPENDENCY

Applies only to 4G, 5G, and 6G Devices.

WARNINGS

`BTI1553_RTConfig` should be called prior to `BTI1553_RTResponseTimeSet.` When adjusting the RT response time, it is important to consider the BC no-response timeout value. To adjust the BC no-response timeout value for a simulated BC use `BTI1553_BCSetTimeout.`

SEE ALSO

`BTI1553_RTConfig`

RTSetMode

```
ERRVAL BTI1553_RTSetMode
(
  ULONG configval,          //Mode to set RT to
  INT taval,                //Terminal address of RT
  INT channel,              //Channel number
  HCORE hCore               //Core handle
)
```

RETURNS

A negative value if an error occurs or zero if successful.

DESCRIPTION

Changes the mode of a configured Remote Terminal to the one specified by `configval` (see table below).

Normally, `BTI1553_RTConfig` sets the mode of a Remote Terminal to simulate, monitor, or disabled. This can be changed by `BTI1553_RTSetMode` while the card is running or stopped. The mode is changed without pausing the channel, but only after the Remote Terminal has finished processing the current message.

configval	
Constant	**Description**
RTCFG1553_SIMULATE	Enable RT simulation
RTCFG1553_MONITOR	Enable this RT as a Shadow Monitor
RTCFG1553_DISABLE	Disable the Remote Terminal

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

An error is returned if the Remote Terminal was not configured by a previous call to `BTI1553_RTConfig`.

SEE ALSO

`BTI1553_RTConfig`

RTSWDRd

```
USHORT BTI1553_RTSWDRd
(
   INT taval,              //Terminal address of simulated RT
   INT channel,            //Channel number
   HCORE hCore             //Core handle
)
```

RETURNS

The status word for the RT.

DESCRIPTION

Reads the status word bits used by the simulated RT with terminal address `taval`.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

The specified terminal address must have already been configured by `BTI1553_RTConfig`.

SEE ALSO

`BTI1553_RTConfig`, `BTI1553_RTSWDWr`

RTSWDWr

```
ERRVAL BTI1553_RTSWDWr
(
   USHORT swdval,          //MIL-STD-1553 status word
   INT taval,              //Terminal address of simulated RT
   INT channel,            //Channel number
   HCORE hCore             //Core handle
)
```

RETURNS

A negative value if an error occurs or zero if successful.

DESCRIPTION

Sets the status word bits used by the simulated RT with terminal address `taval`. Except for the broadcast and message error bits, this function can set all the bits of the status word.

The broadcast and message error bits can be set in `swdval`, but they can not be overridden (cleared) by `swdval`. The broadcast and message error bits in `swdval` are bitwise OR-ed with internal values, which are set according to MIL-STD-1553 protocol. For normal 1553 behavior, the broadcast and message error bits should have a zero value in `swdval`.

`BTI1553_RTConfig` initializes the terminal address field of the terminal's status word. However, `BTI1553_RTSWDWr` overwrites it. Unless an invalid status word is desired, the user must make sure that the terminal address specified in `swdval` matches *`taval`*, since this function overwrites the initialization performed by `BTI1553_RTConfig.`

If the busy bit is set in a Remote Terminal, the Remote Terminal does not process data associated with command words directed to it.

Note: This function does not write to the status word field in any message structure. The status word field in a message structure contains the status word actually transmitted on the last command to that subaddress, not the status word that will be transmitted.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

Since this function overwrites the status word of a terminal, it should be used with extra care to avoid unintended results.

The specified terminal address must have already been configured by `BTI1553_RTConfig`.

SEE ALSO

`BTI1553_RTConfig, BTI1553_RTSWDRd`

RTSyncDefine

```
ERRVAL BTI1553_RTSyncDefine
(
   BOOL enable,             //Enable/disable external sync output
   USHORT syncmask,         //Line(s) used for sync output
   USHORT pinpolarity,      //Active pin polarity (high/low)
   INT taval,               //Terminal address
   ULONG rcvsamask,         //Receive subaddress message(s) to tag for sync
   ULONG xmtsamask,         //Transmit subaddress message(s) to tag for sync
   ULONG rcvmcmask,         //Receive mode code message(s) to tag for sync
   ULONG xmtmcmask,         //Transmit mode code message(s) to tag for sync
   INT channel,             //Channel number
   HCORE hCore              //Core handle
)
```

RETURNS

A negative value if an error occurs or zero if successful.

DESCRIPTION

Defines the sync output settings for `channel` on `hCore` and configures the RT specified by `taval` to output a sync signal when responding to messages to the subaddresses and/or mode codes specified by `rcvsamask`, `xmtsamask`, `rcvmcmask`, and `xmtmcmask`. The sync signal appears on all lines specified by `syncmask` with the polarity specified by `pinpolarity` (see tables below). Note that `BTI1553_RTSyncDefine` cannot be called while the channel is running.

When enabled, the sync outline line(s) is driven active with every RT response for the specified subaddresses and/or mode codes. The sync signal is active from just before the start of the RT transmission until just after the end of the RT transmission.

The constants in the tables below may be bitwise OR-ed together to configure multiple lines. Sync outputs are both generation dependent and device dependent. Refer to Table A.3 and the following tables for more information.

syncmask				
Constant	**Description**	**4G**	**5G**	**6G**
SYNCMASK_SYNCA	Selects discrete sync line A	✓	✓	✓
SYNCMASK_PXITRIGA	Selects PXIe TRIG sync line A			*
SYNCMASK_SYNCB	Selects discrete sync line B	✓	✓	✓
SYNCMASK_PXITRIGB	Selects PXIe TRIG sync line B			*
SYNCMASK_SYNCC	Selects discrete sync line C	✓	✓	✓
SYNCMASK_PXITRIGC	Selects PXIe TRIG sync line C			*
SYNCMASK_PXISTARC	Selects PXIe STAR sync line C			*

** PXIe Syncs are only available on PXIe Devices. Refer to* `BTICard_CardSyncValid` *for device dependent support*

pinpolarity	
Constant	**Description**
SYNCPOL_SYNCAL	Sets active low polarity for sync line A
SYNCPOL_SYNCAH	Sets active high polarity for sync line A
SYNCPOL_SYNCBL	Sets active low polarity for sync line B
SYNCPOL_SYNCBH	Sets active high polarity for sync line B
SYNCPOL_SYNCCL	Sets active low polarity for sync line C
SYNCPOL_SYNCCH	Sets active high polarity for sync line C

Additionally, to configure the BC to output a sync signal during its transmissions, use `BTI1553_BCSyncDefine` (all messages) or `BTI1553_MsgSyncDefine` (selected messages).

DEVICE DEPENDENCY

Applies to 4G, 5G, and 6G Devices. 3G Devices, which have only a single sync line, can use `BTI1553_RTConfig` with the `RTCFG1553_SYNCALL` flag.

WARNINGS

Subsequent calls to `BTI1553_RTCreateMsg` will affect whether or not a particular RT message will drive the defined sync output(s). Use the constant `MSGCRT1553_SYNC` or `MSGCRT1553_NOSYNC` to control the sync option when calling `BTI1553_RTCreateMsg`. The mapping of sync lines is hardware dependent. Please consult the hardware manual for the Device.

SEE ALSO

`BTI1553_BCSyncDefine`, `BTI1553_MsgSyncDefine`,
`BTI1553_RTCreateMsg`, `BTICard_CardSyncValid`

ValPackCWD

```
USHORT BTI1553_ValPackCWD
(
   INT taval,                  //Terminal address
   INT trval,                  //T/R bit
   INT saval,                  //Subaddress
   INT wcval                   //Word count/mode code
)
```

RETURNS

A MIL-STD-1553 command word with its bit fields set to the specified values.

DESCRIPTION

Creates a 16-bit MIL-STD-1553 command word from the values of its four fields. Only the least significant bit of `trval` and the five least significant bits of `taval`, `saval`, and `wcval` are used.

This is a utility function that does not access Device hardware.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

None.

SEE ALSO

`BTI1553_ValUnpackCWD`

ValUnpackCWD

```
VOID BTI1553_ValUnpackCWD
(
  USHORT cwdval,          //MIL-STD-1553 command word
  LPINT taval,            //Pointer to terminal address
  LPINT trval,            //Pointer to T/R bit
  LPINT saval,            //Pointer to subaddress
  LPINT wcval             //Pointer to word count
)
```

RETURNS

None.

DESCRIPTION

Extracts the four fields from a MIL-STD-1553 command word (`cwdval`) and the right-shifed values are passed through `taval`, `trval`, `saval`, and `wcval.` If a pointer is NULL, that value is skipped.

This is a utility function that does not access Device hardware.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

None.

SEE ALSO

`BTI1553_ValPackCWD`

APPENDIX B: BTICARD FUNCTION REFERENCES

Table B.1 summarizes the BTICard functions.

BIT Functions	
BTICard_BITConfig	Configures the Built-In Test functionality of the card
BTICard_BITInitiate	Performs a hardware test on the specified card
BTICard_BITStatusClr	Clears the historical maximum and minimum values
BTICard_BITStatusRd	Reads the current status of Built-In Test (BIT)
CARD Functions	
BTICard_CardClose	Disables access to a Device and releases its hardware resources
BTICard_CardIsRunning	Checks whether core is running
BTICard_CardOpen	Enables access to a Device and secures hardware resources
BTICard_CardProductStr	Returns the name of the Device
BTICard_CardReset	Resets the core hardware; destroys all existing configuration data
BTICard_CardResume	Resumes operation of the core
BTICard_CardStart	Starts operation of the core
BTICard_CardStop	Stops operation of the core
BTICard_CardSyncValid	Returns all available sync lines for the specified core
BTICard_CardTest	Performs a hardware test on the specified core
BTICard_CardTrigger	Generates a software-simulated trigger signal on all available trigger lines
BTICard_CardTriggerEx	Generates a software-simulated trigger signal on the specified trigger line(s)
BTICard_CardTriggerValid	Returns all available trigger lines for the specified core
BTICard_CardTypeStr	Returns the type or model number of the Device
BTICard_CoreOpen	Enables access to a specified core
EVENT Functions	
BTICard_EventLogConfig	Enables events and initializes the Event Log List
BTICard_EventLogRd	Reads an entry from the Event Log List
BTICard_EventLogStatus	Checks the status of the Event Log List
I/O Functions	
BTICard_ExtDIOMonConfig	Enables Sequential monitoring on specific transitions of the digital I/O pins
BTICard_ExtDIOEnWr	Sets both the value and output enable of the specified digital I/O pin
BTICard_ExtDIORd	Reads the value of the specified digital I/O pin
BTICard_ExtDIOWr	Sets the value of the specified digital I/O pin
BTICard_ExtLEDRd	Reads the on/off value of the LED
BTICard_ExtLEDWr	Sets the on/off value of the LED
BTICard_ExtStatusLEDRd	Reads the on/off value and color of the Status LED
BTICard_ExtStatusLEDWr	Sets the on/off value and color of the Status LED
INTERRUPT Functions	
BTICard_IntClear	Clears the interrupt from the core (OS-dependent)
BTICard_IntInstall	Associates an event object with interrupts from the core (OS-dependent)
BTICard_IntUninstall	Removes association between event objects and interrupts (OS-dependent)

Table B.1—Protocol-Independent (BTICard_) Functions (Continued on Next Page)

IRIG TIME Functions	
BTICard_IRIGConfig	Configures the IRIG timer on the specified core
BTICard_IRIGFieldGet??	Returns the ?? field (days, hours, etc.) from an IRIG time-tag
BTICard_IRIGFieldPut??	Writes the ?? field (days, hours, etc.) to an IRIG time-tag
BTICard_IRIGInputThresholdGet	Gets the threshold of the IRIG input circuitry
BTICard_IRIGInputThresholdSet	Sets the threshold of the IRIG input circuitry
BTICard_IRIGRd	Reads the current value of the IRIG timer on the specified core
BTICard_IRIGSyncStatus	Reports whether the IRIG timer is locked in sync with the IRIG bus
BTICard_IRIGTimeBCDToBin	Converts IRIG BCD value to a microsecond binary time
BTICard_IRIGTimeBCDToNanoBin	Converts IRIG BCD value to a nanosecond binary time
BTICard_IRIGTimeBinToBCD	Converts a microsecond binary time value to IRIG BCD value
BTICard_IRIGTimeNanoBinToBCD	Converts a nanosecond binary time value to IRIG BCD value
BTICard_IRIGWr	Sets (initializes) the IRIG timer on the specified core
PXIE Functions	
BTICard_PXIStatus	Returns user-requested status for PXIe cards
SEQUENTIAL RECORD Functions	
BTICard_SeqBlkRd	Reads multiple records out of the Sequential Record (use for few records)
BTICard_SeqCommRd	Reads multiple records out of the Sequential Record (use for many records)
BTICard_SeqConfig	Configures the Sequential Monitor
BTICard_SeqDMARd	Reads multiple records out of the Sequential Record (use for many records)
BTICard_SeqFindCheckVersion	Tests the version number of the specified record
BTICard_SeqFindInit	Initializes the BTICard SeqFindNext?? functions
BTICard_SeqFindMore1553	Finds the extra 1553 record fields, when present
BTICard_SeqFindNext	Finds the next message in the Sequential Record buffer
BTICard_SeqFindNext1553	Finds the next MIL-STD-1553 message in the Sequential Record buffer
BTICard_SeqFindNext429	Finds the next ARINC 429 message in the Sequential Record buffer
BTICard_SeqFindNext708	Finds the next ARINC 708 message in the Sequential Record buffer
BTICard_SeqFindNext717	Finds the next ARINC 717 message in the Sequential Record buffer
BTICard_SeqFindNextDIO	Finds the next DIO message in the Sequential Record buffer
BTICard_SeqFindNextEBR	Finds the next EBR-1553 message in the Sequential Record buffer
BTICard_SeqInterval	Sets the interval value if using Interval mode
BTICard_SeqIsRunning	Determines whether the Sequential Record is running
BTICard_SeqLogFrequency	Specifies the period for Sequential Record Event Log List entries
BTICard_SeqRd	Reads a single record out of the Sequential Record
BTICard_SeqResume	Resumes recording of the Sequential Record where it stopped
BTICard_SeqStart	Starts recording at the beginning of the Sequential Record
BTICard_SeqStatus	Checks the status of the Sequential Record
BTICard_SeqStop	Stops data from being added to the Sequential Record

Table B.1—Protocol-Independent (BTICard_) Functions (Continued on Next Page)

SYSMON Functions	
BTICard_SysMonClear	Resets the historic maximum and minimum values for all sensors
BTICard_SysMonDescGet	Returns the description of a sensor
BTICard_SysMonMaxRd	Reads the historical maximum value of a sensor
BTICard_SysMonMinRd	Reads the historical minimum value of a sensor
BTICard_SysMonNomRd	Reads the nominal value of a voltage sensor
BTICard_SysMonThresholdGet	Reads the user thresholds of a temperature sensor
BTICard_SysMonThresholdSet	Sets the user thresholds of a temperature sensor
BTICard_SysMonTypeGet	Returns the type of sensor
BTICard_SysMonUserStr	Returns a formatted value string of sensor data
BTICard_SysMonValRd	Reads the current sensor value
TIMER Functions	
BTICard_Timer64Rd	Reads the current value of the Device timer (64 bit)
BTICard_Timer64Wr	Writes a value to the Device timer (64 bit)
BTICard_TimerClear	Clears the Device timer
BTICard_TimerRd	Reads the current value of the Device timer (32 bit)
BTICard_TimerResolution	Selects a time-tag timer resolution
BTICard_TimerStatus	Checks status of the timer configuration
BTICard_TimerWr	Writes the a value to the Device timer (32 bit)
Timing Synchronization Manager (TSM) Functions	
BTICard_TSMDriftMaxGet	Reads the current maximum drift bias limit (ppt)
BTICard_TSMDriftMaxSet	Sets the maximum drift bias limit (ppt.
BTICard_TSMDriftRd	Reads the current commanded drift bias (ppt.
BTICard_TSMDriftWr	Sets the drift bias (ppt)
BTICard_TSMInputDelayCompGet	Reads the current input delay compensation (ns)
BTICard_TSMInputDelayCompSet	Sets the input delay compensation (adjusts timer to IRIG delays) (ns)
BTICard_TSMInputThresholdGet	Reads the DAC value for the requested input
BTICard_TSMInputThresholdSet	Sets the DAC value for the requested input
BTICard_TSMIRIGControlRd	Reads ‘control’ field from last received IRIG message
BTICard_TSMIRIGControlWr	Writes ‘control’ field to outgoing IRIG messages
BTICard_TSMIRIGYearsRd	Reads ‘Years’ field from device, Set via IRIG if enabled
BTICard_TSMIRIGYearsWr	Writes (BCD) year to device, Sent via IRIG if enabled
BTICard_TSMJumpThresholdSet	Sets minimum value that forces device timer to ‘jump’ (ns)
BTICard_TSMJumpThresholdGet	Reads currently set value that forces device timer to ‘jump’ (ns)
BTICard_TSMReset	Resets the Timing Synchronization Manager
BTICard_TSMSourceConfig	Sets the output (source) configuration for the TSM (IRIG, PPS, 10 MHz)
BTICard_TSMStatus	Queries the current TSM operating status
BTICard_TSMSyncConfig	Sets the input (sync) configuration for the TSM (IRIG, PPS, 10 MHz, Host)
BTICard_TSMTimerRd	Reads the current device time (ns)
BTICard_TSMTimerRelWr	Jumps the current device time by the input amount (ns)
BTICard_TSMTimerRolloverGet	Reads the current year rollover value (ns)
BTICard_TSMTimerRolloverSet	Writes the year rollover value (ns)
BTICard_TSMTimerWr	Writes the value to the TSM (Core) Timer (ns)
UTILITY Functions	
BTICard_ErrDescStr	Returns the description of the specified error value
BTICard_ValFromAscii	Creates an integer value from an ASCII string
BTICard_ValGetBits	Extracts a bit field from an integer value
BTICard_ValPutBits	Puts a bit field into an integer value
BTICard_ValToAscii	Creates an ASCII string from an integer

Table B.1—Protocol-Independent (BTICard_) Functions (Continued from Previous Page)

Device Dependence

The BTIDriver unified API supports many generations of Ballard hardware Devices. This cross-compatibility allows for application reuse when migrating from one Device to another. Each successive generation of Ballard hardware Devices tries to build upon the feature set of the previous one. Therefore, not all features supported by this API apply to all hardware Devices. Functions that depend upon a particular hardware Device will reference the products listed in Table B.2 by generation or by other functionality.

	Generation/Group				
Product	**3G**	**4G**	**5G**	**6G**	**RPC**
BUSBox (BB1xxx)	✓				
OmniBus PCI (111-xxx, 112-xxx-xxx)		✓			
OmniBus cPCI (121-xxx, 122-xxx-xxx)		✓			
OmniBus PMC (141-xxx)		✓			
OmniBus VME (152-xxx-xxx, 154-xxx-xxx-xxx-xxx)		✓			✓
OmniBusBox (162-xxx-xxx)		✓			✓
Avionics BusBox 1000 (AB1xxx)			✓		✓
Avionics BusBox 2000 (AB2xxx)			✓		✓
AB3000 Series (AB3xxx)			✓		✓
Lx1553-5, Lx429-5, PM1553-5, PM429-2, PE1000			✓		
USB 1553, USB 429/717, USB 708, USB Multi (UA1xxx)			✓		
Mx5 (Mx5x-xx-xx)			✓		
OmniBus II PCIe (212-xxx-xxx)				✓	
OmniBus II PXIe (222-xxx-xxx)				✓	
OmniBusBox II (262-xxx-xxx, 264-xxx-xxx-xxx-xxx)				✓	✓

Table B.2—Devices Grouped by Generation and Functionality

BITConfig

```
ERRVAL BTICard_BITConfig
(
  ULONG ctrlflags,          //Selects configuration options
  HCARD hCard               //Card handle
)
```

RETURNS

A negative value if an error occurs or zero if successful.

DESCRIPTION

Configures the Continuous Built-In Test (CBIT) functionality as defined by `ctrlflags` (see table below) for the card specified by `hCard`. Various functional blocks in the CBIT system can be enabled or disabled using the flags in the table below. Each of these blocks report to the BIT Status register that can be read using `BTICard_BITStatusRd`. These blocks can also be configured to generate an Event Log list entry when an error occurs.

ctrlflags	
Constant	**Description**
BITCFG_DEFAULT	Select all default settings (**bold** below)
BITCFG_MEMECC_ENABLE	Enables the memory interface to operate in ECC mode. In this mode, if a single bit error occurs, the read value will be corrected and `BITSTAT_SINGLE_BIT_ERR` will be set in the BIT Status register. If a double bit error occurs, the read value can't be corrected so the card will be stopped and `BITSTAT_DOUBLE_BIT_ERR` will be set in the BIT Status register.
BITCFG_MEMECC_DISABLE	Disables ECC operation of the memory.
BITCFG_FPGA_ENABLE	Enables monitoring for Single Event Upsets (SEU) in the FPGA configuration. In this mode, if a Single Event Upset is detected the card will be stopped and `BIT-STAT_CBIT_FPGA_ERR` will be set in the BIT Status register.
BITCFG_FPGA_DISABLE	Disables monitoring for Single Event Upsets in the FPGA configuration.
BITCFG_PROTOCOL_ENABLE	Enables CBIT in the 1553 Protocol Engine. Every 1553 word transmitted by the Card will be monitored and checked for accuracy. If the transceiver is damaged or there is a collision on the bus, the protocol error bit will be set in the BIT Status register. In addition, the `MSGERR1553_SYSTEM` bit will also be set in the 1553 Message Record and Sequential Record of the transmission that failed.
BITCFG_PROTOCOL_DISABLE	Disables CBIT in the 1553 Protocol Engine.
BITCFG_MEMECC_NOLOG	Does not generate an event log entry when the ECC decoder detects a single or double bit error in the on-card memory.
BITCFG_MEMECC_LOG	Generates an event log entry when the ECC decoder detects a single or double bit error in the on-card memory.

(Continued on next page)

(Continued from previous page)

BITCFG_FPGA_NOLOG	Does not generate an event log entry when an SEU is detected in the FPGA Configuration.
BITCFG_FPGA_LOG	Generates an event log entry when an SEU is detected in the FPGA Configuration.
BITCFG_PROTOCOL_NOLOG	Does not generate an event log entry when 1553 CBIT detects an error.
BITCFG_PROTOCOL_LOG	Generates an event log entry when 1553 CBIT detects an error.
BITCFG_CARD_STOPPED_NOLOG	Does not generate an event log entry when the Card is stopped due to CBIT Errors.
BITCFG_CARD_STOPPED_LOG	Generates an event log entry when the Card is stopped due to CBIT Errors. When a temperature or voltage sensor value exceeds the System limits for safe operation, the Card will automatically stop protocol activity to reduce power draw. The Card will also stop when an uncorrectable (double bit) error is detected in the memory or an SEU is detected in the FPGA configuration. The card is stopped to prevent the transmission of corrupted data.
BITCFG_SYSMON_NOLOG	Does not generate an event log entry when SysMon detects that a temperature sensor has exceeded the user definable thresholds.
BITCFG_SYSMON_LOG	Generates an event log entry when the SysMon detects that a temperature sensor has exceeded the user definable thresholds. User definable thresholds can be configured by calling `BTI-Card SysMonThresholdSet`.

DEVICE DEPENDENCY

Applies to 5G and 6G Devices that support BIT/SysMon functionality. Please consult the hardware manual for the Device.

WARNINGS

This function will clear any errors in the status read by `BTICard_BITStatusRd`. In order to generate Event Log list entries, the Event Log must be configured by calling `BTICard_EventLogConfig`.

SEE ALSO

`BTICard_BITStatusRd`, `BTICard_EventLogConfig`, `BTICard_SysMonThresholdSet`

BITInitiate

```
ERRVAL BTICard_BITInitiate
(
   HCARD hCard                    //Card handle
)
```

RETURNS

A negative value if an error occurs or zero if successful.

DESCRIPTION

Executes a read and write memory test on the card specified by `hCard`. When the test completes, the Card is left in the same state as after a call to `BTICard_CardReset`.

DEVICE DEPENDENCY

Applies to 5G and 6G Devices that support BIT/SysMon functionality. Please consult the hardware manual for the Device.

WARNINGS

Do not call when the Card is connected to an active databus. The function disrupts normal databus operation of the Card and the results will be unpredictable.

SEE ALSO

`BTICard_CardReset`

BITStatusClear

```
ERRVAL BTICard_BITStatusClear
(
  ULONG statval,              //Mask of bits to clear
  HCARD hCard                 //Card handle
)
```

RETURNS

A negative value if an error occurs or zero if successful.

DESCRIPTION

Each bit in `statval` clears the corresponding bit in the BIT Status register for the Card specified by `hCard`. The constants in the table below may be bitwise OR-ed together to clear each specified status bit.

Constant	Description
BITSTAT_CARD_STOPPED	CBIT fatal system error, card stopped
BITSTAT_CBIT_FPGA_ERR	CBIT FPGA Single Event Upset (SEU)
BITSTAT_CBIT_PROTOCOL_ERR	CBIT Protocol Error
BITSTAT_SINGLE_BIT_ERR	CBIT ECC Single Bit Error (correctable)
BITSTAT_DOUBLE_BIT_ERR	CBIT ECC Double Bit Error
BITSTAT_SYSMON_ERR	CBIT SysMon Error

Please refer to `BTICard_BITStatusRd` for more information on these error flags.

DEVICE DEPENDENCY

Applies to 5G and 6G Devices that support BIT/SysMon functionality. Please consult the hardware manual for the Device.

WARNINGS

None.

SEE ALSO

`BTICard_BITStatusRd`

BITStatusRd

```
ULONG BTICard_BITStatusRd
(
   HCARD hCard                    //Card handle
)
```

RETURNS

The value of the BIT Status register.

DESCRIPTION

Reads the BIT Status register of the Card specified by *hCard*. The status value can be tested using the following predefined constants:

Constant	Description
BITSTAT_CARD_STOPPED	PBIT/CBIT fatal system error, card stopped
BITSTAT_CBIT_FPGA_ERR	CBIT FPGA Single Event Upset (SEU)
BITSTAT_CBIT_PROTOCOL_ERR	CBIT Protocol Error
BITSTAT_SINGLE_BIT_ERR	PBIT/CBIT ECC Single Bit Error (corrected)
BITSTAT_DOUBLE_BIT_ERR	PBIT/CBIT ECC Double Bit Error
BITSTAT_SYSMON_ERR	CBIT SysMon Error

For SysMon errors, the position of the sensor's error bit is determined by the sensor index number.

```
If (BTICard_BITStatusRd(hCard) & (1 << index))
{
     // Handle error on the sensor at <index>
}
```

For more information on the other BIT errors and enabling/disabling reporting refer to `BTICard_BITConfig`.

DEVICE DEPENDENCY

Applies to 5G and 6G Devices that support BIT/SysMon functionality. Please consult the hardware manual for the Device.

WARNINGS

None.

SEE ALSO

`BTICard_BITStatusClear`, `BTICard_BITConfig`

CardClose

```
ERRVAL BTICard_CardClose
(
  HCARD hCard                    //Card handle
)
```

RETURNS

A negative value if an error occurs or zero if successful.

DESCRIPTION

Disables access to the specified Device and releases the associated hardware resources (e.g., memory and I/O space, interrupt number, and DMA channel). `BTICard_CardClose` closes the Device and all of its cores opened with `BTICard_CoreOpen`. This function does not stop the core(s) from operating (use `BTICard_CardStop` to stop each core).

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

Before a program terminates, this function MUST be called to release the associated hardware resources. This is especially important in Microsoft Windows operating systems.

SEE ALSO

`BTICard_CardOpen`, `BTICard_CoreOpen`, `BTICard_CardStop`

CardIsRunning

```
BOOL BTICard_CardIsRunning
(
   HCORE hCore                    //Core handle
)
```

RETURNS

TRUE if the core is still running, otherwise FALSE.

DESCRIPTION

Determines whether the core specified by *hCore* is running.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

None.

SEE ALSO

BTICard_CardStart, BTICard_CardStop

CardOpen

```
ERRVAL BTICard_CardOpen
(
  LPHCARD lpHandle,          //Pointer to the card handle
  INT cardnum                //Card number of Device
)
```

RETURNS

A negative value if an error occurs or zero if successful.

DESCRIPTION

Enables access to a Device. If `BTICard_CardOpen` finds the Device that has been assigned *cardnum*, it performs a quick hardware self-test of the Device. Since this function opens the Device and provides a card handle parameter required by `BTICard_CoreOpen` (which returns the core handle used by all other functions), this function is always the first BTIDriver function called by a program.

Card numbers are assigned to Devices by the operating system or the user. If only one Device has been installed, the system defaults the card number to zero. How the system assigns card numbers for multiple Devices and how the number can be changed by the user is OS-dependent. See the `README.TXT` file for your operating system on the distribution disk for more information. A test program for determining the card number(s) is provided on the distribution disk. The card numbers assigned to BTIDriver Devices are specific to BTIDriver-compliant Devices, so there is no conflict when non-BTIDriver-compliant Devices use those same card numbers.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

`BTICard_CardClose` must be called to release the hardware resources before the program terminates.

SEE ALSO

`BTICard_CardClose`

CardProductStr

```
LPCSTR BTICard_CardProductStr
(
  HCORE hCore                    //Core handle
)
```

RETURNS

A pointer to a character string describing the Device specified by *hCore*.

DESCRIPTION

Returns specific product information for the Device specified by *hCore*.

`BTICard_CardTypeStr` identifies the family to which a Device belongs. Inside of that family, `BTICard_CardProductStr` specifies product information such as model number, level or functionality, or configuration. Combine these functions to identify your Device.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

None.

SEE ALSO

`BTICard_CardTypeStr`

CardReset

```
VOID BTICard_CardReset
(
  HCORE hCore                    //Core handle
)
```

RETURNS

None.

DESCRIPTION

Stops and performs a hardware reset on the core specified by *hCore*. If a message is being processed, the processing is allowed to finish before the core is halted.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

Does not reset historic maximum and minimum Sysmon sensor values; to reset these values use `BTICard_SysMonClear`.

SEE ALSO

`BTICard_CardStart`, `BTICard_CardStop`, `BTICard_SysMonClear`

CardResume

```
ERRVAL BTICard_CardResume
(
   HCORE hCore                    //Core handle
)
```

RETURNS

A negative value if an error occurs or zero if successful.

DESCRIPTION

Reactivates the specified core from the point at which it was stopped (using `BTICard_CardStop`). The following table compares the difference between calling `BTICard_CardResume` and `BTICard_CardStart`:

Feature	**When `CardStart` is called..**	**When `CardResume` is called..**
Transmit Schedule	Execution starts at the start of the transmit Schedule.	Execution resumes at the point the transmit Schedule was stopped.
Event Log List	Any unread entries in the Event Log List are cleared before the core is started.	Any unread records in the Event Log List are preserved as the core is resumed.
Sequential Monitor	Any unread records in the Sequential Record are cleared before the core is started.	Any unread records in the Sequential Record are preserved as the core is resumed.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

A call to `BTICard_CardStop` must precede this function.

SEE ALSO

`BTICard_CardStart`, `BTICard_CardStop`

CardStart

```
ERRVAL BTICard_CardStart
(
  HCORE hCore                    //Core handle
)
```

RETURNS

A negative value if an error occurs or zero if successful.

DESCRIPTION

Activates all configured channels of the specified core. The Sequential Monitor and Event Log List are cleared and begin operation at the start of their allocated buffers.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

The core continues operating even after an application program ends unless `BTICard_CardStop` halts it. Even after `BTICard_CardStart`, individual channels may not transmit or receive if they are disabled or paused. See the channel configuration and control functions for each protocol.

SEE ALSO

`BTICard_CardStop, BTICard_CardIsRunning`

CardStop

```
BOOL BTICard_CardStop
(
  HCORE hCore                    //Core handle
)
```

RETURNS

TRUE if the core was active, otherwise FALSE.

DESCRIPTION

Stops operation of all the channel on the specified core. If a message is being processed, the processing is allowed to finish before the core is halted.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

None.

SEE ALSO

BTICard_CardStart, BTICard_CardIsRunning

CardSyncValid

```
USHORT BTICard_CardSyncValid
(
  HCORE hCore                    //Core handle
)
```

RETURNS

A bitwise OR-ed value of *syncmask* (refer to table) for Device's valid syncs.

syncmask	
Constant	**Description**
SYNCMASK_SYNCA	Selects discrete sync line A
SYNCMASK_PXITRIGA	Selects PXIe TRIG sync line A
SYNCMASK_SYNCB	Selects discrete sync line B
SYNCMASK_PXITRIGB	Selects PXIe TRIG sync line B
SYNCMASK_SYNCC	Selects discrete sync line C
SYNCMASK_PXITRIGC	Selects PXIe TRIG sync line C
SYNCMASK_PXISTARC	Selects PXIe STAR sync line C

DESCRIPTION

This function may be called from an application to verify Device-specific sync support.

DEVICE DEPENDENCY

Applies to 4G, 5G, and 6G Devices.

The mapping of sync lines is hardware dependent. Please consult the hardware manual for the Device.

WARNINGS

None.

SEE ALSO

BTI1553_BCSyncDefine, BTI1553_MsgSyncDefine, BTI1553_RTSyncDefine

CardTest

```
ERRVAL BTICard_CardTest
(
    USHORT level,            //Level of tests to perform
    HCORE hCore              //Core handle
)
```

RETURNS

A negative value if an error occurs or zero if successful.

DESCRIPTION

Executes a hardware test specified by `level` on the core specified by `hCore`. When the test completes, the core is left in the same state as after a `BTICard_CardReset` call.

Level	
Constant	**Description**
TEST_LEVEL_0	Tests the I/O interface of the core. The test reads and writes each I/O with a walking-bit pattern.
TEST_LEVEL_1	In addition to Level 0, this level tests the memory interface of the core. The test performs a pattern test of the RAM.
TEST_LEVEL_2	In addition to previous levels, this level tests the communication process of the core. The test performs a pattern test of the RAM using the communication process.
TEST_LEVEL_3	In addition to previous levels, this level tests the encoders and decoders of the core. Where possible, the core monitors its own transmissions to validate protocol functionality.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

This function disrupts normal operation of the core. For `TEST_LEVEL_3`, do not use this function when the core is connected to an active databus, as the results will be unpredictable.

SEE ALSO

`BTICard_CardReset`

CardTrigger

```
VOID BTICard_CardTrigger
(
   HCORE hCore                    //Core handle
)
```

RETURNS

None.

DESCRIPTION

Generates a software-simulated external trigger signal on all available trigger lines. For Devices with multiple trigger lines, `BTICard_CardTriggerEx` can be used to specify lines individually.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

None.

SEE ALSO

`BTICard_CardTriggerEx`, `BTI1553_BCTriggerDefine`, `BTI1553_MsgTriggerDefine`, `BTI1553_ErrorTriggerDefine`

CardTriggerEx

```
VOID BTICard_CardTriggerEx
(
   USHORT trigmask,          //Line(s) to trigger via software
   HCORE hCore               //Core handle
)
```

RETURNS

None.

DESCRIPTION

Simulates an external trigger signal on the trigger input line(s) specified by *trigmask*. The constants in the table below may be bitwise OR-ed together to trigger multiple lines.

trigmask	
Constant	**Description**
TRIGMASK_TRIGA	Selects trigger line A
TRIGMASK_TRIGB	Selects trigger line B
TRIGMASK_TRIGC	Selects trigger line C

DEVICE DEPENDENCY

Though this function is intended for 4G, 5G, and 6G Devices, which have multiple trigger lines, using a trigmask value of TRIGMASK_TRIGA on 3G Devices produces the same result as BTICard_CardTrigger.

WARNINGS

None.

SEE ALSO

BTICard_CardTrigger, BTI1553_BCTriggerDefine, BTI1553_MsgTriggerDefine, BTI1553_ErrorTriggerDefine, BTICard_CardTriggerValid

CardTriggerValid

```
USHORT BTICard_CardTriggerValid
(
  HCORE hCore                    //Core handle
)
```

RETURNS

A bitwise OR-ed value of *trigmask* (refer to table) for Device's valid triggers.

trigmask	
Constant	**Description**
TRIGMASK_TRIGA	Selects discrete trigger line A
TRIGMASK_PXITRIGA	Selects PXIe TRIG trigger line A
TRIGMASK_PXISTARA	Selects PXIe STAR trigger line A
TRIGMASK_TRIGB	Selects discrete trigger line B
TRIGMASK_PXITRIGB	Selects PXIe TRIG trigger line B
TRIGMASK_PXISTARB	Selects PXIe STAR trigger line B
TRIGMASK_TRIGC	Selects discrete trigger line C
TRIGMASK_PXITRIGC	Selects PXIe TRIG trigger line C
TRIGMASK_PXISTARC	Selects PXIe STAR trigger line C

DESCRIPTION

This function may be called from an application to verify Device-specific trigger support.

DEVICE DEPENDENCY

Applies to 4G, 5G, and 6G Devices.

PXIe Triggers are only available on PXIe Devices..

WARNINGS

None.

SEE ALSO

BTI1553_BCTriggerDefine, BTI1553_ErrorTriggerDefine, BTI1553_MsgTriggerDefine

CardTypeStr

```
LPCSTR BTICard_CardTypeStr
(
   HCORE hCore                    //Core handle
)
```

RETURNS

A pointer to a character string describing the Device specified by *hCore*.

DESCRIPTION

Returns the type of Device specified by *hCore*.

BTICard_CardTypeStr identifies the family to which a Device belongs. Inside of that family, BTICard_CardProductStr specifies product information such as model number, level of functionality, or configuration. Combine these functions to identify your Device.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

None.

SEE ALSO

BTICard_CardProductStr

CoreOpen

```
ERRVAL BTICard_CoreOpen
(
  LPHCORE lphCore,          //Pointer to a core handle
  INT corenum,              //Core number
  HCARD hCard               //Card handle
)
```

RETURNS

A negative value if an error occurs or zero if successful.

DESCRIPTION

Enables access to a core (the presence of multiple cores is Device-dependent). `BTICard_CardOpen` must first be called to obtain the card handle (*hCard*). `BTICard_CoreOpen` finds the core on the Device specified by *hCard* that has been assigned *corenum*, and returns a handle to that core. `BTICard_CoreOpen` must be called for each core that you wish to access in your program. `BTICard_CardClose` will close all cores opened with `BTICard_CoreOpen`.

If you pass the card handle to a function (such as a channel function) instead of a core handle, it will only access the first (or only) core.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

`BTICard_CardOpen` must be called before this function. `BTICard_CoreOpen` must be called for each core that you wish to access in your program.

SEE ALSO

`BTICard_CardOpen`, `BTICard_CardClose`

ErrDescStr

```
LPCSTR BTICard_ErrDescStr
(
  ERRVAL errval,             //An error value
  HCORE hCore                //Core handle
)
```

RETURNS

A pointer to a character string describing the error specified by *errval*.

DESCRIPTION

Describes the error value specified by *errval*.

All functions of type ERRVAL return a negative value if an error occurs, or zero if successful. BTICard_ErrDescStr returns a description of the specified error value.

Note: This is a utility function and does not access the Device hardware.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

None.

SEE ALSO

ERRVAL type functions

EventLogConfig

```
ERRVAL BTICard_EventLogConfig
(
   USHORT ctrlflags          //Selects the configuration options
   USHORT count              //Number of entries in the Event Log List
   HCORE hCore               //Core handle
)
```

RETURNS

A negative value if an error occurs or zero if successful.

DESCRIPTION

Configures and enables the Event Log List of the core specified by *hCore*. The maximum number of entries that may be contained in the Event Log List is set by *count*. Ctrlflags can be one of the following constants:

ctrlflags	
Constant	**Condition**
LOGCFG_DEFAULT	Selects all default **(bold)** settings
LOGCFG_ENABLE	Enables the Event Log List
LOGCFG_DISABLE	Disables the Event Log List

DEVICE DEPENDENCY

The size of the Event Log for 5G and 6G Devices is always 256 entries regardless of count.

WARNINGS

None.

SEE ALSO

BTICard_EventLogRd, BTICard_EventLogStatus

EventLogRd

```
ULONG BTICard_EventLogRd
(
   LPUSHORT typeval,        //Pointer to variable to receive type value
   LPULONG infoval,         //Pointer to variable to receive info value
   LPINT channel,           //Pointer to variable to receive channel value
   HCORE hCore              //Core handle
)
```

RETURNS

The address of the entry in the Event Log List, or zero if it is empty and there are no entries to read.

DESCRIPTION

Reads the next entry from the Event Log List and advances the pointer. The type of event and channel that generated the entry are passed through *typeval* and *channel*. An information word associated with the event is passed through *infoval*.

The value of *typeval* determines the meaning of the *infoval* value (see table below). Note that the Event Log List records events from all protocols implemented on the specified core. The first two event types, regarding the Sequential Record, are protocol-independent. The rest of the table is subdivided by protocol.

	typeval	**Description**	***infoval***	**Refer to…**
General	EVENTTYPE_SEQFULL	Sequential Record full (halted) or overwritten	Address of last entry	BTICard_SeqConfig
	EVENTTYPE_SEQFREQ	n^{th} entry (user-specified)	Address of last entry	BTICard_SeqConfig
	EVENTTYPE_BITERROR	BIT system detected an error	BIT Status at the time of the Error	BTICard_BITConfig
MIL-STD-1553	EVENTTYPE_1553MSG	Message processed	Address of the Message structure	BTI1553_BCCreateMsg BTI1553_RTCreateMsg BTI1553_BCCreateList BTI1553_RTCreateList
	EVENTTYPE_1553OPCODE	BC Schedule encountered a LOG command	User-assigned tag value	BTI1553_BCSchedLog
	EVENTTYPE_1553HALT	BC Schedule encountered a HALT command	Address of the Schedule entry	BTI1553_BCSchedHalt
	EVENTTYPE_1553PAUSE	BC Schedule encountered a PAUSE command	Address of the Schedule entry	BTI1553_BCSchedPause
	EVENTTYPE_1553LIST	List Buffer empty/ full (underflow or overflow)	List Buffer address	BTI1553_BCCreateList BTI1553_RTCreateList
Serial	EVENTTYPE_422TXTHRESHOLD	TX FIFO under threshold	None	BTI422_FIFOConfigTx
	EVENTTYPE_422TXFIFO	TX FIFO underflow	None	BTI422_FIFOConfigTx
	EVENTTYPE_422RXTHRESHOLD	RX FIFO over threshold	None	BTI422_FIFOConfigRx
	EVENTTYPE_422RXFIFO	RX FIFO overflow	None	BTI422_FIFOConfigRx
	EVENTTYPE_422RXERROR	RX error	None	BTI422_FIFOConfigRx

(Continued on next page)

(Continued from previous page)

	typeval	Description	*infoval*	Refer to…
ARINC 429	EVENTTYPE_429MSG	Message received or transmitted	Address of the Message Record	BTI429_MsgCreate BTI429_FilterDefault BTI429_FilterSet
	EVENTTYPE_429OPCODE	A transmit Schedule encountered a LOG command	User-assigned tag value	BTI429_SchedLog
	EVENTTYPE_429HALT	A transmit Schedule encountered a HALT command	Address of the Schedule entry	BTI429_ChConfig BTI429_SchedHalt
	EVENTTYPE_429PAUSE	A transmit Schedule encountered a PAUSE command	Address of the Schedule entry	BTI429_ChConfig BTI429_SchedPause
	EVENTTYPE_429LIST	List Buffer empty or full (underflow or overflow)	List Buffer address	BTI429_ListAsyncCreate BTI429_ListRcvCreate BTI429_ListXmtCreate
	EVENTTYPE_429ERR	A decoder error was detected	Address of the message that contained the error	BTI429_ChConfig
ARINC 717	EVENTTYPE_717WORD	Processed 717 word	Word address	BTI717_SubFrmWordConfig BTI717_SuperFrmWordConfig
	EVENTTYPE_717SUBFRM	Processed 717 subframe	Subframe number	BTI717_SubFrmWordConfig BTI717_SuperFrmWordConfig
	EVENTTYPE_717SYNCERR	717 receive channel lost sync	Channel number	BTI717_ChConfig
ARINC 708	EVENTTYPE_708MSG	Message received/ transmitted	Message index	BTI708_RcvBuild BTI708_XmtBuild
EBR-1553	EVENTTYPE_EBRMSG	Message processed	Address of the Message structure	BTIEBR_BCCreateMsg BTIEBR_RTCreateMsg BTIEBR_BCCreateList BTIEBR_RTCreateList
	EVENTTYPE_EBROPCODE	BC Schedule encountered a LOG command	User-assigned tag value	BTIEBR_BCSchedLog
	EVENTTYPE_EBRHALT	BC Schedule encountered a HALT command	Address of the Schedule entry	BTIEBR_BCSchedHalt
	EVENTTYPE_EBRPAUSE	BC Schedule encountered a PAUSE command	Address of the Schedule entry	BTIEBR_BCSchedPause
	EVENTTYPE_EBRLIST	List Buffer empty/ full (underflow or overflow)	List Buffer address	BTIEBR_BCCreateList BTIEBR_RTCreateList
Discrete IO	EVENTTYPE_DIOEDGE	An edge transition occurred	8-bit mask specifying which I/O had an edge, 8-bit state of the bank I/O after edge.	BTIDIO_BankConfig BTIDIO_DiscreteConfig
	EVENTTYPE_DIOFAULT	A fault occurred	8-bit mask specifying which I/O had a fault	BTIDIO_BankConfig BTIDIO_DiscreteConfig

DEVICE DEPENDENCY

The value passed back in `channel` may not be valid for all types of events. If an event does not have an associated channel value, it is filled in with the value FFh.

WARNINGS

This function should be preceded by a call to `BTICard_EventLogConfig`. To use this function, it is not necessary to install an interrupt handler.

SEE ALSO

`BTICard_EventLogConfig`

EventLogStatus

```
INT BTICard_EventLogStatus
(
   HCORE hCore                    //Core handle
)
```

RETURNS

The status value of the Event Log List.

DESCRIPTION

Checks the status of the Event Log List without removing an entry. The status value can be tested using the predefined constants below:

Constant	Description
STAT_EMPTY	Event Log List is empty
STAT_PARTIAL	Event Log List is partially filled
STAT_FULL	Event Log List is full
STAT_OFF	Event Log List is off

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

When the buffer is full it wraps around and overwrites previous entries.

SEE ALSO

BTICard_EventLogConfig, BTICard_EventLogRd

ExtDIOMonConfig

```
ERRVAL BTICard_ExtDIOMonConfig
(
   USHORT risemask,          //Discrete bitmask to monitor rising edges
   USHORT fallmask,          //Discrete bitmask to monitor falling edges
   INT banknum,              //Bank number of DIO
   HCORE hCore               //Core handle
)
```

RETURNS

A negative value if an error occurs or zero if successful.

DESCRIPTION

Enables Sequential Monitoring on specific transitions of discrete inputs. Discrete inputs are sampled at a minimum rate of 100 kHz and compared to the previously sampled values. If the digital values of any discretes specified in the `risemask` have transitioned from a zero to a one, or the digital values of any discretes specified in the `fallmask` have transitioned from a one to a zero, then a Sequential Record will be created. To disable previously-enabled monitor settings, call `BTICard_ExtDIOMonConfig` again with *risemask* and *fallmask* both set to zero.

The *risemask* and *fallmask* are a bit mask specifying up to 16 discrete input signals. For `banknum` 0, the LSB of each value corresponds to `dionum` 1 and the MSB corresponds to *dionum* 16. See table below for `dionum` higher than 16.

***dionum* to *banknum* and *bitmask* Translation**			
***dionum* Range**	***banknum***	***bitmask* LSB**	***bitmask* MSB**
1-16	0	1	16
17-32	1	17	32
33-48	2	33	48
49-64	3	49	64

DEVICE DEPENDENCY

Applies to 5G and 6G Devices. The mapping of `dionum` to physical discrete I/O is hardware dependent. Please consult the hardware manual for the Device.

WARNINGS

By default, avionics discretes are active low. A grounded avionics discrete will have a digital value of one. If the polarity for that discrete input has been changed, the digital value compared for rising and falling edges will be reversed.

SEE ALSO

`BTICard_SeqFindNextDIO`

ExtDIOEnWr

```
BOOL BTICard_ExtDIOEnWr
(
   INT dionum,                //Specifies the DIO number
   BOOL dioval,               //The value to set
   BOOL dioen,                //The output enable to set
   HCORE hCore                //Core handle
)
```

RETURNS

None.

DESCRIPTION

Sets the digital I/O pin specified by `dionum` to the value specified by `dioval` with an output enable specified by `dioen`. A `dioen` of zero tri-states the pin, and a `dioen` of one sets the pin to the value specified by `dioval`. A `dioval` of zero sets the pin to inactive, and a *`dioval`* of one sets the pin to active.

DEVICE DEPENDENCY

Only applies to 6G Devices. If the DIO referenced by *`dionum`* cannot be tri-stated, then *`dioen`* will be ignored.

The mapping of *`dionum`* to physical discrete I/O is hardware dependent. Please consult the hardware manual for the Device.

WARNINGS

Some discretes are avionics discretes while others are digital I/O discretes. When using the digital I/O as an output (*`dioen`* set to one), do not drive the digital I/O pin from an external source as this may damage the Device. Please consult the hardware manual for the Device.

SEE ALSO

`BTICard_ExtDIORd, BTICard_ExtDIOWr`

ExtDIORd

```
BOOL BTICard_ExtDIORd
(
   INT dionum,                    //Specifies the DIO number
   HCORE hCore                    //Core handle
)
```

RETURNS

Status of the digital I/O pin. Returns a zero if the pin is inactive or a one if the pin is active.

DESCRIPTION

Reads the status of the digital I/O pin specified by *dionum*.

DEVICE DEPENDENCY

The mapping of *dionum* to physical discrete I/O is hardware dependent. Please consult the hardware manual for the Device.

WARNINGS

None.

SEE ALSO

`BTICard_ExtDIOWr`, `BTICard_ExtDIOENWr`

ExtDIOWr

```
VOID BTICard_ExtDIOWr
(
   INT dionum,                //Specifies the DIO number
   BOOL dioval,               //The value to set
   HCORE hCore                //Core handle
)
```

RETURNS

None.

DESCRIPTION

Sets the digital I/O pin specified by `dionum` to the value specified by `dioval`. A `dioval` of zero sets the pin to inactive, and a `dioval` of one sets the pin to active.

DEVICE DEPENDENCY

The mapping of `dionum` to physical discrete I/O is hardware dependent. Please consult the hardware manual for the Device.

WARNINGS

Some discretes are avionics discretes while others are digital I/O discretes. When using the digital I/O as an output (as this function does), do not drive the digital I/O pin from an external source as this may damage the Device. Please consult the hardware manual for the Device.

SEE ALSO

`BTICard_ExtDIORd, BTICard_ExtDIOENWr`

ExtLEDRd

```
BOOL BTICard_ExtLEDRd
(
  HCORE hCore                    //Core handle
)
```

RETURNS

Returns a zero if the LED is off or a one if the LED is on.

DESCRIPTION

Reads the state of the on-board LED.

DEVICE DEPENDENCY

4G, 5G, and 6G Devices have a user-controlled LED for each core. For all other Devices, please refer to the hardware manual.

WARNINGS

None.

SEE ALSO

`BTICard_ExtLEDWr`

ExtLEDWr

```
VOID BTICard_ExtLEDWr
(
   BOOL ledval,                  //New state of the LED
   HCORE hCore                   //Core handle
)
```

RETURNS

None.

DESCRIPTION

Sets the state of the onboard LED. An *ledval* of zero turns the LED off, and an *ledval* of one turns the LED on.

DEVICE DEPENDENCY

4G, 5G, and 6G Devices have a user-controlled LED for each core. For all other Devices, please refer to the hardware manual.

WARNINGS

None.

SEE ALSO

BTICard_ExtLEDRd

ExtStatusLEDRd

```
VOID BTICard_ExtStatusLEDRd
(
  LPINT ledval,          //Pointer to variable to receive LED state
  LPINT ledcolor,        //Pointer to variable to receive LED color
  HCORE hCore            //Core handle
)
```

RETURNS

None.

DESCRIPTION

Reads the state of the onboard Status LED. A zero value will be passed to *ledval* if the LED is off, and a one value if the LED is on.

DEVICE DEPENDENCY

4G Devices have a red status LED.

BUSBox BB1xxx Devices have a multi-color Status LED. The color state of that LED can be read through *ledcolor*. A zero value indicates a red color, and a one value indicates a green color.

For all other Devices, please refer to the Device specific hardware manual.

WARNINGS

None.

SEE ALSO

BTICard_ExtStatusLEDWr, BTICard_ExtLEDRd

ExtStatusLEDWr

```
VOID BTICard_ExtStatusLEDWr
(
  BOOL ledval,                //New state of the LED
  BOOL ledcolor,              //New color of the LED
  HCORE hCore                 //Core handle
)
```

RETURNS

None.

DESCRIPTION

Sets the state of the onboard Status LED. An *ledval* of zero turns the LED off and an *ledval* of one turns the LED on.

DEVICE DEPENDENCY

4G Devices have a red status LED.

BUSBox BB1xxx Devices have a multi-color Status LED. The color state of that LED can be controlled through *ledcolor*. A zero value indicates a red color, and a one value indicates a green color.

RPC Devices have a red Status LED that indicates a succesful booting of the Device. Afterwards, the Status LED can be controlled with this function.

5G USB Adapter Devices have a red Status LED that is both user-controllable and can indicate that a MIL-STD-1553 protocol error has occurred. The on/off state of the Status LED is controlled through *ledval* when the *ledcolor* value is set to zero. The enabling/disabling of the 1553 protocol error indication is controlled through *ledval* when the *ledcolor* value is set to one. When this feature is enabled, the Status LED will be automaticaly set by Device hardware when a MIL-STD 1553 error occurs. The Status LED can then be cleared by calling this function.

For all other Devices, please refer to the Device specific hardware manual.

WARNINGS

None.

SEE ALSO

BTICard_ExtStatusLEDRd, BTICard_ExtLEDWr

IntClear

```
VOID BTICard_IntClear
(
  HCORE hCore                    //Core handle
)
```

RETURNS

None.

DESCRIPTION

Clears the interrupt from the core so it is ready for the next interrupt. Typically, the user's worker thread calls this function. Because the core cannot process another interrupt until the current one is cleared, `BTICard_IntClear` should be called after each interrupt has been processed.

DEVICE DEPENDENCY

Applies to all Devices except those controlled via RPC.

WARNINGS

If another interrupt occurs before `BTICard_IntClear` is called, the new interrupt is lost.

SEE ALSO

`BTICard_IntInstall`, `BTICard_IntUninstall`

IntInstall

```
ERRVAL BTICard_IntInstall
(
  LPVOID hEvent,                 //Handle of a WIN32 event object
  HCORE hCore                    //Core handle
)
```

RETURNS

A negative value if an error occurs, or zero if successful.

DESCRIPTION

`BTICard_IntInstall` associates a `WIN32` event object with interrupts from the core specified by *`hCore`*. If the function is successful, any interrupt issued from *hCore* causes the event object specified by *`hEvent`* to be set to the signaled state.

The user's application must ensure that the event object is set to the unsignaled state before the core issues the first interrupt. This can be done when creating the event object with the `WIN32` API function `CreateEvent`.

Create a worker thread, which immediately goes to sleep by calling a `WIN32` API wait function like `WaitForSingleObject`. When the Device issues an interrupt, the event object is signaled, and the worker thread wakes up to respond to the interrupt. The interrupt is generated whenever an entry is written to the Event Log List.

It is the user's responsibility to clear the interrupt from the core by calling `BTICard_IntClear` in the worker thread. Note that event objects are never polled.

> *Note: `BTICard_IntInstall` should be called separately for each core on the Device, and there should be separate interrupt service threads for each core.*

DEVICE DEPENDENCY

Applies to all Devices except those controlled via RPC.

WARNINGS

If this function is used, `BTICard_IntUninstall` MUST be called before the user's program terminates. It removes the association between the Device and the event object.

SEE ALSO

`BTICard_EventLogRd`, `BTICard_IntUninstall`

IntUninstall

```
ERRVAL BTICard_IntUninstall
(
  HCORE hCore                    //Core handle
)
```

RETURNS

A negative value if an error occurs or zero if successful.

DESCRIPTION

Removes the association between interrupts from the core specified by *hCore* and `WIN32` event objects created by the `BTICard_IntInstall` function. The Event Log List of the core remains unchanged.

DEVICE DEPENDENCY

Applies to all Devices except those controlled via RPC.

WARNINGS

This function must be called before the user’s application terminates if `BTICard_IntInstall` has been called.

SEE ALSO

`BTICard_IntInstall`

IRIGConfig

```
ERRVAL BTICard_IRIGConfig
(
  ULONG ctrlflags,          //Selects IRIG configuration options
  HCORE hCore               //Core handle
)
```

RETURNS

A negative value if an error occurs or zero if successful.

DESCRIPTION

Configures the onboard IRIG circuit as defined by *ctrlflags* (see table below) for the core specified by *hCore*. IRIG timers are configured and enabled for each core independently.

Ctrlflags			
Constant	**Description**	**Rcv**	**Xmt**
IRIGCFG_DEFAULT	Select all default settings (**bold** below)	✓	✓
IRIGCFG_ENABLE	Enables the IRIG timer	✓	✓
IRIGCFG_DISABLE	Disables the IRIG timer	✓	✓
IRIGCFG_SPEEDB	Enables IRIGB timing	✓	✓
IRIGCFG_SPEEDA	Enables IRIGA timing	✓	✓
IRIGCFG_INTERNAL	Use internal IRIG bus	✓	✓
IRIGCFG_EXTERNAL	Use external IRIG bus	✓	✓
IRIGCFG_SLAVE	IRIG timer for this core is a slave	✓	
IRIGCFG_MASTER	IRIG timer for this core is the master		✓
IRIGCFG_PPS	Enables pulse per second signaling	✓	✓
IRIGCFG_PWM	Enables pulse width modulated signaling	✓	✓
IRIGCFG_AM	Enables amplitude modulated signaling	5G 6G	6G

DEVICE DEPENDENCY

Applies only to 4G, 5G, and 6G Devices. Amplitude modulated decoding is only supported by 5G and 6G Devices. Amplitude modulated transmission is only supported by 6G Devices. When IRIG is enabled on a 4G Device, time-tags in message structures and Sequential Records will be in BCD format.

WARNINGS

Rounding is used when the IRIGCFG_PPS option is enabled. Values get rounded up when above 500 ms and are rounded down when below 500 ms if signaling is configured for pulse per second (PPS).

SEE ALSO

BTICard_IRIGRd,BTICard_IRIGWr,
BTICard_IRIGInputThresholdSet,
BTICard_IRIGInputThresholdGet,
BTICard_TimerStatus

IRIGFieldGet??

```
ULONG BTICard_IRIGFieldGetDays
ULONG BTICard_IRIGFieldGetHours
ULONG BTICard_IRIGFieldGetMin
ULONG BTICard_IRIGFieldGetSec
ULONG BTICard_IRIGFieldGetMillisec
ULONG BTICard_IRIGFieldGetMicrosec
(
   ULONG irigvalh,          //Upper 32 bits of the 64-bit BCD IRIG time-tag
   ULONG irigvall,          //Lower 32 bits of the 64-bit BCD IRIG time-tag
)
```

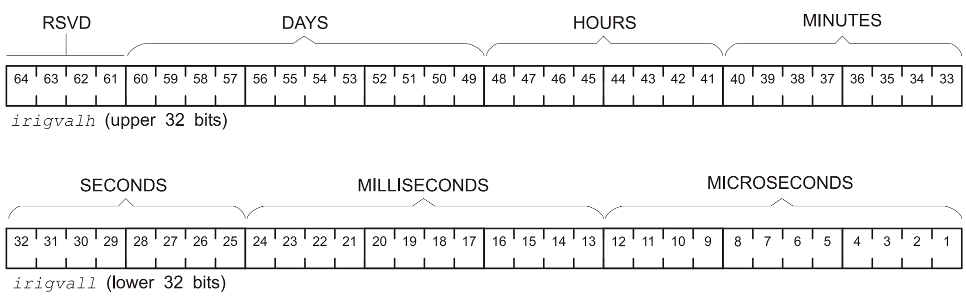
RETURNS

The integer value of the IRIG field for which the function is named.

DESCRIPTION

Extracts the specified BCD field from the 64-bit IRIG time-tag, converts it to an integer, and returns the integer.

An IRIG time-tag is divided into the following BCD fields:



Note: These are utility functions and do not access any Device hardware.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

None.

SEE ALSO

`BTICard_IRIGFieldPut??`

IRIGFieldPut??

```
VOID BTICard_IRIGFieldPutDays
VOID BTICard_IRIGFieldPutHours
VOID BTICard_IRIGFieldPutMin
VOID BTICard_IRIGFieldPutSec
VOID BTICard_IRIGFieldPutMillisec
VOID BTICard_IRIGFieldPutMicrosec
(
   ULONG value,              //Field value to write to the BCD IRIG time
   LPULONG irigvalh,         //Pointer to a variable for the upper 32 bits
   LPULONG irigvall,         //Pointer to a variable for the lower 32 bits
)
```

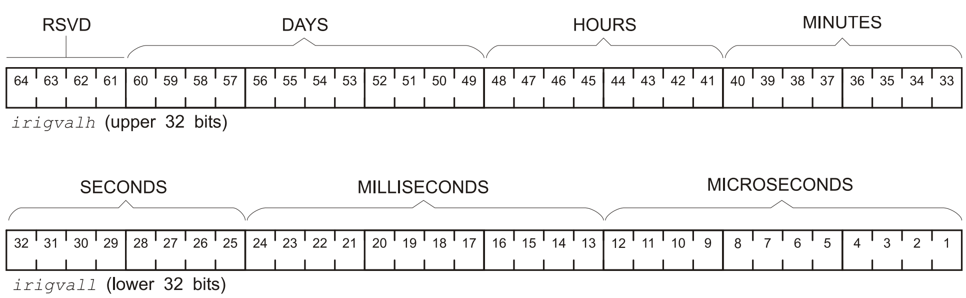
RETURNS

Nothing.

DESCRIPTION

Converts an integer (`value`) to BCD and inserts the BCD value into the specified field in the 64-bit IRIG time-tag.

An IRIG time-tag is divided into the following BCD fields:



Note: These are utility functions and do not access any Device hardware.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

None.

SEE ALSO

`BTICard_IRIGFieldGet??`

IRIGInputThresholdGet

```
ERRVAL BTICard_IRIGInputThresholdGet
(
   LPUSHORT dacval,          //Pointer to a digital-analog converter value
   HCORE hCore               //Core handle
)
```

RETURNS

A negative value if an error occurs or zero if successful.

DESCRIPTION

Gets the threshold of the IRIG input circuitry. The parameter *dacval* represents a linear voltage scale from 0x0000 to 0Xfff0 see product manual for voltage range. Only the most significant 12 bits are used.

DEVICE DEPENDENCY

Applies to all 5G and 6G Devices, except AB1xxx Devices.

WARNINGS

Should only be used when IRIG circuit is configured for Amplitude Modulated (AM) input signaling. Changing the input threshold affects both AM and Pulse Width Modulated decoding.

SEE ALSO

`BTICard_IRIGConfig`, `BTICard_IRIGInputThresholdSet`

IRIGInputThresholdSet

```
ERRVAL BTICard_IRIGInputThresholdSet
(
   USHORT dacval,          //Digital-analog converter value
   HCORE hCore             //Core handle
)
```

RETURNS

A negative value if an error occurs or zero if successful.

DESCRIPTION

Sets the threshold of the IRIG input circuitry. The parameter *dacval* represents a linear voltage scale from 0x0000 to 0Xfff0 see product manual for voltage range. Only the most significant 12 bits are used.

DEVICE DEPENDENCY

Applies to all 5G and 6G Devices, except AB1xxx Devices.

WARNINGS

Should only be used when IRIG circuit is configured for Amplitude Modulated (AM) input signaling. Changing the input threshold affects both AM and Pulse Width Modulated decoding.

SEE ALSO

BTICard_IRIGConfig, BTICard_IRIGInputThresholdGet

IRIGRd

```
ERRVAL BTICard_IRIGRd
(
  LPBTIIRIGTIME irigtime,   //Pointer to an IRIG time structure
  HCORE hCore               //Core handle
)
```

RETURNS

A negative value if an error occurs or zero if successful.

DESCRIPTION

Reads the current value of the IRIG timer from the core specified by `hCore` and puts the value into the `irigtime` structure.

***BTIIRIGTIME* structure**		
Field	**Size**	**Description**
days	USHORT	Day of the year (see Device Dependency)
hours	USHORT	Hours after midnight (0–23)
min	USHORT	Minutes after the hour (0–59)
sec	USHORT	Seconds after the minute (0–59)
msec	USHORT	Milliseconds after the second (0–999)
usec	USHORT	Microseconds after the millisecond (0–999)

Note: To read the binary timer, see `BTICard_TimerRd`.

DEVICE DEPENDENCY

Applies only to 4G, 5G, and 6G Devices.

For 4G Devices, the days count 0-365 with 0=January 1st.

For 5G Devices, the days count 0-364 with 0=January 1st.

For 6G Devices, the days count to a user programmable rollover.

WARNINGS

None.

SEE ALSO

`BTICard_IRIGWr, BTICard_IRIGConfig, BTICard_TimerRd`

IRIGSyncStatus

```
BOOL BTICard_IRIGSyncStatus
(
   HCORE hCore                      //Core handle
)
```

RETURNS

TRUE if the IRIG timer is synchronized or FALSE if it is not synchronized.

DESCRIPTION

Reports the status of the IRIG timer on *hCore* in synchronizing to the signal on the IRIG bus.

DEVICE DEPENDENCY

Applies only to 4G, 5G, and 6G Devices.

WARNINGS

None.

SEE ALSO

`BTICard_IRIGConfig`, `BTICard_IRIGWr`, `BTICard_IRIGRd`

IRIGTimeBCDToBin

```
VOID BTICard_IRIGTimeBCDToBin
(
   LPULONG timevalh,      //Pointer to upper 32 bits of binary time µs value
   LPULONG timevall,      //Pointer to lower 32 bits of binary time µs value
   ULONG irigvalh,        //Upper 32 bits of BCD IRIG time value
   ULONG irigvall,        //Lower 32 bits of BCD IRIG time value
)
```

RETURNS

None.

DESCRIPTION

Converts the 64 bit IRIG BCD time value to the equivalent binary time value (in microseconds).

Note: This is a utility function and does not access any Device hardware.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

None.

SEE ALSO

BTICard_IRIGTimeBinToBCD,
BTICard_IRIGTimeNanoBinToBCD,
BTICard_IRIGTimeBCDToNanoBin

IRIGTimeBCDToNanoBin

```
VOID BTICard_IRIGTimeBCDToNanoBin
(
  LPULONG timevalh,      //Pointer to upper 32 bits of binary time ns value
  LPULONG timevall,      //Pointer to lower 32 bits of binary time ns value
  ULONG irigvalh,        //Upper 32 bits of BCD IRIG time value
  ULONG irigvall,        //Lower 32 bits of BCD IRIG time value
)
```

RETURNS

None.

DESCRIPTION

Converts the 64 bit IRIG BCD time value to the equivalent binary time value (in nanoseconds).

Note: This is a utility function and does not access any Device hardware.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

None.

SEE ALSO

BTICard_IRIGTimeNanoBinToBCD, BTICard_IRIGTimeBinToBCD, BTICard_IRIGTimeBCDToBin

IRIGTimeBinToBCD

```
VOID BTICard_IRIGTimeBinToBCD
(
   LPULONG irigvalh,  //Pointer to upper 32 bits of BCD IRIG time value
   LPULONG irigvall,  //Pointer to lower 32 bits of BCD IRIG time value
   ULONG timevalh,    //Upper 32 bits of binary time µs value
   ULONG timevall,    //Lower 32 bits of binary time µs value
)
```

RETURNS

None.

DESCRIPTION

Converts the binary time value (in microseconds) to the equivalent 64 bit IRIG BCD time value.

Note: This is a utility function and does not access any Device hardware.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

None.

SEE ALSO

BTICard_IRIGTimeBCDToBin,
BTICard_IRIGTimeBCDToNanoBin,
BTICard_IRIGTimeNanoBinToBCD

IRIGTimeNanoBinToBCD

```
VOID BTICard_IRIGTimeNanoBinToBCD
(
   LPULONG irigvalh,   //Pointer to upper 32 bits of BCD IRIG time value
   LPULONG irigvall,   //Pointer to lower 32 bits of BCD IRIG time value
   ULONG timevalh,     //Upper 32 bits of binary time ns value
   ULONG timevall,     //Lower 32 bits of binary time ns value
)
```

RETURNS

None.

DESCRIPTION

Converts the binary time value (in nanoseconds) to the equivalent 64 bit IRIG BCD time value.

Note: This is a utility function and does not access any Device hardware.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

None.

SEE ALSO

BTICard_IRIGTimeBCDToNanoBin, BTICard_IRIGTimeBCDToBin, BTICard_IRIGTimeBinToBCD

IRIGWr

```
ERRVAL BTICard_IRIGWr
(
    LPBTIIRIGTIME irigtime,    //Pointer to an IRIG time array
    HCORE hCore                //Core handle
)
```

RETURNS

A negative value if an error occurs or zero if successful.

DESCRIPTION

Sets the IRIG timer to `irigtime` on the core specified by `hCore`.

***IRIGTIME* structure**		
Field	**Size**	**Description**
days	USHORT	Day of the year (see Device Dependency)
hours	USHORT	Hours after midnight (0–23)
min	USHORT	Minutes after hour (0–59)
sec	USHORT	Seconds after minute (0–59)
msec	USHORT	Milliseconds after minute (0–999)
usec	USHORT	Microseconds after millisecond (0–999)

Note: To write to the binary timer, see `BTICard_TimerWr`.

DEVICE DEPENDENCY

Applies only to 4G, 5G, and 6G Devices.

For 4G Devices, the days count 0-365 with 0=January 1st.

For 5G Devices, the days count 0-364 with 0=January 1st.

For 6G Devices, the days count to a user programmable rollover.

WARNINGS

None.

SEE ALSO

`BTICard_IRIGRd, BTICard_IRIGConfig, BTICard_TimerWr`

PXIStatus

```
ERRVAL BTICard_PXIStatus
(
   LPULONG infoptr,          //Pointer to variable to receive info
   USHORT infotype,          //Info status type
   HCORE hCore               //Core handle
)
```

RETURNS

A negative value if an error occurs or zero if successful.

DESCRIPTION

The desired status is returned through *infoptr* by setting *infotype* to one of the following parameters:

Status Selection	
infotype	**infoptr**
PXITYPE_GEOADDR	Geographical Address
PXITYPE_CLKSEL	PXIe Clock Selection 0 = PXIe_CLK100 1 = Onboard 100 MHz Oscillator
PXITYPE_TRIGVERS	Version of the BTI PXIe Trigger Engine
PXITYPE_OUTEN	Bitmask of PXI_TRIG drive status

DEVICE DEPENDENCY

Only applies to 6G Devices with PXIe functionality.

Please consult the Device hardware manual for more information.

WARNINGS

None.

SEE ALSO

None.

SeqBlkRd

```
ULONG BTICard_SeqBlkRd
(
   LPUSHORT seqbuf,          //Pointer to Sequential Record buffer
   ULONG bufcount,           //Size of the buffer (in 16-bit words)
   LPULONG blockcount,       //Number of records copied to the buffer
   HCORE hCore               //Core handle
)
```

RETURNS

The number of 16-bit words copied to the user-supplied buffer.

DESCRIPTION

Copies as many available complete records as possible from the Sequential Record on the core to a buffer (`seqbuf`). The function returns the number of 16-bit words copied and the puts the number of records copied into `blockcount`. The larger the buffer size (`bufcount`) the greater the number of records that can be copied by a single call to this function. The data read is effectively removed from the Sequential Record on the core. This allows an infinite amount of data to be gathered as long as this function (or one of the others in the table below) is called frequently enough to prevent the Sequential Record on the core from overflowing.

There are four functions that read from the Sequential Record. `BTICard_SeqRd` reads a single record; `BTICard_SeqBlkRd`, `BTICard_SeqCommRd`, and `BTICard_SeqDMARd` read multiple records. Any one of these functions may be used in most applications. The difference lies in their speed of execution under different conditions and availability by Device. The table below compares the four functions that read from the Sequential Record and gives some rationale for selecting one over another:

Function	Function Overhead	Per Record Overhead	Use When…
`BTICard_SeqRd`	low	n/a	Expect one or no records per function call
`BTICard_SeqBlkRd`	low	high	Expect a small number of records per function call
`BTICard_SeqCommRd`	high	low	Expect a large number of records per function call
`BTICard_SeqDMARd`	low	low	Expect a large number of records per function call (Device-dependent)

DEVICE DEPENDENCY

On 3G Devices, `BTICard_SeqRd`, `_SeqBlkRd`, and `_SeqCommRd` all read multiple records in the same manner.

WARNINGS

None.

SEE ALSO

`BTICard_SeqConfig`, `BTICard_SeqCommRd`, `BTICard_SeqDMARd`, `BTICard_SeqBlkRd`, `BTICard_SeqFindInit`, `BTICard_SeqFindNext??`

SeqCommRd

```
USHORT BTICard_SeqCommRd
(
   LPUSHORT seqbuf,          //Pointer to Sequential Record buffer
   USHORT bufcount,          //Size of the buffer (in 16-bit words)
   HCORE hCore               //Core handle
)
```

RETURNS

The number of 16-bit words copied to the user-supplied buffer.

DESCRIPTION

Copies as many available complete records as possible from the Sequential Record on the core to a buffer (*seqbuf*) and returns the number of 16-bit words copied. The larger the buffer size (*bufcount*) the greater the number of records that can be copied by a single call to this function. The data read is effectively removed from the Sequential Record on the core. This allows an infinite amount of data to be gathered as long as this function (or one of the others in the table below) is called frequently enough to prevent the Sequential Record on the core from overflowing.

There are four functions that read from the Sequential Record. BTICard_SeqRd reads a single record; BTICard_SeqBlkRd, BTICard_SeqCommRd, and BTICard_SeqDMARd read multiple records. Any one of these functions may be used in most applications. The difference lies in their speed of execution under different conditions and availability by Device. The table below compares the four functions that read from the Sequential Record and gives some rationale for selecting one over another:

Function	Function Overhead	Per Record Overhead	Use When…
BTICard_SeqRd	low	n/a	Expect one or no records per function call
BTICard_SeqBlkRd	low	high	Expect a small number of records per function call
BTICard_SeqCommRd	high	low	Expect a large number of records per function call
BTICard_SeqDMARd	low	low	Need to offload application from reading monitor data (Device-dependent)

DEVICE DEPENDENCY

On 3G Devices, BTICard_SeqRd, _SeqBlkRd, and _SeqCommRd all read multiple records in the same manner.

WARNINGS

None.

SEE ALSO

BTICard_SeqConfig, BTICard_SeqBlkRd, BTICard_SeqDMARd, BTICard_SeqRd,BTICard_SeqFindInit, BTICard_SeqFindNext??

SeqConfig

```
ERRVAL BTICard_SeqConfig
(
  ULONG ctrlflags,          //Selects configuration options
  HCORE hCore               //Core handle
)
```

RETURNS

A negative value if an error occurs or zero if successful.

DESCRIPTION

Configures the Sequential Monitor of the core by allocating an onboard buffer and initializing internal pointers associated with the buffer.

ctrlflags	
Constant	**Description**
SEQCFG_DEFAULT	Select all default settings (**bold** below)
SEQCFG_DISABLE	Disable Sequential Record
SEQCFG_DMA	Enables DMA mode (Device-dependent)
SEQCFG_TCPNODELAY	Disable Nagle's algorithm on sequential DMA for RPC devices (Device-dependent)
SEQCFG_FILLHALT	Enable Sequential Record in fill and halt mode
SEQCFG_CONTINUOUS	Enable Sequential Record in continuous mode
SEQCFG_DELTA	Enable Sequential Record in delta mode (ARINC 429 only)
SEQCFG_INTERVAL	Enable Sequential Record in interval mode (ARINC 429 only)
SEQCFG_16K	Allocate a 16 K Sequential Record buffer
SEQCFG_32K	Allocate a 32 K Sequential Record buffer
SEQCFG_64K	Allocate a 64 K Sequential Record buffer
SEQCFG_128K	Allocate a 128 K Sequential Record buffer
SEQCFG_ALLAVAIL	Allocate all available memory to the Sequential Record
SEQCFG_NOLOGFULL	Do not generate an entry in the Event Log List when the Sequential Record is full
SEQCFG_LOGFULL	Generate an entry in the Event Log List when the Sequential Record is full
SEQCFG_NOLOGFREQ	Do not generate entries in the Event Log List at user-defined frequency (see BTICard_SeqLogFrequency)
SEQCFG_LOGFREQ	Generate entries in the Event Log List at user-defined frequency (see BTICard_SeqLogFrequency)

Note: It is highly recommended that the SEQCFG_FILLHALT mode be used for the Sequential Record. This mode will allow for continuous recording of databus activity as long as the host keeps up with reading out record data. To allow the host flexibility in reading the Sequential Record, it is also recommended to use a value of SEQCFG_128K for the size of the buffer.

When using the SEQCFG_CONTINOUS mode, databus activity will be continuously written to the Sequential Record without regard for the host reading data from the buffer. If the host attempts to read from it while the Device is running, the data returned could be corrupted. Therefore, when in this mode the Sequential Record should only be read while stopping and resuming the monitor using BTICard_SeqStop and BTICard_SeqResume.

DEVICE DEPENDENCY

5G and 6G Devices always have a 16MB Sequential Record buffer. 5G and 6G 429 Devices do not support interval and delta mode.

Sequential DMA is supported on all 4G, 5G, and 6G Devices; however, sequential DMA is not supported on 4G Devices when operating over USB.

WARNINGS

If the SEQCFG_ALLAVAIL flag is used, BTICard_SeqConfig should be the last function called that allocates memory before BTICard_CardStart is called.

SEE ALSO

BTICard_SeqRd, BTICard_SeqInterval,
BTICard_SeqLogFrequency

SeqDMARd

```
USHORT BTICard_SeqDMARd
(
   LPUSHORT seqbuf,          //Pointer to Sequential Record buffer
   ULONG bufcount,           //Size of the buffer (in 16-bit words)
   HCORE hCore               //Core handle
)
```

RETURNS

The number of 16-bit words copied to the user-supplied buffer.

DESCRIPTION

Copies as many available complete records as possible from the Sequential Record on the core to the buffer *seqbuf* and returns the number of 16-bit words copied. The larger the buffer size (*bufcount*) the greater the number of records that can be copied by a single call to this function. The Sequential Record data is read from Host memory since it was already transferred from the core memory to the Host in a DMA process. This allows an infinite amount of data to be gathered as long as this function (or one of the others in the table below) is called frequently enough to prevent the Sequential Record on the core from overflowing.

There are four functions that read from the Sequential Record. BTICard_SeqRd reads a single record; BTICard_SeqBlkRd, BTICard_SeqCommRd, and BTICard_SeqDMARd read multiple records. Any one of these functions may be used in most applications. The difference lies in their speed of execution under different conditions and availability by Device. The table below compares the four functions that read from the Sequential Record and gives some rationale for selecting one over another:

Function	Function Overhead	Per Record Overhead	Use When…
BTICard_SeqRd	low	n/a	Expect one or no records per function call
BTICard_SeqBlkRd	low	High	Expect a small number of records per function call
BTICard_SeqCommRd	high	Low	Expect a large number of records per function call
BTICard_SeqDMARd	low	Low	Need to offload application from reading monitor data (Device-dependent)

DEVICE DEPENDENCY

Sequential DMA is supported on all 4G, 5G, and 6G Devices; however, sequential DMA is not supported on 4G Devices when operating over USB.

WARNINGS

None.

SEE ALSO

BTICard_SeqConfig, BTICard_SeqBlkRd, BTICard_SeqRd, BTICard_SeqDMARd, BTICard_SeqFindInit, BTICard_SeqFindNext??

SeqFindCheckVersion

```
BOOL BTICard_SeqFindCheckVersion
(
  LPUSHORT pRecord,             //Pointer to a record
  USHORT version,               //Version number to test
)
```

RETURNS

TRUE if record pointed to by *pRecord* is equal to or greater than the version number represented by *version*, otherwise FALSE.

DESCRIPTION

Checks to see if the version number of the record pointed to by *pRecord* is equal to or greater than the constant passed for *version*. Use this function to test the eligibility of a given record for a version-dependent application of a BTICard_SeqFindMore?? function.

version	
Constant	**Description**
SEQVER_0	Sequential Record Version 0
SEQVER_1	Sequential Record Version 1

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

None.

SEE ALSO

BTICard_SeqFindInit, BTICard_SeqFindNext??

SeqFindInit

```
ERRVAL BTICard_SeqFindInit
(
  LPUSHORT seqbuf,           //Pointer to a Sequential Record buffer
  ULONG seqcount,            //Number of 16-bit words in the buffer
  LPSEQFINDINFO sfinfo       //Pointer to structure
)
```

RETURNS

A negative value if an error occurs or zero if successful.

DESCRIPTION

Initializes the structure (*sfinfo*) used by other BTICard_SeqFindNext?? functions for finding records within a Sequential Record buffer. *seqbuf* is a pointer to the start of a buffer containing Sequential Record data, and *seqcount* is the number of 16-bit words in the buffer.

sfinfo contains information that is used by the various BTICard_SeqFindNext?? functions. Each time a BTICard_SeqFindNext?? function is called, the *sfinfo* structure is updated to indicate where to resume the search with the next BTICard_SeqFindNext?? function. Since these find functions pick up where they left off, based on *sfinfo*, it is necessary to call BTICard_SeqFindInit whenever a find function is to start at the beginning of the buffer.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

This function causes the BTICard_SeqFindNext?? functions to start their search at the beginning of the Sequential Record buffer.

SEE ALSO

BTICard_SeqConfig, BTICard_SeqRd, BTICard_SeqFindNext??

SeqFindMore1553

```
ERRVAL BTICard_SeqFindMore1553
(
  LPSEQRECORDMORE1553 *pRecMore,   //Address of pointer to a structure
  LPSEQRECORD1553 pRecBase         //Pointer to a structure
)
```

RETURNS

A negative value if an error occurs, or zero if successful.

DESCRIPTION

Finds the extra fields at the end of a MIL-STD-1553 record in the Sequential Record buffer pointed to by *pRecBase* and updates **pRecMore* to point to those fields. BTICard_SeqFindNext1553 must be called before each call to BTICard_SeqFindMore1553 to seed the *pRecBase* structure with the first portion of a 1553 message. Repeatedly calling BTICard_SeqFindNext1553 and BTICard_SeqFindMore1553 returns the 1553 records in the Sequential Record one at a time until the end of the buffer is reached (at which time BTICard_SeqFindNext1553 returns an error value).

Part of the time-tag and the measured RT response times are recorded in the extra fields, as shown in the table below.

SEQRECORDMORE1553 structure			
Field	**Size**	**Description**	**Version (of base record)**
timestamph	ULONG	Upper 32 bits of the time-tag value	1 or greater
resptime1	USHORT	First RT response time (in 10ths of µs)	1 or greater
resptime2	USHORT	Second RT response time (in 10ths of µs)	1 or greater

DEVICE DEPENDENCY

Applies only to 4G, 5G, and 6G Devices, which add extra fields to 1553 records of version 1 or greater. The version number of the base record pointed to by *pRecBase* can be tested using BTICard_SeqFindCheckVersion.

When IRIG is enabled on a 4G Device, time-tags in message structures and Sequential Records will be in BCD format (see BTICard_TimerStatus).

WARNINGS

BTICard_SeqFindNext1553 must be called before each call to BTICard_SeqFindMore1553.

SEE ALSO

BTICard_SeqFindInit, BTICard_SeqFindNext1553

SeqFindNext

```
ERRVAL BTICard_SeqFindNext
(
   LPUSHORT *pRecord,          //Address of pointer
   LPUSHORT seqtype,           //Pointer to variable to receive type value
   LPSEQFINDINFO sfinfo        //Pointer to structure
)
```

RETURNS

A negative value if an error occurs or zero if successful.

DESCRIPTION

Finds the next record (**pRecord*) in the Sequential Record buffer (regardless of protocol). The protocol for that record is indicated by *seqtype* as shown below. The *sfinfo* structure is also updated.

seqtype	
Constant	**Description**
SEQTYPE_429	Sequential Record type is ARINC 429
SEQTYPE_717	Sequential Record type is ARINC 717
SEQTYPE_1553	Sequential Record type is MIL-STD-1553
SEQTYPE_708	Sequential Record type is ARINC 708
SEQTYPE_CSDB	Sequential Record type is CSDB
SEQTYPE_DIO	Sequential Record type is DIO
SEQTYPE_USER	Sequential Record type is user-defined
SEQTYPE_EBR	Sequential Record type is EBR-1553

Calling this function repeatedly steps through the records one at a time until the end of the buffer is reached (at which time it returns an error value). Thus, messages can be individually saved to disk, displayed on screen, etc. To handle the record data, cast the **pRecord* value to a structure pointer defined in the protocol-specific `BTICard_SeqFindNext??` functions.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

To make this function start its search at the beginning of the Sequential Record buffer, the *sfinfo* structure must first be initialized with `BTICard_SeqFindInit`. Otherwise, it finds the next record from where it left off.

SEE ALSO

`BTICard_SeqConfig`, `BTICard_SeqRd`, `BTICard_SeqFindInit`, `BTICard_SeqFindNext??`

SeqFindNext1553

```
ERRVAL BTICard_SeqFindNext1553
(
  LPSEQRECORD1553 *pRecord,     //Address of pointer to a structure
  LPSEQFINDINFO sfinfo          //Pointer to structure
)
```

RETURNS

A negative value if an error occurs or zero if successful.

DESCRIPTION

Finds the next MIL-STD-1553 record in the Sequential Record buffer and updates **pRecord* to point to that record. This function uses and updates data in the *sfinfo* structure. Calling this function repeatedly returns the 1553 records one at a time until the end of the buffer is reached (at which time it returns an error value). Thus, messages can be individually saved to disk, displayed on screen, etc. Using the predefined Sequential Record structure SEQRECORD1553 allows for easy handling of the data.

SEQRECORD1553 structure			
Field	**Size**	**Description**	**Version**
type	USHORT	The protocol and version number of this record	All versions
timestamp	ULONG	Lower 32 bits of the time-tag value	All versions
activity	USHORT	Activity value (see table below for details)	All versions
error	USHORT	Error value (see table below for details)	All versions
cwd1	USHORT	Command word 1 value	All versions
cwd2	USHORT	Command word 2 value	All versions
swd1	USHORT	Status word 1value	All versions
swd2	USHORT	Status word 2 value	All versions
datacount	USHORT	Number of MIL-STD-1553 data words	All versions
data[]	USHORT	Array of 1553 data words (don't exceed data[datacount -1])	All versions

The version number can be tested using BTICard_SeqFindCheckVersion. The activity and error fields may be tested by AND-ing the values returned with the constants from the tables below:

MIL-STD-1553 activity field	
Constant	**Description**
MSGACT1553_CHMASK	The channel number. Shift the result right by MSGACT-1553_CHSHIFT.
MSGACT1553_CHSHIFT	Channel number shift value. See CHMASK above.
MSGACT1553_XMTCWD1	Command word 1 was transmitted.
MSGACT1553_XMTCWD2	Command word 2 was transmitted.
MSGACT1553_XMTSWD1	Status word 1 was transmitted.
MSGACT1553_XMTSWD2	Status word 2 was transmitted.
MSGACT1553_RCVCWD1	Command word 1 was received.
MSGACT1553_RCVCWD2	Command word 2 was received.
MSGACT1553_RCVSWD1	Status word 1 was received.
MSGACT1553_RCVSWD2	Status word 2 was received.
MSGACT1553_XMTDWD	Data word was transmitted.
MSGACT1553_RCVDWD	Data word was received.
MSGACT1553_BUS	Message was transmitted/received on bus A (0) or B (1).

MIL-STD-1553 error field	
Constant	**Description**
MSGERR1553_NORESP	No response was received from the RT
MSGERR1553_ANYERR	Set if any other error bits are set
MSGERR1553_PROTOCOL	A protocol error occurred
MSGERR1553_SYNC	Wrong polarity of the sync pulse
MSGERR1553_DATACOUNT	Too many/too few data words
MSGERR1553_MANCH	Manchester error
MSGERR1553_PARITY	Parity error
MSGERR1553_WORD	Word error
MSGERR1553_RETRY	All attempts to retry transmission of this message failed (BC only)
MSGERR1553_SYSTEM	Internal system error occurred
MSGERR1553_HIT	Indicates that this message was transmitted or received since this bit was last cleared (always set)

Extract the channel number from the activity word by AND-ing the activity field with `MSGACT1553_CHMASK` and right-shifting the result by `MSGACT1553_CHSHIFT`. The resulting value is the channel number associated with the MIL-STD-1553 record.

```
channel = (activity & MSGACT1553_CHMASK) >> MSGACT1553_CHSHIFT;
```

DEVICE DEPENDENCY

When IRIG is enabled on a 4G Device, time-tags in message structures and Sequential Records will be in BCD format (see `BTICard_TimerStatus`).

WARNINGS

Must be preceded by a call to `BTICard_SeqFindInit`.

SEE ALSO

`BTICard_SeqConfig`, `BTICard_SeqRd`, `BTICard_SeqFindInit`, `BTICard_SeqFindMore1553`

SeqFindNext429

```
ERRVAL BTICard_SeqFindNext429
(
   LPSEQRECORD429 *pRecord,    //Address of pointer to a structure
   LPSEQFINDINFO sfinfo        //Pointer to structure
)
```

RETURNS

A negative value if an error occurs, or zero if successful.

DESCRIPTION

Finds the next ARINC 429 record in the Sequential Record buffer and updates **pRecord* to point to that record. This function uses and updates data from the *sfinfo* structure. Calling this function repeatedly returns the 429 records one at a time until the end of the buffer is reached (at which time it returns an error value). Thus, messages can be individually saved to disk, displayed on screen, etc. Using the predefined Sequential Record structure `SEQRECORD429` allows for easy handling of the data.

SEQRECORD429 structure			
Field	**Size**	**Description**	**Version**
`type`	USHORT	The protocol and version number of this record	All versions
`timestamp`	ULONG	Lower 32 bits of the time-tag value	All versions
`activity`	USHORT	Activity (see table below for details)	All versions
`decgap`	USHORT	Gap preceding the 429 word in half bit-times (up to a maximum of 7.5 bit times)	1 or greater
`data`	ULONG	32-bit ARINC 429 data word value	All versions
`timestamph`	ULONG	Upper 32 bits of the time-tag value	1 or greater

The version number can be tested using `BTICard_SeqFindCheckVersion`.

The *decgap* field is a 4-bit value that measures the gap preceding the decoded word. If *decgap* indicates 7.5 bit times (F hexadecimal), then the gap is 7.5 bit times or greater.

The activity field may be tested by AND-ing the value returned with the constants from the table below:

ARINC 429 activity field	
Constant	**Description**
MSGACT429_CHMASK	The channel number. Shift the result right by MSGACT429_CHSHIFT.
MSGACT429_CHSHIFT	Channel number shift value. See CHMASK above.
MSGACT429_SPD	This bit reflects the speed detected. A one signifies high speed (100 Kbps), and a zero signifies low speed (12.5 Kbps).
MSGACT429_ERR	If set, it signifies that an error occurred in receiving this word. The type of error is defined by the following bits.
MSGACT429_GAP	Gap Error: A gap of less than four bit times preceded the word.
MSGACT429_PAR	Parity error: A parity error was detected in the word.
MSGACT429_LONG	Long word error: A word of more than 32-bits was detected.
MSGACT429_BIT	Bit timing error: An error occurred while decoding the bits of the word (short bits or long bits).
MSGACT429_TO	Time out error: The decoder timed out while receiving a word (short word).
MSGACT429_HIT	Signifies that the message has been processed by the firmware (the Hit bit).

Extract the channel number from the activity word by AND-ing the activity field with MSGACT429_CHMASK and right-shifting the result by MSGACT429_CHSHIFT. The resulting value is the channel number associated with the ARINC 429 record.

```
channel = (activity & MSGACT429_CHMASK) >> MSGACT429_CHSHIFT;
```

DEVICE DEPENDENCY

When IRIG is enabled on a 4G Device, time-tags in message structures and Sequential Records will be in BCD format (see BTICard_TimerStatus).

WARNINGS

Must be preceded by a call to BTICard_SeqFindInit.

SEE ALSO

BTICard_SeqConfig, BTICard_SeqRd, BTICard_SeqFindInit

SeqFindNext708

```
ERRVAL BTICard_SeqFindNext708
(
   LPSEQRECORD708 *pRecord, //Address of pointer to a structure
   LPSEQFINDINFO sfinfo     //Pointer to structure
)
```

RETURNS

A negative value if an error occurs or zero if successful.

DESCRIPTION

Finds the next ARINC 708 record in the Sequential Record buffer and updates `*pRecord` to point to that record. This function uses and updates data from the `sfinfo` structure. Calling this function repeatedly returns the 708 records one at a time until the end of the buffer is reached (at which time it returns an error value). Thus, messages can be individually saved to disk, displayed on screen, etc. Using the predefined Sequential Record structure `SEQRECORD708` allows for easy handling of the data.

SEQRECORD708 structure			
Field	**Size**	**Description**	**Version**
`type`	USHORT	The protocol and version number of this record	All versions
`timestamp`	ULONG	Lower 32 bits of the time-tag value	All versions
`activity`	USHORT	Activity (see table below for details)	All versions
`datacount`	USHORT	Number of data words	All versions
`data[100]`	USHORT	ARINC 708 data word values (100 16-bit data words)	All versions
`extra[16]`	USHORT	Additional data (if variable bit mode is enabled)	1 or greater
`bitcount`	USHORT	Number of bits in this message (if variable bit mode is enabled)	1 or greater
`timestamph`	ULONG	Upper 32 bits of the time-tag value	1 or greater

The version number can be tested using `BTICard_SeqFindCheckVersion`.

The activity field may be tested by AND-ing the value returned with the constants from the table below:

ARINC 708 activity field	
Constant	**Description**
`MSGACT708_CHMASK`	The channel number. Shift the result right by `MSGACT708_CHSHIFT`.
`MSGACT708_CHSHIFT`	Channel number shift value. See `CHMASK` above.
`MSGACT708_ERR`	This bit is set if any of the error bits are set.
`MSGACT708_DSYNC`	No data sync at end of word.
`MSGACT708_MANCH`	Manchester error.
`MSGACT708_WORD`	Word error.
`MSGACT708_LONG`	Long word error: A word of more than 1600 bits was detected.
`MSGACT708_SHORT`	Short word error: A word of less than 1600 bits was detected.
`MSGACT708_TO`	Time out error: The decoder timed out while receiving a word.
`MSGACT708_HIT`	Indicates that this message was transmitted or received since this bit was last cleared.

Extract the channel number from the activity word by AND-ing the activity field with MSGACT708_CHMASK and right-shifting the result by MSGACT708_CHSHIFT. The resulting value is the channel number associated with the ARINC 708 record.

```
channel = (activity & MSGACT708_CHMASK) >> MSGACT708_CHSHIFT;
```

DEVICE DEPENDENCY

When IRIG is enabled on a 4G Device, time-tags in message structures and Sequential Records will be in BCD format (see BTICard_TimerStatus).

WARNINGS

Must be preceded by a call to BTICard_SeqFindInit.

SEE ALSO

BTICard_SeqConfig, BTICard_SeqRd, BTICard_SeqFindInit

SeqFindNext717

```
ERRVAL BTICard_SeqFindNext717
(
   LPSEQRECORD717 *pRecord, //Address of pointer to a structure
   LPSEQFINDINFO sfinfo     //Pointer to structure
)
```

RETURNS

A negative value if an error occurs or zero if successful.

DESCRIPTION

Finds the next ARINC 717 record in the Sequential Record buffer and updates **pRecord* to point to that record. This function uses and updates data from the *sfinfo* structure. Calling this function repeatedly returns the 717 records one at a time until the end of the buffer is reached (at which time it returns an error value). Thus, messages can be individually saved to disk, displayed on screen, etc. Using the predefined Sequential Record structure `SEQRECORD717` allows for easy handling of the data.

SEQRECORD717 structure			
Field	**Size**	**Description**	**Version**
`type`	USHORT	The protocol and version number of this record	All versions
`timestamp`	ULONG	Lower 32-bits of the time-tag value	All versions
`activity`	USHORT	Activity (see table below for details)	All versions
`wordnum`	USHORT	Number of words	All versions
`subframe`	USHORT	Number of subframes	All versions
`superframe`	USHORT	Number of superframes	All versions
`data`	USHORT	12-bit ARINC 717 data word value in LSBs	All versions
`timestamph`	ULONG	Upper 32 bits of the time-tag value	1 or greater

The version number can be tested using `BTICard_SeqFindCheckVersion`.

The activity field may be tested by AND-ing the value returned with the constants from the table below:

ARINC 717 activity field	
Constant	**Description**
MSGACT717_CHMASK	The channel number. Shift the result right by MSGACT717_CHSHIFT.
MSGACT717_CHSHIFT	Channel number shift value. See CHMASK above.
MSGACT717_SPDMASK	The current speed mask value.
MSGACT717_64WPS	The current speed is 64 wps (words per second).
MSGACT717_128WPS	The current speed is 128 wps.
MSGACT717_256WPS	The current speed is 256 wps.
MSGACT717_512WPS	The current speed is 512 wps.
MSGACT717_1024WPS	The current speed is 1024 wps.
MSGACT717_2048WPS	The current speed is 2048 wps.
MSGACT717_4096WPS	The current speed is 4096 wps.
MSGACT717_8192WPS	The current speed is 8192 wps.
MSGACT717_TO	Time out error. The decoder timed out while receiving a (short) word.
MSGACT717_HIT	Indicates that this message was transmitted or received since this bit was last cleared.

Extract the channel number from the activity word by AND-ing the activity field with `MSGACT717_CHMASK` and right-shifting the result by `MSGACT717_CHSHIFT`. The resulting value is the channel number associated with the ARINC 717 record.

```
channel = (activity & MSGACT717_CHMASK) >> MSGACT717_CHSHIFT;
```

DEVICE DEPENDENCY

When IRIG is enabled on a 4G Device, time-tags in message structures and Sequential Records will be in BCD format (see `BTICard_TimerStatus`).

WARNINGS

Must be preceded by a call to `BTICard_SeqFindInit`.

SEE ALSO

`BTICard_SeqConfig`, `BTICard_SeqRd`, `BTICard_SeqFindInit`

SeqFindNextDIO

```
ERRVAL BTICard_SeqFindNextDIO
(
   LPSEQRECORDDIO *pRecord, //Address of pointer to a structure
   LPSEQFINDINFO sfinfo     //Pointer to structure
)
```

RETURNS

A negative value if an error occurs or zero if successful.

DESCRIPTION

Finds the next DIO record in the Sequential Record buffer and updates **pRecord* to point to that record. This function uses and updates data from the *sfinfo* structure. Calling this function repeatedly returns the DIO records one at a time until the end of the buffer is reached (at which time it returns an error value). Thus, messages can be individually saved to disk, displayed on screen, etc. Using the predefined Sequential Record structure SEQRECORDDIO allows for easy handling of the data.

SEQRECORDDIO structure			
Field	**Size**	**Description**	**Version**
type	USHORT	The protocol and version number of this record	All versions
count	USHORT	The length of this record	All versions
bank	USHORT	Number of the bank	All versions
state	USHORT	State of the bank	0 only
timestamp	ULONG	Lower 32 bits of the time-tag value	All versions
timestamph	ULONG	Upper 32 bits of the time-tag value	All versions
change	USHORT	Bitmask of discrete inputs that changed value	1 or greater
value	USHORT	Current value of discrete inputs	1 or greater

DEVICE DEPENDENCY

Applies to all 5G and 6G Devices. The mapping of *dionum* to physical discrete I/O is hardware dependent. Please consult the hardware manual for the Device. Also applies to 4G Devices with one or more discrete I/O modules (832 module). Please consult the OmniBus Discrete IO User's Manual for usage with 4G Devices.

When IRIG is enabled on a 4G Device, time-tags in Sequential Records will be in BCD format (see BTICard_TimerStatus).

WARNINGS

Must be preceded by a call to BTICard_SeqFindInit.

SEE ALSO

BTICard_ExtDIOMonConfig,BTICard_SeqConfig,
BTICard_SeqRd, BTICard_SeqFindInit

SeqFindNextEBR

```
ERRVAL BTICard_SeqFindNextEBR
(
  LPSEQRECORDEBR *pRecord,      //Address of pointer to a structure
  LPSEQFINDINFO sfinfo          //Pointer to structure
)
```

RETURNS

A negative value if an error occurs or zero if successful.

DESCRIPTION

Finds the next EBR-1553 record in the Sequential Record buffer and updates `*pRecord` to point to that record. This function uses and updates data in the `sfinfo` structure. Calling this function repeatedly returns the EBR records one at a time until the end of the buffer is reached (at which time it returns an error value). Thus, messages can be individually saved to disk, displayed on screen, etc. Using the predefined Sequential Record structure `SEQRECORDEBR` allows for easy handling of the data.

SEQRECORDEBR structure			
Field	**Size**	**Description**	**Version**
`type`	UINT16	The protocol and version number of this record	All versions
`count`	UINT16	Word count in record	All versions
`activity`	UINT16	Activity value (see table below for details)	All versions
`error`	UINT16	Error value (see table below for details)	All versions
`timestamp`	UINT32	Lower 32 bits of the time-tag value	All versions
`timestamph`	UINT32	Upper 32 bits of the time-tag value	All versions
`cwd`	UINT16	Command word value	All versions
`cwdinfo`	UINT16	RT Link the command word was sent on	All versions
`swd`	UINT16	Status wordvalue	All versions
`swdinfo`	UINT16	RT Link the status word was sent on	All versions
`resptime`	UINT16	RT response time (in 100ths of µs)	All versions
`datacount`	UINT16	Number EBR-1553 data words	All versions
`data[]`	UINT16	Array of EBR data words (don’t exceed data[datacount -1])	All versions

The version number can be tested using `BTICard_SeqFindCheckVersion`.

The activity and error fields may be tested by AND-ing the values returned with the constants from the tables below:

EBR-1553 activity field	
Constant	**Description**
MSGACTEBR_CHMASK	The channel number mask value. Shift the result right with MSGACTEBR_CHSHIFT.
MSGACTEBR_CHSHIFT	Channel number shift value. See CHMASK above.
MSGACTEBR_LINKMASK	The RT Link number. Shift the result right by MSGACTEBR_LINKSHIFT.
MSGACTEBR_LINKSHIFT	Link number shift value. See LINKMASK above.
MSGACTEBR_XMTCWD	The command word was transmitted.
MSGACTEBR_XMTSWD	The status word was transmitted.
MSGACTEBR_RCVCWD	The command word was received.
MSGACTEBR_RCVSWD	The status word was received.
MSGACTEBR_XMTDWD	Data words were transmitted.
MSGACTEBR_RCVDWD	Data words were received.
MSGACTEBR_MODEMASK	LHub Mode mask value
MSGACTEBR_MODESHIFT	LHub Mode shift value

EBR-1553 error field	
Constant	**Description**
MSGERREBR_NORESP	No response was received from the RT
MSGERREBR_ANYERR	Set if any other error bits are set
MSGERREBR_PROTOCOL	A protocol error occurred
MSGERREBR_SYNC	Wrong polarity of the sync pulse
MSGERREBR_DATACOUNT	Too many/too few data words
MSGERREBR_MANCH	Manchester error
MSGERREBR_PARITY	Parity error
MSGERREBR_WORD	Word error
MSGERREBR_RETRY	All attempts to retry transmission of this message failed (BC only)
MSGERREBR_SYSTEM	Internal system error occurred
MSGERREBR_LHUB	Indicates unexpected LHUB activity was detected
MSGERREBR_HIT	Indicates that this message was transmitted or received since this bit was last cleared (always set)

Extract the channel number from the activity word by AND-ing the activity field with MSGACTEBR_CHMASK and right-shifting the result by MSGACTEBR_CHSHIFT. The resulting value is the channel number associated with the EBR-1553 record.

```
channel = (activity & MSGACTEBR_CHMASK) >> MSGACTEBR_CHSHIFT;
```

Cwdinfo field	
Constant	**Description**
CWDINFOEBR_RESVMASK	Reserved for future use. Do not use.
CWDINFOEBR_LINKMASK	The link number mask value. Shift the result right with CWDINFOEBR_LINKSHIFT.
CWDINFOEBR_LINKSHIFT	Link number shift value. See LINKMASK above.

Swdinfo field	
Constant	**Description**
SWDINFOEBR_RESVMASK	Reserved for future use. Do not use.
SWDINFOEBR_LINKMASK	The link number mask value. Shift the result right with SWDINFOEBR_LINKSHIFT.
SWDINFOEBR_LINKSHIFT	Link number shift value. See LINKMASK above.

The RT Link number that the command was sent on can be extracted from cwdinfo field through the same process.

```
cwdlink = (cwdinfo & CWDINFOEBR _LINKMASK) >> CWDINFOEBR _LINKSHIFT;
```

This is also done to determine the RT Link number the response was seen on using the swdinfo field.

```
swdlink = (swdinfo & SWDINFOEBR _LINKMASK) >> SWDINFOEBR _LINKSHIFT;
```

The configured Logical Hub mode can also be extracted from the activity word. The MODE bits must be masked and shifted by doing the following

```
mode = (activity & MSGACTEBR_MODEMASK) >> MSGACTEBR_MODESHIFT;
```

Then the mode can be directly compared against the following values

LHub Mode	
Constant	**Description**
LHUBMODE_SPEC	The LHub is in "spec" mode
LHUBMODE_SWITCH	The LHub is in "switch" mode
LHUBMODE_LINK	The LHub is in "link" mode

Comparison is as follows

```
if (mode == LHUBMODE_SPEC)
```

DEVICE DEPENDENCY

When IRIG is enabled on a 4G Device, time-tags in message structures and Sequential Records will be in BCD format (see `BTICard_TimerStatus`).

WARNINGS

Must be preceded by a call to `BTICard_SeqFindInit`.

SEE ALSO

`BTICard_SeqConfig`, `BTICard_SeqRd`, `BTICard_SeqFindInit`

SeqInterval

```
INT BTICard_SeqInterval
(
   INT interval,               //Interval time (in seconds)
   INT mode,                   //Mode to determine interval value
   HCORE hCore                 //Core handle
)
```

RETURNS

The actual interval value that the core will use.

DESCRIPTION

Sets the interval time for the Sequential Monitor, and is used when the Sequential Record has been configured with the `SEQCFG_INTERVAL` flag. In Interval mode, the Sequential Monitor records only the first occurrence of selected messages within the specified interval. The availability of the Interval mode is both Device- and protocol-dependent.

The core cannot accommodate all interval values that could be passed through *interval*. The specified mode helps determine the actual interval that will be used. The constants below should be used to set the mode:

mode	
Constant	**Description**
`INTERVALMODE_CLOSEST`	Uses the value closest to the specified interval
`INTERVALMODE_LESS`	Uses the value just less than specified interval
`INTERVALMODE_GREATER`	Uses the value just greater the specified interval

DEVICE DEPENDENCY

Applies only to 3G and 4G Devices.

WARNINGS

None.

SEE ALSO

`BTICard_SeqConfig`, `BTICard_SeqRd`

SeqIsRunning

```
BOOL BTICard_SeqIsRunning
(
  HCORE hCore                    //Core handle
)
```

RETURNS

TRUE if the Sequential Record is still active, otherwise FALSE.

DESCRIPTION

Determines whether the Sequential Record is active and is typically used when the Sequential Record has been configured with the `SEQCFG_FILLHALT` flag. In which case, recording halts when the on-board Sequential Record is full. This prevents unread data from being overwritten when the host gets behind in reading data from the Sequential Record. Thus, in fill and halt mode `BTICard_SeqIsRunning` effectively indicates whether the buffer is full or not.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

None.

SEE ALSO

`BTICard_SeqConfig`, `BTICard_SeqRd`

SeqLogFrequency

```
USHORT BTICard_SeqLogFrequency
(
  USHORT logfreq,          //Frequency of Event Log List entries
  HCORE hCore              //Core handle
)
```

RETURNS

The previous value of the frequency of Event Log List entries.

DESCRIPTION

Sets the Event Log List frequency for the Sequential Monitor. It is used when the Sequential Record has been configured with the `SEQCFG_LOGFREQ` flag in `BTICard_SeqConfig`. The Sequential Record generates an Event Log List entry after it records *`logfreq`* amount of records. The user specifies the value of *`logfreq`*. For example, a value of 1 enables an Event Log List entry after every record, a value of 2 after every second record, and so on. It continues in this manner until the Sequential Record is stopped.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

None.

SEE ALSO

`BTICard_SeqConfig`, `BTICard_SeqRd`, `BTICard_EventLogConfig`, `BTICard_EventLogRd`

SeqRd

```
USHORT BTICard_SeqRd
(
  LPUSHORT seqbuf,          //Pointer to Sequential Record buffer
  HCORE hCore               //Core handle
)
```

RETURNS

The number of 16-bit words copied to the user-supplied buffer.

DESCRIPTION

Copies up to a single record at a time from the Sequential Record on the core to a buffer (*seqbuf*). The function returns the number of 16-bit words copied. The data read is effectively removed from the Sequential Record on the core. This allows an infinite amount of data to be gathered as long as this function (or one of the others in the table below) is called frequently enough to prevent the Sequential Record on the core from overflowing.

There are four functions that read from the Sequential Record. `BTICard_SeqRd` reads a single record; `BTICard_SeqBlkRd`, `BTICard_SeqCommRd`, and `BTICard_SeqDMARd` read multiple records. Any one of these functions may be used in most applications. The difference lies in their speed of execution under different conditions and availability by Device. The table below compares the four functions that read from the Sequential Record and gives some rationale for selecting one over another:

Function	Function Overhead	Per Record Overhead	Use When…
BTICard_SeqRd	low	n/a	Expect one or no records per function call
BTICard_SeqBlkRd	low	High	Expect a small number of records per function call
BTICard_SeqCommRd	high	Low	Expect a large number of records per function call
BTICard_SeqDMARd	low	Low	Need to offload application from reading monitor data (Device-dependent)

DEVICE DEPENDENCY

On 3G Devices, `BTICard_SeqRd`, `BTICard_SeqBlkRd`, and `BTICard_SeqCommRd` all read multiple records in the same manner.

WARNINGS

None.

SEE ALSO

`BTICard_SeqConfig`, `BTICard_SeqBlkRd`, `BTICard_SeqCommRd`, `BTICard_SeqFindInit`, `BTICard_SeqFindNext??`

SeqResume

```
BOOL BTICard_SeqResume
(
  HCORE hCore                   //Core handle
)
```

RETURNS

TRUE if the Sequential Record was previously running, otherwise FALSE.

DESCRIPTION

Resumes operation of the Sequential Record at the point at which it was stopped using `BTICard_SeqStop`. Use this function to continue recording data to the Sequential Record without overwriting previous records.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

A call to `BTICard_SeqStop` must precede this function.

SEE ALSO

`BTICard_SeqStart`, `BTICard_SeqStop`

SeqStart

```
BOOL BTICard_SeqStart
(
  HCORE hCore                    //Core handle
)
```

RETURNS

TRUE if the Sequential Record was previously running, otherwise FALSE.

DESCRIPTION

Starts recording of the Sequential Record. If necessary, it also stops and clears the Sequential Record before restarting it.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

If this function is called after `BTICard_SeqStop`, recording starts at the beginning of the buffer and previous data is overwritten. To add to previous data without erasing it, use `BTICard_SeqResume` instead.

SEE ALSO

`BTICard_SeqStop`, `BTICard_SeqResume`

SeqStatus

```
BOOL BTICard_SeqStatus
(
  HCORE hCore                    //Core handle
)
```

RETURNS

The status value of the Sequential Record.

DESCRIPTION

Checks the status of the Sequential Record. The status value can be tested using the predefined constants below:

Constant	Description
STAT_EMPTY	Sequential Record is empty
STAT_PARTIAL	Sequential Record is partially filled
STAT_FULL	Sequential Record is full
STAT_OFF	Sequential Record is off

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

The operation of the SeqStatus is configuration and Device dependent. All 4G Devices and 5G/6G RPC Devices configured for DMA can return `STAT_EMPTY` when the internal sequential buffer is empty but the DMA buffer contains data.

SEE ALSO

`BTICard_SeqStart`, `BTICard_SeqStop`, `BTICard_SeqResume`

SeqStop

```
BOOL BTICard_SeqStop
(
   HCORE hCore                  //Core handle
)
```

RETURNS

TRUE if the Sequential Record was previously running, otherwise FALSE.

DESCRIPTION

Suspends the recording of data to the Sequential Record before the buffer is filled. If `BTICard_SeqResume` is subsequently called, recording is resumed at the point at which it was stopped without overwriting previous records. If `BTICard_SeqStart` is called after `BTICard_SeqStop`, recording starts at the beginning of the buffer and previous data is overwritten.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

None.

SEE ALSO

`BTICard_SeqStart, BTICard_SeqResume`

SysMonClear

```
ERRVAL BTICard_SysMonClear
(
  HCARD hCard                   //Card handle
)
```

RETURNS

A negative value if an error occurs or zero if successful.

DESCRIPTION

Resets the historic maximum and minimum values for all sensors on the card specified by *hCard*.

DEVICE DEPENDENCY

Applies to 5G and 6G Devices that support BIT/SysMon functionality. Please consult the hardware manual for the Device.

WARNINGS

None.

SEE ALSO

BTICard_SysMonTypeGet

SysMonDescGet

```
LPCSTR BTICard_SysMonDescGet
(
   INT index,                    //Specifies the sensor index
   HCARD hCard                   //Card handle
)
```

RETURNS

A pointer to a character string describing the sensor or NULL if the sensor is not present.

DESCRIPTION

Provides a formatted string that describes the sensor specified by *index*.

DEVICE DEPENDENCY

Applies to 5G and 6G Devices that support BIT/SysMon functionality. Please consult the hardware manual for the Device.

WARNINGS

None.

SEE ALSO

BTICard_SysMonTypeGet

SysMonMaxRd

```
INT BTICard_SysMonMaxRd
(
  INT index,                //Specifies the sensor index
  HCARD hCard               //Card handle
)
```

RETURNS

The historic maximum value of a sensor or `SYSMONRD_NOTVALID` if the sensor is not present or the historic maximum value is not valid.

DESCRIPTION

Reads the historic maximum value from the sensor specified by *`index`*. The value is in units of mV, mA, or m°C depending on the sensor type. Call `BTICard_SysMonUserStr` to convert the value to a formatted string.

To reset the historic maximum value, call `BTICard_SysMonClear`.

DEVICE DEPENDENCY

Applies to 5G and 6G Devices that support BIT/SysMon functionality. Please consult the hardware manual for the Device.

WARNINGS

This function will return `SYSMONRD_NOTVALID` for up to 800 ms after the first call to `BTICard_CardOpen` and up to 150 ms after any call to `BTICard_SysMonClear` to allow SysMon to get valid data.

Note: Due to occasionally spurious values that may be reported by the ***voltage/current*** *sensor, a second read of the system monitor to confirm an error is recommended prior to software acting on the assumed error data. See `BTICard_SysMonValRd` for additional information.*

SEE ALSO

`BTICard_SysMonClear`, `BTICard_SysMonMinRd`, `BTICard_SysMonUserStr`, `BTICard_SysMonValRd`

SysMonMinRd

```
INT BTICard_SysMonMinRd
(
   INT index,                  //Specifies the sensor index
   HCARD hCard                 //Card handle
)
```

RETURNS

The historic minimum value of a sensor or `SYSMONRD_NOTVALID` if the sensor is not present or the historic minimum value is not valid.

DESCRIPTION

Reads the historic minimum value from the sensor specified by *`index`*. The value is in units of mV, mA, or m°C depending on the sensor type. Call `BTICard_SysMonUserStr` to convert the value to a formatted string.

To reset the historic minimum value, call `BTICard_SysMonClear`.

DEVICE DEPENDENCY

Applies to 5G and 6G Devices that support BIT/SysMon functionality. Please consult the hardware manual for the Device.

WARNINGS

This function will return `SYSMONRD_NOTVALID` for up to 800 ms after the first call to `BTICard_CardOpen` and up to 150 ms after any call to `BTICard_SysMonClear` to allow SysMon to get valid data.

*Note: Due to occasionally spurious values that may be reported by the **voltage/current** sensor, a second read of the system monitor to confirm an error is recommended prior to software acting on the assumed error data. See `BTICard_SysMonValRd` for additional information.*

SEE ALSO

`BTICard_SysMonClear`, `BTICard_SysMonMaxRd`, `BTICard_SysMonUserStr`, `BTICard_SysMonValRd`

SysMonNomRd

```
INT BTICard_SysMonNomRd
(
   INT index,                    //Specifies the sensor index
   HCARD hCard                   //Card handle
)
```

RETURNS

The nominal voltage for a voltage sensor or `SYSMONRD_NOTVALID` if the sensor is not present.

DESCRIPTION

Reads the nominal voltage from the voltage sensor specified by *`index`*. The value is in units of mV. Call `BTICard_SysMonUserStr` to convert the value to a formatted string.

DEVICE DEPENDENCY

Applies to 5G and 6G Devices that support BIT/SysMon functionality. Please consult the hardware manual for the Device.

WARNINGS

Does not apply to temperature and current sensors.

SEE ALSO

`BTICard_SysMonUserStr`, `BTICard_SysMonValRd`

SysMonThresholdGet

```
ERRVAL BTICard_SysMonThresholdGet
(
   BOOL *enable,      //Pointer to variable to receive enable value
   LPINT minval,      //Pointer to variable to receive minimum threshold value
   LPINT maxval,      //Pointer to variable to receive maximum threshold value
   INT index,         //Specifies the sensor index
   HCARD hCard        //Card handle
)
```

RETURNS

A negative value if an error occurs or zero if successful.

DESCRIPTION

Reads the user definable thresholds, in units of m°C, for the temperature sensor specified by *index*. These user thresholds are disabled and set to System Limits at power on and can be modified by calling BTICard_SysMonThresholdSet. The enable value and thresholds are passed through *enable*, *minval*, and *maxval* respectively.

Call BTICard_SysMonUserStr to convert *minval* or *maxval* to a formatted string.

DEVICE DEPENDENCY

Applies to 5G and 6G Devices that support BIT/SysMon functionality. Please consult the hardware manual for the Device.

WARNINGS

Due to rounding, *minval* and *maxval* may not match what was set in BTICard_SysMonThresholdSet. The values may be rounded to the nearest resolution.

Does not apply to voltage and current sensors.

SEE ALSO

BTICard_BITStatusRd, BTICard_SysMonThresholdSet

SysMonThresholdSet

```
ERRVAL BTICard_SysMonThresholdSet
(
  BOOL enable,          //Enable for the Sensor
  INT minval,           //Minimum threshold value
  INT maxval,           //Maximum threshold value
  INT index,            //Specifies the sensor index
  HCARD hCard           //Card handle
)
```

RETURNS

A negative value if an error occurs, or zero if successful.

DESCRIPTION

Sets the user definable thresholds, in units of m°C, for the temperature sensor specified by `index`. The enable value and thresholds are passed through `enable`, `minval`, and `maxval` respectively. Once enabled, if the sensor value exceeds the the user definable thresholds, the BIT Status register will indicate a fault which can be read by calling `BTICard_BITStatusRd`.

DEVICE DEPENDENCY

Applies to 5G and 6G devices that support BIT/SysMon functionality. Please consult the hardware manual for the Device.

WARNINGS

Due to rounding, `minval` and `maxval` may not match what was set in `BTICard_SysMonThresholdSet`. The values may be rounded to the nearest resolution.

Does not apply to voltage and current sensors.

SEE ALSO

`BTICard_BITStatusRd`, `BTICard_SysMonThresholdSet`

SysMonTypeGet

```
ULONG BTICard_SysMonTypeGet
(
   INT index,                    //Specifies the sensor index
   HCARD hCard                   //Card handle
)
```

RETURNS

The type of sensor or `SYSMONTYPE_NONE` if the sensor is not present.

DESCRIPTION

Reports the type of the sensor specified by *index* by returning one of the predefined constants below:

Constant	Description
SYSMONTYPE_NONE	Sensor is not present
SYSMONTYPE_TEMP	Temperature Sensor
SYSMONTYPE_VOLTAGE	Voltage Sensor
SYSMONTYPE_CURRENT	Current Sensor

DEVICE DEPENDENCY

Applies to 5G and 6G devices that support BIT/SysMon functionality. Please consult the hardware manual for the Device.

WARNINGS

None.

SEE ALSO

`BTICard_SysMonDescGet`

SysMonUserStr

```
LPCSTR BTICard_SysMonUserStr
(
   INT value,                //Value to be formatted
   INT index,                //Specifies the sensor index
   HCARD hCard               //Card handle
)
```

RETURNS

A pointer to a character string containing the value and units for a sensor or NULL if sensor is not present.

DESCRIPTION

Returns a formatted character string containing the value and units for the sensor specified by *index*. The parameter *value* is typically read by calling `BTICard_SysMonValRd`, `BTICard_SysMonNomRd`, `BTICard_SysMonMinRd`, or `BTICard_SysMonMaxRd`.

DEVICE DEPENDENCY

Applies to 5G and 6G devices that support BIT/SysMon functionality. Please consult the hardware manual for the Device.

WARNINGS

None.

SEE ALSO

`BTICard_SysMonValRd`, `BTICard_SysMonNomRd`, `BTICard_SysMonMinRd`, `BTICard_SysMonMaxRd`

SysMonValRd

```
INT BTICard_SysMonValRd
(
   INT index,              //Specifies the sensor index
   HCARD hCard             //Card handle
)
```

RETURNS

The current value for a sensor or `SYSMONRD_NOTVALID` if the sensor is not present.

DESCRIPTION

Reads the current value of the sensor specified by *`index`*. The units for the current value are in mV, mA, or m°C depending on the sensor type. Call `BTICard_SysMonUserStr` to convert the value to a formatted string.

DEVICE DEPENDENCY

Applies to 5G and 6G devices that support BIT/SysMon functionality. Please consult the hardware manual for the Device.

WARNINGS

This function will return `SYSMONRD_NOTVALID` for up to 800 ms after the first call to `BTICard_CardOpen` and up to 150 ms after any call to `BTICard_SysMonClear` to allow SysMon to get valid data.

Note: Due to occasionally spurious values that may be reported by the ***voltage/current*** *sensor, a second read of the system monitor to confirm an error is recommended prior to software acting on the assumed error data. A minimum wait time of 150 ms is required before issuing a second read of the system monitor to ensure that a new value has been sampled. Call `BTICard_SysMonClear` to clear min/max values if a spurious voltage or current values is read.*

SEE ALSO

`BTICard_SysMonUserStr`, `BTICard_SysMonMaxRd`, `BTICard_SysMonMinRd`

Timer64Rd

```
ERRVAL BTICard_Timer64Rd
(
  LPULONG valueh,          //Pointer to upper 32 bits of the timer value
  LPULONG valuel,          //Pointer to lower 32 bits of the timer value
  HCORE hCore              //Core handle
)
```

RETURNS

A negative value if an error occurs or zero if successful.

DESCRIPTION

Reads the current value of the binary timer from the specified Device.

DEVICE DEPENDENCY

Applies to all Devices.

6G Devices have a 64-bit binary timer with a 1 ns resolution.

5G Devices have a 48-bit binary timer with a 1 μs resolution.

3G and 4G Devices have a 32-bit binary timer with an adjustable resolution. To use the IRIG timer for a specified 4G Device instead of the default binary timer, see `BTICard_IRIGConfig`.

Please see `BTICard_TimerRd` for a discussion of Device-dependent timer differences.

WARNINGS

None.

SEE ALSO

`BTICard_TimerClear`, `BTICard_TimerRd`, `BTICard_TimerWr`, `BTICard_Timer64Wr`, `BTICard_IRIGConfig`, `BTICard_IRIGRd`, `BTICard_IRIGWr`

Timer64Wr

```
VOID BTICard_Timer64Wr
(
   ULONG valueh,                 //Upper 32 bits of the timer value
   ULONG valuel,                 //Lower 32 bits of the timer value
   HCORE hCore                   //Core handle
)
```

RETURNS

None.

DESCRIPTION

Writes the timer value of the binary timer for the specified Device.

DEVICE DEPENDENCY

Applies to all Devices.

6G Devices have a 64-bit binary timer with a 1 ns resolution.

5G Devices have a 48-bit binary timer with a 1 µs resolution.

3G and 4G Devices have a 32-bit binary timer with an adjustable resolution. To use the IRIG timer for a specified 4G Device instead of the default binary timer, see `BTICard_IRIGConfig`.

Please see `BTICard_TimerWr` for a discussion of Device-dependent timer differences.

WARNINGS

None.

SEE ALSO

`BTICard_TimerClear`, `BTICard_TimerRd`, `BTICard_TimerWr`, `BTICard_Timer64Rd`, `BTICard_IRIGConfig`, `BTICard_IRIGRd`, `BTICard_IRIGWr`

TimerClear

```
VOID BTICard_TimerClear
(
  HCORE hCore                    //Core handle
)
```

RETURNS

None.

DESCRIPTION

Clears the Device timer to zero.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

None.

SEE ALSO

BTICard_TimerRd, BTICard_TimerResolution, BTICard_IRIGConfig, BTICard_IRIGRd, BTICard_IRIGWr

TimerRd

```
ULONG BTICard_TimerRd
(
  HCORE hCore                    //Core handle
)
```

RETURNS

The current Device timer value.

DESCRIPTION

Reads the lower 32-bits of the current value of the binary timer from the specified Device.

DEVICE DEPENDENCY

Applies to all Devices.

6G Devices have a 64-bit binary timer with a 1 ns resolution.

5G Devices have a 48-bit binary timer with a 1 µs resolution.

For 3G and 4G Devices, the binary timer exists in two parts: a hardware DSP timer, and a software extended value. Together these values make a 48-bit time value. `BTICard_TimerResolution` is used to adjust which bits of this 48-bit value are used to make the 32-bit time-tag used in message structures and Sequential Records. `BTICard_TimerRd` and `BTICard_TimerWr` functions only read from and write to the software extended portion of this time value and not the hardware portion (due to the complexity of accounting for the rollover from the Host, an accurate reading of both the hardware and software part is not possible). The software extended portion used by `BTICard_TimerRd` and `BTICard_TimerWr` has a resolution of 4.096 ms.

To use the IRIG timer on a 4G Device to generate time-tag values for message structures and Sequential Records instead of the default binary timer, see `BTICard_IRIGConfig`.

WARNINGS

Use caution to account for rollover when using this function. For example, the lower 32-bits of a 1 ns binary timer will rollover approximately every 4 seconds.

Refer to `BTICard_Timer64Rd` for access to the full binary timer.

SEE ALSO

`BTICard_Timer64Rd`, `BTICard_Timer64Wr`, `BTICard_TimerClear`, `BTICard_TimerResolution`, `BTICard_TimerWr`, `BTICard_IRIGConfig`, `BTICard_IRIGRd`, `BTICard_IRIGWr`

TimerResolution

```
INT BTICard_TimerResolution
(
   INT timerresol,              //Selects the timer resolution
   HCORE hCore                  //Core handle
)
```

RETURNS

The value of the previous resolution. Refer to *timerresol* for return values.

DESCRIPTION

Selects the resolution for the time-tag timer on the specified Device. Timer resolution can be read (without modifying) by setting *timerresol* to *TIMERRESOL_CURRENT*. *timerresol* must be one of the following predefined constants:

timerresol		
Constant	**Resolution**	**3G/4G Range (hr:min:sec)**
TIMERRESOL_CURRENT	*current*	-
TIMERRESOL_1US	1 µs	1:11:34
TIMERRESOL_16US	16 µs	19:05:19
TIMERRESOL_1024US	1024 µs	50 days
TIMERRESOL_1NS	1 ns	*6G Only (Refer to device dependency)*

DEVICE DEPENDENCY

Only 3G and 4G Devices have resolutions that can be modified. To use the IRIG timer for a specified 4G core instead of the default binary timer, see BTICard_IRIGConfig. The 5G binary timer resolution is always 1 µs (*TIMERRESOL_1US*), and has a range of 365 days. The 6G binary timer resolution is always 1 ns and defaults to a range of 365 days. The 6G binary timer is capable of larger ranges, see BTICard_TSMTimerRolloverSet.

WARNINGS

After changing the resolution on a 3G/4G Device, a call to BTICard_TimerClear should be made to clear the timer.

SEE ALSO

BTICard_TimerClear, BTICard_IRIGConfig,
BTICard_IRIGRd, BTICard_IRIGWr,
BTICard_TSMTimerRolloverSet

TimerStatus

```
INT BTICard_TimerStatus
(
  HCORE hCore                    //Core handle
)
```

RETURNS

The status value of the timer configuration.

DESCRIPTION

This function determines the status of how the timer for a core is configured. Some devices have configurable modes that affect elements of data structures. For example, 4G Devices allow for BCD or binary time-tag formatting.

The status value can be tested using the predefined constants below:

Constant	Description
TIMETAG_FORMAT_BCD	Time-tags are in BCD format
TIMETAG_FORMAT_BIN	Time-tags are in binary format

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

3G Devices and 4G Devices (when configured to use binary formatted time-tags) latch the time-tag value when processed by the Device firmware. This will create some minor variability in time-tags from message to message. 6G Devices, 5G Devices, and 4G Devices (when configured to use IRIG BCD formatted time-tags) latch the time-tag value when processed by the Device hardware. This results in a very consistent and accurate time-tag value.

Some ARINC 429 messages of 4G Devices (when configured for binary formatted time-tags) may have identical time-tags due to batch processing of messages in the same Device firmware time slot. This is most noticeable in the Sequential Record when comparing time-tags of messages.

SEE ALSO

BTICard_IRIGConfig, BTICard_IRIGTimeBCDToBin, BTICard_IRIGTimeBinToBCD, BTICard_SeqFind??

TimerWr

```
VOID BTICard_TimerWr
(
  ULONG value,                //Value of the timer
  HCORE hCore                 //Core handle
)
```

RETURNS

None.

DESCRIPTION

Writes `value` to the lower 32-bits of the binary timer of the specified Device.

DEVICE DEPENDENCY

Applies to all Devices.

6G Devices have a 64-bit binary timer with a 1 ns resolution.

5G Devices have a 48-bit binary timer with a 1 µs resolution.

For 3G and 4G Devices, the binary timer exists in two parts: a hardware DSP timer, and a software extended value. Together these values make a 48-bit time value. `BTICard_TimerResolution` is used to adjust which bits of this 48-bit value are used to make the 32-bit time-tag used in message structures and Sequential Records. `BTICard_TimerRd` and `BTICard_TimerWr` functions only read from and write to the software extended portion of this time value and not the hardware portion (due to the complexity of accounting for the rollover from the Host, an accurate reading of both the hardware and software part is not possible). The software extended portion used by `BTICard_TimerRd` and `BTICard_TimerWr` has a resolution of 4.096 ms.

To use the IRIG timer on a 4G Device to generate time-tag values for message structures and Sequential Records instead of the default binary timer, see `BTICard_IRIGConfig`.

WARNINGS

Use caution to account for rollover when using this function. For example, the lower 32-bits of a 1 ns binary timer will rollover approximately every 4 seconds.

Refer to `BTICard_Timer64Wr` for access to the full binary timer.

SEE ALSO

`BTICard_Timer64Wr`, `BTICard_Timer64Rd`, `BTICard_TimerRd`, `BTICard_TimerResolution`, `BTICard_IRIGConfig`, `BTICard_IRIGRd`, `BTICard_IRIGWr`

TSMDriftMaxGet

```
ERRVAL BTICard_TSMDriftMaxGet
(
   LPULONG driftptr,  //Pointer to variable to hold current drift max (ppt)
   HCORE handleval    //Device handle
)
```

RETURNS

A negative value if an error occurs, otherwise zero.

DESCRIPTION

Reads the current value of the maximum drift adjustment limit and places the value into the location pointed to by *driftptr*. Value is in parts-per-trillion.

DEVICE DEPENDENCY

Applies to Core A of all 6G Devices.

WARNINGS

None.

SEE ALSO

BTICard_TSMDriftMaxSet

TSMDriftMaxSet

```
ERRVAL BTICard_TSMDriftMaxSet
(
   ULONG drift,              //Value for maximum drift (ppt)
   HCORE handleval           //Device handle
)
```

RETURNS

A negative value if an error occurs, otherwise zero.

DESCRIPTION

Writes the value *drift* to the Device's maximum drift adjustment limit. This value is the limit the device will attempt to bias the clock frequency to match an incoming source frequency. Value is in parts-per-trillion and is limited to 425 ppm (425,000,000 ppt).

DEVICE DEPENDENCY

Applies to Core A of all 6G Devices.

WARNINGS

None.

SEE ALSO

BTICard_TSMDriftMaxGet

TSMDriftRd

```
ERRVAL BTICard_TSMDriftRd
(
   LPINT driftptr,     //Pointer to variable to hold current drift (ppt)
   HCORE handleval     //Device handle
)
```

RETURNS

A negative value if an error occurs, otherwise zero.

DESCRIPTION

Places the current two's complement representation of the commanded drift into the location pointed to by *driftptr*. Value is in parts-per-trillion.

DEVICE DEPENDENCY

Applies to Core A of all 6G Devices.

WARNINGS

None.

SEE ALSO

`BTICard_TSMDriftWr`

TSMDriftWr

```
ERRVAL BTICard_TSMDriftWr
(
  INT drift,          //Commanded drift value (ppt)
  HCORE handleval     //Device handle
)
```

RETURNS

A negative value if an error occurs, otherwise zero.

DESCRIPTION

Writes the input two's complement representation to the Device's drift adjustment value. When used in conjunction with `BTICard_TSMSyncConfig` called with `TSMCFG_HOST`, the device will adjust the clock frequency by the amount of the input PPT. A negative number represents a slower frequency. Value is in parts-per-trillion and is limited to the range from
-425,000,000 < *drift* < 425,000,000.

DEVICE DEPENDENCY

Applies to Core A of all 6G Devices.

WARNINGS

The Device will not adjust the frequency past the value set by `BTICard_TSMDriftMaxSet`.

SEE ALSO

`BTICard_TSMDriftRd`, `BTICard_TSMSyncConfig`, `BTICard_TSMDriftMaxSet`

TSMInputDelayCompGet

```
ERRVAL BTICard_TSMInputDelayCompGet
(
  LPINT delayptr,     //Pointer to variable to hold current input
                      // compensation (ns)
  HCORE handleval     //Device handle
)
```

RETURNS

A negative value if an error occurs, otherwise zero.

DESCRIPTION

Reads the current value of the input compensation and places the value into the location pointed to by *delayptr*. Value is in nanoseconds.

DEVICE DEPENDENCY

Applies to Core A of all 6G Devices.

WARNINGS

None.

SEE ALSO

BTICard_TSMInputDelayCompSet

TSMInputDelayCompSet

```
ERRVAL BTICard_TSMInputDelayCompSet
(
   INT delay,               //Value for input compensation (ns)
   HCORE handleval          //Device handle
)
```

RETURNS

A negative value if an error occurs, otherwise zero.

DESCRIPTION

Writes the value *delay* to the Device's input compensation. The Device will adjust the internal time by the set amount to account for flight and logic delays in the circuitry and wiring when synchronizing time to an external IRIG or PPS source. Value is in nanoseconds and valid range is 0 to 1000000 (1 ms).

DEVICE DEPENDENCY

Applies to Core A of all 6G Devices.

WARNINGS

None.

SEE ALSO

`BTICard_TSMInputDelayCompGet`, `BTICard_TSMSyncConfig`

TSMInputThresholdGet

```
ERRVAL BTICard_TSMInputThresholdGet
(
   LPUSHORT dacval,    //Pointer to variable to hold current DAC value
   INT pinindex,       //Index indicating which input DAC threshold to read
   HCORE handleval     //Device handle
)
```

RETURNS

A negative value if an error occurs, otherwise zero.

DESCRIPTION

Reads the current value of the input threshold DAC setting and places the value into the location pointed to by *dacval*. Value is an unsigned representation.

Available *pinindex values* are shown below:

pinindex	
Constant	**Description**
TSMPIN_PWMIRIG0	Threshold for IRIG PCM/PPS
TSMPIN_PWMIRIG1	Threshold for IRIG PCM/PPS
TSMPIN_AMIRIGH*	Threshold for IRIG AM, Mark
TSMPIN_AMIRIGL	Threshold for IRIG AM, Space
TSMPIN_10MHZ	Threshold for 10MHz

*By default the Device will auto-calculate the optimal AM IRIG Mark threshold. To disable auto-calculation for AM IRIG Mark threshold, use the flag TSMCFG_AMDACUSER when calling BTICard_TSMSyncConfig.

DEVICE DEPENDENCY

Applies to Core A of all 6G Devices.

WARNINGS

When using AM IRIG, unless BTICard_TSMSyncConfig is called with the TSMCFG_AMDACUSER flag, the Device will automatically calculate the optimal threshold. Overriding the automatic threshold is not recommended for reliable performance.

SEE ALSO

BTICard_TSMInputDelayCompGet, BTICard_TSMSyncConfig

TSMInputThresholdSet

```
ERRVAL BTICard_TSMInputThresholdSet
(
   LPUSHORT dacval,     //Value for selected input threshold
   INT pinindex,        //Index indicating which input DAC threshold to set
   HCORE handleval      //Device handle
)
```

RETURNS

A negative value if an error occurs, otherwise zero.

DESCRIPTION

Sets the selected input threshold DAC value to the input *dacval*. *dacval* is an unsigned representation.

Available *pinindex* are shown below:

pinindex	
Constant	**Description**
TSMPIN_PWMIRIG0	Threshold for IRIG PCM/PPS
TSMPIN_PWMIRIG1	Threshold for IRIG PCM/PPS
TSMPIN_AMIRIGH*	Threshold for IRIG AM, Mark
TSMPIN_AMIRIGL	Threshold for IRIG AM, Space
TSMPIN_10MHZ	Threshold for 10 MHz

*By default the Device will auto-calculate the optimal AM IRIG threshold. To disable auto-calculation for AM IRIG use the flag TSMCFG_AMDACUSER when calling BTICard_TSMSyncConfig.

DEVICE DEPENDENCY

Applies to Core A of all 6G Devices.

WARNINGS

When using AM IRIG, unless BTICard_TSMSyncConfig is called with the TSMCFG_AMDACUSER flag, the Device will automatically calculate the optimal threshold. Overriding the automatic thresholds is not recommended for reliable performance.

SEE ALSO

BTICard_TSMInputDelayCompSet, BTICard_TSMSyncConfig

TSMIRIGControlRd

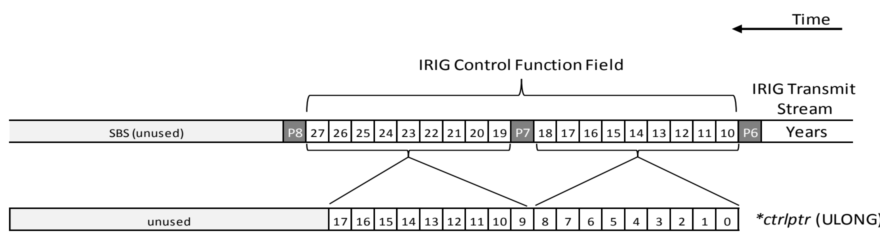
```
ERRVAL BTICard_TSMIRIGControlRd
(
   LPULONG ctrlptr,        //Pointer to variable to hold IRIG control field
   HCORE handleval         //Device handle
)
```

RETURNS

A negative value if an error occurs, otherwise zero.

DESCRIPTION

Reads the last received "Control Function" field from a received IRIG message and places the value into the location pointed to by `ctrlptr`. Control Function bits 10 to 27 are placed in bits 0 to 17 of the ULONG.



DEVICE DEPENDENCY

Applies to Core A of all 6G Devices.

WARNINGS

None.

SEE ALSO

`BTICard_TSMIRIGControlWr`, `BTICard_TSMIRIGYearsWr`, `BTICard_TSMIRIGYearsRd`, `BTICard_TSMSyncConfig`

TSMIRIGControlWr

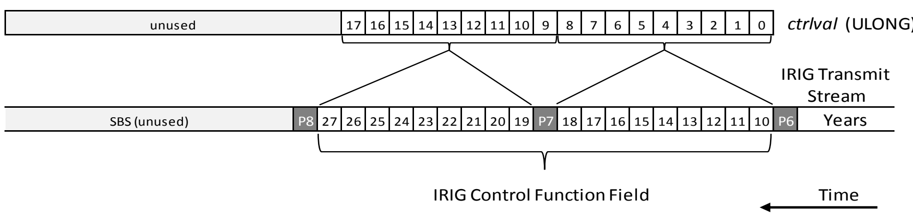
```
ERRVAL BTICard_TSMIRIGControlWr
(
   ULONG ctrlval,          //Value for outgoing IRIG control field
   HCORE handleval         //Device handle
)
```

RETURNS

A negative value if an error occurs, otherwise zero.

DESCRIPTION

Writes the value to the Device to send in the "Control Function" field of outgoing IRIG messages. Control Function bits 10 to 27 are taken from bits 0 to 17 of the *ctrlval*, with Control Function bit 10 located at *ctrlval* bit 0.



DEVICE DEPENDENCY

Applies to Core A of all 6G Devices.

WARNINGS

None.

SEE ALSO

`BTICard_TSMIRIGControlRd`, `BTICard_TSMIRIGYearsWr`, `BTICard_TSMIRIGYearsRd`, `BTICard_TSMSourceConfig`

TSMIRIGYearsRd

```
ERRVAL BTICard_TSMIRIGYearsRd
(
   LPULONG year,           //Pointer to variable to hold year value
   HCORE handleval         //Device handle
)
```

RETURNS

A negative value if an error occurs, otherwise zero.

DESCRIPTION

Reads Device's Year and places the value into the location pointed to by *year*. Value is an unsigned integer value between 2000 and 2099.

When using IRIG time synchronization via `BTICard_TSMSyncConfig` the Device's year value will be automatically populated with the Year field of the received message. Otherwise, the Device's year value can be set by the user via `BTICard_TSMIRIGYearsWr` and the value will increment at the time indicated via `BTICard_TSMTimerRolloverSet`. For IRIG formats without a Years field, this function can be ignored.

DEVICE DEPENDENCY

Applies to Core A of all 6G Devices.

WARNINGS

None.

SEE ALSO

`BTICard_TSMIRIGControlWr`, `BTICard_TSMIRIGControlRd`, `BTICard_TSMIRIGYearsWr`, `BTICard_TSMSyncConfig`, `BTICard_TSMTimerRolloverSet`

TSMIRIGYearsWr

```
ERRVAL BTICard_TSMIRIGYearsWr
(
   ULONG year,              //Value to set Device year
   HCORE handleval          //Device handle
)
```

RETURNS

A negative value if an error occurs, otherwise zero.

DESCRIPTION

Sets the Device's Years to the input value *year*, which is an unsigned integer value between 2000 and 2099.

If Device has been configured via BTICard_TSMSourceConfig to output IRIG, the Device will send *year* in an IRIG message

The Device's Years value will increment at the time indicated via BTICard_TSMTimerRolloverSet.

DEVICE DEPENDENCY

Applies to Core A of all 6G Devices.

WARNINGS

When using IRIG time synchronization via BTICard_TSMSyncConfig the Device's year value will be automatically populated with the Year field of the received message, overwriting the User set value.

SEE ALSO

BTICard_TSMIRIGYearsRd, BTICard_TSMSyncConfig, BTICard_TSMSourceConfig, BTICard_TSMTimerRolloverSet

TSMJumpThresholdGet

```
ERRVAL BTICard_TSMJumpThresholdGet
 (
  LPULONG drift,      //Pointer to variable to hold current offset drift
                      // threshold (ppt)
  LPULONG offset,     //Pointer to variable to hold current offset jump
                      // threshold (ns)
  HCORE handleval     //Device handle
)
```

RETURNS

A negative value if an error occurs, otherwise zero.

DESCRIPTION

Reads the current values of the offset drift threshold and offset jump threshold and places the values into the location pointed to by *drift* and *offset* respectively. *drift* is in Parts-per-Trillion and *offset* is in nanoseconds. See `BTICard_TSMJumpThresholdSet` for details on how these thresholds work.

DEVICE DEPENDENCY

Applies to Core A of all 6G Devices.

WARNINGS

None.

SEE ALSO

`BTICard_TSMJumpThresholdSet`

TSMJumpThresholdSet

```
ERRVAL BTICard_TSMJumpThresholdSet
(
   ULONG drift,              //Value for offset drift threshold (ppt)
   ULONG offset,             //Value for offset jump threshold (ns)
   HCORE handleval           //Device handle
)
```

RETURNS

A negative value if an error occurs, otherwise zero.

DESCRIPTION

Writes the values `drift` and `offset` to the Device's IRIG/PPS drift and time thresholding limits.

When using the Device to synchronize to an single external IRIG/PPS source for both time and drift modes, it is possible to 'drift' the timer to match the external time source time by overcorrecting the frequency of the Device's clock to relatively speed or slow the timer's incrementing. This will cause the timer to move slowly towards the target time value rather than immediately 'jumping' to the received time. This may be useful to prevent discontinuities in protocol timestamps.

The `drift` threshold is used to set the maximum drift adjustment a time delta can force onto the clock frequency. Note that this is in addition to any drift difference due to frequency variation between timing samples and the sum is clipped by the value set by `BTICard_TSMDriftMaxSet`. The value is in parts-per-trillion unsigned representation with a valid range of 0 to 425,000,000.

The `offset` threshold is used to set the minimum offset value that will force the timer to 'jump' to the received time. When the time received via IRIG/PPS is more than the `offset` threshold from the Device's internal time, the Device's time will snap to the received time. Value is in nanoseconds, unsigned representation with a valid range of 0 to 0xFFFFFFFF.

DEVICE DEPENDENCY

Applies to Core A of all 6G Devices.

WARNINGS

None.

SEE ALSO

`BTICard_TSMJumpThresholdGet`, `BTICard_TSMSyncConfig`, `BTICard_TSMDriftMaxSet`

TSMReset

```
ERRVAL BTICard_TSMReset
(
  HCORE handleval            //Device handle
)
```

RETURNS

A negative value if an error occurs, otherwise zero.

DESCRIPTION

Commands the Timing Synchronization Manager to reset. Resets Device time and mode settings.

DEVICE DEPENDENCY

Applies to Core A of all 6G Devices.

WARNINGS

TSM functionality is not reset on `BTICard_CardReset`.

This function will not reset input threshold DAC values.

SEE ALSO

`BTICard_CardReset`

TSMSourceConfig

```
ERRVAL BTICard_TSMSourceConfig
(
  ULONG sourcecfg        //Specifies configuration
  HCORE handleval        //Device handle
)
```

RETURNS

A negative value if an error occurs, otherwise zero.

DESCRIPTION

Sets the Device's TSM as the output timing source with the options defined by *sourcecfg*. The constants below define the available operating modes of the TSM, which is able to source a single IRIG or PPS timing mode and simultaneously output the 10 MHz signal. Use TSMCFG_IRIGA and TSMCFG_IRIGB to select the IRIG speed.

The TSM is able to "relay" an IRIG timing input to a PPS timing output. To use this feature, call BTICard_TSMSyncConfig first with the desired IRIG input, and call BTICard_TSMSourceConfig second with the desired PPS output, ensuring that the input and output are on different pins.

sourcecfg	
Constant	**Description**
TSMCFG_NONE	Select all default settings (**bold** below).
TSMCFG_PPS0	Outputs PPS signal
TSMCFG_PPS1	Outputs PPS signal
TSMCFG_PWMIRIG0	Outputs PWM IRIG signal at speed selected by TSMCFG_IRIGA and TSMCFG_IRIGB
TSMCFG_PWMIRIG1	Outputs PWM IRIG signal at speed selected by TSMCFG_IRIGA and TSMCFG_IRIGB
TSMCFG_AMIRIG	Outputs AM IRIG signal at speed selected by TSMCFG_IRIGA and TSMCFG_IRIGB
TSMCFG_10MHZ	Outputs 10 MHz signal
TSMCFG_IRIGA	Selects 100 ms period IRIG
TSMCFG_IRIGB	Selects 1 s period IRIG

See specific hardware manuals for information regarding TSM pin specifications.

DEVICE DEPENDENCY

Applies to Core A of all 6G Devices.

WARNINGS

None.

SEE ALSO

BTICard_TSMSyncConfig

TSMStatus

```
ERRVAL BTICard_TSMStatus
(
   LPULONG statusptr        //Pointer to value to store current status
   HCORE handleval          //Device handle
)
```

RETURNS

A negative value if an error occurs, otherwise zero.

DESCRIPTION

Reads the Device's TSM status and loads the results into the location specified by `statusptr`. The status value can be tested using the predefined constants below:

*statusptr	
Constant	**Description**
TSMSTAT_IRIGPRES	Input IRIG signal is toggling and decodable
TSMSTAT_IRIGBITSYNC	Input IRIG signal is toggling
TSMSTAT_IRIGSYNC	TSM is locked to IRIG input
TSMSTAT_10MHZPRES	Input 10 MHz signal is toggling
TSMSTAT_10MHZSYNC	TSM is locked to 10 MHz input
TSMSTAT_PPSPRES	Input PPS signal is toggling
TSMSTAT_PPSSYNC	TSM is locked to PPS input
TSMSTAT_OUTOFBOUNDS	Drift control is outside of set drift threshold

DEVICE DEPENDENCY

Applies to Core A of all 6G Devices.

WARNINGS

If the TSM clock and input signal source are drifting more than the drift threshold returned by `BTICard_TSMDriftMaxGet`, the TSM will not lock to the source and the associated synchronization status {`TSMSTAT_IRIGSYNC`, `TSMSTAT_PPSSYNC`, `TSMSTAT_10MHZSYNC`} will not be asserted.

SEE ALSO

`BTICard_TSMSyncConfig`, `BTICard_TSMSourceConfig`, `BTICard_TSMDriftMaxGet`

TSMSyncConfig

```
ERRVAL BTICard_TSMSyncConfig
(
   ULONG timeconfig        //Specifies time configuration
   ULONG driftconfig       //Specifies drift configuration
   HCORE handleval         //Device handle
)
```

RETURNS

A negative value if an error occurs, otherwise zero.

DESCRIPTION

Sets the Device's TSM to synchronize time and/or drift to an external source with options defined by *timeconfig* and *driftconfig* as shown in the tables below.

The TSM can synchronize time to a single IRIG or PPS input as set by *timeconfig* and drift to a single IRIG, PPS or 10 MHz input as set by *driftconfig*.

Using time synchronization with an IRIG source will set the Device's timer to the time received via IRIG. Using time synchronization with a PPS source will round the timer to the nearest second at the PPS timing mark. When using a PPS input source, it is recommended to call BTICard_TSMTimerWr to set the Device's time to the system time.

Using the TSM drift controls (set by *driftconfig*) will speed or slow the Device's clock by up to 425 PPM to account for the clock drift between the Device clock and the input source clock.

If BTICard_TSMSyncConfig is not called, the host can still adjust the time using BTICard_TSMTimerWr.

timeconfig	
Constant	**Description**
TSMCFG_NONE	Select all default settings (**bold** below). Timer is free-running
TSMCFG_PPS0	Synchronizes time to PPS signal
TSMCFG_PPS1	Synchronizes time to PPS signal
TSMCFG_PWMIRIG0	Synchronizes time to PCM IRIG signal at speed selected by TSMCFG_IRIGA and TSMCFG_IRIGB
TSMCFG_PWMIRIG1	Synchronizes time to PCM IRIG signal at speed selected by TSMCFG_IRIGA and TSMCFG_IRIGB
TSMCFG_AMIRIG	Synchronizes time to AM IRIG signal at speed selected by TSMCFG_IRIGA and TSMCFG_IRIGB
TSMCFG_IRIGA	Selects 100 ms period IRIG
TSMCFG_IRIGB	Selects 1 s period IRIG
TSMCFG_AMDACAUTO	Automatically selects appropriate AM IRIG voltage threshold
TSMCFG_AMDACUSER	Allows user to select AM IRIG voltage threshold

driftconfig	
Constant	**Description**
TSMCFG_NONE	Select all default settings (**bold** below). Clock frequency is unbiased.
TSMCFG_PPS0	Adjusts clock frequency to match that of source of PPS signal
TSMCFG_PPS1	Adjusts clock frequency to match that of source of PPS signal
TSMCFG_PWMIRIG0	Adjusts clock frequency to match that of source of PCM IRIG signal
TSMCFG_PWMIRIG1	Adjusts clock frequency to match that of source of PCM IRIG signal
TSMCFG_AMIRIG	Adjusts clock frequency to match that of source of AM IRIG signal
TSMCFG_10MHZ	Adjusts clock frequency to match that of source of 10 MHz signal
TSMCFG_HOST	Allows user to adjust clock frequency via BTICARD_TSMDriftWr
TSMCFG_IRIGA	Selects 100 ms period IRIG
TSMCFG_IRIGB	Selects 1 s period IRIG
TSMCFG_AMDACAUTO	Automatically selects appropriate AM IRIG voltage threshold
TSMCFG_AMDACUSER	Allows user to select AM IRIG voltage threshold

The TSM can use drift and time controls simultaneously with valid configurations of sources defined in a table below.

Drift Input	Time Input
None	None
None	IRIG
None	PPS
10 MHz	None
10 MHz	IRIG
10 MHz	PPS
IRIG	IRIG
PPS	None
PPS	IRIG
PPS	PPS

TSM valid drift and time combinations

See specific hardware manuals for information regarding TSM pin specifications.

DEVICE DEPENDENCY

Applies to Core A of all 6G Devices.

WARNINGS

None.

SEE ALSO

BTICard_TSMSourceConfig

TSMTimerRd

```
ERRVAL BTICard_TSMDTimerRd
(
  LPULONGLONG valueptr,   //Pointer to variable to hold current time (ns)
  HCORE handleval         //Device handle
)
```

RETURNS

A negative value if an error occurs, otherwise zero.

DESCRIPTION

Reads the current value of the Device timer and places the value into the location pointed to by `valueptr`. Value is in nanoseconds.

DEVICE DEPENDENCY

Applies to Core A of all 6G Devices.

WARNINGS

None.

SEE ALSO

`BTICard_TSMTimerWr`

TSMTimerRelWr

```
ERRVAL BTICard_TSMTimerRelWr
(
   LONGLONG value,           //Value to jump timer (ns)
   HCORE handleval           //Device handle
)
```

RETURNS

A negative value if an error occurs, otherwise zero.

DESCRIPTION

Commands the Device to 'jump' the timer by the number of nanoseconds in *value*. A negative *value* will 'jump' the timer backwards in time.

DEVICE DEPENDENCY

Applies to Core A of all 6G Devices.

WARNINGS

'Jumping' the timer backwards can cause out of order protocol timestamps.

Using a *value* that is large enough to push the timer value over the rollover value will cause the timer to immediately rollover and start incrementing from zero. Writing a negative number with magnitude larger than the current timer value will cause the timer to wrap around and rollover (depending on current rollover value).

SEE ALSO

BTICard_TSMTimerWr

TSMTimerRolloverGet

```
ERRVAL BTICard_TSMTimerRolloverGet
(
  LPULONGLONG valueptr,  //Pointer to variable to hold timer rollover (ns)
  HCORE handleval        //Device handle
)
```

RETURNS

A negative value if an error occurs, otherwise zero.

DESCRIPTION

Reads the Device's current timer rollover and places the value into the location pointed to by *valueptr*. Value is in nanoseconds. Default rollover value is exactly 365 days, but can be adjusted by `BTICard_TSMTimerRolloverSet` to account for leap years and seconds. Values possible range from 0 to 2^{64}-1.

DEVICE DEPENDENCY

Applies to Core A of all 6G Devices.

WARNINGS

None.

SEE ALSO

`BTICard_TSMTimerRolloverSet`

TSMTimerRolloverSet

```
ERRVAL BTICard_TSMTimerRolloverSet
(
   ULONGLONG value,        //Value for timer rollover (ns)
   HCORE handleval         //Device handle
)
```

RETURNS

A negative value if an error occurs, otherwise zero.

DESCRIPTION

Writes *value* to the Device's timer rollover. This will signal the Device to increment the Years field and reset the timer to zero. Default rollover value is exactly 365 days, but can be adjusted to account for leap years and leap seconds. *value* is in nanoseconds and valid range is 0 to 2^{64}-1.

DEVICE DEPENDENCY

Applies to Core A of all 6G Devices.

WARNINGS

None.

SEE ALSO

BTICard_TSMTimerRolloverGet

TSMTimerWr

```
ERRVAL BTICard_TSMTimerWr
(
  ULONGLONG value,        //Value to set timer (ns)
  HCORE handleval         //Device handle
)
```

RETURNS

A negative value if an error occurs, otherwise zero.

DESCRIPTION

Commands the Device to set the timer to the input *value*.
Valid range is 0 to 2^{64}-1.

DEVICE DEPENDENCY

Applies to Core A of all 6G Devices.

WARNINGS

Writing a timer value higher than the timer rollover (set by BTICard_TSMTimerRolloverSet) will cause the timer to rollover to zero.

SEE ALSO

BTICard_TSMTimerRd, BTICard_TSMTimerRelWr, BTICard_TSMTimerRolloverSet

ValFromAscii

```
ULONG BTICard_ValFromAscii
(
  LPCSTR asciistr,          //ASCII string to convert
  INT radixval              //Radix of string
)
```

RETURNS

The converted integer numeric value.

DESCRIPTION

Converts a string representation of a 32-bit value with the specified radix to an integer. Processing stops at the first null terminator. *radixval* can be any positive integer, but is commonly 16 for hexadecimal, 8 for octal, or 10 for decimal.

Note: This is a utility function and does not access the Device hardware.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

No check is made for invalid characters.

SEE ALSO

BTICard_ValToAscii

ValGetBits

```
ULONG BTICard_ValGetBits
(
  ULONG oldvalue,          //The old value
  INT startbit,            //Position of starting bit of field
  INT endbit               //Position of ending bit of field
)
```

RETURNS

The value of the extracted bit field.

DESCRIPTION

Extracts the specified bit field from the 32-bit integer *oldvalue*. The result is obtained by masking the field and shifting the *endbit* to the LSB of the return value. The LSB is bit number zero.

Note: This is a utility function and does not access the Device hardware.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

None.

SEE ALSO

BTICard_ValPutBits

ValPutBits

```
USHORT BTICard_ValPutBits
(
   ULONG oldvalue,          //The old value
   ULONG newfld,            //The value of the new field
   INT startbit,            //Position of starting bit of field
   INT endbit               //Position of ending bit of field
)
```

RETURNS

The integer value with the inserted bit field.

DESCRIPTION

Inserts a bit field into a 32-bit integer value. The *oldval* is masked and OR-ed with the shifted value of *newfld*. The LSB is bit number zero.

Note: This is a utility function and does not access the Device hardware.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

None.

SEE ALSO

BTICard_ValGetBits

ValToAscii

```
LPSTR BTICard_ValToAscii
(
   ULONG value,                //The value to be converted
   LPSTR asciistr,             //A string to receive the results
   INT numbits,                //The number of significant bits
   INT radixval,               //The radix value
)
```

RETURNS

An ASCII string representing the integer.

DESCRIPTION

Creates a string representation of an integer in a specified radix. The string is copied to *asciistr* and is also returned. The string is always null-terminated. *asciistr* is assumed to be large enough to hold the resulting string.

The length of the string is determined by *numbits* and *radixval* and is padded by leading zeros. *radixval* can be any positive integer but is commonly 16 for hexadecimal, 8 for octal, or 10 for decimal. For example, a string representation of a value with 16 significant bits and a radix of 16 will always be 4 characters long followed by a null terminator.

Note: This is a utility function and does not access the Device hardware.

DEVICE DEPENDENCY

Applies to all Devices.

WARNINGS

None.

SEE ALSO

BTICard_ValFromAscii

APPENDIX C: MULTI-PROTOCOL / DEVICE PROGRAMS

A single software application can be written to simultaneously operate many similar or dissimilar Ballard BTIDriver-compliant products, each supporting a single or multiple avionics databus protocols. This appendix provides information needed to write software programs to control multiple Devices and Devices that support more than one protocol.

Programming Rules

Guidelines for writing multi-Device and multi-protocol programs are summarized in the following rules. The discussion in the rest of this appendix further explains these rules.

1. A card number for each Device is assigned by the operating system. If only one BTIDriver-compliant Device exists on the system, it is assigned card number zero (0) by the operating system.
2. A core number for each core on the Device is set by the architecture of the Device. If only one core exists on the Device, it is core number zero.
3. A test utility is provided with the Device for indicating and associating the card number with each individual Device and the core number of each core. A utility for reassigning the card number may also be included with the Device. The card numbers assigned to BTIDriver-compliant Devices are specific to them, so there is no conflict when devices that are not BTIDriver-compliant use those same card numbers.
4. The card handle returned by `BTICard_CardOpen` is passed to `BTICard_CoreOpen` to obtain the core handle used by all channels and all protocols on that core.
5. The recommended programming practice is to use the card handle only in `BTICard_CoreOpen` and `BTICard_CardClose` (i.e., to obtain core handles and to release the resources back to the operating system at the end of the program). All other functions needing a handle should use the core handle.
6. If a card handle is used in place of a core handle, it has the same effect as when the handle for core number zero is used. Programs for single-core Devices can be written without using core handles, but they would be more easily ported to other Devices by following the recommendation of using core handles.
7. Card functions (those prefixed with `BTICard_`) are shared with all protocols and channels on the core specified by the core handle. For instance, `BTICard_CardStart` starts all channels on the core (independent of protocol). Note that using a card handle with this function only starts channels on core number zero.
8. Different protocol functions may be interleaved in the program between the common `BTICard_` functions.

BTICard_ Functions

BTICard_ functions are common to all protocols supported by the core. When a BTICard_ function is used, all protocols on the core specified by the core handle are affected. Programs supporting different protocols may be combined into a single program by interleaving the protocol-specific functions with common BTICard_ functions. A normal application would use BTICard_CardOpen and BTICard_CardClose once for each Device and BTICard_CoreOpen once for each core. Similarly, BTICard_ functions like BTICard_CardStart and BTICard_CardStop apply to all channels and protocols on the specified core.

Sequential Record

Each core has one Sequential Record, independent of how many different protocols it supports. The format of individual records within the Sequential Record differs between protocols. There are two ways of scanning through a Sequential Record: by protocol-specific records or by every record. To scan by protocol, use the BTICard_SeqFindNext?? function to find the next record with the ?? protocol. For instance, the BTICard_SeqFindNext429 and BTICard_SeqFindNext1553 functions are used to find the next ARINC 429 or MIL-STD-1553 record respectively. To scan through every record, use the BTICard_SeqFindNext function, which finds the next record and returns the type (429, 1553, etc.) of the record it found. The different BTICard_SeqFindNext?? functions should not be mixed within a sequence without first using BTICard_SeqFindInit. Note that the BTICard_SeqFindNext?? functions do not use a handle, so they do not access the Device. They work from a copy of the Sequential Record in the computer's memory. Thus, they may be used to process a Sequential Record that had been previously saved to a hard disk.

Event Log List

As with the Sequential Record, there is one Event Log List per core, independent of how many protocols are supported. However, all records in the Event Log List have the same format. To determine the cause of the event and the protocol associated with it, test the type value passed through BTICard_EventLogRd. There are some event types that are common between protocols and some that are unique to specific protocols.

Using Multiple Devices

A program that uses more than one Device can be viewed as a combination of programs for the individual Devices. Every BTIDriver function in the individual programs would appear in the combined program and may be interleaved so as to provide the desired functionality. All BTICard_ functions affect only the Device specified by the handle (e.g. each Device needs its own BTICard_CardOpen and BTICard_CardClose functions, and each core needs its own BTICard_CoreOpen, BTICard_CardStart, and BTICard_CardStop functions). If interrupts are used, there should be separate interrupt service threads for each core on each Device. In a similar way, non-BTIDriver-compliant devices may be combined into the program using the API for that device(s).

APPENDIX D: REVISION HISTORY

The following revisions have been made to this manual:

Rev A. Date: August 10, 2001

Original release of this manual.

Rev. B. Date: June 13, 2003

Major revision of both the text and the appendices to accommodate the features and functionality of multi-core Devices. Many functions are new or modified.

Rev. C. Date: October 13, 2008

Major revision of both the text and the appendices to accommodate the features and functionality of 5G Devices. Many functions are new or modified.

Rev. C.1 Date: August 7, 2009

Minor revision of both the text and appendices to fix typos and add clarifications to function usage.

Rev. D Date: January 5, 2010

Major revision of both the text and the appendices to accommodate the features and functionality of USB 1553 Devices.

Rev. E Date: June 13, 2011

Major revision of both the text and the appendices to accommodate the features and functionality of Mx5 Devices. Added BIT and SysMon functions.

Rev. F Date: September 25, 2012

Major revision of the appendices to accommodate the features and functionality of PM1553-5 and AB3000 Series Devices.

Rev. G Date: November 21, 2013

Minor revision of both the text and the appendices to accommodate SysMon function warnings and list function warning. Added CoPilot section and updated Monitor GUI. Added AB3000, PM1553-5, USB 708, and USB Multi device references. Some function descriptions modified.

Rev. H.0 Date: May 12, 2016

Added/Deleted text in the main body of the manual with some new sections including *Unscheduled List Transmission* and *TSM*. Major revision to the appendices to accommodate MIL-STD-1553 functions in Appendix A and new BTICard Functions in Appendix B. Added/deleted text in Appendices as well.

This page intentionally blank.