1553Px

Software Tools

Programmer's Reference

311 Meacham Ave ♦ Elmont ♦ NY 11003 ♦ tel. (516) 327-0000 ♦ fax (516) 327-4645
e-mail: excalibur@mil-1553.com website: www.mil-1553.com

Table of Contents

1 Introduction

Chapter 1 provides an overview of the *1553Px* module (or *Px* for short).

The following topics are covered:

Overview

Welcome to the *1553Px Software Tools*. The *1553Px Software Tools* are C language functions designed to aid users to write test programs. These functions provide access to all of the module's functions in a structured and straightforward programming environment.

The *1553Px Software Tools* distributed with the product are compatible with the Windows® operating system. The Windows drivers were written and tested using Microsoft Visual C++.

This manual describes the module-level software functions. Board-level software functions are described in the *Excalibur Carrier Board Software Tools Software Tools Programmer's Reference*.

Note: The *EXC-1553ExCARD/Px*, *EXC-1553UNET/Px* and *ES-1553RUNET/Px* use both module-level and board-level software tools. For board-level software functions, refer to the *Excalibur Carrier Board Software Tools Programmer's Reference*.

All references to a module apply both to the removable *Px* module and to the *Px* module integrated on a board or card. Therefore, this manual applies to all current *Px* products.

Note: This manual is for firmware revision 8.0 or later. For earlier firmware revisions, refer to the appropriate *Programmer's Reference* for your firmware revision.

Getting Started

Note: The following sections describe the installation procedure. This section is not relevant to the *UNET* or *RUNET*. For these products, refer to the Installation chapter of the *EXC-1553UNET/Px & ES-1553RUNET/Px User's Manual*.

Before starting to write applications:

1. Install your board. For instructions, see **`Installation Instructions.pdf`** in the root folder of the *Excalibur Installation CD*.
2. Use the *Excalibur Installation CD* to install the software for your board and modules. For instructions, see **`Installation Instructions.pdf`**.
3. Locate the hardware manual (user's manual) and the software manual (programmer's reference) on the *Excalibur Installation CD*, for your board and modules.
4. Copy them to your computer.
5. Fill out the registration card and return it to your Excalibur representative.

Note: If anything is missing or damaged, contact your Excalibur representative.

Installation

For hardware and software installation instructions, see **`Installation Instructions.pdf`** in the root folder of the installation CD. When downloading new software from the Excalibur website, **`Installation Instructions.pdf`** is contained in the zip file.

The *Excalibur Installation CD* you received with your package is the most recent release of the CD as of the date of shipping. Software and documentation updates can be found and downloaded from our website: www.mil-1553.com.

The standard software provided with Excalibur boards and modules is for Windows operating systems. For more details, see **`Installation Instructions.pdf`**. Software for other operating systems may be available. Check on our website or write to excalibur@mil-1553.com.

Assigning a Device Number

ExcConfig is used to assign a device number of 0 – 15 to the board, which is used when running Excalibur's *Software Tools*. The first function generally called in an application program is Init_Module, and Init_Module requires the device number as one of its parameters.

ExcConfig assigns a device number by creating an association between the selected device number and the Unique Identifier of the board. It stores this information in the Windows Registry.

The Unique Identifier is set by a DIP switch or jumpers on the board. (For more details, see your board's user's manual. In the user's manual, the Unique Identifier is called the Selected ID.) For ExpressCard and PCMCIA cards, there are no DIP switches or jumpers for setting the Unique Identifier; the Socket Number is used instead.

Note: When only one board of the same type is installed in your computer, you have the option of using the board's default device number instead of running ExcConfig. However, you cannot use the default device number when you have two or more boards in the computer that have the same default device number, or if your board does not have a default device number. The following table lists the default device numbers for most board types.

Board Type	Default Device Number	#define Value
UNET, RUNET, USB	None – the board's device number must be set via ExcConfig	N/A
VME	None – the board's device number must be set via a DIP switch (or jumper)	N/A
Ethernet, 664 (AFDX)	34 (dec)	EXC_ETHERNET_PCIE *or* EXC_664_PCIE
1394	32 (dec)	EXC_1394PCI
EXC-1553[c]PCI/MCH	29 (dec)	EXC_1553PCIMCH
All Other Current PCI[e] Boards (Including VPX and XMC)	25 (dec)	EXC_4000PCI

Note:

- You can use any combination of up to 16 PCIe, PCI or cPCI boards (or cards) simultaneously. To use more than four boards, you must be using *Software Tools* released on or after September 2007.
- The **ExcConfig** utility is not for use with VME/VXI boards. To configure single or multiple VME/VXI boards, see **`Installation Instructions.pdf`**.

An Overview of the Data Communications Bus

The *1553Px Software Tools* enable the user to create custom diagnostic programs for the Excalibur *Px* module. To use the *1553Px Software Tools*, some familiarity with the functioning of the 1553 data communications bus and an understanding of the hardware is required. This overview is an introduction to the 1553 bus. It will assist in understanding the rationale of the software functions for the hardware.

Military Standard 1553 (MIL-STD-1553) has been a mainstay of military avionics communications since its introduction in 1975. This specification defines a data communications bus capable of supporting up to 32 devices, called Remote Terminals, and coordinated by a device called a Bus Controller. The **Data communications bus** is physically composed of two wires used to transmit and receive a differential signal between the various devices. For backup purposes, a second pair of wires is often used and is called the secondary bus. The two buses are also referred to as "Bus A" and "Bus B."

Each **Remote Terminal** can transfer data to or receive data from the Bus Controller or another Remote Terminal in response to a command from the Bus Controller.

Remote Terminals for their part never initiate communications but merely react to the Bus Controller.

The **Bus Controller** directs the flow of data on the data bus. Since all transmission takes place over a single pair of wires, some mechanism must be included to ensure that only a single device attempts to transmit at any given time. MIL-STD-1553 deals with this issue by dictating that the Bus Controller must initiate all communication. The Bus Controller determines the sequence of messages, the size of each message, and the timing between messages. The Bus Controller also determines which of the Remote Terminals will be transmitting or receiving data.

The **Bus Monitor** is a passive device which records 1553 bus traffic. A monitor may collect all the data from the bus or may collect selected data.

Data is transmitted in the form of **messages**. Messages consist of a Command word containing routing information transmitted by the Bus Controller, between 1 and 32 Data Words containing the information to be communicated, and a Status Word transmitted by a Remote Terminal acknowledging receipt of a command. A number of message types are defined in MIL-STD-1553. The most common message types involve data transfer either from the Bus Controller (BC) to a single Remote Terminal (RT) or from an RT to the BC. Less frequently, data may be broadcast from the Bus Controller to all RTs or from one RT directly to a second RT. In all cases, the type of message is encoded in the Command word that is transmitted by the BC.

The MIL-STD-1760 Option

MIL-STD-1760 implements an enhanced MIL-STD-1553 digital interface for the transfer of digital messages to the remote terminal. The enhancements include:

Use of Checksums	The use of checksums for the standard message, stored in place of the last Data Word of the message.
Support for Header Word	Support for the Header Word as the first word in a message, including the predefined Header Words.

Checksums

Checksum is mandated on critical control messages and provisional on the remainder of the messages. Implementing this level of error detection ensures a higher degree of error free data integrity requirements than that which odd parity provides.

The 1760 option implements checksum error detection capabilities. Checksums are computed as each Data Word is sent or received. If the checksum flag is set on an outgoing message, the checksum will be sent in place of the last Data Word. On an incoming message, the computed checksum is checked against the last Data Word received. If it does not match, the Checksum Error bit is set in the Message Status word.

The 1760 option has the additional feature of checksum error injection in BC/ Concurrent RT mode. The user can set the checksum to intentionally send an

error, giving the additional capability to test for checksum errors on the receiving RTs.

Header Word

In the MIL-STD-1760 specification, the first Data Word of a message may be a Header Word, which is used for message identification. The header word is associated with a specific RT subaddress.

To indicate that a specific subaddress will require a Header Word, set the corresponding entry in the 1760 Header Active table to 1. Then set the corresponding entry in the 1760 Header Value table to the value you expect to receive in the first Data Word of the message.

The MIL-STD-1760 specification provides predefined header values. These are preset on Excalibur hardware according to the operating mode: Remote Terminal, BC/Concurrent-RT or Bus Monitor.

Single Function Option

The *PxS* is a single function version of the standard *Px* module. It operates as a single Remote Terminal (RT), a Bus Controller (BC) or Triggerable Bus Monitor, with an Internal Concurrent monitor for RT and BC operation.

The differences between the standard *Px* module and the single function module (*PxS*) are as follows:

- The *PxS* simulates only one RT at a time
- It has a fixed amplitude
- It has no error injection capability
- It has only one LED (Module Ready)
- The single RT address is set by calling Set_RT_Active_Px with an option to set the RT address via the module connector
- Illegalization is done based on the SAid, not the RTid, by calling Set_SAid_Illegal_Px
- The module's response time cannot be greater than 12,000 nanoseconds (see Set_RT_Resp_Time_Px)

For information on which functions are available to the single function module, see the lists of functions at the beginning of chapters **2**, **3** and **5**.

Compiler Options

The DLL is compiled under Microsoft Visual Studio using **_cdecl** options. The driver functions in the *1553Px Software Tools* are supplied both in source form and linked as a DLL. When writing application-programs, keep in mind that the module is a physical resource, and therefore you cannot run multiple copies of the program simultaneously, accessing the same module.

Each function is presented with its formal definition, including data types of all input and output variables. A brief description of the purpose of the function is provided along with the legal values for inputs where applicable.

Functions are written as 'C' functions, i.e., they return values. A negative value signifies an error. Full error messages may be printed using the Get_Error_String_Px function. (See **Appendix E-3: Error Messages,** on page E-11.)

In Windows all user-defined programs must include the file **`proto_px.h.`** This file includes all the necessary header files and DLL functions to operate the *1553Px Software Tools* for *Px* modules.

Direct Memory Access (DMA)

Direct Memory Access (DMA) is available with PCI Express-based products. DMA enables the board, card or module to read from, or write to, system memory independently of the computer's CPU. This results in faster data transfer from the board, with much less CPU overhead than when not using DMA.

When DMA is available, the *1553Px Software Tools* use DMA access for functions that read from, or write to, memory.

The following functions use DMA:

Get_Next_Message_BCM_Px
Get_Next_Message_Px
Get_Next_Message_RTM_Px
Get_Next_Mon_Message_Px

Conventions Used in this Manual

To help differentiate between different kinds of information, the following text styles are used in this manual:

Functions look like this.

Parameters look like this.

`File names` look like this.

FLAGS look like this.

Technical Support

Excalibur Systems is ready to assist you with any technical questions you may have. For technical support, visit the Technical Support page of our website (www.mil-1553.com). You can also contact us by phone. To find the location nearest you, visit to the Contact Us page of our website. Before contacting Technical Support, please see Information Required for Technical Support.

2 General Functions

Chapter 2 contains the software functions that can be used in at least two modes. The input values included in each function are given, in Hex format, by each flag within the function description.

All references to a module apply both to the removable *Px* module and to the *Px* module integrated on a board or card. For more information, see Introduction on page 1-1.

The following functions are described in this chapter:

Clear_Timetag_Sync_Px
Convert_IrigTimetag_BCD_to_Decimal_Px
DelayMicroSec_Px
Enable_1553A_Support_Px
External_Loopback_Px
Get_Board_Options_Hi_Px
Get_Board_Status_Px
Get_Card_Type_Px
Get_DmaAvailable_Px
Get_Error_String_Px
Get_Header_Exists_Px
Get_Header_Value_Px
Get_Id_Px
Get_Instr_Px[1]
Get_Mode_Px
Get_ModuleIrigTime_Px
Get_ModuleTime_Px
Get_Next_Mon_Message_Px
Get_PCMCIA_HWInterface_Rev_Px
Get_Rev_Level_Px
Get_SerialNumber_Px
Get_UseDmaIfAvailable_Px
GetTimeTag_Px
Global_Ttag_Reset_PCMCIA_Px
Host_Reset_PCMCIA_Px
Init_Module_Px for Most Boards
Init_Module_Px for VME and VXI Boards
Internal_Loopback_Px
Is_Enhanced_Mode_Supported_Px
Is_Expanded_Mode_Supported_Px
Is_IrigSignal_Present_Px
Is_Single_Function_Px
OnBoard_Loopback_Px
Parse_CommandWord_Px
Preset_IrigTimeTag_Px
Print_Error_Px
PrintIRIGtime_Px
Read_Start_Reg_Px
Release_Module_Px
Reset_Time_Tag_Px
Restart_Px
Set_Bit_Cnt_Px[2]
Set_Expanded_Block_Mode_Px
Set_Header_Exists_Px
Set_Header_Value_Px
Set_Interrupt_Px
Set_IRIG_TimeTag_Mode_Px
Set_Mode_Px
Set_ModuleTime_Px
Set_RT_Active_Px[3]
Set_RT_Resp_Time_Px[3]
Set_Timetag_Res_Px
Set_UseDmaIfAvailable_Px
Set_Var_Amp_Px[2]
Stop_Px

1. Only for VME/VXI systems running VISA
2. Not for single function module (*PxS*)
3. Not for single function module (*PxS*) in BC mode

To handle interrupts in Windows see Using Interrupts in Windows on page 2-42. The functions described are:

Get_Interrupt_Count_Px
InitializeInterrupt_Px
Wait_For_Interrupt_Px
Wait_For_Multiple_Interrupts_Px

Clear_Timetag_Sync_Px

Description Clear_Timetag_Sync_Px resets the Time Tag register to 0:

- **BC mode** When the BC sends out a Mode Code synchronize message.
- **RT mode** When the RT receives the selected Mode Code synchronize message.
- **Monitor mode** When the selected Mode Code synchronize message is detected on the bus.

Syntax Clear_Timetag_Sync_Px (`int` handle, `int` modecode, `int` flag)

Input Parameters

handle	The handle designated by Init_Module_Px	
modecode	The selected Mode Code: 1 or 17	
flag	ENABLE	Reset Time Tag at Mode Code 'Sync' [0001 H]
	DISABLE	Do not reset Time Tag at Mode Code 'Sync' [0000 H]

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
einval	If an invalid parameter was used as an input.
0	If successful

Convert_IrigTimetag_BCD_to_Decimal_Px

Description Convert_IrigTimetag_BCD_to_Decimal_Px converts IRIG B Time Tag from BCD representation (an array of 4 words) to decimal representation (`struct` t_IrigOnModuleTime).

Syntax Convert_IrigTimetag_BCD_to_Decimal_Px (`unsigned short int` *moduleTime, `t_IrigOnModuleTime` *IrigTime)

Input Parameters

moduleTime	A pointer to an array of four words containing the IRIG B Time Tag (four BCD digits).

Output Parameters

IrigTime	A structure (`t_IrigOnModuleTime`) containing the IRIG B time stamp in decimal digits, defined in **pxIncl.h** as:

```
typedef struct {
        unsigned short int days;
        unsigned short int hours;
        unsigned short int minutes;
        unsigned short int seconds;
        unsigned int microsecs;
} t_IrigOnModuleTime;
```

Return Values

0	If successful

DelayMicroSec_Px

Description DelayMicroSec_Px is used to suspend execution within an application for a specified amount of time. This function uses the on-board Time Tag to determine how much time has passed since the start of waiting period. It is more accurate than the Windows API Sleep function.

Syntax DelayMicroSec_Px (`int` handle, `int` delayMicroSec)

Input Parameters

handle	The handle designated by Init_Module_Px
delayMicroSec	The number of microseconds to delay

Output Parameters none

Return Values

ebadhandle	If invalid handle specified; should be value returned by Init_Module_Px
ebadttag	If there is a problem calling GetTimeTag_Px
ettagexternal	If the Time Tag source is set to an external clock source
0	If successful

Enable_1553A_Support_Px

Description Enable_1553A_Support_Px enables MIL-STD-1553A support.

- **BC mode** No Mode Codes with data
- **RT mode** The Broadcast bit is not set in the 1553 Status word.
- **Monitor mode** No Mode Codes with data

Syntax Enable_1553A_Support_Px (`int` handle, `int` enableflag)

Input Parameters

handle	The handle designated by Init_Module_Px	
enableflag	ENABLE	Enable MIL-STD-1553A support [0001 H]
	DISABLE	Disable MIL-STD-1553A support [0000 H]

Output Parameters none

Return Values

ebadhandle	If invalid handle specified; should be value returned by Init_Module_Px
einval	If an invalid value was used as an input
emode	If module is not in BC or RT mode
0	If successful

External_Loopback_Px

Description

External_Loopback_Px is used to check the 1553 transceivers, transformers and associated bus cables. For more information, see **Appendix D: External and OnBoard Loopback Tests**.

Note:

- External_Loopback_Px requires a loopback cable to connect Bus A to Bus B. For more information, see **Appendix E Application of External Loopback Test** in the *M8K1553Px Module User's Manual* or the *M4K1553Px Module User's Manual*.
- When running a loopback, all values previously written to registers and memory addresses are erased.
- This function is not available for Monitor mode-only cards/modules.

Syntax

External_Loopback_Px (`int` handle, `struct e_loopback` *elbvals)

Input Parameters

handle — The handle designated by Init_Module_Px

Output Parameters

elbvals

struct E_LOOPBACK

{

	Address in Dual-Port RAM	Status Value
usint frame_val;	0	X (not for user)
usint frame_status;	2	8000H passed, 8001H failed
usint cmd_send[8];	4	cmd_send[0]: 5555H
	6	cmd_send[1]: 8000H passed, else failed
	8	cmd_send[2]: 1234H
	A	cmd_send[3]: 8000H passed, else failed
	C	cmd_send[4]: 5555H
	E	cmd_send[5]: 8000H passed, else failed
	10	cmd_send[6]: 1234H
	12	cmd_send[7]: 8000H passed, else failed
usint ttag_val_lo	14	30D4H ± 2
usint ttag_val_hi	16	0
usint ttag_status;	18	8000H passed, 8001H failed

} *E_loopback;

Return Values

elbfailure	If External Loopback test failed
ebadhandle	If invalid handle specified; should be value returned by Init_Module_Px.
emonmode	If the card/module only operates in Monitor mode
0	If successful

Get_Board_Options_Hi_Px

Description Get_Board_Options_Hi_Px indicates the type of firmware on the module, and some features that are currently in use.

Syntax Get_Board_Options_Hi_Px (`int` handle, `int` *boardopt)

Input Parameters

handle	The handle designated by Init_Module_Px	

Output Parameters

boardopt	A '1' in the specified bit indicates:	
	Bit 8	Firmware for 1553
	Bit 9	Firmware for 1760
	Bit 10	Intel processor (0 indicates old hardware, AMD processor)
	Bit 11	Replay mode is in use (BC mode only)
	Bit 12	*EXC-1553ExCARD/Px* or *EXC-1553PCMCIA/Px* card (0 indicates *M8K1553Px* or *M4K1553Px* module)
	Bit 13	Expanded Block mode is in use in RT mode. (To check whether Expanded Block mode is supported in the current mode, use Is_Expanded_Mode_Supported_Px. See page 2-23.)
	Bit 15	NIOS II processor

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
0	If successful

Get_Board_Status_Px

Description Get_Board_Status_Px indicates the status of the specified module.

Syntax Get_Board_Status_Px (`int` handle)

Input Parameters

handle	The handle designated by Init_Module_Px

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
einval	If an invalid value was used as an input
0 or more of the following flags:	
BOARD_READY	Module has completed reset sequence [0001 H]

Get_Board_Status_Px (cont.)

RAM_OK	Module has passed internal memory self-test [0002 H]
TIMERS_OK	Module has passed internal timers self-test [0004 H]
SELF_TEST_OK	Module has passed full internal self-test [0008 H]
BOARD_HALTED	Module is stopped [0010 H]

Note: Bit 7 is always set to 1

Get_Card_Type_Px

Description Get_Card_Type_Px indicates the module firmware type.

Syntax Get_Card_Type_Px (`int` handle, `usint` *cardtype)

Input Parameters

handle — The handle designated by Init_Module_Px

Output Parameters

cardtype

One of the following flags:

MOD_1760	Standard MIL-STD-1760 module [1760 H]
MOD_1760_MON_ONLY	Monitor-only MIL-STD-1760 module [4D17 H]
MOD_1760_SF	Single function MIL-STD-1760 module [1F17 H]
MOD_1553	Standard MIL-STD-1553 module [1553 H]
MOD_1553_MON_ONLY	Monitor-only MIL-STD-1553 module [4D53 H]
MOD_1553_SF	Single function MIL-STD-1553 module [1F53 H]

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
0	If successful

Get_DmaAvailable_Px

Description Get_DmaAvailable_Px returns an indicator as to whether Direct Memory Access (DMA) is available for the transfer of data between the host memory and the *Px* module memory. See also Set_UseDmaIfAvailable_Px on page 2-40.

Note: This function is only for *Px* modules on a PCI Express board or ExpressCards®.

Syntax Get_DmaAvailable_Px (`int` handle, `char` *pDmaEnabled)

Input Parameters	handle	The handle designated by Init_Module_Px
Output Parameters	pDmaEnabled	A pointer to whether DMA is available.
	ENABLE	DMA is available [0001 H]
	DISABLE	DMA is not available [0000 H]
Return Values	0	If successful
	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px

Get_Error_String_Px

Description Get_Error_String_Px accepts the error returned from other *1553Px Software Tools* functions. This function returns the string containing a corresponding error message.

See **Appendix E-3 Error Messages** on page E-11.

Syntax Get_Error_String_Px (`int` errcode, `char` *errstring)

Example

```
char ErrorStr[255];
Get_Error_String_Px (errorcode, ErrorStr);
printf("error is: %s", ErrorStr)
```

Input Parameters	errorcode	The error code returned from a *1553Px Software Tools* function.
Output Parameters	errstring	A string of characters, with the corresponding error message. In case of bad input, a string denoting that.
Return Values	0	Always

Get_Header_Exists_Px

Description

Get_Header_Exists_Px checks to see if the Subaddress (SA) has an assigned Header Word Value.

Note: This function is applicable *only* to modules that support the 1760 option.

Syntax

Get_Header_Exists_Px (`int` handle, `int` sa, `int` *enable)

Input Parameters

handle	The handle designated by Init_Module_Px
sa	The Subaddress to be checked

Output Parameters

enable:	Legal flags are:	
	1	Header Word assigned
	0	Header Word *not* assigned

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
rterror	Bad Subaddress number
0	If successful

Get_Header_Value_Px

Description

Get_Header_Value_Px gets the Header Word assigned to this Subaddress.

The predefined header values are:

Transmit Subaddress	Header Value
1	0421 H
11	0420 H
14	0423 H

Receive Subaddress	Header Value
11	0400 H
14	0422 H

Note: This function is applicable *only* to modules that support the 1760 option.

Syntax

Get_Header_Value_Px (`int` handle, int `sa`, int `direction`, usint *header_value)

Input Parameters

handle	The handle designated by Init_Module_Px	
sa	The Subaddress to be checked	
direction	Message type to assign this header to:	
	TRANSMIT	[0001 H]
	RECEIVE	[0000 H]

Output Parameters

header_value	The assigned Header Word

Get_Header_Value_Px (cont.)

Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	rterror	Bad Subaddress number
	0	If successful

Get_Id_Px

Description	Get_Id_Px returns the module ID of the specified module.	
Syntax	Get_Id_Px (`int` handle)	
Input Parameters	handle	The handle designated by Init_Module_Px
Output Parameters	none	
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	Anything other than module ID	Failure
	Module ID: E [0045 H]	If successful

Get_Instr_Px

Description	Get_Instr_Px returns the handle supplied by the VISA driver when it opened a session to the VME/VXI board.	
	Note: This function is only for VME/VXI systems running VISA.	
Syntax	Get_Instr_Px (`int` handle, `unsigned long int` *instr)	
Input Parameters	handle	The handle designated by Init_Module_Px
Output Parameters	instr	A pointer to the handle supplied by the VISA driver.
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	0	If successful

Get_Mode_Px

Description Get_Mode_Px indicates the current mode of the specified module reflecting the last call to Set_Mode_Px.

Syntax Get_Mode_Px (`int` handle)

Input Parameters

handle	The handle designated by Init_Module_Px

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
einval	If an invalid value was used as an input

One of the following flags:

RT_MODE	Module set up as a Remote Terminal [0002 H]
BC_RT_MODE	Module set up as BC/Concurrent RT [0004 H]
MON_BLOCK	Module set up as a Monitor in Sequential Block submode [0008 H]
MON_LOOKUP	Module set up as Monitor in Table Lookup submode [0020 H]

Get_ModuleIrigTime_Px

Description Get_ModuleIrigTime_Px returns the current module level IRIG B time stamp in decimal representation.

Note: This function is supported only on Nios-based module with firmware released after February 2015.

Syntax Get_ModuleIrigTime_Px (`int` handle, `t_IrigOnModuleTime` *IrigTime)

Input Parameters

handle	The handle designated by Init_Module_Px

Output Parameters

IrigTime	A structure (`t_IrigOnModuleTime`) containing the IRIG B time stamp in decimal digits, defined in **pxIncl.h** as:

```
typedef struct {
        unsigned short int days;
        unsigned short int hours;
        unsigned short int minutes;
        unsigned short int seconds;
        unsigned int microsecs;
} t_IrigOnModuleTime;
```

Get_ModuleIrigTime_Px (cont.)

Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	ewrongprocessor	If the processor is not a NIOS II
	efeature_not_ supported_PX	If IRIG B Time Tag feature is not supported on this module
	0	If successful

Get_ModuleTime_Px

Description

Get_ModuleTime_Px returns the time from the Module Time register.

Note:
- Calling this function resets the module and all data is lost.
- The Module Time register is not the same as the Time Tag register.

Syntax

Get_ModuleTime_Px (`int` handle, `unsigned int` *moduleTime)

Input Parameters	handle	The handle designated by Init_Module_Px
Output Parameters	moduleTime	A 32 bit value representing the module time
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	ewrongprocessor	If the processor is not a NIOS II
	0	If successful

Get_Next_Mon_Message_Px

Description

Get_Next_Mon_Message_Px reads the message block following the message block read in the previous call to Get_Next_Mon_Message_Px.

This function can be used in all modes, instead of Get_Next_Message_Px in Monitor mode, Get_Next_Message_BCM_Px in BC mode or Get_Next_Message_RTM_Px in RT mode.

However, in Monitor mode, it is recommended to use Get_Next_Message_Px instead of this function, since it has been optimized to be more efficient.

Syntax

Get_Next_Mon_Message_Px (`int` handle, `struct MONMSG` *msgptr, `unsigned long` *msg_counter)

Input Parameters	handle	The handle designated by Init_Module_Px

Get_Next_Mon_Message_Px (cont.)

Output Parameters	msgptr	Pointer to an address in host memory where you want a copy of the message to be placed. The message is returned in a MONMSG structure, as defined below. Space should always be allocated for 36 words of data to accommodate the maximum case of RT-to-RT transmission of 32 Data Words plus 2 Status words and 2 Command words.
	struct MONMSG {	
	unsigned short int msgstatus;	Status word containing the flags as defined in the corresponding function each mode; for the list of flags, see: Get_Next_Message_Px in Monitor mode Get_Next_Message_BCM_Px in BC mode or Get_Next_Message_RTM_Px in RT mode
	unsigned int elapsedtime;	The Time Tag associated with the message, with a precision of 4 microseconds per bit
	unsigned short int words [36];	A pointer to an array of 1553 words in the sequence they were received over the bus
	unsigned short int msgCounter;	The low 16 bits of the message number assigned to this message. The high 16 bits are stored in the Message Number Hi register [3EBC H].
	}	
Return Values	emode	If current mode is not BC, RT or Sequential Monitor mode
	enomsg	If no new messages have been received
	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	func_invalid	If the module is a *MMSI* module
	ewrongprocessor	If the processor is not a NIOS II
	eoverrunTtag	If the Time Tag of the message was overwritten while the message was being read

Get_Next_Mon_Message_Px (cont.)

eoverrunEOMflag	If the End Of Message flag was overwritten while the message was being read
eIrigTimetagSet_PX	If this function was called when in IRIG B Time Tag mode (Set_IRIG_TimeTag_Mode_Px); use Get_Next_Mon_IRIGttag_Message_Px instead
block number	If successful

Get_PCMCIA_HWInterface_Rev_Px

Description Get_PCMCIA_HWInterface_Rev_Px returns the host interface revision of the hardware.

Note: This function only applies to the *EXC-1553PCMCIA/Px* and *EXC-1553PCMCIA/EP* cards.

Syntax Get_PCMCIA_HWInterface_Rev_Px (`int` handle, `usint` * hwInterfaceRev)

Input Parameters

handle	The handle designated by Init_Module_Px

Output Parameters

hwInterfaceRev	A pointer to the hardware's host interface revision

Return Values

func_invalid	If is not an *EXC-1553PCMCIA/Px* card
ekernelinitmodule	If error initializing kernel related data
ekernelbadparam	If input parameter is invalid
ekernelbadpointer	If output parameter buffer is invalid
ekerneldevicenotopen	If specified device has not been opened
ekernelcantmap	If a pointer to memory cannot be obtained
0	If successful

Get_Rev_Level_Px

Description Get_Rev_Level_Px indicates the current revision level of the firmware for the specified module. Each Hex nibble (i.e. 4 bits) reflects a level, so the 0x11 should be interpreted as revision 1.1.

Syntax Get_Rev_Level_Px (`int` handle)

Input Parameters	handle	The handle designated by Init_Module_Px
Output Parameters	none	
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	firmware revision level	If successful

Get_SerialNumber_Px

Description Get_SerialNumber_Px returns the serial number of the module.

Syntax Get_SerialNumber_Px (`int` handle, `usint` *serialNum)

Input Parameters	handle	The handle designated by Init_Module_Px
Output Parameters	serialNum	The serial number of the module
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
Return Values	ewrongprocessor	If the processor is not a NIOS II
	0	If successful

Get_UseDmaIfAvailable_Px

Description Get_UseDmaIfAvailable_Px returns the indicator as to whether DMA will be used for data transfer between the host memory and the *Px* module memory, if DMA is available.

Note: This function is only for *Px* modules on a PCI Express board or ExpressCards®.

Syntax Get_UseDmaIfAvailable_Px (`int` handle, `int` *pUseDmaFlag)

Input Parameters	handle	The handle designated by Init_Module_Px
Output Parameters	pUseDmaFlag	A pointer to a flag indicating whether DMA will be used for data transfer (if available):
		ENABLE — DMA will be used if available [0001 H]
		DISABLE — DMA will not be used [0000 H]

Get_UseDmaIfAvailable_Px (cont.)

Return Values	ebadhandle	If invalid handle specified; should be value returned by Init_Module_Px.
	0	If successful

Get_Message_Count_Px

Description	Get_Message_Count_Px returns the running message count of transmitted or received messages (depending on the current mode).	
Syntax	Get_Message_Count_Px (`int` handle, `unsigned int` *msgCount)	
Input Parameters	handle	The handle designated by Init_Module_Px
Output Parameters	msgCount	A pointer to the total messages sent or received (depending on the current mode)
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	0	If successful

GetTimeTag_Px

Description	GetTimeTag_Px returns the running Time Tag of the module. The default Time Tag resolution is 4 µsec., but can be changed for RT and Monitor modes using Set_Timetag_Res_Px. When using a Time Tag resolution of 4 µsec., the Time Tag wraps around after approximately 4 hours and 45 minutes.	
Syntax	GetTimeTag_Px (`int` handle, `unsigned int` *Time_Tag)	
Input Parameters	handle	The handle designated by Init_Module_Px
Output Parameters	Time_Tag	A pointer to the 32-bit Time Tag
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	0	If successful

Global_Ttag_Reset_PCMCIA_Px

Description

Global_Ttag_Reset_PCMCIA_Px resets the Time Tag on each *EXC-1553PCMCIA/Px* channel. When using a two channel card, this synchronizes the channels.

Note:

- Calling Init_Module_Px, Host_Reset_PCMCIA_Px or Set_Mode_Px resets the module's Time Tag, which causes the modules to become desychronized. In order to keep the modules synchronized, call this function any of these functions.
- This function only applies to the *EXC-1553PCMCIA/Px* card.

Syntax

Global_Ttag_Reset_PCMCIA_Px (`int` handle)

Input Parameters

handle	The handle designated by Init_Module_Px

Output Parameters

none

Return Values

func_invalid	If is not an *EXC-1553PCMCIA/Px* card
ekernelinitmodule	If error initializing kernel related data
ekernelbadparam	If input parameter is invalid
ekernelbadpointer	If output parameter buffer is invalid
ekerneldevicenotopen	If specified device has not been opened
ekernelcantmap	If a pointer to memory cannot be obtained
0	If successful

Host_Reset_PCMCIA_Px

Description Host_Reset_PCMCIA_Px performs a host reset and reloads the firmware onto the card's channels, for PCMCIA/Px only

Note: This function only applies to the *EXC-1553PCMCIA/Px* card.

Syntax Host_Reset_PCMCIA_Px (`int` handle)

Input Parameters	handle	The handle designated by Init_Module_Px
Output Parameters	none	
Return Values	func_invalid	If is not an *EXC-1553PCMCIA/Px* card
	ekernelinitmodule	If error initializing kernel related data
	ekernelbadparam	If input parameter is invalid
	ekernelbadpointer	If output parameter buffer is invalid
	ekerneldevicenotopen	If specified device has not been opened
	ekernelcantmap	If a pointer to memory cannot be obtained
	0	If successful

Init_Module_Px for Most Boards

This function is for 1553Px modules on all boards and devices except for VME/VXI boards.

Description Init_Module_Px is the first function the user must call for each *Px* module on each device that is accessed in the application program. Before exiting a program, call Release_Module_Px for each module initialized with Init_Module_Px.

Note: This function initializes only one module. Call this function once for each module you want to initialize.

The function may be called with the SIMULATE argument. If the SIMULATE argument is used, a portion of the memory equal to the size of the board's dual-port RAM is set aside. This area is then initialized with an ID and version number for use in testing programs when no module is available.

Multiple modules may be simulated. It is possible to have real or SIMULATED modules in one application. And more than one module may be SIMULATED simultaneously.

Syntax Init_Module_Px (`usint` device_num, `usint` module_num)

Init_Module_Px for Most Boards (cont.)

	device_num	The device number is the index number of the board set in `ExcConfig`: 0 – 15 *or* SIMULATE [FFFF H] **Note:** If only one board of the same type is used, the #define value EXC_4000PCI (or the value 25) can be used instead of a device number. If more than one board is used the programmer must run the `ExcConfig` utility to set the device number.
	module_num	The module number of the *1553Px* module on the board: 0 – 15 (depending on the board). You can run **demo_information_1553.exe** to check the module number. For ***EXC-1553PCMCIA/Px***: EXC_PCMCIA_MODULE [100] EXC_PCMCIA_MODULE_2ND [101]
Output Parameters	none	
Return Values	esim_no_mem	If not enough memory available for simulation
	exreset	If Xilinx failed to reset
	eboardtoomany	If too many modules initialized; this module not initialized
	egetirq	If there was an error getting the IRQ number
	emodnum	Invalid module number specified
	enomodule	If no module is present at specified location
	ewrngmodule	If module specified on carrier board is not a *Px* module
	etimeoutreset	If timed out waiting for reset
	eallocresources	If there was an error allocating resources
	ekernelnot4000card	If board is not from EXC-4000/8000 family
	ekernelinitmodule	If error initializing kernel related data

Init_Module_Px for Most Boards (cont.)

ekernelbadparam	If input parameter is invalid
eopenkernel	If there was an error opening a device
ekernelfrontdeskload	It there was an error loading **`frontdesk.dll`**
ekernelfrontdesk	If there was an error opening kernel device; check **`ExcConfig`** set up
eallocresources	If there was an error allocating resources
regnotset	If the board is not configured; reboot after running **`ExcConfig`** and board is in slot
ekernelbadpointer	If output parameter buffer is invalid
ekerneldevicenotopen	If specified device has not been opened
ekernelcantmap	If a pointer to memory cannot be obtained
handle	If successful, the handle to the specified module on the board. This handle is used as the first parameter in all module functions. A valid handle is a positive number ranging from 0 – 16.

Init_Module_Px for VME and VXI Boards

Description

Init_Module_Px for VME is the first function the user must call for a VME or VXI board. The function may be called with the SIMULATE argument.

Init_Module_Px must be called once per module (in multi-module) applications. Up to 8 modules can be used or simulated per board. The number of VME or VXI boards which can be used or simulated simultaneously, depends on the system used.

Before exiting a program, call Release_Module_Px for each module for which Init_Module_Px was called.

Syntax

Init_Module_Px (`usint` device_num, `usint` module_num)

Input Parameters

device_num — A device number, which is the board number (0 – 255), as set with the DIP switches.
or
SIMULATE [FFFF H]

Note: For more information on the DIP switch settings, see the **`Installation Instructions.pdf`** file in the installation folder and in the root folder of the *Excalibur Installation CD*.

module_num — The module number of the *1553Px* module on the board: 0 - 7 (depending on the board).

Output Parameters

none

Return Values

eboardnotfound	If device number is not valid
esim_no_mem	If not enough memory available for simulation
etimeoutreset	If timed out waiting for reset
eopendefaultrm	If error in VISA function viOpenDefaultRM
eviopen	If error in VISA function viOpen
evimapaddress	If error in VISA function viMapAddress
einval	If an invalid parameter was used as an input
emodnum	Invalid module number specified
eboardtoomany	If too many modules initialized; this module not initialized
enomodule	If no module is present at specified location
ewrngmodule	If module specified on carrier board is not a *Px* Module
evicommand	If error in VISA command
efuncinvalid	If function is invalid for this board
edevnum	If specified device number is greater than 255
0	If successful

Internal_Loopback_Px

Description

Internal_Loopback_Px is used to check the 1553 front-end logic, excluding transceivers and coupling transformers. For more information, see **Appendix C: Internal Loopback Test**.

Note: When running a loopback, all values previously written to registers and memory addresses are erased.

Syntax

Internal_Loopback_Px (`int` handle, `struct i_loopback` *ilbvals)

Input parameters

handle — The handle designated by Init_Module_Px

Output parameters

ilbvals

struct I_LOOPBACK

{

	Address in Dual-Port Ram	Status Value
usint frame_val;	0	X (not for user)
usint frame_status;	2	8000H passed, 8001H failed
usint resp_status;	4	8000H passed, 8001H failed
usint early_val;	6	6 LSB must be 15H
usint receive_data1;	8	5555H
usint status_1;	A	8000H passed, else failed
usint receive_data2;	C	AAAAH
usint status_2;	E	8000H passed, else failed
usint mc_status;	10	8000H passed, else failed
usint ttag_val_lo;	12	30D4H ± 2
usint ttag_val_hi;	14	0
usint ttag_status;	16	8000H passed, 8001H failed
usint prl;	18	8 LSB contain the CPU version

} *I_loopback;

Return values

ilbfailure	If Internal Loopback test failed.
ebadhandle	If invalid handle specified; should be value returned by Init_Module_Px.
0	If successful

Is_Enhanced_Mode_Supported_Px

Description

Is_Enhanced_Mode_Supported_Px is used to check whether the module supports Enhanced Monitor mode. For information on Enhanced Monitor mode, see Set_Enhanced_Mon_Px on page 4-20.

Note: The module must be in Sequential Monitor mode to call this function.

Syntax

Is_Enhanced_Mode_Supported_Px (`int` handle, `int` *isFeatureSupported)

Input Parameters

handle	The handle designated by Init_Module_Px

Output Parameters

isFeatureSupported	A pointer to whether Enhanced Monitor mode is supported:
	TRUE Enhanced Monitor mode is supported in the current mode [0001 H]
	FALSE Enhanced Monitor mode is not supported in the current mode [0000 H]

Return Values

ebadhandle	If invalid handle specified; should be value returned by Init_Module_Px.
emode	If module is in Table Lookup Bus Monitor mode
0	If successful

Is_Expanded_Mode_Supported_Px

Description Is_Expanded_Mode_Supported_Px is used to check whether the module supports Expanded Block mode in the current mode, RT, Bus Monitor or BC/Concurrent-RT. For information on Expanded Block mode, see Set_Expanded_Block_Mode_Px on page 2-31.

Syntax Is_Expanded_Mode_Supported_Px (`int` handle, `int` *isFeatureSupported)

Input Parameters

handle	The handle designated by Init_Module_Px

Output Parameters

isFeatureSupported	A pointer to whether Expanded Block Mode is supported: TRUE Expanded Block Mode is supported in the current mode [0001 H] FALSE Expanded Block Mode is not supported in the current mode [0000 H]

Return Values

ebadhandle	If invalid handle specified; should be value returned by Init_Module_Px.
emode	If module is in Table Lookup Bus Monitor mode
0	If successful

Is_IrigSignal_Present_Px

Description Is_IrigSignal_Present_Px returns whether an external IRIG B signal is being received by the board. (When no IRIG B signal is received, the board updates the IRIG B Time Tag based on an internal oscillator.)

Note: This function is supported only on Nios-based module with firmware released after February 2015.

Syntax Is_IrigSignal_Present_Px (`int` handle, `unsigned short` *isIrigSignalPresent)

Input Parameters

handle	The handle designated by Init_Module_Px

Output Parameters

isIrigSignalPresent	A pointer to an indicator of whether an IRIG B signal is present. 1 IRIG B signal is present 0 IRIG B signal is not present

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
ewrongprocessor	If the processor is not a NIOS II

Is_IrigSignal_Present_Px (cont.)

	efeature_not_ supported_PX	If IRIG B Time Tag feature is not supported on this module
	0	If successful

Is_Single_Function_Px

Description	Is_Single_Function_Px returns whether the module is single function (*PxS*) module or a multifunction module.	
Syntax	Is_Single_Function_Px (`int` handle)	
Input Parameters	handle	The handle designated by Init_Module_Px
Output Parameters	none	
Return Values	0	If multifunction module
	1	If single function (*PxS*) module

OnBoard_Loopback_Px

Description

OnBoard_Loopback_Px is used to check 1553 operation, including transceivers and transformers. For more information, see **Appendix D: External and OnBoard Loopback Tests**.

Note:

- This function is available for the *M8K1553Px-LB, M8K1553PxS-LB, M4K1553Px-LB, M4K1553PxS-LB*, *EXC-1553[cc]PMC/Px* and *EXC-1553ccVME/Px*.
- When this loopback test is running, no signals are transmitted out of the board onto the bus.
- When running a loopback, all values previously written to registers and memory addresses are erased.

Syntax

OnBoard_Loopback_Px (`int` handle, `struct e_loopback` *oblbvals)

Input Parameters

handle	The handle designated by Init_Module_Px

Output Parameters

oblbvals

struct E_LOOPBACK

{	Address in Dual-Port RAM	Status Value
usint frame_val;	0	X (not for user)
usint frame_status;	2	8000H passed, 8001H failed
usint cmd_send[8];	4	cmd_send[0]: 5555H
	6	cmd_send[1]: 8000H passed, else failed
	8	cmd_send[2]: 1234H
	A	cmd_send[3]: 8000H passed, else failed
	C	cmd_send[4]: 5555H
	E	cmd_send[5]: 8000H passed, else failed
	10	cmd_send[6]: 1234H
	12	cmd_send[7]: 8000H passed, else failed
usint ttag_val_lo	14	30D4H ± 2
usint ttag_val_hi	16	0
usint ttag_status;	18	8000H passed, 8001H failed
} *E_loopback;		

Return Values

elbfailure	If Onboard Loopback test failed
ebadhandle	If invalid handle specified; should be value returned by Init_Module_Px.
enoonboardloopback	If the board/module does not support the Onboard Loopback feature
0	If successful

Parse_CommandWord_Px

Description Parse_CommandWord_Px parses a 1553 Command word into its components (RT number, subaddress, direction (T/R), word count or Mode Code, and RTid).

Syntax Parse_CommandWord_Px (`usint` cmdWord, `int` *RT, `int` *SA, `int` *direction, `int` *wordCount, `int` *RTid)

Input Parameters	cmdWord	A 1553 Command word
Output Parameters	RT	The RT number
	SA	The subaddress
	direction	1 for Transmit, 0 for Receive
	wordCount	The number of words in the message; for Mode Codes, this is the Mode Code
	RTid	The MSB 11 bits of the Command word, comprising the RT, subaddress and direction
Return Values	none	

Preset_IrigTimeTag_Px

Description Preset_IrigTimeTag_Px loads values (day, hours, minutes and seconds) into the IRIG Preload registers. These value are loaded when any value is written to the IRIG B Load Counter (at 70D8 (H)) and the IRIG B counter is updating from an internal source. (See "Is_IrigSignal_Present_Px" on page 24.) The IRIG B Time Tag continues to update from the preload values based on the internal clock.

Syntax Preset_IrigTimeTag_Px (`int` handle, `usint` days, `usint` hours, `usint` minutes, `usint` seconds)

int handlepx,

Input Parameters	handle	The handle designated by Init_Module_Px
	days	The number of the days in the year, counting from January 1; 1 = January 1 (1–366)
	hours	Hour of the day (0–23)
	minutes	Minute of the hour (0–59)
	seconds	Second of the minute (0–59)
Output Parameters	none	
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	ewrongprocessor	If the processor is not a NIOS II
	efeature_not_ supported_PX	If IRIG B Time Tag feature is not supported on this module
	0	If successful

Print_Error_Px

Description

Print_Error_Px has been superseded by Get_Error_String_Px. The function has been retained for backward compatibility.

Print_Error_Px accepts the error code returned from other *1553Px Software Tools* functions. This function returns a char pointer to a string containing a corresponding error message. (See **Appendix E-3 Error Messages** on page E-11 for a complete list of error messages.)

Syntax

char * Print_Error_Px (`int` errorcode)

Input Parameters

errorcode	The error code returned from a *1553Px Software Tools* call

Output Parameters

none

Return Values

a pointer	Pointer to a message string that contains a corresponding error message. In case of bad input this function points to a string denoting "No such message".

PrintIRIGtime_Px

Description

PrintIRIGtime_Px prints (**sprintf**) the IRIG B Time Tag components into a string. This function is used on the value returned by Get_ModuleIrigTime_Px or the Time Tag in a message using the function Get_Next_Mon_IRIGttag_Message_Px.

Syntax

PrintIRIGtime_Px (`t_IrigOnModuleTime` *IrigTime, `char` *outString)

Input Parameters

IrigTime	A structure (`t_IrigOnModuleTime`) containing the IRIG B time stamp in decimal digits, defined in **pxIncl.h** as:

```
typedef struct {
        unsigned short int days;
        unsigned short int hours;
        unsigned short int minutes;
        unsigned short int seconds;
        unsigned int microsecs;
} t_IrigOnModuleTime;
```

outString	A string containing the visual display of the IRIG B Time Tag value, in the format: **Day Month Hour:Minute:Second.Microseconds**

Return Values

none

Read_Start_Reg_Px

Description	Read_Start_Reg_Px returns the value in the Start register, indicating whether or not the module is currently operating.	
Syntax	Read_Start_Reg_Px (`int` handle, `usint` *startval)	
Input Parameters	handle	The handle designated by Init_Module_Px
Output Parameters	startval	Value in the start register
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	0	If successful

Release_Module_Px

Description	Release_Module_Px must be called for each module initialized with Init_Module_Px, before exiting a program. Following a call to this function, the user must call Init_Module_Px before selecting another module or release memory allocated for simulated modules.	
Syntax	Release_Module_Px (`int` handle)	
Input Parameters	handle	The handle designated by Init_Module_Px
Output Parameters	none	
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	0	If successful

Reset_Time_Tag_Px

Description	Reset_Time_Tag_Px resets the module's Time Tag register.	
Syntax	Reset_Time_Tag_Px (`int` handle)	
Input Parameters	handle	The handle designated by Init_Module_Px
Output Parameters	none	
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	0	If successful

Restart_Px

Description Restart_Px continues processing following Set_Stop_on_Error_Px.

Syntax Restart_Px (`int` handle)

Input Parameters	handle	The handle designated by Init_Module_Px
Output Parameters	none	
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	0	If successful

Set_Bit_Cnt_Px

Description Set_Bit_Cnt_Px determines how many bits will be sent for messages that have requested BIT_CNT_ERR errors. The number sent will be the legal number of bits in a word (20 bits) plus the offset.

In BC/Concurrent-RT mode, this function works in conjunction with other error injection functions. For detailed information on error injection, see Error Injection on page 5-5.

Note: This function is related to error injection, and is not available for the single function module (*PxS*).

Syntax Set_Bit_Cnt_Px(`int` handle, `int` offset)

Input Parameters	handle	The handle designated by Init_Module_Px
	offset	Number of bits to add or subtract from the correct number of bits in a 1553 word (20). Valid values: -3 to +3
Output Parameters	none	
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	emode	If module is not in BC or RT mode
	einval	If parameter is out of range
	enotforsinglefuncerror	If this function is called with a single function module (*PxS*)
	0	If successful

Set_Expanded_Block_Mode_Px

Description

Set_Expanded_Block_Mode_Px expands the number of available blocks:

RT Mode

To use blocks 256 through 511, call this function with the argument flag set to ENABLE. This causes part of the memory used for the Concurrent Monitor to be assigned to the additional RT blocks. (This reduces the number of messages in the Concurrent Monitor to 204 from the standard 409.)

To reduce the available blocks in RT mode to 256 and restore the Concurrent Monitor size to 409 blocks, call this function with flag set to DISABLE.

Sequential Monitor Mode

To use blocks 200 through 799, call this function with the argument flag set to ENABLE.

BC/Concurrent-RT

To expand the Message Block Area, call this function with the argument flag set to ENABLE. When the Message Block Area is expanded, there is no Concurrent Monitor. Note that in most cases it is not recommended to use Expanded Block mode in BC/Concurrent-RT mode, since the standard Message Block Area is generally sufficient.

Note:

- Expanded Block mode is disabled by default.
- If the module is running when this function is called, it does not take effect until the module is restarted.
- In Bus Monitor mode, Expanded Block mode cannot be enabled when Enhanced Monitor is enabled. See Set_Enhanced_Mon_Px on page 4-20.
- See also Is_Expanded_Mode_Supported_Px on page 2-24.

Syntax

Set_Expanded_Block_Mode_Px (int handle, int flag)

Input Parameters

handle	The handle designated by Init_Module_Px	
flag	ENABLE	Turns on expanded block mode [0001 H]
	DISABLE	Turns off expanded block mode [0000 H]

Output Parameters

none

Set_Expanded_Block_Mode_Px (cont.)

Return Values

efeature_not_supported_PX	If the feature is not supported on this module
func_invalid	If firmware does not support expanded mode
ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
einval	If parameter is out of range (flag is not ENABLE or DISABLE)
0	If successful

Set_Header_Exists_Px

Description

Set_Header_Exists_Px indicates that messages associated with this Subaddress are expected to have the specified Header Word.

Note: This function is applicable *only* to modules that support the 1760 option.

Syntax

Set_Header_Exists_Px (`int` handle, `int` sa, `usint` enable)

Input Parameters

handle	The handle designated by Init_Module_Px	
sa	The Subaddress assigned a Header Word	
enable	Valid values depend of the current mode:	
BC/Concurrent-RT mode	HEADER_ENABLE	Header Word expected on Transmit messages [0001 H]
	HEADER_DISABLE	No Header Word expected [0000 H]
RT mode	HEADER_ENABLE	Header Word expected on Receive messages [0001 H]
	HEADER_DISABLE	No Header Word expected [0000 H]

Set_Header_Exists_Px (cont.)

	Monitor mode	HEADER_ENABLE_RCV	Header Word expected on Receive messages [0003 H]
		HEADER_ENABLE_XMT	Header Word expected on Transmit messages [0002 H]
		HEADER_ENABLE	Header Word expected on both Transmit and Receive messages [0001 H]
		HEADER_DISABLE	No Header Word expected [0000 H]
Output Parameters	none		
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px	
	rterror	If a bad Subaddress number is selected	
	0	If successful	

Set_Header_Value_Px

Description Set_Header_Value_Px sets the Header Word assigned to this Subaddress.

Note: This function is applicable *only* to modules that support the 1760 option.

Syntax Set_Header_Value_Px (`int` handle, `int` sa, `int` direction, `usint` header_value)

Input Parameters	handle	The handle designated by Init_Module_Px	
	sa	The Subaddress associated with this Header Word	
	direction	Message type to assign this header to:	
		TRANSMIT	[0001 H]
		RECEIVE	[0000 H]
	header_value	The assigned Header Value	
Output Parameters	none		
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px	
	einval	If an invalid parameter was used as an input	
	rterror	If a bad Subaddress number is selected	
	0	If successful	

Set_Interrupt_Px

Description

Set_Interrupt_Px describes the conditions under which the module is to generate or disable interrupts.

Note:
- In RT mode, in order to generate interrupts, this function must be called in conjunction with Set_RTid_Interrupt_Px or Set_RT_Active_Px.
- This function replaces Set_RT_Interrupt_Px and Reset_RT_Interrupt_Px.

Syntax

Set_Interrupt_Px (`int` handle, `int` flag)

Input Parameters

handle	The handle designated by Init_Module_Px
flag	Conditions must be appropriate to the current mode. Multiple conditions (flags) maybe ORed together.

BC/Concurrent-RT mode

MSG_CMPLT	Message sent [0002 H]
END OF FRAME	Complete frame of messages sent [0004 H]

Note: When an interrupt is configured for an END OF FRAME, the interrupt pulse occurs immediately after the last message transmission is complete.

MSG_ERR	Error occurred [0008 H]
SRQ_MSG	Interrupt when the Bus Controller completes processing an SRQ request from an RT [0020 H]
END_MINOR_FRAME	Minor Frame completed [0010 H]

Note: When an interrupt is configured for an END_MINOR_FRAME, the interrupt pulse occurs immediately after the last message transmission in the minor frame is complete.

RT mode

MSG_CMPLT	Message sent [0002 H]

Sequential Monitor mode

TRIG_RCVD	Received message, matched trigger [0001 H]
MSG_IN_PROGRESS	Message in progress [0002 H]
MSG_INTERVAL	The specified number of messages was received [0008 H]

Note: See Set_Message_Interval_Interrupt_Value_Px on page 4-21.

CNT_TRIG_MATCH	Set_Cnt_Trig_Px block number message received [0004 H]

Set_Interrupt_Px (cont.)

Look-up Table Monitor	MSG_IN_PROGRESS	Message in progress [0002 H]
All modes	0	Do not set interrupts of any kind
Output Parameters	none	
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	eintr	If attempted to set an undefined interrupt
	noirqset	If no interrupt allocated
	emode	If the module mode is not recognized
	0	If successful

Set_IRIG_TimeTag_Mode_Px

Description	Set_IRIG_TimeTag_Mode_Px configures the module to record an IRIG B Time Tag (4 words) in each message received in Sequential Monitor mode.	
Syntax	Set_IRIG_TimeTag_Mode_Px (`int` handle, `int` flag)	
Input Parameters	handle	The handle designated by Init_Module_Px
	flag	ENABLE Enable IRIG Time Tag mode [0001 H]
		DISABLE Disable IRIG Time Tag mode [0000 H]
Output Parameters	none	
Return Values	einval	If an invalid parameter was used as an input
	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	efeature_not_ supported_PX	If IRIG B Time Tag feature is not supported on this module
	0	If successful

Set_Mode_Px

Description

Set_Mode_Px sets the mode that the relevant module on the board is to operate. It performs a reset, clearing all the memory on the module and initializes the module to its default values.

This function must be called prior to any mode specific function.

Note:

- Set_Mode_Px may be called just for reset purposes; if you wish to reset *only* the function variables in BC mode, Clear_Card_Px or Clear_Frame_Px are faster.
- If the card/module is a Monitor mode-only card/module, the mode cannot be set to BC_RT_MODE or RT_MODE.

Syntax

Set_Mode_Px (`int` handle, `int` mode)

Input Parameters

handle	The handle designated by Init_Module_Px	
mode		
	BC_RT_MODE	Set up module as a BC/Concurrent-RT [0004 H]
	RT_MODE	Set up module as Remote Terminal [0002 H]
	MON_BLOCK	Set up module as Monitor in Sequential Block submode [0008 H]
	MON_LOOKUP	Set up module as Monitor in Table Lookup submode [0020 H]

Output Parameters

none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
einval	If an invalid parameter was used as an input
bad_mem	If an invalid pointer was specified
etimeoutreset	If timed out waiting for reset
emonmode	If an invalid mode was used as input; the card/module only operates in Monitor mode
0	If successful

Set_ModuleTime_Px

Description

Set_ModuleTime_Px sets the Module Time register with a user-supplied value.

Note:

- Calling this function resets the module and all data is lost.
- The Module Time register is not the same as the Time Tag register.

Syntax

Set_ModuleTime_Px (`int` handle, `int` unsigned `int` moduleTime)

Input Parameters

handle	The handle designated by Init_Module_Px
moduleTime	A 32 bit value representing the module time

Output Parameters

none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
ewrongprocessor	If the processor is not a NIOS II
etimeout	If timed out waiting for BOARD_HALTED
etimeoutreset	If timed out waiting for reset
esetmoduletime	If Module Time could not be set
0	If successful

Set_RT_Active_Px

Description

Set_RT_Active_Px causes a particular RT to be simulated.

RT Mode

In RT mode, when an RT is simulated the 1553 Data Words are *either* transmitted from an assigned data block *or* received by the RT and are then stored in the assigned data block. See Assign_RT_Data_Px on page 3-7 to assign a data block.

Input parameter, intrpt, lets the user request an interrupt when a message is processed for every message associated with this RT. Interrupts are only generated if Set_Interrupt_Px is also set. See Set_Interrupt_Px on page 2-34.

Note: To set an interrupt at the RTid level using Set_RTid_Interrupt_Px, the intrpt parameter must be set to 0. See Set_RTid_Interrupt_Px on page 3-31.

When using a single function module (*PxS*), only one RT can be active. When calling this function, the new RT number is activated and the previously set RT is set to "inactive."

When a single function module's RT number is set externally, you cannot use this function. For more information, see **Mechanical and Electrical Specifications** in the *M8K1553Px Module User's Manual* or the *M4K1553Px Module User's Manual.*

BC/Concurrent-RT Mode

In BC/Concurrent-RT mode, this function may be called to turn on Concurrent-RT simulation; ignore this function if only a Bus Controller is simulated.

Note: This function is not available for the single function module (*PxS*) in BC mode. When using a single function (*PxS*) module in BC mode, you cannot use the module as a Concurrent-RT.

Syntax

Set_RT_Active_Px (`int` handle, `int` rtnum, `int` intrpt)

Input Parameters

handle	The handle designated by Init_Module_Px
rtnum	Address of the RT Valid values: 0 – 31
intrpt	1 Generate interrupt when a message is processed by this RT 0 Do *not* generate interrupt at the RT level **Note:** This parameter is only used in RT mode.

Output Parameters

none

Set_RT_Active_Px (cont.)

Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	emode	If module is not in RT or BC/ Concurrent-RT mode
	einval	If parameter is out of range
	enotforsinglefuncerror	If this function is called with a single function module (*PxS*) in BC mode
	ebitlocked	If the single function RT Number register is locked (only for single function modules (*PxS*) in RT mode)
	0	If successful

Set_RT_Resp_Time_Px

Description

Set_RT_Resp_Time_Px sets response time for RTs. This is the time between receipt of the command (and data for BC-to-RT messages) and the transmission of the 1553 Status word by the RT.

This function can be called in RT or BC/Concurrent-RT modes. In BC/Concurrent-RT mode, this function can be used to adjust the timing of simulated Concurrent-RTs.

Note: This function is not available for the single function module (*PxS*) in BC mode. When using a single function (*PxS*) module in BC mode, you cannot use the module as a Concurrent-RT.

Syntax

Set_RT_Resp_Time_Px (`int` handle, `int` nsecs)

Input Parameters	handle	The handle designated by Init_Module_Px
	nsecs	Time in nanoseconds Valid range: 4000 – 42000
Output Parameters	none	
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	emode	If module not in RT or BC/ Concurrent-RT modes
	enotforsinglefuncerror	If this function is called with a single function module (*PxS*) in BC mode
	0	If successful

Set_Timetag_Res_Px

Description

Set_Timetag_Res_Px sets the resolution of the Time Tag used in the RT and Monitor Mode. The minimum Time Tag resolution is 4 µsec., which is the default. The resolution is rounded up the nearest multiple of 4.

Note: Set Set_Timetag_Res_Px before starting RT or Monitor mode. The new resolution takes effect the next time the RT or Monitor is started.

Syntax

Set_Timetag_Res_Px (`int` handle, `usint` resolution)

Input Parameters

handle	The handle designated by Init_Module_Px
resolution	Valid values: 4 – 1024, multiples of 4 only

Output Parameters

none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
einval	If an invalid parameter was used as an input
0	If successful

Set_UseDmaIfAvailable_Px

Description

Set_UseDmaIfAvailable_Px sets whether DMA should be used for data transfer between the host memory and the *Px* module memory, if DMA is available. (DMA is enabled by default.) See also Get_DmaAvailable_Px on page 2-8.

Note: This function is only for *Px* modules on a PCI Express board or ExpressCards®.

Syntax

Set_UseDmaIfAvailable_Px (`int` handle, `int` useDmaFlag)

Input Parameters

handle	The handle designated by Init_Module_Px	
useDmaFlag	Indicates whether DMA should be used for data transfer (if available):	
	ENABLE	Use DMA if available [0001 H]
	DISABLE	Do not use DMA [0000 H]

Output Parameters

None

Return Values

ebadhandle	If invalid handle specified; should be value returned by Init_Module_Px.
0	If successful

Set_Var_Amp_Px

Description

Set_Var_Amp_Px enables the user to set the amplitude of the 1553 output signal.

Note:

- This function is not available for a single function module (*PxS*). On a single function module, the amplitude is set to 7.5 volts.
- This function is not available for the *EXC-1553[cc]PMC/Px*, *EXC-1553ccVME/Px*, *EXC-1553ExCARD/Px*, *EXC-1553PCMCIA/Px* or *EXC-1553PCMCIA/Px*.

Syntax

Set_Var_Amp_Px (`int` handle, `int` millivolts)

Input Parameters

handle	The handle designated by Init_Module_Px
millivolts	Amplitude in millivolts. Valid range: 0 – 7.5 volts Value is rounded up to the nearest 30 millivolts.

Output Parameters

none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in BC or RT mode
einvamp	If tried to set an invalid amplitude
enotforsinglefuncerror	If this function is called with a single function module (*PxS*)
0	If successful

Stop_Px

Description

Stop_Px stops the current operation of the module in all modes.

Syntax

Stop_Px (`int` handle)

Input Parameters

handle	The handle designated by Init_Module_Px

Output Parameters

none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
0	If successful

Using Interrupts in Windows

When writing a Windows program that processes interrupts, a separate thread is generally created to handle the interrupt processing. This thread calls Wait_For_Interrupt_Px, page 2-45, in order to wait for the next interrupt. When the function returns, the interrupt is processed as needed. This method is demonstrated in the test programs **`demo_int.c`** and **`demo_intms.c`** included with the *1553Px Software Tools.*

Note: There is no need to reset the physical interrupt line in the interrupt thread; this is handled internally.

In cases of very high interrupt frequency, several interrupts may occur before the interrupt thread resumes execution. The Get_Interrupt_Count_Px function may be used to determine if multiple interrupts have occurred. Conversely, it is possible that the Wait_For_Interrupt_Px function will indicate an interrupt that has already been processed by the thread. (This will occur in the case where a subsequent interrupt occurs in between the return of the Wait_For_Interrupt_Px function and the call to Get_Interrupt_Count_Px.) Once again, the Get_Interrupt_Count_Px function may be used to determine if the interrupt has already been processed.

The following functions are described below:

Get_Interrupt_Count_Px
InitializeInterrupt_Px
Wait_For_Interrupt_Px
Wait_For_Multiple_Interrupts_Px

Get_Interrupt_Count_Px

Description

Get_Interrupt_Count_Px returns the total interrupt count for the specified module from the time the module was initialized with Init_Module_Px.

Syntax

Get_Interrupt_Count_Px (`int` handle, `unsigned long` *pdwInterruptCount)

Input Parameters

handle	The handle designated by Init_Module_Px

Output Parameters

pdwInterruptCount	Pointer to an unsigned long which receives the interrrupt count

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
egetintcount	If there was a kernel error
ekernelinitmodule	If error initializing kernel related data
ekernelbadparam	If input parameter is invalid
ekernelbadpointer	If output parameter buffer is invalid
ekerneldevicenotopen	If specified device has not been opened
0	If successful

InitializeInterrupt_Px

Description

InitializeInterrupt_Px instructs the Excalibur kernel driver to keep track of interrupts which occur on the Excalibur module. Once this function has been called, the kernel driver will make note of any interrupts which occur on the module, even if the application is not currently waiting for interrupts (via the Wait_For_Interrupt_Px function). When the application later calls Wait_For_Interrupt_Px, the function will return immediately, indicating an interrupt has occurred in the meantime.

Note: 1. This function need only be called once during a given session with a module. Subsequently, interrupts will continue to be tracked for the module until Release_-Module_Px is called to close the session.

2. The use of this function is not absolutely necessary for interrupt processing. If Wait_For_Interrupt_Px is called before InitializeInterrupt_Px has been called, then the initialization will be performed automatically, at that point. The use of InitializeInterrupt_Px is necessary only if the user requires interrupt tracking prior to the first call to Wait_For_Interrupt_Px.

Syntax

InitializeInterrupt_Px (`int` handle)

Input Parameters

handle	The handle designated by Init_Module_Px

Output Parameters

none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
egeteventhand1	If there is an error in kernel Get_Event_Handle, first part
egeteventhand2	If there is an error in kernel Get_Event_Handle, second part
ekernelinitmodule	If error initializing kernel related data
ekernelbadparam	If input parameter is invalid
ekerneldevicenotopen	If specified device was not opened
ekernelbadpointer	If output parameter buffer is invalid
0	If successful

Wait_For_Interrupt_Px

Description

Wait_For_Interrupt_Px waits for an interrupt on the module. It suspends control of the calling thread while waiting, and returns control to the thread upon receipt of the interrupt, or upon expiration of the time out. If timeout is set to INFINITE, then the call will return only upon receipt of the interrupt.

Syntax

Wait_For_Interrupt_Px (`int` handle, `unsigned int` timeout)

Example

Since this function suspends execution of the calling thread, it is generally called from a separate thread, to allow the main thread to continue its processing. An example of a thread routine which waits for interrupts and processes them as they come in is as follows:

```
DWORD InterruptThread(int referenceParam)
{
  while (1)
  {
    int status;
    status = Wait_For_Interrupt_Px(module_handle, INFINITE);
    if (status < 0)
    {
      // We don't check for ekerneltimeout since we passed
      // in a timeout value of INFINITE.
      // All other return values indicate error.
      // Process error…
      ExitThread(1);
    }
    // Process interrupt…
    // Get the status register to determine the
    // cause of this interrupt
    bc_stat = Get_BC_Status;
    Reset_BC_Status();
    if (bc_stat & END_OF_FRAME)
    {
      // interrupt due to end of frame
      // process…
    }
    if (bc_stat & MSG_CMPLT)
    {
      // interrupt due to message completed
      // process…
    }
    if (bc_stat & MSG_ERR)
    {
      // interrupt due to message error
      // process…
    }
    // Check total number of interrupts
    Get_Interrupt_Count_Px(module_handle, &numints);
  }
}
```

Wait_For_Interrupt_Px (cont.)

Input Parameters	handle	The handle designated by Init_Module_Px
	timeout	Timeout is specified in milliseconds *or* INFINITE [FFFFFFFF H]
Output Parameters	none	
Return Values	ebadhandle	If an invalid handle is specified; should be value returned by Init_Module_Px
	egeteventhand1	If there is an error in kernel mGetEventHandle, first part
	egeteventhand2	If there is an error in kernel mGetEventHandle, second part
	ekernelinitmodule	If error initializing kernel related data
	ekernelbadparam	If input parameter is invalid
	ekerneldevicenotopen	If specified device was not opened
	Successful if *either*:	
	ekerneltimeout	The wait timed out without receiving an interrupt
	or	
	0	

Wait_For_Multiple_Interrupts_Px

Description	Wait_For_Multiple_Interrupts_Px waits for an interrupt on any of the specified modules. It suspends control of the calling thread while waiting, and returns control to the thread either upon receipt of the interrupt, or upon expiration of the time out. If timeout is set to INFINITE, then the call will return only upon receipt of the interrupt.	
Syntax	Wait_For_Multiple_Interrupts_Px(`int` *handle_list, `int` num_modules, `unsigned int` timeout, `unsigned long` *pdwInterruptBitfield)	
Input Parameters	handle_list	An array of module handles
	num_modules	Number of modules in the handle_list
	timeout	Timeout is specified in milliseconds *or* INFINITE [FFFFFFFF H]

Wait_For_Multiple_Interrupts_Px (cont.)

Output Parameters	pdwInterruptBitfield	Pointer to an unsigned long which receives a bitfield indicating which of the modules have interrupted (note that more than one module may have interrupted simultaneously). The modules are distributed in the bitfield such that the lowest bit corresponds to the first module in the handle_list, and so on.
Return Values	egeteventhand1	If there is an error in kernel mGetEventHandleForModule, first part
	egeteventhand2	If there is an error in kernel mGetEventHandleForModule, second part
	ebadhandle	If invalid handle specified; should be value returned by Init_Module_RTx.
	ekernelinitmodule	If error initializing kernel related data
	ekernelbadparam	If input parameter is invalid
	ekernelbadpointer	If output parameter buffer is invalid
	ekerneldevicenotopen	If the specified device was not opened
	Successful if *either*:	
	ekerneltimeout	The wait timed out without receiving an interrupt
	or	
	0	

Using Interrupts Under VISA for VME Boards

When using interrupts under VISA an "interrupt service function" must be created into which the user can place any code that is to run in response to receiving an interrupt. The interrupt line should be reset when an interrupt is received so that the next interrupt can be recorded.

The *1553Px Software Tools* include test programs, such as **`demo_int.c`** and **`demo_intms.c`**, that show how to use interrupts using VISA library functions.

3 Remote Terminal Functions

Chapter 3 describes *Px* module operation in Remote Terminal (RT) mode. The RT module can be used to simulate one or more Remote Terminals. The user can:

- Control which RTs will be simulated.
- Determine which data should be sent for each RT/subaddress combination.
- Store data to each RT/subaddress combination.

All references to a module apply both to the removable *Px* module and to the *Px* module integrated on a board or card. For more information, see Introduction on page 1-1.

The following functions are described in this chapter:

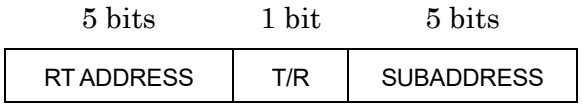
Assign_DB_Datablk_Px	SA_Id_Px
Assign_RT_Data_Px	Set_1553Status_Px
Clear_RT_Sync_Entry_Px	Set_Bit_Px
Get_Blknum_Px	Set_Both_RT_Stacks_Px
Get_Checksum_Blocks_Px[1]	Set_Broad_Interrupt_Px
Get_Multibuf_Nextbuf_Px	Set_Checksum_Blocks_Px[1]
Get_Next_Message_RTM_Px	Set_Invalid_Data_Res_Px
Get_Next_RT_Message_Px	Set_Mode_Addr_Px
Get_RT_Message_Px	Set_RT_Active_Bus_Px
Get_RT_Sync_Info_Px	Set_RT_Broadcast_Px
Load_Datablk_Px	Set_RT_Errors_Px[2]
Load_Multiple_Datablk_Px	Set_RT_Nonactive_Px[2]
Load_RTid_Px	Set_RTid_Interrupt_Px
Read_Datablk_Px	Set_RTid_Multibuf_Px
Read_RT_Status_Px	Set_RTid_Status_Px
Read_RTid_Px	Set_SAid_Illegal_Px[3]
Reset_RTid_Multibuf_Px	Set_Vector_Px
RT_Id_Px	Set_Wd_Cnt_Err_Px[2]
Run_RT_Px	

1. Only for modules that support the 1760 option
2. Not for single function module (*PxS*)
3. Only for single function module (*PxS*)

The input values included in each function are given, in Hex format, by each flag within the function description.

RT Identifier (RTid)

Many of the functions in this section take an RT identifier (RTid) as an argument. The RTid is defined as an RT address, a T/R bit (T=1, R=0) and RT subaddress combination. The structure of the RTid is illustrated below:

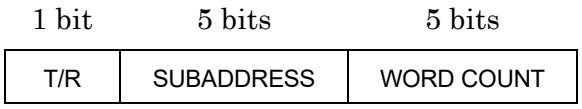


Example: RT 5, Transmit, Subaddress 6 would be represented as 00101 1 00110 or 0166 H. This value can be isolated from a Command word by shifting the Command word 5 bits to the right.

The RT_Id_Px function (on page 3-22) is provided to carry out this calculation.

Subaddress Identifier (SAid)

The Set_SAid_Illegal_Px function take a subaddress identifier (SAid) as an argument. The SAid is defined as a T/R bit (T=1, R=0), subaddress and word count combination. The structure of the SAid is illustrated below:



Example: Transmit, Subaddress 6, Word Count 5 would be represented as 1 00110 00101 or 04C5 H. This value can be isolated from a Command word by masking out the upper 5 bits (for example, Command word & 07FF H).

The SA_Id_Px function (on page 3-22) is provided to carry out this calculation.

New and Old Message Stacks

The original *Px* module had a single message stack with 42 message entries and a 16-bit Time Tag. In later versions of the module a second message stack was added with 512 message entries and a 32-bit Time Tag. Currently only the new message stack is enabled by default.

For new applications it is recommended only to use the new message stack. When there is a need to use the old message stack, you can enable the old message stack (in addition to the new one) by calling Set_Both_RT_Stacks_Px.

Note: For backwards compatibility, if you call Get_RT_Message_Px, which accesses the old stack, the old stack is enabled automatically. However, it is recommended to enable the old message using Set_Both_RT_Stacks_Px before calling Run_RT_Px.

See the following table to determine the message stack behavior on your *Px* module. You can determine which module you have by checking the module revision number printed on the module itself, or by calling Get_Rev_Level_Px to check the firmware revision. For module revision C, you must check the firmware revision to determine whether the new message stack exists.

Module Revision	Firmware Revision	Message Stack Details
A, A1	4.x – 6.9	Only one stack exists with 42 message entries
C	From 1.0 – 1.6	Only one stack exists with 42 message entries
	From 1.7 – 3.3	Both new and old message stacks are enabled
D or later	8.0 or later	By default only the new stack is enabled (512 message entries). You can call Set_Both_RT_Stacks_Px to enable the old stack in addition to the new one.

Frequently Used Functions by Category

For quick reference, the frequently called functions are grouped by use:

Init_Module_Px	To initialize the module
Set_Mode_Px	To place the module into RT mode
Set_RT_Active_Px	To select which RT(s) to simulate
Run_RT_Pxx	To start simulation
Stop_Px	To Stop the RT
Release_Module_Px	To release resources assigned to the module

To control the data being transmitted by the simulated RT(s) or read the data received by them:

Assign_RT_Data_Px	To associate a buffer with an RTid
Load_Datablk_Px	To enter data to be transmitted
Read_Datablk_Px	To read data received by an RTid
Set_1553Status_Px	To set up a value to be transmitted as the 1553 Status Word for the specified RT and to be used as a response to the Transmit Status mode commands

To use messages in the Message Stack:

A Command stack retains a history of the last 512 commands processed by the module. This history is used to compile statistics about bus traffic and alerts the user to incoming messages so that any necessary processing can be performed in real-time. Use of interrupts notifies the user when a particular message comes in and can save processing time. The functions to use are:

Get_Next_RT_Message_Px	To read an entry from the message stack
Set_Interrupt_Px	To select which RTs generate interrupts

To invoke Error injection capabilities:

The *Px* module provides a wide array of error injection capabilities for testing at a systems level. To invoke error injection capabilities call:

Set_Bit_Cnt_Px	To alter the number of bits in transmitted data
Set_RT_Errors_Px	To select which errors to inject
Set_Wd_Cnt_Err_Px	To select how many words to send in a message

To simulate other systems:

A number of system parameters may be altered to change the character of the system.

Set_Invalid_Data_Res_Px	To select how to respond to invalid data
Set_Mode_Addr_Px	To select which subaddress represents a mode command
Set_RT_Resp_Time_Px	To select how long to wait before responding to a command

To send checksum values in modules with the 1760 option:

With the Excalibur *Px* modules with the 1760 option, the user can set which data blocks (associated with some RTids) are associated with a valid checksum value.

Set_Checksum_Blocks_Px	To set which block numbers are enabled to check for a valid checksum value

To increase the number of RT data blocks:

Set_Expanded_Block_Mode_Px	To increase the number of RT data blocks from 256 to 512. When this feature is used, the Concurrent Monitor stack is reduced from 409 to 204 messages.

Assign_DB_Datablk_Px

Description

Assign_DB_Datablk_Px enables double buffering for the specified Receive type RTid and assigns the data blocks for the messages received by that RTid. The data for each message received by the RTid will be stored alternately in the given block and in the next block sequence. The block number specified must be an even-numbered block.

For example, selecting block number 4 will cause that the data for messages to this RTid will be stored alternately in block 4, block 5, block 4, block 5, etc.

This data can be retrieved by the user via the Read_RTid_Px function.

Note: When this function is used to disable double buffering, the data block is not assigned to the RTid.

Syntax

Assign_DB_Datablk_Px (`int` handle, `int` rtid, `int` enable, `int` blknum)

Input Parameters

handle	The handle designated by Init_Module_Px	
rtid	11 bit identifier including RT#, T/R bit and subaddress. See RT Identifier (RTid) on page 3-2 and RT_Id_Px on page 3-22.	
enable	ENABLE	Enable double buffering for the RTid [0001 H]
	DISABLE	Disable double buffering for the RTid [0000 H]
blknum	Even numbered data block: If Expanded Block mode is in use: 0 – 510 If Expanded Block mode is not in use: 0 – 254	

Note:

- Data block 0, while technically a legal block, should be avoided. It is the default block for all RTs and all active RTs with no block assigned will end up here.
- Ignored when double buffering is set to DISABLE.

Output Parameters

none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in RT mode
einval	If parameter is out of range
ercvfunc	If T/R bit is set to Transmit

Assign_DB_Datablk_Px (cont.)

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in RT mode
einval	If parameter is out of range
ercvfunc	If T/R bit is set to Transmit
ebadblknum	If block number selected is not an even number
ertvalue	If RT value does not match the single function module's RT value (for *PxS* only)
0	If successful

Assign_RT_Data_Px

Description Assign_RT_Data_Px associates an RTid with a data block.

Syntax Assign_RT_Data_Px (`int` handle, `int` rtid, `int` blknum)

Input Parameters

handle	The handle designated by Init_Module_Px
rtid	11 bit identifier including RT#, T/R bit and subaddress. See RT Identifier (RTid) on page 3-2 and RT_Id_Px on page 3-22.
blknum	If Expanded Block mode is in use: 0 – 511 If Expanded Block mode is not in use: 0 – 255

Note: Data block 0 represents a default for all unassigned RTid's and is not recommended for use by anyone interested in using the data.
If the RT does not assign a data block to an RTid, the default data block is used for both receive and transmit messages.

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in RT mode
einval	If parameter is out of range
ertvalue	If RT value does not match the single function module's RT value (for *PxS* only)
0	If successful

Clear_RT_Sync_Entry_Px

Description Clear_RT_Sync_Entry_Px clears (resets to 0) the internally stored Time Tag and Data Word, if any, relating to the occurrence of a Sync Mode Code message (MC-1 or MC-17) for this RT.

Syntax Clear_RT_Sync_Entry_Px (`int` handle, `int` rtnum)

Input Parameters

handle	The handle designated by Init_Module_Px
rtnum	The number of the RT for which to clear the Sync information. Valid values: 0 – 31

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in RT mode.
einval	If parameter is out of range
ertvalue	If RT value does not match the single function module's RT value (for *PxS* only)
0	If successful

Get_Blknum_Px

Description Get_Blknum_Px returns the data block number which was assigned to specified RTid.

Syntax Get_Blknum_Px (`int` handle, `int` rtid)

Input Parameters

handle	The handle designated by Init_Module_Px
rtid	11 bit identifier including RT#, T/R bit and subaddress. See RT Identifier (RTid) on page 3-2 and RT_Id_Px on page 3-22.

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in RT mode
einval	If parameter is out of range
ertvalue	If RT value does not match the single function module's RT value (for *PxS* only)
block number	If successful

Get_Checksum_Blocks_Px

Description

Get_Checksum_Blocks_Px returns the number of data blocks for which the associated RT subaddress will receive a checksum.

This function returns the value set by Set_Checksum_Blocks_Px on page 3-26.

Note: This function is applicable *only* to modules that support the Excalibur 1760 option.

Syntax

Get_Checksum_Blocks_Px (`int` handle)

Input Parameters	handle	The handle designated by Init_Module_Px
Output Parameters	none	
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	0	No value set
	block number	If successful, the number of data blocks to receive a checksum. For example: A value of 5 means that data blocks 0 – 4 will receive a checksum; the rest of the data blocks will not receive a checksum.

Get_Multibuf_Nextbuf_Px

Description

Get_Multibuf_Nextbuf_Px returns the next buffer to be accessed by the module. When the module is in middle of accessing a buffer, it will return the number of the following buffer.

Syntax

Get_Multibuf_Nextbuf_Px **(**`int` handle**,** `int` rtid**,** `int` *nextbuf**)**

Input Parameters	handle	The handle designated by Init_Module_Px
	rtid	11 bit identifier including RT#, receive bit and subaddress. See RT Identifier (RTid) on page 3-2 and RT_Id_Px on page 3-22.
Output Parameters	nextbuf	The next buffer to be accessed by the board
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	emode	If module is not in RT mode
	einval	If parameter is out of range
	ertvalue	If RT value does not match the single function module's RT value (for *PxS* only)
	0	If successful

Get_Next_Message_RTM_Px

Description

Get_Next_Message_RTM_Px relates to the Internal Concurrent Monitor available in the memory of the module. The Internal Concurrent Monitor operates automatically when the module is started in Remote Terminal mode. All bus traffic in RT mode is recorded in the Internal Concurrent Monitor memory. The Concurrent Monitor stores 1553 Message blocks in memory sequentially, starting from block 0.

The Concurrent Monitor memory has room for 409 messages. When Expanded Block mode is used, the memory area is reduced to 204 blocks.

The first call to the function Get_Next_Message_RTM_Px returns the contents of block 0. If the next block does not contain a new message, an error value will be returned.

Note:

- This function can only be used in RT mode. See also Get_Next_Mon_Message_Px, which can be used in all modes.
- The structure returned by this function is similar to that returned by the Sequential Monitor mode function Get_Next_Message_Px. However, the bits used in the Message Status word of the Internal Concurrent Monitor are unique to this function – they are similar but not identical to the bits set in the Message Status word of Sequential Monitor mode and BC/Concurrent-RT mode and its Internal Concurrent Monitor.

  For each mode, verify the Message Status bits for the appropriate mode and function.

- If there is a user-initiated Time Tag reset (Reset_Time_Tag_Px, Clear_Timetag_Sync_Px, or SetIrig_4000 with the flag IRIG_TIME_RESET), Get_Next_Message_RTM_Px will interpret this as an overrun condition. To avoid this, call Ignore_Timetag_Overrun_Px. See Ignore_Timetag_Overrun_Px on page 4-18.

Syntax

Get_Next_Message_RTM_Px (`int` handle, `struct MONMSG` *msgptr, `usint *msgCounter`)

Input Parameters

handle — The handle designated by Init_Module_Px

Output Parameters

msgptr — Pointer to an address in host memory where you want a copy of the message to be placed. The message is returned in a `MONMSG` structure, as defined below.

Space should always be allocated for 36 words of data to accommodate the maximum case of RT-to-RT transmission of 32 Data Words plus 2 Status words and 2 Command words.

Get_Next_Message_RTM_Px (cont.)

struct MONMSG {

unsigned short int msgstatus;	Status Word containing one or more of the following flags:
END_OF_MSG	Indicates end of message [8000 H]
BUS_A	Indicates bus A [4000 H]
BAD_CHECKSUM_CONCM	Checksum error [2000 H] (for 1760 option only)
ME_SET	Message Error bit in the RT Status word [1000 H]
BAD_STATUS	RT Status Word bits set (not ME bit) [0800 H]
TX_TIME_OUT	In RT-to-RT message receiving RT did not detect a transmitter Status Word [0400 H]
LATE_RESP	Response time error occurred in the message, even if no RT active on the module [0200 H]
INVALID_MSG_RT2RT	Invalid message; perhaps RT-to-RT with two receive Command words [0100 H]
INVALID_WORD	At least one invalid 1553 Word received [0080 H]
BAD_HEADER_RTCM	Bad Header Word [0040 H] (for 1760 option only)
WORD_CNT_ERR	Incorrect number of words received in a message [0020 H]
BAD_RT_ADDR	Received 1553 Status word did not contain the correct RT address [0010 H]
BAD_SYNC	Sync of either the Command or the Data Word(s) is incorrect [0008 H]
BAD_GAP	Invalid gap received between 1553 Words [0004 H]
RT2RT_MSG_CONCM	RT-to-RT message was received [0002 H]
MSG_ERROR	Error occurred – defined in other flags [0001 H]
unsigned int elapsedtime;	A 32-bit Time Tag; resolution is 4 microseconds
unsigned short int words [36];	A pointer to an array of 1553 Words in the sequence they were received over the bus. See **Appendix A: MIL-STD-1553 Word Formats**

}

Get_Next_Message_RTM_Px (cont.)

	msgCounter	The low 16 bits of the message number assigned to this message. The high 16 bits are stored in the Message Number Hi register [3EBC H].
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	emode	If module is not in RT mode
	enomsg	If no new messages have been received
	eoverrunTtag	If the Time Tag of the message was overwritten while the message was being read
	eoverrunEOM flag	If the End Of Message flag was overwritten while the message was being read
	block number	If successful. Valid values: 0 – 409 **Note:** For *M4K1553Px* modules rev A and rev A1, valid values are 0 – 128.

Get_Next_RT_Message_Px

Description

Get_Next_RT_Message_Px reads the next available entry from the message stack; the entry immediately following the entry that was returned via a previous call to this routine. If the next block does not contain a new entry an error will be returned.

Note:

- If the user falls 512 entries behind, it will be impossible to know if the message returned is newer or older than the previous message returned.
- This function accesses the new message stack. It is recommended to use this function instead of Get_RT_Message_Px, which accesses the old message stack. For more information, see New and Old Message Stacks on page 3-3.

Syntax

Get_Next_RT_Message_Px (`int` handle, `struct CMDENTRYRT` *cmdstruct)

Input Parameters	handle	The handle designated by Init_Module_Px
Output Parameters	cmdstruct	A pointer to a structure of type CMDENTRY containing information regarding the next available entry from the message stack.

Get_Next_RT_Message_Px (cont.)

struct CMDENTRYRT{

usint command;	1553 Command word
usint command2;	1553 transmit Command word for RT-to-RT
usint timetaghi;	16 upper bits of 32-bit Time Tag of message
usint timetaglo;	16 lower bits of 32-bit Time Tag of message
usint status;	status containing one or more of the following flags:

END_OF_MSG	Indicates end of message [8000 H]
BUS_A	Indicates bus A [4000 H]
RT_BAD_CHECKSUM	Bad 1760 Checksum on the RT receive side (for 1760 only) [2000 H]
TX_TIME_OUT	In RT-to-RT no response [0400 H]
INVALID_WORD	At least one invalid 1553 Word received [0080 H]
BAD_WD_CNT	Wrong number of words received [0020 H]
BROADCAST	Broadcast command received [0010 H]
BAD_SYNC	Sync of either the Command or the Data Word(s) is incorrect [0008 H]
BAD_GAP	Invalid gap received between 1553 Words [0004 H]
RT2RTMSG	RT-to-RT message was received [[0002 H]
MSG_ERROR	Error occurred – defined in other flags [0001 H]

}

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in RT mode
enomsg	If no new messages have been received
entry number	If successful: Valid values: 0 – 511

Get_RT_Message_Px

Description

Get_RT_Message_Px reads the next available entry from the old message stack; the entry immediately following the entry that was returned via a previous call to this routine. If the next block does not contain a new entry an error will be returned.

This function accesses the old message stack. It is recommended to use the new message stack instead by calling Get_Next_RT_Message_Px.

In the current *Px* module, the old message stack is disabled by default. Calling this function automatically enables the old message stack. However, it is recommended to enable the old message using Set_Both_RT_Stacks_Px before calling Run_RT_Px. For more information, see New and Old Message Stacks on page 3-3.

Note: If the user falls 42 entries behind, it will be impossible to know if the message returned is newer or older than the previous message returned.

Syntax

Get_RT_Message_Px (`int` handle, `struct CMDENTRY` *cmdstruct)

Input Parameters

handle — The handle designated by Init_Module_Px

Output Parameters

cmdstruct — A pointer to a structure of type CMDENTRY containing information regarding the next available entry from the message stack.

struct CMDENTRY{

usint command; 1553 Command word

usint command2; 1553 transmit Command word for RT-to-RT

usint timetag; 16-bit Time tag of message

usint status; status containing the following flags:

END_OF_MSG	Indicates end of message [8000 H]
BUS_A	Indicates bus A [4000 H]
RT_BAD_CHECKSUM	Bad Checksum on the RT receive side [2000 H] (for 1760 only)
TX_TIME_OUT	In RT-to-RT no response [0400 H]
INVALID_WORD	At least one invalid 1553 Word received [0080 H]
BAD_WD_CNT	Wrong number of words received [0020 H]
BROADCAST	Broadcast command received [0010 H]
BAD_SYNC	Sync of either the Command or the Data Word(s) is incorrect [0008 H]

Get_RT_Message_Px (cont.)

BAD_GAP	Invalid gap received between 1553 Words [0004 H]
RT2RTMSG	RT-to-RT message was received [0002 H]
MSG_ERROR	Error occurred - defined in other flags [0001 H]

}

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in RT mode
enomsg	If no unread messages are available to return
entry number	If successful. Valid values: 0 – 41

Get_RT_Sync_Info_Px

Description

Get_RT_Sync_Info_Px extracts the internally stored Time Tag and Data Word, if any, of the most recent occurrence of a Sync Mode Code message (MC-1 or MC-17) for this RT. Previous values are overwritten.

In order for this function to return meaningful data, you must first call Get_Next_RT_Message_Px (or Get_RT_Message_Px) to read a message. Each call to one of these functions reads a message and places the current message data into a structure entry for the specified RT, which contains sync information (Time Tag and Data Word).

Once the internal structure contains this sync data, it can be read using this function Get_RT_Sync_Info_Px.

Note: This function replaces the obsolete Get_RT_Sync_Entry_Px.

Syntax

Get_RT_Sync_Info_Px (`int` handle, `int` rtnum, `unsigned int` *sync_timetag, `usint` *sync_data)

Input Parameters

handle	The handle designated by Init_Module_Px
rtnum	The number of the RT for which to get the Sync information. Valid values: 0 – 31

Output Parameters

sync_timetag	The Time Tag reading when the Sync message arrived
sync_data	The Data Word of the Sync message. For Sync message with data, MC-17, only.

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in RT mode.
einval	If parameter is out of range
ertvalue	If RT value does not match the single function module's RT value (for *PxS* only)
0	If successful

Load_Datablk_Px

Description

Load_Datablk_Px assigns data to a specified data block. This data will be used to respond to a Transmit command or a Transmit mode command.

Note:

- The same data block may be assigned to multiple RTids.
- If the data block is assigned to multiple receive RTids (or to a receive and a transmit RTid), then the data in this data block may change in an unpredictable manner (reflecting the most recently received message.)
- Data block 0 represents a default for all unassigned RTid's and is not recommended for use by anyone interested in using the data.

Syntax

Load_Datablk_Px (`int` handle, `int` blknum, `usint` *words)

Input Parameters

handle	The handle designated by Init_Module_Px
blknum	Number of block to assign data to. If Expanded Block mode is in use: 0 – 511 If Expanded Block mode is not in use: 0 – 255
words	Pointer to an array of up to 32 words of data to be placed in the block

Output Parameters

none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in RT mode
einval	If parameter is out of range
0	If successful

Load_Multiple_Datablk_Px

Description

Load_Multiple_Datablk_Px assigns up to three blocks of data (with 32 words each) for an RT block in a single-block write. This data will be used to respond to transmit commands or transmit mode commands.

Note:

- The same data block may be assigned to multiple RT's.
- Data block 0 is the default data block used by any active RT for which Assign_RT_Data_Px has not been called. Since this block is the default for receive commands as well, its data may change in an unpredictable way. It is not advisable to use this block.

Syntax

Load_Multiple_Datablk_Px (`int` handle, `int` blknum, `int` numOfBlocks, `usint` *words)

Input Parameters

handle	The handle designated by Init_Module_Px
blknum	Number of block to assign data to. If Expanded Block mode is in use: 0 – 511 If Expanded Block mode is not in use: 0 – 255
numOfBlocks	Number of blocks to write (1 – 3)
words	Pointer to an array of up to numOfBlocks x 32 words of data to be placed in the blocks

Output Parameters

none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in RT mode
einval	If parameter is out of range
0	If successful

Load_RTid_Px

Description

Load_RTid_Px is used to write data to the appropriate data block assigned to the specified RTid.

This is necessary for RTid's using double buffering or multibuffering, since the user does not know the block number to be written to. (When using double or multibuffering, use this function instead of Load_Datablk_Px.)

Syntax

Load_RTid_Px (`int` handle, `int` rtid, `usint` *words)

Input Parameters

handle	The handle designated by Init_Module_Px
rtid	11 bit identifier including RT#, receive bit and subaddress. See RT Identifier (RTid) on page 3-2 and RT_Id_Px on page 3-22.
words	Pointer to an array of up to 32 words from which to copy the data (to the appropriate data block)

Output Parameters

none

Return Values

emode	If module is not in RT mode
einval	If parameter is out of range
ertvalue	If RT value does not match the single function module's RT value (for *PxS* only)
0	If successful

Read_Datablk_Px

Description

Read_Datablk_Px returns the words contained in the specified data block. The most common use of this function is to retrieve data from a block assigned to a receive RTid. To do this, set up the block and check Read_RT_Status_Px (or wait for an interrupt). When a message is received, call Get_Next_RT_Message_Px to find out which RT received the message. Then call Get_Blknum_Px to get the data block number which was assigned to this RTid. Then call Read_Datablk_Px to retrieve the data.

Syntax

Read_Datablk_Px (`int` handle, `int` blknum, `usint` *words)

Input Parameters

handle	The handle designated by Init_Module_Px
blknum	Number of the data block to be read: If Expanded Block mode is in use: 0 – 511 If Expanded Block mode is not in use: 0 – 255 **Note:** Data block 0, while technically a legal block, should be avoided. It is the default block for all active RTs for which no data block has been assigned.

Output Parameters

words	Pointer to an array of 32 words to be read from the block. Although the message may contain fewer than 32 Data Words, 32 words will be returned.

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in RT mode
einval	If parameter is out of range
0	If successful

Read_RT_Status_Px

Description

Read_RT_Status_Px returns 1 if a 1553 Message has been received since the last time Read_RT_Status_Px was called.

Note: This function clears the associated register when it returns a value.

Syntax

Read_RT_Status_Px (`int` handle)

Input Parameters

handle	The handle designated by Init_Module_Px

Output Parameters

none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in RT mode
status	1 Message was received 0 No message was received

Read_RTid_Px

Description

Read_RTid_Px is used to read data from the appropriate data block assigned to the specified RTid. It returns an array of 32 words that are found in the data block.

This is necessary for RTid's using double buffering or multibuffering, since the user does not know the block number to be read. (When using double or multibuffering, use this function instead of Read_Datablk_Px.)

This function can be used to read the data blocks of any RTid, transmit or receive, when using single, double or multibuffering.

Syntax

Read_RTid_Px (`int` handle, `int` rtid, `usint` *words)

Input Parameters

handle	The handle designated by Init_Module_Px
rtid	11 bit identifier including RT#, receive bit and subaddress. See RT Identifier (RTid) on page 3-2 and RT_Id_Px on page 3-22.

Output Parameters

words	Pointer to an array of 32 words. Although the message may contain fewer than 32 words, 32 words will be transferred.

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in RT mode
einval	If RTid is out of range
ertvalue	If RT value does not match the single function module's RT value (for *PxS* only)
0	If successful

Reset_RTid_Multibuf_Px

Description Reset_RTid_Multibuf_Px forces the next read/write to use the first buffer on both firmware and driver sides.

Syntax Reset_RTid_Multibuf_Px (`int` handle, `int` rtid)

Input Parameters

handle	The handle designated by Init_Module_Px
rtid	11 bit identifier including RT#, receive bit and subaddress. See RT Identifier (RTid) on page 3-2 and RT_Id_Px on page 3-22.

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in RT mode
einval	If parameter is out of range
ertvalue	If RT value does not match the single function module's RT value (for *PxS* only)
0	If successful

RT_Id_Px

Description RT_Id_Px is a utility function to help the user calculate the RTid value. (See RT Identifier (RTid) on page 3-2.)

Syntax RT_Id_Px (`int` rtnum, `int` type, `int` subaddr, `int` *rtid)

Input Parameters

rtnum	Address of the RT Valid values: 0 – 31	
type	TRANSMIT	Allocate memory for data to be transmitted [0001 H]
	RECEIVE	Allocate memory for data to be received [0000 H]
subaddr	Designated Subaddress of the given RT Valid values: 0 – 31	

Output Parameters rtid — The calculated RTtid

Return Values

einval	If an invalid value or parameter was used as an input
0	If successful

Run_RT_Px

Description	Run_RT_Px starts RT simulation for all RTs that have been activated with Set_RT_Active_Px. See Set_RT_Active_Px on page 2-38.	
Syntax	Run_RT_Px (`int` handle)	
Input Parameters	handle	The handle designated by Init_Module_Px
Output Parameters	none	
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	emode	If module is not in RT mode
	ebiterror	If the error bit of the single function RT Number register is lit (for *PxS* only)
	etimeout	If timed out waiting for BOARD_HALTED
	0	If successful

SA_Id_Px

Description	SA_Id_Px is a utility function to help the user calculate the SAid value. (See Subaddress Identifier (SAid) on page 3-2.)		
Syntax	SA_Id_Px (`int` type, `int` subaddr, `int` wordcount, `int` *said)		
Input Parameters	type	TRANSMIT	Allocate memory for data to be transmitted [0001 H]
		RECEIVE	Allocate memory for data to be received [0000 H]
	subaddr	Designated Subaddress of the given RT Valid values: 0 – 31	
	wordcount	The number of words in the message. Valid values: 0 – 31 **Note:** 0 indicates a word count of 32.	
Output Parameters	said	The calculated SAid	
Return Values	einval	If an invalid value or parameter was used as an input	
	0	If successful	

Set_1553Status_Px

Description Set_1553Status_Px provides a value, with a flag specifying duration, to be transmitted as the 1553 Status Word for the specified RT.

Note: This function replaces Set_Status_Px.

Syntax Set_1553Status_Px (`int` handle, `int` rtnum, `int` statusvalue, `int` duration)

Input Parameters

handle	The handle designated by Init_Module_Px	
rtnum	Address of the RT Valid values: 0 – 31	
statusvalue	1553 Status word value to be transmitted	
duration	DUR_ALWAYS	The 1553 Status word specified should *always* be used for the RT [0000 H]
	DUR_ONETIME	The 1553 Status word specified should be used only *once* for this RT [0001 H]

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
einval	If parameter is out of range
emode	If module is not in RT mode
ertvalue	If RT value does not match the single function module's RT value (for *PxS* only)
0	If successful

Set_Bit_Px

Description Set_Bit_Px provides a value to be returned in response to the Transmit BIT (Built in Test) Word mode command for the given RT.

Syntax Set_Bit_Px (`int` handle, `int` rtnum, `int` bitvalue)

Input Parameters

handle	The handle designated by Init_Module_Px
rtnum	Address of the RT Valid values: 0 – 31
bitvalue	BIT response value – the value to be set

Output Parameters none

Set_Bit_Px (cont.)

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in RT mode
einval	If parameter is out of range
ertvalue	If RT value does not match the single function module's RT value (for *PxS* only)
0	If successful

Set_Both_RT_Stacks_Px

Description

Set_Both_RT_Stacks_Px determines whether the old message stack is used in addition to the new message stack. For more information, see New and Old Message Stacks on page 3-3.

Note: If you call Get_RT_Message_Px, which accesses the old message stack, the old message stack is enabled automatically.

Syntax

Set_Both_RT_Stacks_Px (`int` handle, `int` enableFlag)

Input Parameters

handle	The handle designated by Init_Module_Px	
enableFlag	ENABLE	Enable the old RT message stack area and use both stacks [0001 H]
	DISABLE	Disable the old RT message stack area, and only use the new area [0000 H]

Output Parameters

none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in RT mode
einval	If parameter is out of range
ecallmewhenstopped	If the function is called when the module is running
0	If successful

Set_Broad_Interrupt_Px

Description Set_Broad_Interrupt_Px determines whether an interrupt is to be generated when a broadcast message is detected.

Syntax Set_Broad_Interrupt_Px (`int` handle, `int` intrpt)

Input Parameters

handle	The handle designated by Init_Module_Px	
intrpt	ENABLE	Generate an interrupt [0001 H]
	DISABLE	Do *not* generate an interrupt [0000 H]

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in RT mode
einval	If parameter is out of range
0	If successful

Set_Checksum_Blocks_Px

Description Set_Checksum_Blocks_Px is used to specify which RTid will receive a checksum. Associate an RTid with a data block using Assign_RT_Data_Px, see page 3-7. RTids associated with all data blocks numbered up to the value specified in this function will receive a checksum.

The value set in this function is returned by Get_Checksum_Blocks_Px, see page 3-9.

Note: This function is applicable *only* for modules with the 1760 option.

Syntax Set_Checksum_Blocks_Px (`int` handle, `int` csum_blocks)

Input Parameters

handle	The handle designated by Init_Module_Px
csum_blocks	Number of data blocks to receive checksum. For example: A value of 5 means that data blocks 0 – 4 will receive a checksum; the rest will not receive a checksum. Valid values: If Expanded Block mode is in use: 1– 512 If Expanded Block mode is not in use: 1 – 256

Set_Checksum_Blocks_Px (cont.)

Output Parameters	none	
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	emode	If module is not in RT mode
	einval	If parameter is out of range
	0	If successful

Set_Invalid_Data_Res_Px

Description	Set_Invalid_Data_Res_Px determines whether RTs are to respond to messages containing invalid Data Words.		
Syntax	Set_Invalid_Data_Res_Px (`int` handle, `int` flag)		
Input Parameters	handle	The handle designated by Init_Module_Px	
	flag	ENABLE	Respond to message with invalid data [0001 H]
		DISABLE	Do *not* respond to this message with invalid data [0000 H]
Output Parameters	none		
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px	
	emode	If module is not in RT mode	
	einval	If parameter is out of range	
	0	If successful	

Set_Mode_Addr_Px

Description	Set_Mode_Addr_Px defines which subaddresses in a Command word are to be used to indicate that the command is a Mode Command, *instead* of being interpreted as an RT subaddress.	
Syntax	Set_Mode_Addr_Px (`int` handle, `int` flag)	
Input Parameters	handle	The handle designated by Init_Module_Px
	flag	'0' 11111 and 00000 are mode commands '1' only 00000 is a mode command '2' only 11111 is a mode command
Output Parameters	none	
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px

Set_Mode_Addr_Px (cont.)

	emode	If module is not in RT mode
	einval	If parameter is out of range
	0	If successful

Set_RT_Active_Bus_Px

Description Set_RT_Active_Bus_Px causes a particular RT to respond to a specified bus or buses.

Syntax Set_RT_Active_Bus (`int` handle, `int` rtnum, `usint` bus)

Input Parameters

handle	The handle designated by Init_Module_Px	
rtnum	Address of the RT Valid values: 0 – 31	
bus	BUS_A_ONLY	Bus A only [0002 H]
	BUS_B_ONLY	Bus B only [0001 H]
	BUS_AB	Buses A and B [0000 H]
	BUS_NONE	No buses active [0003 H]

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in RT mode
einval	If parameter is out of range
0	If successful

Set_RT_Broadcast_Px

Description Set_RT_Broadcast_Px designates RT address 31 as either the broadcast address or as a regular RT.

Syntax Set_RT_Broadcast_Px (`int` handle, `int` toggle)

Input Parameters

handle	The handle designated by Init_Module_Px	
toggle	ENABLE	Enable RT31 as the broadcast address [0001 H]
	DISABLE	Disable RT31 from being the broadcast address [0000 H]

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in RT mode
0	If successful

Set_RT_Errors_Px

Description Set_RT_Errors_Px sets up which errors are to be inserted by all RTs.

Note: This function is related to error injection, and is not available for the single function module (*PxS*).

Syntax Set_RT_Errors_Px (`int` handle, `int` errormask)

Input Parameters

handle	The handle designated by Init_Module_Px	
errormask	All requested errors may be ORed together. The following errors are defined:	
	BIT_CNT	Insert bad bit count based on Set_Bit_Cnt_Px [0002 H]
	BAD_GAP_TIME	Insert bad gap between Data Words 1 and 2 [0004 H]
	STATUS_PARITY	Send Status word with even parity [0010 H]
	STATUS_SYNC	Send Status word with data sync [0020 H]
	DATA_PARITY	Send Data Word with even parity [0040 H]
	DATA_SYNC	Send Data Word with command sync [0080 H]
	0	Clears error injection: no errors are inserted by any RT on a transmit message [0000 H]

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in RT mode

Set_RT_Errors_Px (cont.)

	einval	If parameter is out of range
	enotforsinglefuncerror	If this function is called with a single function module (*PxS*)
	0	If successful

Set_RT_Nonactive_Px

Description

Set_RT_Nonactive_Px sets a particular RT address nonactive, thereby turning off simulation for the specified RT. The module will not respond to messages sent to nonactive RTs nor will it store messages to or from nonactive RTs.

This function may be called in BC mode to turn off concurrent RT simulation.

Note: This function is not relevant to the single function module (*PxS*).

Syntax

Set_RT_Nonactive_Px (`int` handle, `int` rtnum)

Input Parameters

	handle	The handle designated by Init_Module_Px
	rtnum	Address of the RT Valid values: 0 – 31

Output Parameters none

Return Values

	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	emode	If module is not in RT mode
	enotforsinglefuncerror	If this function is called with a single function module (*PxS*)
	0	If successful

Set_RTid_Interrupt_Px

Description

Set_RTid_Interrupt_Px enables or disables the generation of an interrupt for the given RTid upon the conditions specified by Set_Interrupt_Px.

Note:

- DISABLE only turns off interrupts that were previously ENABLED for a particular RTid by calling Set_RTid_Interrupt_Px with ENABLE value.
- To set an interrupt at the RTid level, the intrpt parameter of Set_RT_Active_Px must be set to 0. See Set_RT_Active_Px on page 2-38.

Syntax

Set_RTid_Interrupt_Px (`int` handle, `int` rtid, `int` enable, `int` int_type)

Input Parameters

handle	The handle designated by Init_Module_Px	
rtid	11 bit identifier including RT#, T/R bit and subaddress. See RT Identifier (RTid) on page 3-2 and RT_Id_Px on page 3-22.	
enable	ENABLE	Generate interrupt for this RTid [0001 H]
	DISABLE	Do not generate interrupt for this RTid [0000 H]
int_type	type of interrupt to generate	
	INT_ON_ERR	Generate interrupt on error [0001 H]
	INT_ON_ENDOFMSG	Generate interrupt on end of message [000 H]

Output Parameters

none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in RT mode
einval	If parameter is out of range
ertvalue	If RT value does not match the single function module's RT value (for *PxS* only)
0	If successful

Set_RTid_Multibuf_Px

Description

Set_RTid_Multibuf_Px configures an RTid to use multiple buffers.

Note:

- Before calling this function, you must call Assign_RT_Data_Px to assign a base data block number to this RTid.
- There are two data block sections (0 – 199 and 200 – 255), plus a third data block section (256 – 511) when using Expanded Block mode. When defining the number of buffers, you can cross between the first two blocks, but you cannot cross between the second and third data blocks. For example, if you request 16 buffers to start at buffer number 246, this will generate an error, since it crosses between second and third data block boundaries.
- To enable Expanded Block mode, use Set_Expanded_Block_Mode_Px.

Syntax

Set_RTid_Multibuf_Px (`int` handle, `int` rtid, `int` numbufs)

Input Parameters

handle	The handle designated by Init_Module_Px
rtid	11 bit identifier including RT#, T/R bit and subaddress. See RT Identifier (RTid) on page 3-2 and RT_Id_Px on page 3-22.
numbufs	The number of buffers to use for this RTid Valid values: 0 – 16 **Note:** Use '0' to turn off multibuffering for this RTid.

Output Parameters

none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in RT mode
einval	If parameter is out of range
edoublebuf	If an attempt is made to set multiple buffers on an RTid that is already using double buffering
eboundary	If multibufferring attempts to cross an illegal data block boundary (see the note in the description of this function)
func_invalid	If firmware does not support this feature
ertvalue	If RT value does not match the single function module's RT value (for *PxS* only)
0	If successful

Set_RTid_Status_Px

Description Set_RTid_Status_Px enables or disables the ILLEGALIZATION or INACTIVE status of a given RTid.

Syntax Set_RTid_Status_Px (`int` handle, `int` rtid, `int` enable, `int` status_type)

Input Parameters

handle	The handle designated by Init_Module_Px	
rtid	11 bit identifier including RT#, T/R bit and subaddress. See RT Identifier (RTid) on page 3-2 and RT_Id_Px on page 3-22.	
enable	ENABLE	Enable the feature for this RTid [0001 H]
	DISABLE	Disable the feature for this RTid [0000 H]
status_type	Status to set for the RTid:	
	RTID_ILLEGAL	RTid will accept data on receive message, will not send data for transmit messages [0002 H]
	RTID_INACTIVE	RTid will not respond to any messages sent to it [0003 H]
	RTID_BUSY	When a message is sent to this RTid, the Busy Bit of the RT Status Word will be set [0007 H]

Note: For Transmit messages with RTID_INACTIVE enabled, in order to ensure that the RT does not send the Status word back to the BC, the RT Response Time *must* be set to a minimum of 5µsec. The default setting is 4 µsec.

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in RT mode
einval	If parameter is out of range
ertvalue	If RT value does not match the single function module's RT value (for *PxS* only)
efeature_not_supported_PX	If the feature is not supported on this module
0	If successful

Set_SAid_Illegal_Px

Description

Set_SAid_Illegal_Px sets the specified subaddress/word count/ direction as illegal. The specified SA/WC/direction will accept data on receive message, but will not send data for transmit messages.

Note: This function is only relevant for the single function module (*PxS*).

Syntax

Set_SAid_Illegal_Px (`int` handle, `int` said, `int` enabled_flag, `int` broadcast_flag)

Input Parameters

handle	The handle designated by Init_Module_Px	
said	11 bit identifier including T/R bit, subaddress and word count. See Subaddress Identifier (SAid) on page 3-2 and SA_Id_Px on page 3-23.	
enabled_flag	ENABLE	Enable illegalization of this SAid [0001 H]
	DISABLE	Disable illegalization of this SAid [0000 H]
broadcast_flag	ENABLE	Enable illegalization of this SAid for a broadcast message [0001 H]
	DISABLE	Disable illegalization of this SAid for a broadcast message [0000 H]

Output Parameters

none

Return Values

func_invalid	If this is not a single function module (*PxS*)
ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
einval	If parameter is out of range
emode	If module is not in RT mode
0	If successful

Set_Vector_Px

Description Set_Vector_Px provides a value to be transmitted in response to the Transmit Vector Word mode command for the specified RT.

Syntax Set_Vector_Px (`int` handle, `int` rtnum, `int` vecvalue)

Input Parameters

handle	The handle designated by Init_Module_Px
rtnum	Address of the RT Valid values: 0 – 31
vecvalue	Service Request vector response value to be set

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in RT mode
einval	If parameter is out of range
ertvalue	If RT value does not match the single function module's RT value (for *PxS* only)
0	If successful

Set_Wd_Cnt_Err_Px

Description Set_Wd_Cnt_Err_Px requests the selected RT to insert a Word Count Error in its transmissions. Each individual RT can be set to send up to three words more than the word count or up to three words less than the word count.

Note: This function is related to error injection, and is not available for the single function module (*PxS*).

Syntax Set_Wd_Cnt_Err_Px(`int` handle, `int` rtnum, `int` offset)

Input Parameters

handle	The handle designated by Init_Module_Px
rtnum	Address of the RT Valid values: 0 – 31
offset	Offset to add to correct word count. Valid values: -3 to +3

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in RT mode
einval	If parameter is out of range

Set_Wd_Cnt_Err_Px (cont.)

enotforsinglefuncerror	If this function is called with a single function module (*PxS*)
0	If successful

4 Bus Monitor Functions

Chapter 4 describes *Px* module operation in Bus Monitor mode. The Bus Monitor mode is used to simulate the Bus Monitor in an application.

All references to a module apply both to the removable *Px* module and to the *Px* module integrated on a board or card. For more information, see Introduction on page 1-1.

The *1553Px Software Tools* support two different submodes for Monitoring:

- Look Up Submode is useful for applications in which the last data for a given RTid is important. In this mode all messages to the same RTid are stored in the same buffer.
- Sequential Submode is used when the sequence of the incoming messages is important. In this mode messages are stored sequentially in a circular buffer. This submode supports extensive filtering and triggering capabilities which causes the module to record only wanted messages and saves valuable processing time.
  In Sequential mode, 1553 messages are stored at fixed sequential blocks in the memory. Sequential mode supports Trigger capability.
  In Sequential mode, you have the following options:
  - **Regular Monitor:** 200 blocks are used to store Bus Monitor data.
  - **Expanded Monitor:** 800 blocks are used to store Bus Monitor data.
  - **Enhanced Monitor:** 400 blocks are used to store Bus Monitor data, and an additional 400 blocks are used to store additional information about each word. For example, block 1 in the regular message buffer corresponds to block 1 of the additional information buffer.

    When using Enhanced Monitor, all words are saved, including those with errors. For example, Data Words without Command Words are saved. When using Regular Monitor or Expanded Monitor, these are disregarded. When an error message has more than 36 words, the message is stored in two or more data blocks, the first 36 words in the first data block, and the remainder in the next data block(s).

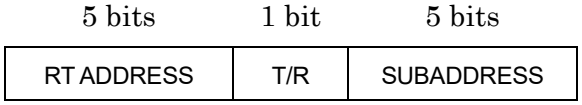
In each mode, the user can select which messages will cause the module to generate an interrupt.

The functions described in this chapter are listed, under their submode. Functions that can be used with both submodes are listed under **All Submodes**.

All Submodes	Look Up Submode	Sequential Submode
Clear_Msg_Blks_Px	Assign_Blk_Px	Enable_Mon_1553A_Support_Px
Get_MON_Status_Px	Enable_Lkup_Int_Px	Enable_Mon_1553A_Support_Px
Run_MON_Px	Get_Last_Blknum_Px	Get_Message_Info_Px
Set_Broad_Ctl_Px	Get_Message_Px	Get_Next_Message_Px
Set_Mode_Addr_Px		Get_Next_Mon_IRIGttag_Message_Px
Set_MON_Concurrent_Px		Ignore_Timetag_Overrun_Px
Set_Mon_Response_Time_1553A_Px		Set_Cnt_Trig_Px
Set_Mon_Response_Time_Px		Set_Enhanced_Mon_Px
		Set_Message_Interval_Interrupt_Value_Px
		Set_Trigger_Mode_Px
		Set_Trigger1_Px
		Set_Trigger2_Px

RT Identifier

Many of the functions in this section take an RT identifier (RTid) as an argument. The RTid is defined as an RT address, T/R bit value and RT subaddress combination. The structure of the RTid is illustrated below:



Example: RT 5, Transmit, Subaddress 6 would be represented as 00101 1 00110 or 0166 H. This value can be isolated from a Command word by shifting the Command word 5 bits to the right.

The RT_Id_Px function (on page 3-22) is provided to carry out this calculation.

All Submodes

Clear_Msg_Blks_Px

Description	Clear_Msg_Blks_Px clears out all 1553 messages previously received and stored in memory. The Monitor Status register is also cleared. If the module is running, it is stopped, cleared and restarted.	
Syntax	Clear_Msg_Blks_Px (`int` handle)	
Input Parameters	handle	The handle designated by Init_Module_Px
Output Parameters	none	
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	emode	If module is not in Monitor mode
	0	If successful

Get_MON_Status_Px

Description

Get_MON_Status_Px returns the status of the Monitor. The function is useful to determine what condition triggered an interrupt.

Note: The status is automatically reset each time it is read.

Syntax

Get_MON_Status_Px (`int` handle)

Input Parameters

handle — The handle designated by Init_Module_Px

Output Parameters

none

Return Values

Status word containing one or more of the following flags ORed together:

MSG_IN_PROGRESS	A message is in the process of being received [0002 H]
MSG_INTERVAL	The specified number of messages was received [0008 H]

Note: See the function Set_Message_Interval_Interrupt_Value_Px on page 4-21.

TRIG_RCVD	A message which matched trigger1 or trigger2 has been received [0001 H]
CNT_TRIG_MATCH	The number of messages recorded by the module has reached the number set by Set_Cnt_Trig_Px [0004 H]
0	If no messages were received since the last time this function was called.

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in Monitor mode

Run_MON_Px

Description	Run_MON_Px causes the Monitor module to start.	
Syntax	Run_MON_Px (`int` handle)	
Input Parameters	handle	The handle designated by Init_Module_Px
Output Parameters	none	
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	emode	If module is not in Monitor mode
	etimeout	If timed out waiting for BOARD_HALTED
	0	If successful

Set_Broad_Ctl_Px

Description	Set_Broad_Ctl_Px designates the RT address 31 as either the broadcast address or as a regular RT.		
Syntax	Set_Broad_Ctl_Px (`int` handle, `int` flag)		
Input Parameters	handle	The handle designated by Init_Module_Px	
	flag	ENABLE	Sets RT31 as the broadcast address [0001 H]
		DISABLE	Sets RT31 as a regular RT [0000 H]
Output Parameters	none		
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px	
	emode	If module is not in Monitor mode	
	einval	If an invalid parameter was used as an input	
	0	If successful	

Set_Mode_Addr_Px

Description	Set_Mode_Addr_Px defines which 'subaddresses' in a Command word are to be used to indicate that the command in a Mode Code, *instead* of being interpreted as an RT subaddress.	
Syntax	Set_Mode_Addr_Px (`int` handle, `int` flag)	
Input Parameters	handle	The handle designated by Init_Module_Px
	flag	'0' 11111 and 0000 are mode commands '1' only 00000 is a mode command '2' only 11111 is a mode command
Output Parameters	none	

Set_Mode_Addr_Px

Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	emode	If module is not in Monitor mode
	einval	If an invalid parameter was used as an input
	0	If successful

Set_MON_Concurrent_Px

Description Set_MON_Concurrent_Px links pairs of modules to each other such that one module of the pair can monitor the other module in the pair internally *without* attaching to the bus.

PCI[e] and **cPCI**	Module 1 can monitor module 0 and module 3 can monitor module 2 without attaching to the bus.
VME and **VXI**	In addition to the above, module 5 can monitor module 4 and module 7 can monitor module 6 without attaching to the bus.
EXC-1553ExCARD/Px and ***EXC-1553PCMCIA/Px***	Module 1 can monitor module 0 without attaching to the bus.

Syntax Set_MON_Concurrent_Px (`int` handle, `int` enable)

Input Parameters	handle	The handle designated by Init_Module_Px	
	enable	ENABLE	Enable the internal monitor links [0001 H]
		DISABLE	Disable the internal monitor links [0000 H]

Output Parameters none

Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	econcurrmonmodule	If module 1 or 3 (or modules 5 or 7 on VME/VXI boards) is not currently selected
	einval	If an invalid parameter was used as an input
	0	If successful

Set_Mon_Response_Time_1553A_Px

Description

Set_Mon_Response_Time_1553A_Px sets the Monitor's response time window for 1553A messages. The value determines the maximum wait time until an RT's Status Response is considered invalid by the Monitor.

Note: This function applies to RTs that send 1553A messages. (An RT is designated to be sending 1553A mesages by calling Enable_Mon_1553A_Support_Px.) For RTs that send 1553B messages (the default message type), use Set_Mon_Response_Time_Px instead.

Syntax

Set_Mon_Response_Time_Px (`int` handle, `usint` rtime)

Input Parameters

handle	The handle designated by Init_Module_Px
rtime	Permissible response time in microseconds. The default response time for 1553A messages is 7 microseconds.

Output Parameters

none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
func_invalid	If the module is a *MMSI* module
emode	If not in Bus Monitor mode
efeature_not_supported_PX	If the feature is not supported on this module
0	If successful

Set_Mon_Response_Time_Px

Description

Set_Mon_Response_Time_Px sets the Monitor's response time window for 1553B messages. The value determines the maximum wait time until an RT's Status Response is considered invalid by the Monitor.

Note: This function applies to RTs that send 1553B messages (the default message type). For 1553A messages, use Set_Mon_Response_Time_1553A_Px instead.

Syntax

Set_Mon_Response_Time_Px (`int` handle, `usint` rtime)

Input Parameters

handle	The handle designated by Init_Module_Px
rtime	Permissible response time in microseconds. The default response time for 1553B messages is 14 microseconds.

Output Parameters

none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If not in Bus Monitor mode
0	If successful

Look Up Submode

Assign_Blk_Px

Description Assign_Blk_Px assigns a data block to a RT subaddress.

Syntax Assign_Blk_Px (`int` handle, `int` rtid, `int` blknum)

Input Parameters

handle	The handle designated by Init_Module_Px
rtid	11 bit identifier including RT#, T/R bit and subaddress. See RT Identifier on page 4-3 and RT_Id_Px on page 3-22.
blknum	Block number: Valid values: 1 – 127 Assign data block to RTid *or* 0 De-assigns block‘

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in Monitor mode
einval	If an invalid parameter was used as an input
0	If successful

Enable_Lkup_Int_Px

Description Enable_Lkup_Int_Px enables or disables interrupts for the specified RTid.

Syntax Enable_Lkup_Int_Px (`int` handle, `int` rtid, `int` toggle)

Input Parameters

handle	The handle designated by Init_Module_Px	
rtid	11 bit identifier including RT#, T/R bit and subaddress. See RT Identifier on page 4-3 and RT_Id_Px on page 3-22.	
toggle	ENABLE	Enable interrupts for the RTid [0001 H]
	DISABLE	Disable interrupts for the RTid [0000 H]

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
einval	If an invalid parameter was used as an input
0	If successful

Get_Last_Blknum_Px

Description Get_Last_Blknum_Px returns the number of the last data block written to.

Syntax Get_Last_Blknum_Px (`int` handle)

Input Parameters

handle	The handle designated by Init_Module_Px

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in Monitor mode
otherwise	Number of the current data block in use: Valid values: 1 – 127
0	No block

Get_Message_Px

Description Get_Message_Px retrieves a specified message from the Message Block Area.

Syntax Get_Message_Px (`int` handle, `int` blknum, `struct MONMSG` *msgptr)

Input Parameters

handle	The handle designated by Init_Module_Px
blknum	The blknum for the most recent message is obtained via the Get_Last_Blknum_Px function. Valid values: 0 – 127

Output Parameters

msgptr	Pointer to an address in host memory where you want a copy of the message to be placed. The message is returned in a `MONMSG` structure, as defined below. Space should always be allocated for 36 words of data to accommodate the maximum case of RT-to-RT transmission of 32 Data Words plus 2 Status words and 2 Command words.

struct MONMSG {

unsigned short int msgstatus;	Status word containing the following flags:

END_OF_MSG	Indicates end of message [8000 H]
TRIGGER_FOUND	Trigger message was received and restored [4000 H]
RT2RT_MSG	Message was RT-to-RT transfer [2000 H]
ME_SET	Message Error bit in the RT Status word [1000 H]

Get_Message_Px (cont.)

BAD_STATUS	RT Status word bits set (not ME bit) [0800 H]
INVALID_MSG	Word count or Sync error occurred [0400 H]
BM_BAD_CHECKSUM	Bad checksum [0200 H] (1760 option only)
BUS_A_XFER	Message was transferred on bus A [0100 H]
INVALID_WORD	Bad bit count, Manchester or parity [0080 H]
BAD_HEADER_MON*	Header error [0060 H] (for 1760 option only)

Note: If both WORD_CNT_HI and WORD_CNT_LO are set, this indicates a BAD_HEADER_MON error, which is only applicable to 1760.

WORD_CNT_HI	RT transmitted too many words [0040 H]
WORD_CNT_LO	RT transmitted too few words [0020 H]
BAD_RT_ADDR	Received 1553 Status word did not contain the correct RT address [0010 H]
BAD_SYNC	Sync of either the Command or the Data Word(s) is incorrect [0008 H]
BAD_GAP	Invalid gap received between 1553 Words [0004 H]
MON_LATE_RESP	Response time error occurred in the message, even if no RT active on the module [0002 H]
MSG_ERROR	Error occurred, defined in other flags [0001 H]

unsigned int elapsedtime;	A 32-bit Time Tag associated with the message.
unsigned short int *words;	A pointer to an array of 1553 Words in the sequence they were received over the bus. See **Appendix A: MIL-STD-1553 Word Formats**.
}	

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in Monitor mode
einval	If an invalid parameter was used as an input
0	If successful

Sequential Submode

Enable_Mon_1553A_Support_Px

Description Enable_Mon_1553A_Support_Px specify whether an RT is sending 1553A or 1553B messages.

Syntax Enable_Mon_1553A_Support_Px (`int` handle, `int` rtNum, `int` enableflag)

Input Parameters

handle	The handle designated by Init_Module_Px	
rtNum	Address of the RT Valid values: 0 – 31	
enableflag	ENABLE	The RT is sending 1553A messages [0001 H]
	DISABLE	The RT is sending 1553B messages [0000 H] (default)

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
func_invalid	If the module is a *MMSI* module
einval	If an invalid parameter was used as an input
efeature_not_supported_PX	If the feature is not supported on this module
emode	If module is not in Sequential Monitor mode
0	If successful

Get_Counter_Px

Description Get_Counter_Px returns the number of the last block that was filled with a monitored message.

Valid values:

If Regular Monitor is in use: 0 – 199
If Expanded Monitor is in use: 0 – 799
If Enhanced Monitor is in use: 0 – 399

Syntax Get_Counter_Px (`int` handle)

Input Parameters handle — The handle designated by Init_Module_Px

Output Parameters none

Get_Counter_Px (cont.)

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in Sequential Monitor mode
counter	0 – 199, 0 – 399 or 0 – 799

Get_Message_Info_Px

Description

Get_Message_Info_Px reads the additional information provided for each word when using Enhanced Monitor. When using Enhanced Monitor, 400 blocks are used to store Bus Monitor data, and an additional 400 blocks are used to store additional information about each word. To enable Enhanced Monitor, use Set_Enhanced_Mon_Px.

Note: Before calling this function, call Get_Next_Message_Px to get the correct block number.

Syntax

Get_Message_Info_Px (`int` handle, `int` blknum, `struct MONMSGINFO` *msgInfoPtr)

Input Parameters

handle	The handle designated by Init_Module_Px
blknum	The block number returned by Get_Next_Message_Px Valid values: 0 – 399

Output Parameters

msgInfoPtr	Pointer to the structure defined below in which to return the information

struct MONMSGINFO {

unsigned short int msgstatus;	Status Word containing the following elements:
END_OF_MSG	Indicates end of message [8000 H]
NUM_OF_WORDS	Number of words (1 – 36) in the message [003F H]
unsigned short int reserved1;	Reserved
unsigned short int reserved2;	Reserved
unsigned short int words [36];	A pointer to an array of additional information for each 1553 word in the Bus Monitor buffer
INFO_COMMAND	Indicates that the sync pattern of the word is that of a Command or a Status Word [C000 H]
INFO_DATA	Indicates that the sync pattern of the word is that of a Data Word [D000 H]

Get_Message_Info_Px (cont.)

	INFO_BUSA	Indicates that the word was received on Bus A [0A00 H]
	INFO_BUSB	Indicates that the word was received on Bus B [0B00 H]
	INFO_NOT_CONTIG	Indicates that the word was not received contiguously with the previous word [0010 H]
	INFO_PARITY_ERR	Indicates that the word had a parity error [0001 H]
	INFO_MANCH_ERR	Indicates that the word had a Manchester error [0002 H]
	unsigned short int msgCounter;	A 16-bit serial message count associated with the message
	}	
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	emode	If module is not in Sequential Monitor mode
	func_invalid	If the function is not supported by this version of the firmware. Use Is_Enhanced_Mode_Supported_Px on page 2-23 to check whether it is supported.
	einval	If an invalid parameter was used as an input
	enomsg	If there is no message to return
	0	If successful

Get_Next_Message_Px

Description

Get_Next_Message_Px reads the message block following the block read in the previous call to Get_Next_Message_Px. The first call to Get_Next_Message_Px will return block 0.

Note:

- This function can only be used in Bus Monitor mode, and only when using a standard Time Tag. When using the IRIG B Time Tag, use Get_Next_Mon_IRIGttag_Message_Px instead.
- If there is a user-initiated Time Tag reset (Reset_Time_Tag_Px, Clear_Timetag_Sync_Px, or SetIrig_4000 with the flag IRIG_TIME_RESET), Get_Next_Message_Px will interpret this as an overrun condition. To avoid this, call Ignore_Timetag_Overrun_Px. See Ignore_Timetag_Overrun_Px on page 4-18.

Syntax

Get_Next_Message_Px (`int` handle, `struct MONMSG` *msgptr)

Input Parameters

handle — The handle designated by Init_Module_Px

Output Parameters

msgptr — Pointer to an address in host memory where you want a copy of the message to be placed. The message is returned in a `MONMSG` structure, as defined below.

Space should always be allocated for 36 words of data to accommodate the maximum case of RT-to-RT transmission of 32 Data Words plus 2 Status words and 2 Command words.

```
struct MONMSG {
unsigned short int msgstatus;
```

Status word containing the following flags:

END_OF_MSG	Indicates end of message [8000 H]
TRIGGER_FOUND	Trigger message was received and restored [4000 H]
RT2RT_MSG	Message was RT-to-RT transfer [2000 H]
ME_SET	Message Error bit in the RT Status word [1000 H]
BAD_STATUS	RT Status word bits set (not ME bit) [0800 H]
INVALID_MSG	Word count or Sync error occurred [0400 H]
BM_BAD_CHECKSUM	Bad checksum [0200 H] (1760 option only)
BUS_A_XFER	Message was transferred on bus A [0100 H]

Get_Next_Message_Px (cont.)

INVALID_WORD	Bad bit count, Manchester or parity [0080 H]
BAD_HEADER_MON*	Header error [0060 H] (for 1760 option only)

Note: If both WORD_CNT_HI and WORD_CNT_LO are set, this indicates a BAD_HEADER_MON error, which is only applicable to 1760.

WORD_CNT_HI	RT transmitted too many words [0040 H]
WORD_CNT_LO	RT transmitted too few words [0020 H]
BAD_RT_ADDR	Received 1553 Status word did not contain the correct RT address [0010 H]
BAD_SYNC	Sync of either the Command or the Data Word(s) is incorrect [0008 H]
BAD_GAP	Invalid gap received between 1553 Words [0004 H]
MON_LATE_RESP	Response time error occurred in the message, even if no RT active on the module [0002 H]
MSG_ERROR	Error occurred, defined in other flags [0001 H]

unsigned int elapsedtime;	A 32-bit Time Tag associated with the message.
unsigned short int *words [36];	A pointer to an array of 1553 words in the sequence they were received over the bus. See **Appendix A: MIL-STD-1553 Word Formats**.
unsigned short int msgCounter;	The low 16 bits of the message number assigned to this message. The high 16 bits are stored in the Message Number Hi register [3EBC H].

}

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in Monitor mode
enomsg	If no new messages have been received
eoverrun	If message was overwritten in the middle of being read here. At least 200 messages were received since the previous Get_Next_Message_Px, so the Time Tags are out of sync.

Get_Next_Message_Px (cont.)

eoverrunTtag	If the Time Tag of the message was overwritten while the message was being read
eoverrunEOM flag	If the End Of Message flag was overwritten while the message was being read
eIrigTimetagSet_PX	If this function was called when in IRIG B Time Tag mode (Set_IRIG_TimeTag_Mode_Px); use Get_Next_Mon_IRIGttag_Message_Px instead
block number	If successful Valid values: If Regular Monitor is in use: 0 – 199 If Expanded Monitor is in use: 0 – 799 If Enhanced Monitor is in use: 0 – 399

Get_Next_Mon_IRIGttag_Message_Px

Description

Get_Next_Mon_IRIGttag_Message_Px reads the message block following the block read in the previous call to Get_Next_Mon_IRIGttag_Message_Px. The first call to Get_Next_Mon_IRIGttag_Message_Px will return block 0. The Time Tag is returned in decimal representation.

This function should be used only when using an IRIG B Time Tag. When using a standard Time Tag, use Get_Next_Message_Px instead.

Note: This function is only available in Bus Monitor Sequential Fixed-Block mode.

Syntax

Get_Next_Mon_IRIGttag_Message_Px (`int` handle, `struct MONIRIGMSG_TTAG` *msgptr, `unsigned long` *msg_counter)

Input Parameters

handle	The handle designated by Init_Module_Px

Output Parameters

msgptr	Pointer to an address in host memory where you want a copy of the message to be placed. The message is returned in a `MONIRIGMSG_TTAG` structure, as defined in Figure 4-1. Space should always be allocated for 36 words of data to accommodate the maximum case of RT-to-RT transmission of 32 Data words plus 2 Status words and 2 Command words.

Get_Next_Mon_IRIGttag_Message_Px (cont.)

struct MONIRIGMSG {

unsigned short int msgstatus;	Status word containing the flags as defined in Get_Next_Message_Px
t_IrigOnModule Time IrigTime	A structure containing the IRIG B time stamp in decimal digits, defined in **pxIncl.h** as: typedef struct { unsigned short int days; unsigned short int hours; unsigned short int minutes; unsigned short int seconds; unsigned int microsecs; } t_IrigOnModuleTime;
unsigned short int words [36];	A pointer to an array of 1553 words in the sequence they were received over the bus
unsigned short int msgCounterLo;	The low 16 bits of the message number assigned to this message
unsigned short int msgCounterHi;	The high 16 bits of the message number assigned to this message

}

Return Values

emode	If module is not in Monitor mode
enomsg	If no new messages have been received
ewrongprocessor	If the processor is not a NIOS II
ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
eoverrunTtag	If the Time Tag of the message was overwritten while the message was being read
eoverrunEOM flag	If the End Of Message flag was overwritten while the message was being read
efeature_not_ supported_PX	If the feature is not supported on this module
eNoIrigModeSet_ PX	If Set_IRIG_TimeTag_Mode_Px was not called
block number	If successful

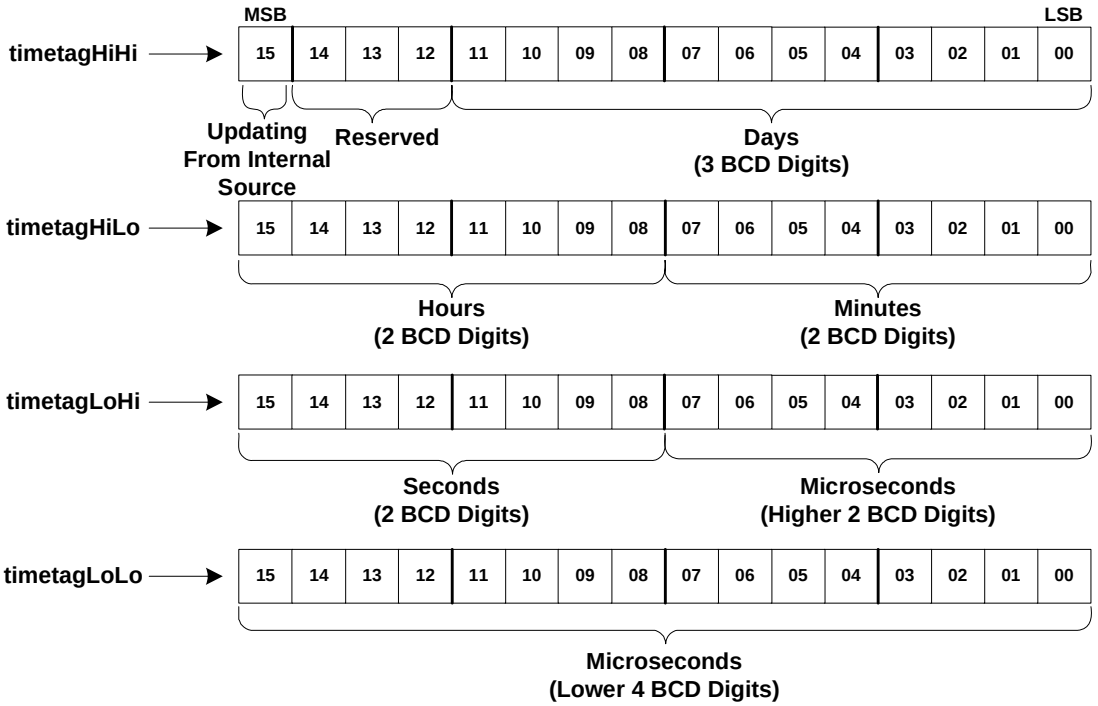


Figure 4-1 IRIG B Time Tag

Note: The **Updating from Internal Source** bit in timetagHiHi indicates whether a valid external IRIG B signal is being received by the module. When this bit is set to 1, the module is not receiving an IRIG B signal, and the module is updating the Time Tag based on its internal clock.

Ignore_Timetag_Overrun_Px

Description Ignore_Timetag_Overrun_Px is used in conjunction with a user-initiated Time Tag reset (Reset_Time_Tag_Px, Clear_Timetag_Sync_Px, or SetIrig_4000 with the flag IRIG_TIME_RESET) to stop Get_Next_Message_Px from returning an overrun error even when the Time Tag associated with the message following the Time Tag reset is earlier than the previous read message.

Syntax Ignore_Timetag_Overrun_Px (`int` handle, `int` enableflag)

Input Parameters

handle	The handle designated by Init_Module_Px	
enableflag	ENABLE	Ignores overrun error [0001 H]
	DISABLE	Does not ignore overrun error [0000 H]

Output Parameters none

Ignore_Timetag_Overrun_Px (cont.)

Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	emode	If module is not in Sequential Monitor mode
	einval	If an invalid parameter was used as an input
	0	If successful

Set_Cnt_Trig_Px

Description Set_Cnt_Trig_Px sets a block number as the counter trigger. When this block is filled, the CNT_TRIG_MATCH bit within the Monitor status word will be set and, if requested via the Set_Interrupt_Px function an interrupt will be generated.

Note: If the module is running when this function is called, it does not take effect until the module is restarted.

Syntax Set_Cnt_Trig_Px (`int` handle, `int` blknum)

Input Parameters	handle	The handle designated by Init_Module_Px
	blknum	Block number Valid values: If Regular Monitor is in use: 0 – 199 If Expanded Monitor is in use: 0 – 799 If Enhanced Monitor is in use: 0 – 399
Output Parameters	none	
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	emode	If module is not in Sequential Monitor mode
	einval	If an invalid parameter was used as an input
	0	If successful

Set_Enhanced_Mon_Px

Description

Set_Enhanced_Mon_Px is used to enable or disable Enhanced Monitor. When using Enhanced Monitor, 400 blocks are used to store Bus Monitor data, and an additional 400 blocks are used to store additional information about each word. To read the additional info, use Get_Message_Info_Px.

Note: If this function is called when Expanded Monitor is in use, the module switches from Expanded Monitor to Enhanced Monitor. See Set_Expanded_Block_Mode_Px.

Syntax

Set_Enhanced_Mon_Px (`int` handle, `int` enableFlag)

Input Parameters

handle	The handle designated by Init_Module_Px	
enableflag	ENABLE	Enables Enhanced Monitor [0001 H]
	DISABLE	Disables Enhanced Monitor [0000 H]

Output Parameters

none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in Sequential Monitor mode
einval	If an invalid parameter was used as an input
efeature_not_supported_PX	If the feature is not supported on this module
func_invalid	If the function is not supported by this version of the firmware. Use Is_Enhanced_Mode_Supported_Px on page 2-23 to check whether it is supported.
0	If successful

Set_Message_Interval_Interrupt_Value_Px

Description

Set_Message_Interval_Interrupt_Value_Px is used to set a message status flag (MSG_INTERVAL) after a specific number of messages are received. In addition, an interrupt message can be sent when the specific number of messages are received. See Set_Interrupt_Px on page 2-34. Note that only messages that are saved in memory are counted, not messages that are filtered out.

Syntax

Set_Message_Interval_Interrupt_Value_Px (`int` handle, `int` interval)

Input Parameters

handle	The handle designated by Init_Module_Px	
interval	The number of messages after which to generate an interrupt	
	NO_MSG_INTERVAL_INTERRUPT	Disables interval interrupts [0000 H]

Output Parameters

none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in Sequential Monitor mode
0	If successful

Set_Trigger_Mode_Px

Description

Set_Trigger_Mode_Px can be used to set additional conditions for triggers that have been selected by calling Set_Trigger1_Px and/or Set_Trigger2_Px. Before calling this function, you must first call Set_Trigger1_Px or Set_Trigger2_Px to define the trigger. (See Set_Trigger1_Px on page 4-23 and Set_Trigger2_Px on page 4-24.)

Set_Trigger_Mode_Px determines:

1. The action to be taken upon receiving a message matching the trigger.
2. If the source of the trigger will be the Command word or the message Status Word.

The trigger message is always stored, and the user can choose to store all messages from this point on or only those messages matching the trigger.

Note: To disable the triggers, call Set_Trigger1_Px or Set_Trigger2_Px (whichever one was previously set) with the desired trigger, and a mask set to all 0's.

Syntax

Set_Trigger_Mode_Px (`int` handle, `int` mode)

Set_Trigger_Mode_Px (cont.)

Input Parameters	handle	The handle designated by Init_Module_Px	
	mode	Combination of one flag from each group ORed together	
	When to trigger	STORE_AFTER	Default – Causes storage of messages to begin following the first occurrence of a trigger message [0010 H]
		STORE_ONLY	Causes only messages matching the trigger to be stored [0008 H]
	On what to trigger	TCOMMAND	Default – Causes triggering to be based on the Command word [0000 H]
		STATUS_TRIGGER	Causes triggering to be based on the Message Status word [0080 H]
Output Parameters	none		
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px	
	emode	If module is not in Sequential Monitor mode	
	einval	If an invalid parameter was used as an input	
	0	If successful	

Set_Trigger1_Px

Description

In Set_Trigger_Mode_Px the user can select a 1553 Command word or a Message Status word for the monitor to trigger on. (See Set_Trigger_Mode_Px on page 4-21.)

In Set_Trigger1_Px the user specifies which Control word or Message Status word is the trigger. When a message matching this parameter is encountered, the trigger is met and the action specified in Set_Trigger_Mode_Px is taken.

The user has the option of setting an additional trigger using Set_Trigger2_Px.

Note:

- Set_Trigger1_Px must be called *before* Set_Trigger_Mode_Px. Otherwise, the mode may not be set properly.
- To disable this trigger, call this function with a mask set to all 0's.

Syntax

Set_Trigger1_Px (`int` handle, `int` trigger, `int` mask)

Input Parameters

handle	The handle designated by Init_Module_Px
trigger	16 bit trigger value – a Command word or Message Status Word See Set_Trigger_Mode_Px on page 4-21
mask	Mask for trigger. Specifies which bits in the trigger word are required to match and which are 'don't care' '1' in required bit positions '0' in don't care bit positions Mask of all 0's turns off trigger
Example	When triggering on Command words, Set_Trigger1_Px or Set_Trigger2_Px (0843 H, F800 H) would trigger all messages to/from RT1.

Output Parameters

none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in Sequential Monitor mode
0	If successful

Set_Trigger2_Px

Description

In Set_Trigger_Mode_Px the user can select a 1553 Command word or a Message Status word for the monitor to trigger on. (See Set_Trigger_Mode_Px on page 4-21.)

In Set_Trigger2_Px the user specifies which Control word or Message Status word is the trigger. When a message matching this parameter is encountered, the trigger is met and the action specified in Set_Trigger_Mode_Px is taken.

The user has the option of setting an additional trigger using Set_Trigger1_Px.

Note:

- Set_Trigger2_Px must be called *before* Set_Trigger_Mode_Px. Otherwise, the mode may not be set properly.
- To disable this trigger, call this function with a mask set to all 0's.

Syntax

Set_Trigger2_Px (`int` handle, `int` trigger, `int` mask)

Input Parameters

handle	The handle designated by Init_Module_Px
trigger	16 bit trigger value – a Command word or Message Status Word See Set_Trigger_Mode_Px on page 4-21
mask	Mask for trigger. Specifies which bits in the trigger word are required to match and which are 'don't care' '1' in required bit positions '0' in don't care bit positions Mask of all 0's turns off trigger
Example	When triggering on Command words, Set_Trigger1_Px or Set_Trigger2_Px (0843 H, F800 H) would trigger all messages to/from RT1.

Output Parameters

none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in Sequential Monitor mode
0	If successful

5 BC/Concurrent-RT Functions

Chapter 5 describes the *1553Px Software Tools* functions for operating the *Px* module in BC/Concurrent-RT mode. To set the module in BC/Concurrent-RT mode, see Set_Mode_Px on page 2-36.

In BC/Concurrent-RT mode the module operates as a Bus Controller. The user can define 1553 messages and group them into frames and has control over which commands and data are transmitted over the bus as well as the timing between messages and frames. Error injection options permit detailed testing of RTs, and error detection features provide rich reporting capabilities.

In addition, the module can simulate up to 32 Remote Terminals using the function Set_RT_Active_Px on page 2-38.

All references to a module apply both to the removable *Px* module and to the *Px* module integrated on a board or card. For more information, see Introduction on page 1-1.

Note: When using a single function module (*PxS*) in BC mode, you cannot use the module as a Concurrent-RT.

The following functions are described in this chapter:

Alter_Cmd_Px
Alter_IMG_Px
Alter_Message_Px
Alter_MsgSendTime_Px
BC_Check_Alter_Msgentry_Px
Clear_Card_Px
Clear_Frame_Px
Command_Word_Px
Create_1553_Message_Px
Create_Frame_Px
Enable_Checksum_Error_Px[1, 2]
Enable_Checksum_Px[2]
Enable_SRQ_Support_Px
Get_BC_Msgentry_Px
Get_BC_Status_Px
Get_Minor_Frame_Time_Px
Get_Msgentry_Status_Px
Get_Next_Message_BCM_Px
Insert_Msg_Err_Px[1]
Re_Create_Message_Px
Read_Message_Px
Read_SRQ_Message_Px
Reset_BC_Status_Px
Run_BC_Px
Select_Async_Frame_Px
Send_Async_Frame_Px
Send_Msg_Once_Px
Send_Timetag_Sync_Px
Set_BC_Resp_Px
Set_Bus_Px
Set_Continue_Px
Set_Error_Location_Px[1]
Set_Frame_Time_Px
Set_Frequency_Mode_Px
Set_Halt_Px
Set_Interrupt_On_Msg_Px
Set_Jump_Px
Set_Minor_Frame_Time_Px
Set_Replay_Px
Set_Restore_Px
Set_Retry_Px
Set_RT_Nonactive_Px[1]
Set_Skip_Px
Set_Stop_On_Error_Px
Set_Sync_Pattern_Px[1]
Set_Word_Cnt_Px[1]
Set_Zero_Cross_Px[1]
Start_Frame_Px

1. Not for single function module (*PxS*)
2. Only for modules that support the 1760 option

The input values included in each function are given, in Hex format, by each flag within the function description.

BC/Concurrent-RT Simulation

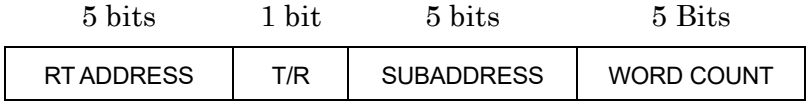
In BC/Concurrent-RT mode, the Create_1553_Message_Px function is used to define both the BC message and any simulated RT response. In BC/Concurrent-RT mode, the maximum number of Words in the message is 36:

	Maximum number of Data Words	32
+	Maximum number of Command words	2
+	Maximum number of Status words	2
	Maximum number of Words in the message	36

Refer to the hardware *User's Manual* that came with your module for further explanation and examples of this mode.

Command Word Calculation

Some of the functions in BC/Concurrent-RT mode take the 1553 Command word as an argument. The structure of the Command word is:



Example: RT5, Receive, Subaddress 6 and Word count 15 would be represented as 0010 1000 1100 1111 in binary or 28CF in hex.

The Command_Word_Px function, on page 5-12, is provided to carry out this calculation.

Servicing the Service Request (SRQ) Bit

The SRQ bit is set for a Remote Terminal (RT) in the 1553 RT Status Word. Setting the SRQ bit indicates to the Bus Controller (BC) that the RT/Subaddress requires service.

In response, the BC provides the following service: the module sends out a Mode Code 16 (transmit Vector Word) to get the Vector Word from the RT which contains more information about what needs service. The BC then builds and sends out a transmit message (RT-to-BC) to this RT, with the Subaddress and Word Count as indicated in the corresponding bit positions of the Vector Word (provided that the Subaddress is not set to 0).

Functions by Category

The more frequently called functions are grouped according to use. Page references are given only if the functions do not appear in this chapter.

Sending out a fixed number of messages in a loop

Init_Module_Px for Most Boards	To initialize the module/card (page 2-18)
Set_Mode_Px	To place the module in BC/Concurrent-RT mode (page 2-36)
Create_1553_Message_Px	To create the messages to send
Create_Frame_Px	To select the order in which to transmit the messages
Set_Frame_Time_Px	To determine the time between transmission of frames
Start_Frame_Px	To choose the frame to transmit
Run_BC_Px	To start the module running
Stop_Px	To Stop the RT
Release_Module_Px	To release resources assigned to the module

Additional frame functions:

Select_Async_Frame_Px	To designate a frame to be sent out by Send_Async_Frame
Send_Async_Frame_Px	To send the asynchronous frame that was setup by a call to Select_Async_Frame
Set_Minor_Frame_Time_Px	To set the minor frame time for a Minor frame type of message

Obtaining data and status information during transmission

Read_Message_Px	To get data and 1553 status information
Get_BC_Msgentry_Px	To get both data and 1553 status information AND status information for a particular message
Get_BC_Status_Px	To get module level status information
Reset_BC_Status_Px	To clear the BC status
Get_Next_Message_BCM_Px	To read messages monitored on the bus by the Internal Concurrent Monitor

Changing message parameters in real-time

Set_BC_Resp_Px	To select how long to wait for an RT to respond
Set_Halt_Px	To stop transmission at a selected message
Set_Continue_Px	To resume transmission following a halt
Set_Skip_Px	To skip transmission of a specified message
Set_Restore_Px	To restore a skipped message
Set_Stop_On_Error_Px	To cause the module to stop if it detects an error
Clear_Card_Px	To fully clear (entire module) in order to restart
Clear_Frame_Px	To partially clear (only frame) in order to restart

Setting RTs

In BC/Concurrent-RT mode, the user can simulate an entire system by simulating one or more RTs in addition to simulating the Bus Controller. To simulate RTs, set up the Data and Status Words that the simulated RTs are expected to send in Create_1553_Message_Px and Alter_Message_Px.

The functions to simulate RTs are described in detail in **Chapter 3: Remote Terminal Functions**.

Set_RT_Active_Px	To select the RTs to be simulated	page 2-38
Set_RT_Nonactive_Px	To turn off simulation of a RT	page 3-30
Set_RT_Resp_Time_Px	To select how long to wait before responding to a command	page 2-39

Note: These RT functions are the only ones usable in BC/Concurrent-RT mode. Use Run_BC_Px to start BC/Concurrent-RT module operation and *not* Run_RT_Px.

To send checksum values – for 1760 option modules

With the Excalibur *Px* modules which support the 1760 option in BC and BC/Concurrent-RT modes, the user can send a checksum value in place of the last Data Word in the message. In addition, an error may be injected into this checksum value. The following functions control these functions:

Enable_Checksum_Px	To select whether or not to place checksum as the last Data Word of a message
Enable_Checksum_Error_Px	To select whether or not to use checksum error injection on the message

Error Injection

The *Px* module provides a wide array of error injection capabilities for testing at a systems level. This section describes how to invoke error injection in BC mode. For information on using error injection in RT mode, see **Chapter 3: Remote Terminal Functions**.

Bit Count Error Injection

To inject a bit count error in a Command word call:

Set_Bit_Cnt_Px (handle, offset)	To alter the number of bits in transmitted data
Insert_Msg_Err_Px (handle, id, BIT_CNT_ERR)	To inject the error

To inject a bit count error in a Data Word (receive message) call:

Set_Bit_Cnt_Px (handle, offset)	To alter the number of bits in transmitted data
Set_Error_Location_Px (handle, errorWord, 0)	To set the location of the error injection
Insert_Msg_Err_Px (handle, id, BIT_CNT_ERR | DATA_ERR)	To inject the error

Zero Cross Error Injection

To inject a zero cross error in a Command word call:

Set_Zero_Cross_Px (handle, zctype)	To set the type of zero cross error to be used
Set_Error_Location_Px (handle, 0, zcerrorBit)	To set the location of the error injection
Insert_Msg_Err_Px (handle, id, BIT_CNT_ERR)	To inject the error

To inject a zero cross error in a Data Word (receive message) call:

Set_Zero_Cross_Px (handle, zctype)	To set the type of zero cross error to be used
Set_Error_Location_Px (handle, errorWord, zcerrorBit)	To set the location of the error injection
Insert_Msg_Err_Px (handle, id, BIT_CNT_ERR | DATA_ERR)	To inject the error

Word Count Error Injection

To inject a word count error in a Data Word (receive message) call:

Set_Word_Cnt_Px (handle, offset)	To set the number of words to send for messages that have requested word count errors
Insert_Msg_Err_Px (handle, id, WD_CNT_ERR)	To inject the error

Noncontiguous Data Error Injection

To inject a noncontiguous data error in a Data Word (receive message) call:

Set_Error_Location_Px (handle, errorWord, 0)	To set the location of the error injection
Insert_Msg_Err_Px (handle, id, GAP_ERR | DATA_ERR)	To inject the error

Parity Error Injection

To inject a parity error in a Command word call:

Insert_Msg_Err_Px (handle, id, PARITY_ERR)	To inject the error

To inject a parity error in a Data Word (receive message) call:

Set_Error_Location_Px (handle, errorWord, 0)	To set the location of the error injection
Insert_Msg_Err_Px (handle, id, PARITY_ERR | DATA_ERR)	To inject the error

Incorrect Synchronization Error Injection

To inject an incorrect synchronization error in a Command word call:

Insert_Msg_Err_Px (handle, id, SYNC_ERR)	To inject the error

To inject an incorrect synchronization error in a Data Word (receive message) call:

Set_Error_Location_Px (handle, errorWord, 0)	To set the location of the error injection
Insert_Msg_Err_Px (handle, id, SYNC_ERR | DATA_ERR)	To inject the error

Synchronization Pattern Error Injection

To inject a synchronization pattern error in a Command word call:

Set_Sync_Pattern_Px (handle, pattern)	To set the sync pattern to be used
Insert_Msg_Err_Px (handle, id, SYNC_ERR)	To inject the error

To inject a synchronization pattern error in a Data Word (receive message) call:

Set_Sync_Pattern_Px (handle, pattern)	To set the sync pattern to be used
Set_Error_Location_Px (handle, errorWord, 0)	To set the location of the error injection
Insert_Msg_Err_Px (handle, id, SYNC_ERR | DATA_ERR)	To inject the error

BC/Concurrent-RT Functions

Alter_Cmd_Px

Description Alter_Cmd_Px changes the Command word of a previously defined message. The function takes an RT address and a SA as arguments and copies these into the original Command word. The number of Data Words cannot be changed.

Syntax Alter_Cmd_Px (`int` handle, `int` id, `usint` rt, `usint` sa)

Input Parameters

handle	The handle designated by Init_Module_Px	
id	Message identifier returned from a prior call to Create_1553_Message_Px	
rt	The new RT number Valid values: 0 – 31 *or*	
	SAME_RT	Do not change the RT of the Command [0020 H]
sa	The new Subaddress number Valid values: 0 – 31 *or*	
	SAME_SA	Do not change the SA of the Command [0020 H]

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If not in BC mode
ebadid	If ID is not a valid message ID
0	If successful

Alter_IMG_Px

Description Alter_IMG_Px changes the intermessage gap time of a previously defined message.

Syntax Alter_IMG_Px (`int` handle, `int` frameid, `int` msgentry, `unsigned long` img)

Input Parameters

handle	The handle designated by Init_Module_Px
frameid	Frame identifier returned from a prior call to Create_Frame_Px
msgentry	Entry within the frame, i.e., 0 for the first message in the frame, 1 for the second message, etc.
img	New intermessage gap time in microseconds

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
frm_erange	If frameid is not a valid frame id
frm_erangecnt	If msgentry is greater than the number of messages in the frame
0	If successful

Alter_Message_Px

Description Alter_Message_Px changes the data of a previously defined message. The function receives as input a pointer to an array of Data Words and copies these words into the original message. The Command word cannot be changed and therefore the number of Data Words cannot be changed. Only the values of the Data Words (and Status word(s) for simulated RTs in BC/Concurrent-RT mode) can be changed.

Note: See **Appendix A: MIL-STD-1553 Word Formats.**

Syntax Alter_Message_Px (`int` handle, `int` id, `usint` *data)

Input Parameters

handle	The handle designated by Init_Module_Px
id	Message identifier returned from a prior call to Create_1553_Message_Px
data	Pointer to an array of up to 34 words of new Data Words (plus Status word(s) for BC/Concurrent-RT mode) for message ID. **Note:** Space must be left for the Status word, even if the RT is not simulated.

Output Parameters none

Alter_Message_Px (cont.)

Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	ebadid	If ID is not a valid message ID
	emode	If not in BC/Concurrent-RT mode
	eminorframe	If message is Minor Frame type
	0	If successful

Alter_MsgSendTime_Px

Description Alter_MsgSendTime_Px changes a message send time previously defined in Create_Frame_Px. This is for use with the replay feature (Set_Replay_Px on page 5-38) to specify exactly when a message will be sent out.

Syntax Alter_MsgSendTime_Px (`int` handle, `int` frameid, `int` msgentry, `unsigned int` mst)

Input Parameters	handle	The handle designated by Init_Module_Px
	frameid	Frame identifier returned from a prior call to Create_Frame_Px
	msgentry	Entry within the frame, i.e. 0 for the first message in the frame, 1 for the second message, etc.
	mst	The Time Tag value at which the message should be sent out in replay mode
Output Parameters	none	
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	emode	If not in BC/Concurrent-RT mode
	frm_erange	If frameid is not a valid frame id
	frm_erangecnt	If msgentry is greater than the number of messages in the frame
	0	If successful

BC_Check_Alter_Msgentry_Px

Description

To insure data integrity, call BC_Check_Alter_Msgentry_Px before a call to Alter_Message_Px. The function verifies that a message that is to be altered is not in the process of being transmitted or received. Without this check, a message may be altered in the middle of its transmission.

After a successful call to this function the user has at least 20 µsec. to call Alter_Message_Px or Read_Message_Px.

Note: BC_Check_Alter_Msgentry_Px replaces the obsolete function BC_Check_Alter_Msg_Px. BC_Check_Alter_Msgentry_Px is more accurate when multiple frames are used.

Syntax

BC_Check_Alter_Msgentry_Px (`int` handle, `int` frameid, `int` msgentry)

Input Parameters

handle	The handle designated by Init_Module_Px
frameid	Frame identifier returned from a prior call to Create_Frame
msgentry	Entry within the frame, i.e. 0 for the first message in the frame, 1 for the second message, etc.

Output Parameters

none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px	
emode	If module is not in BC/Concurrent-RT mode.	
einval	If an invalid parameter was used as an input	
frm_erange	If `frameid` is not a valid frame id	
frm_erangecnt	If msgentry is greater than the number of messages in the frame	
Successful if:	ALTER_MSG	If message is available to be altered [0001 H]
	DO_NOT_ALTER_MSG	If message *is not* available to be altered [0000 H]

Clear_Card_Px

Description Clear_Card_Px clears out all messages and message frames (and the *1553Px Software Tools* variables associated with them) from the memory of the current module. Use this function to delete old, unnecessary messages to make room for new messages. It may be called to undo calls to Create_1553_Message_Px and Create_Frame_Px.

Note: Call this function only when the module is stopped.

Syntax Clear_Card_Px (`int` handle)

Input Parameters	handle	The handle designated by Init_Module_Px
Output Parameters	none	
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	0	If successful

Clear_Frame_Px

Description Clear_Frame_Px clears out all message frames previously created via Create_Frame_Px while leaving messages created via Create_1553_Message_Px. May be used in situations when a limited number of messages must be reformulated into different frames in realtime.

Note:

- It is often more efficient to use the Set_Skip_Px and Set_Restore_Px functions for this situation.
- Call this function only when the module is stopped.

Syntax Clear_Frame_Px (`int` handle)

Input Parameters	handle	The handle designated by Init_Module_Px
Output Parameters	none	
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	0	If successful

Command_Word_Px

Description Command_Word_Px is a utility function to help the user calculate the 1553 Command word value.

Syntax Command_Word_Px (`int` rtnum, `int` type, `int` subaddr, `int` wordcount, `usint` *commandword)

Input Parameters

rtnum	Address of the RT Valid values: 0 – 31	
type	TRANSMIT	Indicates that the RT/SA should transmit Data Words [0001 H]
	RECEIVE	Indicates that the RT/SA should receive Data Words [0000 H]
subaddr	Designated subaddress of the given RT Valid values: 0 – 31	
wordcount	The number of words to be received or transmitted in a message. Valid values: 0 – 31 **Note:** For messages with at least one Data Word, 0 indicates a word count of 32.	

Output Parameters

commandword	The calculated Command word

Return Values

0	If successful

Create_Frame_Px

Description

Create_Frame_Px creates a frame of previously defined messages to be sent by the BC. Each element in the frame structure consists of a message ID (the value returned from a call to Create_1553_Message_Px), and a gap time indicating the time between the transmission of the message and the following one. The frame structure must contain a final entry with id '0', indicating the end of the frame.

Up to 20 frames may be defined by calls to Create_Frame_Px, each call returning a frame id which can be used as a parameter to Start_Frame_Px to specify which frame to send next. Run_BC_Px then sends the messages in that frame.

Note: When setting up a frame for use in Replay mode, indicate the required 'message send time' for each message, in place of the gap time. (See Set_Replay_Px on page 5-38 and Alter_MsgSendTime_Px on page 5-9.)

Syntax

Create_Frame_Px (`int` handle, `struct FRAME` * framestruct)

Input Parameters

handle	The handle designated by Init_Module_Px

FRAME framestruct [ ];

struct FRAME {

short int id;	Message identifier returned by Create_1553_Message_Px or 0 for last message
long gaptime;	Gap Time between this message and following message in microseconds *or* Message Send Time – Time Tag at which this message should be sent on in replay mode.

};

Output Parameters

none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
frm_badid	If tried to place an undefined message into a message frame
frm_nostack	If there is not enough space in frame stack for this frame
frm_maxframe	If exceeded maximum number of frames permitted (20)
frameid	If successful. A frame identifier of a frame just created. The frameid is used in many of the functions in BC/Concurrent-RT mode. Valid values: 0 – 19

Create_1553_Message_Px

Description

Create_1553_Message_Px creates the basic building block of the 1553 protocol – the message. A message contains a minimum of one, and a maximum of two (for RT-to-RT messages), Command words and 32 Data Words, depending on the type of message. Each message created with this function returns a unique id which is used to set or read characteristics of the message or to build a frame (i.e., a collection of messages) for subsequent transmission. The message must define the Data and Status words to be sent by the RT that is being simulated.

Note: See **Appendix A: MIL-STD-1553 Word Formats**.

Syntax

Create_1553_Message_Px (`int` handle, `usint` cmdtype, `usint` data[ ], `short int` *id)

Input Parameters

handle	The handle designated by Init_Module_Px	
cmdtype	One of the following flags:	
	RT2BC	Send a transmit message [0000 H]
	BC2RT	Send a Receive message [0001 H]
	RT2RT	Send an RT to RT transfer message [0002 H]
	MODE	Mode Command [0003 H]
	BRD_RCV	Broadcast Receive message [0004 H]
	BRD_RT2RT	Broadcast RT to RT transfer message [0005 H]
	BRD_MODE	Broadcast Mode Command [0006 H]
	MINOR_FRAME	Minor frame message/ delay [000F H]
data[ ]	A pointer to an array of command + data + status	

Note: In BC/Concurrent-RT mode the user may create an RT-to-RT message with 32 Data Words, 2 Command words and 2 Status words for 2 simulated RTs.

Output Parameters

id	On success, returns a message ID identifying the message just created, for use in other function calls.

Create_1553_Message_Px (cont.)

Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	emode	If not in BC/Concurrent-RT mode
	ebadcommandword	If the T/R bit in the Command word is set incorrectly for the given cmdtype parameter
	einval	If an invalid parameter was used as an input
	msgnospace	If there is not enough space in message stack for this message
	msg2many	If exceeded maximum number of messages permitted (1660)
	0	If successful

Enable_Checksum_Px

Description Enable_Checksum_Px specifies if the last word of a message should be a checksum word or a regular Data Word. This function is used for BC-to-RT and RT-to-RT messages.

Note: This function is applicable *only* to modules with 1760 option.

Syntax Enable_Checksum_Px (`int` handle, `int` frameid, `int` msgentry, `int` enable)

Input Parameters	handle	The handle designated by Init_Module_Px
	frameid	Frame identifier returned from a prior call to Create_Frame_Px
	msgentry	Entry within the frame, i.e. 0 for the first message in the frame, 1 for the second message, etc.
	enable	ENABLE Checksum is sent [0001 H]
		DISABLE Checksum is *not* sent [0000 H]
Output Parameters	none	
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	emode	If not in BC/Concurrent-RT mode
	frm_erange	If frameid is not a valid frame id
	frm_erangecnt	If msgentry is greater than the number of messages in the frame
	0	If successful

Enable_Checksum_Error_Px

Description

Enable_Checksum_Error_Px sets checksum error injection for the message. Enable_Checksum_Px must be called for the message, to specify that checksum is sent.

This function is used for BC-to-RT and RT-to-RT messages.

Note: This function *only* applies to modules with 1760 option and does not apply to single function (*PxS*) modules.

Syntax

Enable_Checksum_Error_Px (`int` handle, `int` frameid, `int` msgentry, `int` enable)

Input Parameters

handle	The handle designated by Init_Module_Px	
frameid	Frame identifier returned from a prior call to Create_Frame_Px	
msgentry	Entry within the frame, i.e. 0 for the first message in the frame, 1 for the second message, etc.	
enable	ENABLE	Erroneous checksum is sent [0001 H]
	DISABLE	Correct checksum is sent [0000 H]

Output Parameters

none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If not in BC/Concurrent-RT mode
frm_erange	If frameid is not a valid frame id
frm_erangecnt	If msgentry is greater than the number of messages in the frame
enotforsinglefuncerror	If this function is called with a single function module (*PxS*)
0	If successful

Enable_SRQ_Support_Px

Description Enable_SRQ_Support_Px enables Service Request processing. See Servicing the Service Request (SRQ) Bit on page 5-2. If this function is not called, SRQ processing is enable by default.

Syntax Enable_SRQ_Support_Px (`int` handle, `int` enableflag)

Input Parameters

handle	The handle designated by Init_Module_Px
enableflag	ENABLE Enable SRQ processing [0001 H] DISABLE Disable SRQ processing [0000 H]

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If not in BC mode
einval	If an invalid parameter was used as an input
0	If successful

Get_BC_Msgentry_Px

Description Get_BC_Msgentry_Px combines the status and the contents of a message in one structure, similar to the function of Get_Next_Message_RTM_Px (RT) and Get_Next_Message_BCM_Px (BC).

Get_BC_Msgentry_Px replaces the obsolete function Get_BC_Message_Px. Get_BC_Msgentry_Px is more accurate when multiple frames are used.

Syntax Get_BC_Msgentry_Px (`int` handle, `int` frameid, `int` msgentry, `struct BCMSG` *msgptr)

Input Parameters

handle	The handle designated by Init_Module_Px
frameid	Frame identifier returned from a prior call to Create_Frame_Px
msgentry	Entry within the frame, i.e., 0 for the first message in the frame, 1 for the second message, etc.

Output Parameters

msgptr	Pointer to an address in host memory where you want a copy of the message to be placed. The message is returned in a `BCMSG` structure, as defined below. Space should always be allocated for 36 words of data to accommodate the maximum case of RT-to-RT transmission of 32 Data Words plus 2 Status words and 2 Command words.

Get_BC_Msgentry_Px (cont.)

struct BCMSG {

unsigned int msgstatus	Status word containing the following words:	
	END_OF_MSG	Indicates end of message [8000 H]
	BC_BAD_CHECKSUM	Bad checksum on the transmit side [4000 H]
	BAD_BUS	RT response received on non-active channel [2000 H]
	ME_SET	Message Error bit in the RT Status word [1000 H]
	BAD_STATUS	Bit other than ME set in 1553 Status word [0800 H]
	INVALID_MSG	Word count or Sync error occurred [0400 H]
	LATE_RESP	RT responded late [0200 H]
	BAD_HEADER_BC	Error in Header Word [0100 H] (for 1760 only)
	INVALID_WORD	Bad bit count or Manchester error or parity error [0080 H]
	WD_CNT_HI	RT transmitted too many words [0040 H]
	WD_CNT_LO	RT transmitted too few words [0020 H]
	BAD_RT_ADDR	1553 Status word contained wrong RT address [0010 H]
	BAD_SYNC	Status or Data sync were wrong [0008 H]
	BAD_GAP	Invalid Gap time occurred between words [0004 H]
	MSG_ERROR	Error occurred; defined in other flags [0001 H]
unsigned short int words [36]	A pointer to an array of 1553 words.	

}

Get_BC_Msgentry_Px (cont.)

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If not in BC/Concurrent-RT mode
einval	If an invalid parameter was used as an input
frm_erange	If `frameid` is not a valid frame id
frm_erangecnt	If msgentry is greater than the number of messages in the frame
0	If successful

Note: The function can be used instead of separately calling Get_Msgentry_Status_Px for the Status word, and then calling Read_Message_Px that returns just the content of the message.

Do *not* call Get_Msgentry_Status_Px before calling this function. An error message "enobcmsg" will be returned.

Get_BC_Status_Px

Description

Get_BC_Status_Px indicates the current status of the module. This function returns status flags generated since the last call to Reset_BC_Status_Px.

The function may be used for polling applications.

Note: Reset this status with the Reset_BC_Status_Px command.

Syntax

Get_BC_Status_Px (`int` handle)

Input Parameters

handle	The handle designated by Init_Module_Px

Output Parameters

none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px	
emode	If not in BC/Concurrent-RT mode	
If successful	0 or more of the following flags:	
	END_MINOR_FRAME	Minor Frame completed [0010 H]
	MSG_ERR	An error in some message [0008 H]
	END_OF_FRAME	Complete frame of messages sent [0004 H]

Get_BC_Status_Px (cont.)

	MSG_CMPLT	Message sent [0002 H]
	WAIT_FOR_CONTINUE	Message named by Set_Halt_Px was reached [0001 H]

Get_Minor_Frame_Time_Px

Description Get_Minor_Frame_Time_Px gets the previously stored minor frame time in microseconds. This value was set in a call to Set_Minor_Frame_Time_Px.

Note: This function is valid only in BC/Concurrent-RT mode

Syntax Get_Minor_Frame_Time_Px (`int` handle, `long` *frame_time)

Input Parameters	handle	The handle designated by Init_Module_Px
Output Parameters	frame_time	The minor frame time in microseconds
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	emode	If not in BC/Concurrent-RT mode
	0	If successful

Get_Msgentry_Status_Px

Description Get_Msgentry_Status_Px returns a Status word recorded by the Bus Controller associated with a selected message in a selected frame. If a valid message status is returned, the function clears the status.

Note: Get_Msgentry_Status_Px replaces the obsolete function Get_Msg_Status_Px. Get_Msgentry_Status_Px is more accurate when multiple frames are used.

Syntax Get_Msgentry_Status_Px (`int` handle, `int` frameid, `int` msgentry, `usint` *msgstatusword)

Input Parameters	handle	The handle designated by Init_Module_Px
	frameid	Frame identifier returned from a prior call to Create_Frame_Px
	msgentry	Entry within the frame, i.e., 0 for the first message in the frame, 1 for the second message, etc.
Output Parameters	msgstatusword	On success, 0 or more of the following flags
	END_OF_MSG	Complete 1553 message received [8000 H]

Get_Msgentry_Status_Px (cont.)

Note: If END_OF_MSG bit (0×8000) is *not* set, the remaining flags are not valid.

BC_BAD_CHECKSUM	Bad checksum on the transmit side [4000 H]
BAD_BUS	RT response received on non-active channel [2000 H]
ME_SET	Message Error bit in the RT Status word [1000 H]
BAD_STATUS	Bit other than ME set in 1553 Status word [0800 H]
INVALID_MSG	1553 message-level error occurred [0400 H]
LATE_RESP	RT responded late [0200 H]
BAD_HEADER_BC	Error in Header Word [0100 H] (for 1760 only)
INVALID_WORD	Bad bit count or Manchester error or parity error [0080 H]
WD_CNT_HI	RT transmitted too many words [0040 H]
WD_CNT_LO	RT transmitted too few words [0020 H]
BAD_RT_ADDR	1553 Status word contained wrong RT address [0010 H]
BAD_SYNC	Status or Data sync were wrong [0008 H]
BAD_GAP	Invalid Gap time occurred between words [0004 H]
MSG_ERROR	Error occurred; defined in other flags [0001 H]
0	If no message has been received since the previous call to this function.

Get_Msgentry_Status_Px (cont.)

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If not in BC/Concurrent-RT mode
einval	If an invalid parameter was used as an input
frm_erange	If called this function with an undefined frame
frm_erangecnt	If called to this function with a `msgentry` value greater than the number of messages in the frame
0	If successful

Get_Next_Message_BCM_Px

Description

Get_Next_Message_BCM_Px relates to the Internal Concurrent Monitor available in the memory of the module. The Internal Concurrent Monitor operates automatically when the module is started in BC/Concurrent-RT mode. All bus traffic in BC/Concurrent-RT mode is recorded in the Internal Concurrent Monitor memory. The Concurrent Monitor operates in sequential mode which means that the 1553 Message blocks are stored in sequential locations in memory, beginning from block 0.

The first call to the function Get_Next_Message_BCM_Px returns the contents of block 0. If the next block does not contain a new message, an error value will be returned.

Note:

- This function can only be used in BC/Concurrent-RT mode. See also Get_Next_Mon_Message_Px, which can be used in all modes.
- The structure returned by this function is similar to that returned by the Sequential Monitor mode function Get_Next_Message_Px. However, the bits used in the Message Status word of the Internal Concurrent Monitor are unique to this function – they are similar but not identical to the bits set in the Message Status word of Sequential Monitor mode and BC/Concurrent-RT mode and its Internal Concurrent Monitor.

  For each mode, verify the Message Status bits for the appropriate mode and function.
- If there is a user-initiated Time Tag reset (Reset_Time_Tag_Px, Clear_Timetag_Sync_Px, or SetIrig_4000 with the flag IRIG_TIME_RESET), Get_Next_Message_BCM_Px will interpret this as an overrun condition. To avoid this, call Ignore_Timetag_Overrun_Px. See Ignore_Timetag_Overrun_Px on page 4-18.

Syntax

Get_Next_Message_BCM_Px (`int` handle, `struct MONMSG` *msgptr, `usint` *msgcounter)

Input Parameters

handle — The handle designated by Init_Module_Px

Output Parameters

msgptr — Pointer to an address in host memory where you want a copy of the message to be placed. The message is returned in a `MONMSG` structure, as defined below.

Space should always be allocated for 36 words of data to accommodate the maximum case of RT-to-RT transmission of 32 Data Words plus 2 Status words and 2 Command words.

Get_Next_Message_BCM_Px (cont.)

struct MONMSG {
unsigned short int msgstatus;
Status Word containing one or more of the following flags:

END_OF_MSG	Indicates end of message [8000 H]
BUS_A	Indicates BUS A [4000 H]
BAD_CHECKSUM_CONCM	Checksum error [2000 H] (for 1760 only)
ME_SET	Message Error bit in the RT Status word [1000 H]
BAD_STATUS	Status Word bits set [0800 H]
INVALID_MSG	Invalid message; perhaps RT-to-RT with two receive Command words [0400 H]
MON_LATE_RESP	Response time error occurred in the message, even if no RT active on the module [0200 H]
BAD_HEADER_BCCM	Bad Header Word [0100 H] (for 1760 only)
INVALID_WORD	At least one invalid 1553 Word received [0080 H]
WD_CNT_HI	RT transmitted too many words [0040 H]
WD_CNT_LO	RT transmitted too few words [0020 H]
BAD_RT_ADDR	Received 1553 Status word did not contain the correct RT address [0010 H]
BAD_SYNC	Sync of either the Command or the Data Word(s) is incorrect [0008 H]
BAD_GAP	Invalid gap received between 1553 Words [0004 H]
RT2RT_MSG_CONCM	RT-to-RT message was received [0002 H]
MSG_ERROR	Error occurred; defined in other flags [0001 H]

unsigned int elapsedtime;	A 32-bit Time Tag; resolution is 4 microseconds
unsigned short int words [36];	A pointer to an array of 1553 words in the sequence they were received over the bus. See **Appendix A: MIL-STD-1553 Word Formats**.

}

msgCounter	The low 16 bits of the message number assigned to this message. The high 16 bits are stored in the Message Number Hi register [3EBC H].

Get_Next_Message_BCM_Px (cont.)

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module is not in RT mode
enomsg	If no new messages have been received
eoverrunTtag	If the Time Tag of the message was overwritten while the message was being read
eoverrunEOM flag	If the End Of Message flag was overwritten while the message was being read
block number	If successful. Valid values: 0 – 408

Note: For *M4K1553Px* modules rev A and rev A1, valid values are 0 – 128.

Insert_Msg_Err_Px

Description

Insert_Msg_Err_Px selects various errors to inject into a previously defined message. This function may also be called in realtime after a call to Run_BC_Px.

This function works in conjunction with several other functions. For detailed information on configuring error injection, see Error Injection on page 5-5.

Note:

- For RT-to-RT messages, error injection only applies to the first command.
- This function is related to error injection, and is not available for the single function module (*PxS*).

Syntax

Insert_Msg_Err_Px (`int` handle, `int` id, `int` flags)

Input Parameters

handle	The handle designated by Init_Module_Px
id	Message identifier returned from a prior call to Create_1553_Message_Px

flags

One or more of the following flags ORed together:

PARITY_ERR	Select parity error on 1553 words [4000 H]
WD_CNT_ERR	Inserts bad word count based on Set_Word_Cnt_Px [1000 H]
BIT_CNT_ERR	Inserts bad bit count [0800 H]
SYNC_ERR	Data type sync is inserted in Command word [0400 H]
GAP_ERR	Invalid gap time between Command and Data Words [0300 H]

Insert_Msg_Err_Px (cont.)

	DATA_ERR	For BC-to-RT commands the SYNC_ERR, PARITY_ERR and BIT_COUNT_ERR may be accompanied by this flag to cause the error to be inserted in the Data Words rather than the Command word [0100 H]
	NO_ERR_IN_MSG	Disables error injection in this message [0000 H]

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If not in BC/Concurrent-RT mode
ebadid	If ID is not a valid message ID
enotforsinglefuncerror	If this function is called with a single function module (*PxS*)
0	If successful

Re_Create_Message_Px

Description Re_Create_Message_Px enables the user to alter all the parameters of a message for a particular message ID that has already been allocated. Re_Create_Message_Px has the same parameters as Create_1553_Message_Px except that the id is an input instead of an output.

To change a message, the user must give the message's message ID. The total message size must not exceed the message size of the one allocated for the original message that was created by Create_1553_Message_Px.

Note: See **Appendix A: MIL-STD-1553 Word Formats**.

Syntax Re_Create_Message_Px (`int` handle, `usint` cmdtype, `usint` data[], `short int` id)

Input Parameters

handle	The handle designated by Init_Module_Px	
cmdtype	One of the following flags:	
	RT2BC	Send a transmit message [0000 H]
	BC2RT	Send a Receive message [0001 H]
	RT2RT	Send an RT-to-RT transfer message [0002 H]

Re_Create_Message_Px (cont.)

		MODE	Mode Command [0003 H]
		BRD_RCV	Broadcast Receive message [0004 H]
		BRD_RT2RT	Broadcast RT-to-RT transfer message [0005 H]
		BRD_MODE	Broadcast Mode Command [0006 H]
		MINOR_FRAME	Minor frame message/ delay [000F H]
	data []	A pointer to an array of command + data (and Status words if RT is simulated by this module)	
	id	Message identifier returned by Create_1553_Message_Px	
Output Parameters	none		
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px	
	emode	If not in BC/Concurrent-RT mode	
	ebadid	If ID is not a valid message ID	
	einval	If parameter is out of range	
	ebadcommandword	If the T/R bit in the Command word is set incorrectly for the given cmdtype parameter	
	msg2big	If attempted to create a message with too many words	
	0	If successful	

Read_Message_Px

Description Read_Message_Px allows the user to read back data associated with a given message. This is used for reading data returned from a Transmit command or for reading the 1553 Status word returned by an RT for all message types. The output consists of the entire message in the same sequence as it appeared on the bus including Command words, Status words and data.

Note: See **Appendix A: MIL-STD-1553 Word Formats.**

Syntax Read_Message_Px (`int` handle, `int` id, `usint` *words)

Input Parameters	handle	The handle designated by Init_Module_Px
	id	Message identifier returned from a prior call to Create_1553_Message_Px
Output Parameters	words	An array of up to 36 words containing Command, Status and Data Words associated with the message ID
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	emode	If not in BC/Concurrent-RT mode
	ebadid	If ID is not a valid message ID
	eminorframe	If message is a Minor Frame type
	0	If successful

Read_SRQ_Message_Px

Description Read_SRQ_Message_Px allows the user to read back the Command and Data Words associated with the last SRQ message

Syntax Read_SRQ_Message_Px (`int` handle, `usint` *vector_status, `usint` *msg_status, `usint` *data, `usint` *srq_counter)

Input Parameters	handle	The handle designated by Init_Module_Px
Output Parameters	vector_status	Status word of the Mode Code Get Vector Command
	msg_status	Status of the BC-to-RT message
		Note: For msg_status/vector_status flag values, see Get_Msgentry_Status_Px on page 5-20.
	data	Pointer to an array of up to 36 words associated with the SRQ message
	srq_counter	The number of times an SRQ message came in

Read_SRQ_Message_Px (cont.)

Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	emode	If not in BC/Concurrent-RT mode
	enosrq	If SRQ was disabled by the user See Enable_SRQ_Support_Px on page 5-17
	0	If successful

Reset_BC_Status_Px

Description	Reset_BC_Status_Px clears the BC status so that the next call to Get_BC_Status_Px is meaningful.	
Syntax	Reset_BC_Status_Px (`int` handle)	
Input Parameters	handle	The handle designated by Init_Module_Px
Output Parameters	none	
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	emode	If not in BC/Concurrent-RT mode
	0	If successful

Run_BC_Px

Description	Run_BC_Px causes the Bus Controller to start transmitting messages. All data structures must be set up before calling this function.		
Syntax	Run_BC_Px (`int` handle, `int` type)		
	handle	The handle designated by Init_Module_Px	
	type	0	Continuous operation
		1 – 255	The number of times to execute the current frame
Output Parameters	none		
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px	
	emode	If not in BC/Concurrent-RT mode	
	bcr_erange	If called with type greater than 255	
	etimeout	If timed out waiting for BOARD_HALTED	
	0	If successful	

Select_Async_Frame_Px

Description

With multiple calls to Create_Frame_Px the user can create multiple frames. Select_Async_Frame_Px selects a frame to be set up for asynchronous transmission.

Note: Always call Select_Async_Frame_Px *before* Send_Async_-Frame_Px.

Syntax

Select_Async_Frame_Px (`int` handle, `int` frameid, `int` nummsgs)

Input Parameters

handle	The handle designated by Init_Module_Px
frameid	Frame identifier returned from a prior call to Create_Frame_Px
nummsgs	Number of messages within the frame to execute *or* FULLFRAME Execute the entire frame [0000 H]

Output Parameters

none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
frm_erange	If called this function with an undefined frame
frm_erangecnt	If called to this function with a nummsgs value greater than the number of messages in the frame
0	If successful

Send_Async_Frame_Px

Description

Send_Async_Frame_Px causes the asynchronous frame that was set up by a call to Select_Async_Frame_Px to be sent after the current message has completed its transmission.

Note: Always call Select_Async_Frame_Px *before* Send_Async_-Frame_Px.

Syntax

Send_Async_Frame_Px (`int` handle)

Input Parameters

handle	The handle designated by Init_Module_Px

Output Parameters

none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
enoasync	If there are not enough messages in async frame
0	If successful

Send_Msg_Once_Px

Description

Send_Msg_Once_Px sends a message once, asynchronously, while in Frequency mode. The message ID should be created, but should not be added to the FRAME structure. This function is for messages that were set to be skipped by calling Set_Skip_Px.

Note: This function is only available when the module supports Frequency mode, and Frequency mode was activated by calling Set_Frequency_Mode_Px.

Syntax

Send_Msg_Once_Px (`int` handle, `int` id, `int` msgtype)

Input Parameters

handle	The handle designated by Init_Module_Px

Output Parameters

none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If not in BC/Concurrent-RT mode
efrequency modemustbeon	If Frequency mode was not turned ON; call Set_Frequency_Mode_Px first
ebadid	If ID is not a valid message ID
einval	In an invalid msgtype
enoskip	If tried to restore a message that was not set to be skipped
0	If successful

Send_Timetag_Sync_Px

Description

Send_Timetag_Sync_Px sets the BC to send the current Time Tag value as the Data Word in a Mode Code Synchronize with Data message (Mode Code 17). The Time Tag is sent out as a single 16-bit Data Word, with a resolution of 64 µsec.

Note:

- When this feature is disabled, the BC sends the Data Word as defined in the message.
- Call this function only when the module is stopped.

Syntax

Send_Timetag_Sync_Px (`int` handle, `int` flag)

Input Parameters

handle	The handle designated by Init_Module_Px	
flag	ENABLE	Send Time Tag as data for Mode Code 17 [0001 H]
	DISABLE	Do *not* send Time Tag as data for Mode Code 17 [0000 H]

Output Parameters

none

Send_Timetag_Sync_Px (cont.)

Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	emode	If not in BC/Concurrent-RT mode
	einval	If parameter is out of range
	0	If successful

Set_BC_Resp_Px

Description	Set_BC_Resp_Px sets the amount of time the BC will wait for a response before declaring the RT late.	
Syntax	Set_BC_Resp_Px (`int` handle, `int` time)	
	handle	The handle designated by Init_Module_Px
	time	Time in nanoseconds Valid range: 2000 – 32000
Output Parameters	none	
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	emode	If not in BC/Concurrent-RT mode
	einval	If an invalid parameter was used as an input
	eresptimeerror	If the single function module's response time is greater than 12,000 nanoseconds (for *PxS* only)
	0	If successful

Set_Bus_Px

Description	Set_Bus_Px sets the bus over which a particular message will be transmitted.		
Syntax	Set_Bus_Px (`int` handle, `int` id, `int` bus)		
Input Parameters	handle	The handle designated by Init_Module_Px	
	id	Message identifier returned from a prior call to Create_1553_Message_Px	
	bus	XMT_BUS_A	Transmit message over Bus A [0080 H]
		XMT_BUS_B	Transmit message over Bus B [0000 H]
Output Parameters	none		
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px	
	emode	If not in BC/Concurrent-RT mode	

Set_Bus_Px (cont.)

ebadid	If ID is not a valid message ID
ebadchan	If tried to set bus to illegal value
0	If successful

Set_Continue_Px

Description Set_Continue_Px restarts transmission that has been halted after a call to Set_Halt_Px.

Syntax Set_Continue_Px (`int` handle, `int` id)

Input Parameters

handle	The handle designated by Init_Module_Px
id	Message identifier returned from a prior call to Create_1553_Message_Px

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If not in BC/Concurrent-RT mode
ebadid	If ID is not a valid message ID
0	If successful

Set_Error_Location_Px

Description Set_Error_Location_Px sets the location of errors injected into messages. This function works in conjunction with Insert_Msg_Err_Px. See Insert_Msg_Err_Px on page 5-25.

This function works in conjunction with other error injection functions. For detailed information on configuring error injection, see Error Injection on page 5-5.

Note: This function is related to error injection, and is not available for the single function module (*PxS*).

Syntax Set_Error_Location_Px (`int` handle, `unsigned int` errorWord, `unsigned int` zcerrorBit)

Input Parameters

handle	The handle returned by Init_Module_Px
errorWord	The index of the Data Word into which to inject the error. This is used for Data Word errors only. 0 represents the first Data Word, 1 the second, etc.

Set_Error_Location_Px (cont.)

	zcerrorBit	The index of the bit into which to inject a zero crossing error. 0 represents the high bit, (i.e. the first bit to be transmitted after the sync) 1 the next, etc. The Set_Zero_Cross_Px function uses this parameter for location of the error. See Set_Zero_Cross_Px on page 5-44.
Output Parameters	none	
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	func_invalid	If the function is not supported by this version of the firmware
	emode	If the module is not in BC/ Concurrent RT mode
	einval	If errorWord is greater than 31 or zcerrorBit is greater than 15
	enotforsinglefuncerror	If this function is called with a single function module (*PxS*)
	0	If successful

Set_Frame_Time_Px

Description	Set_Frame_Time_Px determines the time each frame will take when the frame is to be executed more than once. (See Run_BC_Px on page 5-29). This value is the minimum time in microseconds from the start of the frame to the start of the next repetition of the frame.	
Syntax	Set_Frame_Time_Px (`int` handle, `long` time)	
Input Parameters	handle	The handle designated by Init_Module_Px
	time	Frame time in microseconds
Output Parameters	none	
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	emode	If not in BC/Concurrent-RT mode
	0	If successful

Set_Frequency_Mode_Px

Description Set_Frequency_Mode_Px activates Frequency mode whereby messages in the bus list are transmitted at a specified frequency (in microseconds). The frequency of each message in the bus list is assigned to the gaptime field in the FRAME structure. After activating Frequency mode, you can call Send_Msg_Once_Px to transmit a single message asynchronously.

Syntax Set_Frequency_Mode_Px (`int` handle, `int` enableflag)

Input Parameters

handle	The handle designated by Init_Module_Px
enableflag	ENABLE Enable Frequency mode [0001 H]
	DISABLE Disable Frequency mode [0000 H]

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
func_invalid	If the module is a *MMSI* module
einval	If the enableflag value was invalid
emode	If not in BC/Concurrent-RT mode
efrequencymode notsupported	If Frequency mode is not supported by the module's firmware
0	If successful

Set_Halt_Px

Description Set_Halt_Px halts transmission upon reaching this message. Transmission will begin again after Set_Continue_Px is called.

Syntax Set_Halt_Px (`int` handle, `int` id)

Input Parameters

handle	The handle designated by Init_Module_Px
id	Message identifier returned from a prior call to Create_1553_Message_Px

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If not in BC/Concurrent-RT mode
ebadid	If ID is not a valid message ID
0	If successful

Set_Interrupt_On_Msg_Px

Description

Set_Interrupt_On_Msg_Px causes an interrupt to be generated when a chosen message is generated.

Syntax

Set_Interrupt_On_Msg_Px (`int` handle, `int` frameid, `int` msgentry, `int` enable)

Input Parameters

handle	The handle designated by Init_Module_Px	
frameid	frame_id returned from a call to Create_Frame_Px	
msgentry	Message number within the frame	
enable	ENABLE	Generate an interrupt [0001 H]
	DISABLE	Do *not* generate an interrupt [0000 H]

Output Parameters

none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
frm_erange	If frameid is not a valid frame id
frm_erangecnt	If msgentry is greater than the number of messages in the frame
emode	If not in BC/Concurrent-RT mode
0	If successful

Set_Jump_Px

Description

Set_Jump_Px sets the module to jump to a new frame when the module gets to the selected message.

Note: The original message denoted by the id is destroyed by this function.

Syntax

Set_Jump_Px (`int` handle, `int` id, `int` frameid, `int` intcnt)

Input Parameters

handle	The handle designated by Init_Module_Px
id	Message identifier returned from a prior call to Create_1553_Message_Px for jumping message
frameid	Frame identifier returned from a prior call to Create_Frame_Px for frame to jump to
intcnt	Number of instructions in new frame to execute *or* FULLFRAME Execute all instructions in frame [0000H]

Output Parameters

none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If not in BC/Concurrent-RT mode
ebadid	If ID is not a valid message ID
enoalter	If the message is currently being sent, the jump cannot be set; try again
frm_erange	If frameid is not a valid frame id
frm_erangecnt	If intcnt is greater than the number of messages in the frame
0	If successful

Set_Minor_Frame_Time_Px

Description Set_Minor_Frame_Time_Px sets time for a Minor Frame message type. This message functions as a "delay time" message. The time given is from the beginning of a Minor Frame to the beginning of the next Minor Frame.

Syntax Set_Minor_Frame_Time_Px (`int` handle, `long` micro_second)

Input Parameters

handle	The handle designated by Init_Module_Px
micro_second	Minor Frame time in microseconds; maximum value allowed is 800,000 microseconds (800 milliseconds)

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If not in BC/Concurrent-RT mode
0	If successful

Set_Replay_Px

Description Set_Replay_Px sets the module in replay mode. When replay mode is enabled, a message will be sent out when the Time Tag equals the message send time (mst) as specified in that msgentry within the frame.

Syntax Set_Replay_Px (`int` handle, `int` flag)

Input Parameters

handle	The handle designated by Init_Module_Px	
flag	ENABLE	Enable replay mode [0001 H]
	DISABLE	Disable replay mode [0000 H]

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If not in BC/Concurrent-RT mode
einval	If a parameter is out of range
0	If successful

Set_Restore_Px

Description Set_Restore_Px restores a message, that was set to be skipped, to be sent as usual.

Syntax Set_Restore_Px (`int` handle, `int` id, `int` msgtype)

Input Parameters

handle	The handle designated by Init_Module_Px	
id	Message identifier returned from a prior call to Create_1553_Message_Px	
msgtype	One of the following:	
	RT2BC	Send a transmit message [0000 H]
	BC2RT	Send a Receive message [0001 H]
	RT2RT	Send an RT to RT transfer message [0002 H]
	MODE	Mode Command [0003 H]
	BRD_RCV	Broadcast Receive message [0004 H]
	BRD_RT2RT	Broadcast RT to RT transfer message [0005 H]
	BRD_MODE	Broadcast Mode Command [0006 H]
	MINOR_FRAME	Minor frame message/ delay [000F H]

Note: The msgtype must be the same as was used in Create_1553_Message_Px. Set_Skip_Px destroys this information, therefore Set_Restore_Px must supply it again.

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If not in BC/Concurrent-RT mode
ebadid	If ID is not a valid message ID
einval	If an invalid parameter was used as an input
enoskip	If tried to restore a message that was not set to be skipped
0	If successful

Set_Retry_Px

Description Set_Retry_Px selects the number and type of retries to attempt in the event of an error.

Syntax Set_Retry_Px (`int` handle, `int` id, `int` num, `int` altbus)

Input Parameters

handle	The handle designated by Init_Module_Px
id	Message identifier returned from a prior call to Create_1553_Message_Px
num	Number of retries to attempt: 1 – 3 Number of valid retries 0 No retries
altbus	Whether to retry on the initial bus or to alternate buses for each retry. RETRY_SAME_BUS Retries should all be on the original bus [0000 H] RETRY_ALT_BUS Retries should alternate between buses [0040 H]

Output Parameters none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If not in BC/Concurrent-RT mode
ebadid	If ID is not a valid message ID
einval	If an invalid parameter was used as an input
0	If successful

Set_RT_Nonactive_Px

Description Set_RT_Nonactive_Px sets a particular RT address nonactive, thereby turning off simulation for the specified RT. This function may be called in BC/Concurrent-RT mode to turn off Concurrent-RT simulation.

Note: This function is not available for the single function module (*PxS*) in BC mode. When using a single function (*PxS*) module in BC mode, you cannot use the module as a Concurrent-RT.

Syntax Set_RT_Nonactive_Px (`int` handle, `int` rtnum)

Input Parameters

handle	The handle designated by Init_Module_Px
rtnum	Address of the RT Valid values: 0 – 31

Set_RT_Nonactive_Px (cont.)

Output Parameters	none	
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	emode	If module not in BC/Concurrent-RT mode
	enotforsinglefuncerror	If this function is called with a single function module (*PxS*)
	0	If successful

Set_Skip_Px

Description	Set_Skip_Px causes a message within a specified frame to be skipped. The message can be restored later by calling Set_Restore_Px.	
Syntax	Set_Skip_Px (`int` handle, `int` id)	
Input Parameters	handle	The handle designated by Init_Module_Px
	id	Message identifier returned from a prior call to Create_1553_Message_Px
Output Parameters	none	
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	emode	If not in BC/Concurrent-RT mode
	ebadid	If ID is not a valid message ID
	0	If successful

Set_Stop_On_Error_Px

Description	Set_Stop_On_Error_Px sets the module to halt transmission upon reaching a message for which this function has been called and which returns an error status after all retry attempts.	
Syntax	Set_Stop_On_Error_Px (`int` handle, `int` id)	
Input Parameters	handle	The handle designated by Init_Module_Px
	id	Message identifier returned from a prior call to Create_1553_Message_Px
Output Parameters	none	

Set_Stop_On_Error_Px (cont.)

Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	emode	If module not in BC/Concurrent-RT mode
	ebadid	If ID is not a valid message ID
	0	If successful

Set_Sync_Pattern_Px

Description	Set_Sync_Pattern_Px sets the sync pattern to be used when a Sync error is injected into a message. This function works in conjunction with other error injection functions. For detailed information on configuring error injection, see Error Injection on page 5-5.	
Syntax	Set_Sync_Pattern_Px (`int` handle, `int` pattern)	
Input Parameters	handle	The handle returned by Init_Module_Px
	pattern	Six bits (in the low six bits of this register) each representing a half bit time of the desired register. For example, a value 0023 H (100011 binary) would be sent as one bit times high, three bit times low and two bit times high.
Output Parameters	none	
Return Values	ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
	func_invalid	If the function is not supported by this version of the firmware
	emode	If module is not in BC/Concurrent-RT mode
	einval	If pattern contains a set bit above the low six bits
	0	If successful

Set_Word_Cnt_Px

Description

Set_Word_Cnt_Px sets the number of words to send for messages that have requested WD_CNT_ERR errors. The number sent will be the actual number of words in the message + offset.

This function works in conjunction with other error injection functions. For detailed information on configuring error injection, see Error Injection on page 5-5.

Note: This function is related to error injection, and is not available for the single function module (*PxS*).

Syntax

Set_Word_Cnt_Px (`int` handle, `int` offset)

Input Parameters

handle	The handle designated by Init_Module_Px
offset	Offset from correct word count Valid values: -3 to +3

Output Parameters

none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
emode	If module not in BC/Concurrent-RT mode
einval	If an invalid parameter was used as an input
enotforsinglefuncerror	If this function is called with a single function module (*PxS*)
0	If successful

Set_Zero_Cross_Px

Description

Set_Zero_Cross_Px sets the type of zero crossing to be used when a zero crossing error is injected into a message.

This function works in conjunction with other error injection functions. For detailed information on configuring error injection, see Error Injection on page 5-5.

Note: This function is related to error injection, and is not available for the single function module (*PxS*).

Syntax

Set_Zero_Cross_Px (`int` handle, `unsigned int` zctype)

Input parameters

handle	The handle returned by Init_Module_Px	
zctype	ZC_NO_ERROR	No error [0000 H]
	ZC_LATE100	Zero cross 100 nanoseconds late (legal) [0001 H]
	ZC_LATE150	Zero cross 150 nanoseconds late (illegal) [0002 H]
	ZC_LATE200	Zero cross 200 nanoseconds late (illegal) [0003 H]
	ZC_EARLY100	Zero cross 100 nanoseconds early (legal) [0004 H]
	ZC_EARLY150	Zero cross 150 nanoseconds early (illegal) [0005 H]
	ZC_EARLY200	Zero cross 200 nanoseconds early (illegal) [0006 H]
	ZC_STUCK_HI	No zero cross, full bit hi [0008 H]
	ZC_STUCK_LO	No zero cross, full bit low [0009 H]
	ZC_STUCK_DEAD	No zero cross, full bit dead (center) [000A H]

Output parameters

none

Set_Zero_Cross_Px (cont.)

Return values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
func_invalid	If the function is not supported by this version of the firmware
emode	If the module is not in BC/ Concurrent RT mode
einval	If zctype is not one of the flags listed
enotforsinglefuncerror	If this function is called with a single function module (*PxS*)
0	If successful

Start_Frame_Px

Description

Start_Frame_Px selects which of the multiple frames created by calls to Create_Frame_Px are to be run when Run_BC_Px is called.

Note: This function must be called prior to each call on Run_BC_Px. If the module is stopped, either by calling Stop_Px, or because all requested messages have been transmitted, the module may be restarted by calling Start_Frame_Px and then Run_BC_Px

Syntax

Start_Frame_Px (`int` handle, `int` frameid, `int` nummsgs)

Input Parameters

handle	The handle designated by Init_Module_Px
frameid	Frame identifier returned from a prior call to Create_Frame_Px
nummsgs	Number of messages within the frame to execute *or* FULLFRAME Execute the entire frame [0000 H]

Output Parameters

none

Return Values

ebadhandle	If an invalid handle was specified; should be value returned by Init_Module_Px
frm_erange	If frameid is not a valid frame id
frm_erangecnt	If intcnt is greater than the number of messages in the frame
0	If successful

Appendix A MIL-STD-1553 Word Formats

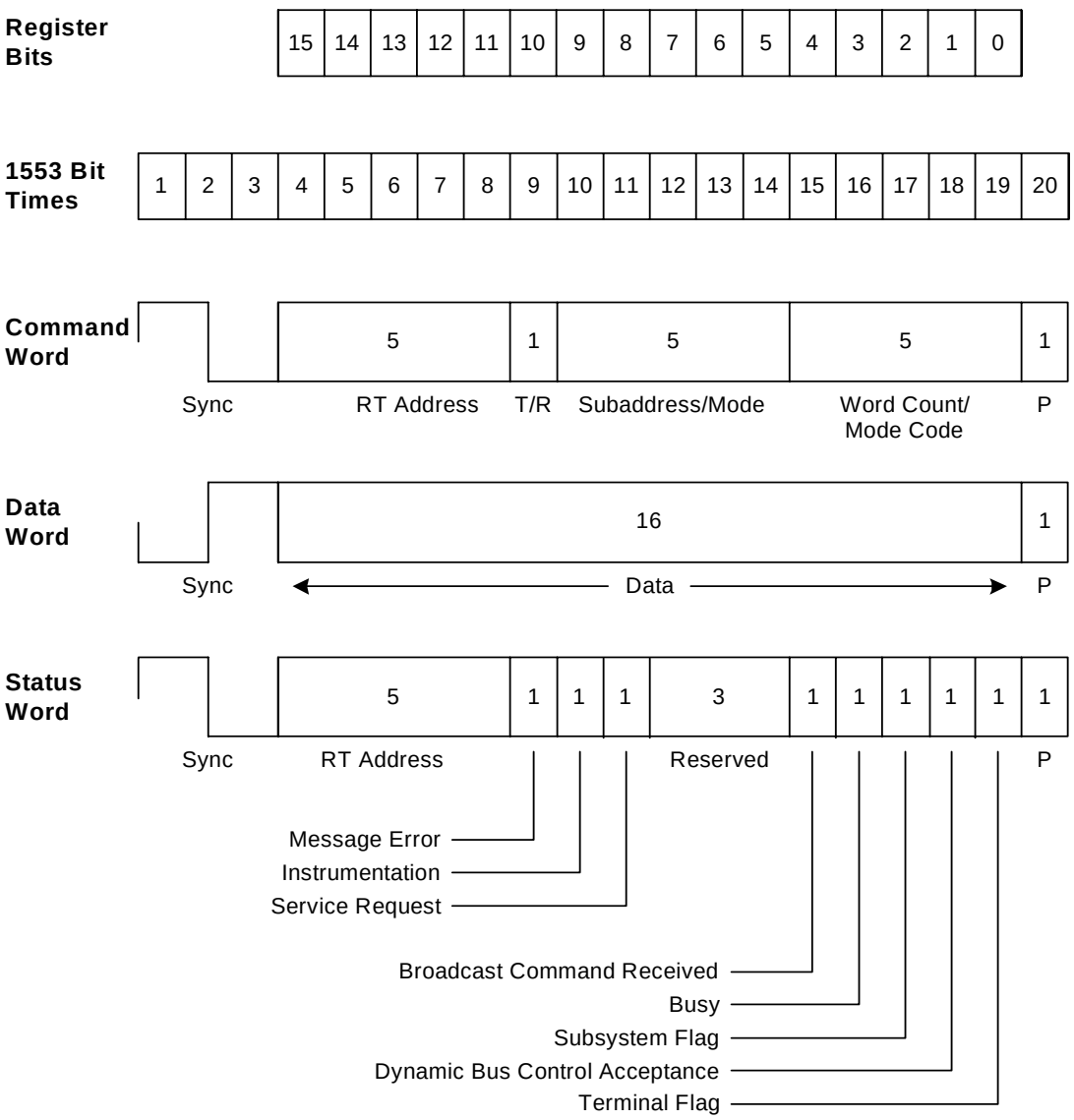


Figure A-1 MIL-STD-1553 Word Formats

Note: T/R = Transmit/Receive
P = Parity

Appendix B MIL-STD-1553 Message Formats

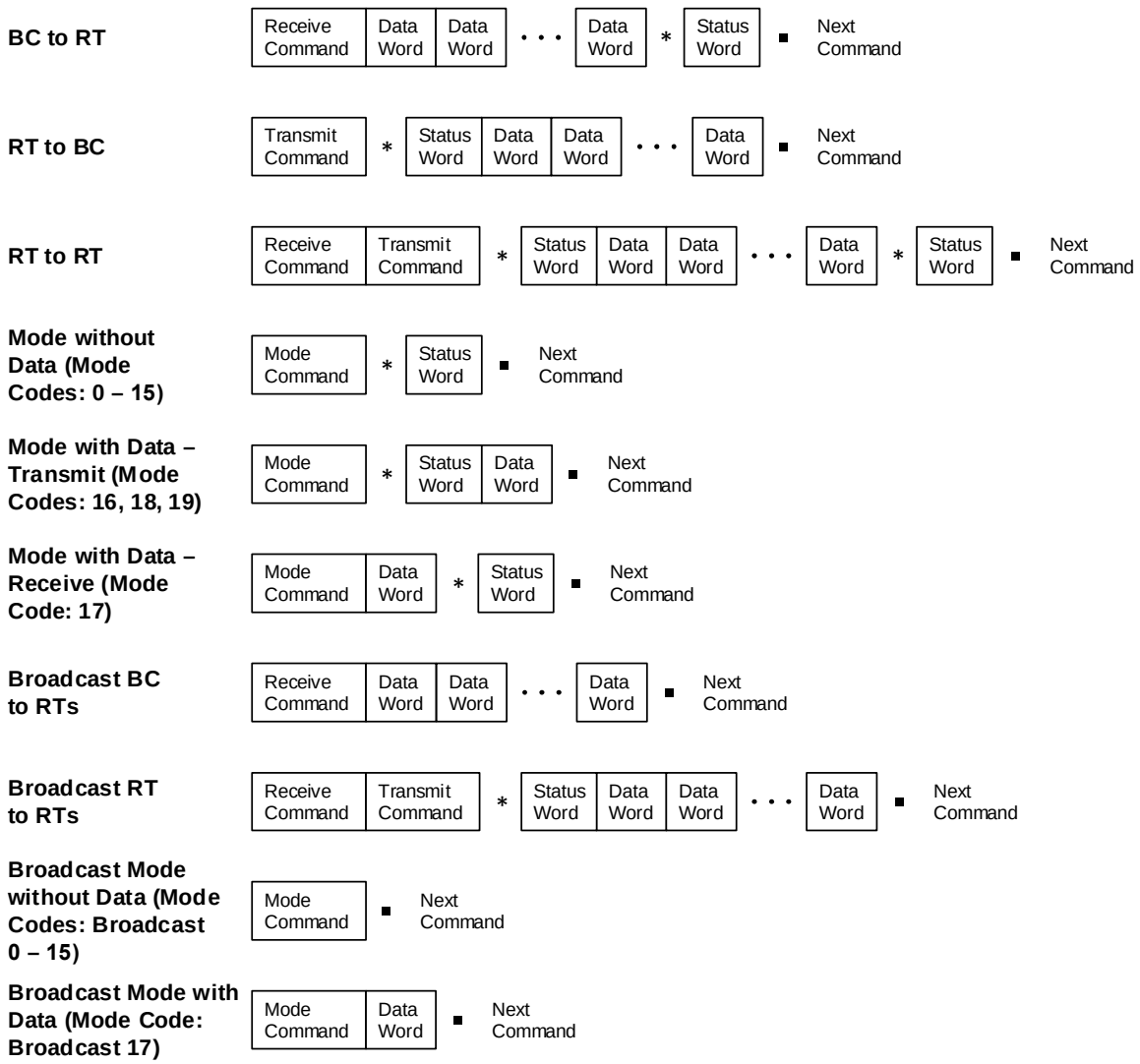


Figure B-1 MIL-STD-1553B Message Formats

Note: * = Response time
▪ = Intermessage Gap time

Appendix C Internal Loopback Test

The Internal Loopback Test is used to check the 1553 front-end logic, excluding transceivers and coupling transformers.

Note:

- The Internal Loopback Test can be performed with no cables attached to the module.
- When running a loopback, all values previously written to registers and memory addresses are erased.

To run an Internal Loopback Test, use Internal_Loopback_Px on page 2-22. The results of this test are returned to the host in dual-port RAM using the following structure beginning at address 0:

	Definition	Address in Dual-Port RAM	Status Value
{struct I_LOOPBACK			
usint frame_val;		0	X (not for user)
usint frame_status;	frame time counter status	2	8000H passed, 8001H failed
usint resp_status;	response time counter status	4	8000H passed, 8001H failed
usint early_val;		6	6 LSB must be 15H
usint receive_data1;	first looped word test, using command sync	8	5555H
usint status_1;		A	8000H passed, else failed
usint receive_data2;	second looped word test, using data sync	C	AAAAH
usint status_2;		E	8000H passed, else failed
usint mc_status;	mode code function test	10	8000H passed, else failed
usint ttag_val_lo;		12	30D4H ± 2
usint ttag_val_hi;		14	0
usint ttag_status;	time tag status	16	8000H passed, 8001H failed
usint prl;		18	(The CPU version)
} *I_loopback;			

Appendix D External and OnBoard Loopback Tests

The External Loopback Test is used to check the 1553 transceivers, transformers and associated bus cables. The External Loopback Test requires a special loopback cable connection that connects bus A to bus B, which is not compatible with standard 1553 communication. See Figure D-1 . To run an External Loopback Test, use **External_Loopback_Px** on page 2-5.

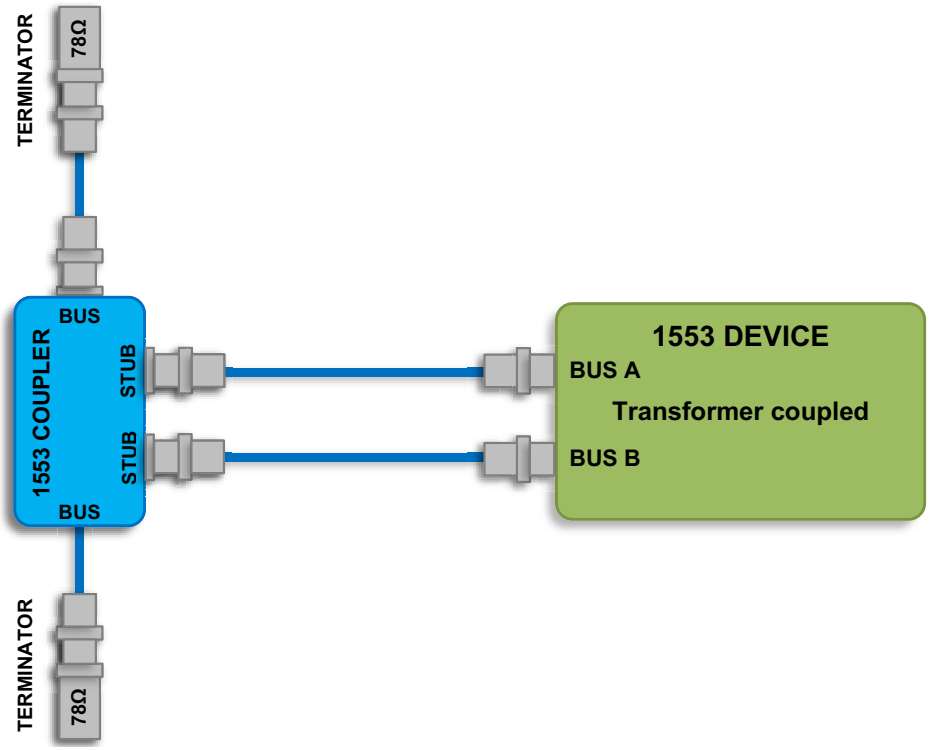


Figure D-1 External Loopback Test

Alternatively, on -LB modules you can activate an Onboard Loopback. When using an Onboard Loopback, the signals are looped back before the I/O connectors. Therefore, when an Onboard Loopback is activated, the external wiring and the I/O connector itself are not tested. To run an Onboard Loopback Test, use **OnBoard_Loopback_Px** on page 2-26.

Note: When running a loopback, all values previously written to registers and memory addresses are erased.

The results of the test are returned to the host in dual-port RAM using the following structure beginning at address 0:

	Definition/conditions for passing E_loopback test	Address in Dual-Port RAM	Status Value
{struct E_LOOPBACK			
usint frame_val;		0	X (not for user)
usint frame_status;	frame time counter status	2	8000H passed, 8001H failed
usint cmd_send[8];		4	cmd_send[0]: 5555H
	TX-A, RX-B command sync	6	cmd_send[1]:8000H passed, else failed
		8	cmd_send[2]: 1234H
	TX-A, RX-B data sync	A	cmd_send[3]:8000H passed, else failed
		C	cmd_send[4]: 5555H
	TX-B, RX-A command sync	E	cmd_send[5]:8000H passed, else failed
		10	cmd_send[6]: 1234H
	TX-B, RX-A data sync	12	cmd_send[7]:8000H passed, else failed
usint ttag_val_lo		14	30D4H ± 2
usint ttag_val_hi		16	0
usint ttag_status;	time tag status	18	8000H passed, 8001H failed
} *E_loopback;			

Note: For more information on External Loopbacks, see **Appendix E Application of External Loopback Test** in the *M8K1553Px Module User's Manual* or the *M4K1553Px Module User's Manual*.

Appendix E Source Code References

Appendix E-1 *1553Px Software Tools* Library

Appendix E-1 includes a list of the files in the *1553Px Software Tools* needed to write user-defined applications. The files are divided into three categories:

Source code and Header files for the *1553Px Software Tools* functions. Header files should be included in application programs as needed.

File Extension	Description
***.c**	source code
***.h**	header file

DLL and associated *.lib files

File Extension	Description
***MS.dll**	Microsoft compiled DLL
***MS.lib**	Index file used to create applications using Microsoft DLL functions

Demo Programs are examples of programs using the *1553Px Software Tools*, which can be used as a basis for user-defined programs. Demo programs include the following types of files:

File Extension	Description
***.c, *.h**	Demo source code
***.exe**	Demo executable files
***.sln** ***.suo**	Microsoft project solution files

	File Name	Description
Source Code files	**bcget.c**	Functions to retrieve information in BC mode
	bcrtmsg.c	Functions for BC/Concurrent-RT mode
	bcframe.c	Functions to set up frame identifiers in BC mode
	bcset.c	Functions for setting up the module in BC mode
	deviceio_px.c	Functions for interacting with the Excalibur kernel drivers for all Windows operating systems
	deviceio.c	Functions relating to resources such as memory and interrupts – these files act on kernel drivers
	error_px.c	Function for returning error messages
	gget.c	Functions for retrieving information from all modes
	gset.c	Functions for setting up the module for non mode-specific values
	initcard.c	Initialization and Release resources functions
	mon.c	Functions for Monitor mode
	mon_lkup.c	Functions for Monitor Lookup mode
	monseq.c	Functions for Monitor Sequential mode
	rtset.c	Functions for RT mode
Source Header files	**bcrun.h**	Header file for BC mode functions
	deviceio.h	Header file for interaction with kernel driver
	error_px.h	Header file containing error message codes
	error_devio.h	Header file containing error codes for the Excalibur module kernel drivers
	excsysio.h	Header file which defines shared codes and structures between DLL and kernel driver
	exdef.h	Header file to differentiate between different Excalibur products.
	exc4000.h	Header file for EXC-4000/8000 family board level functions
	galahad.h	Header file containing all other header files
	flags_px.h	Header file containing flags for all modes
	instance_px.h	Header file for global module structure
	monlkup.h	Header file for Monitor Lookup mode functions

	File Name	Description
	monseq.h	Header file for Monitor Sequential mode functions
	proto_px.h	Header file containing prototypes of all functions
	rtrun.h	Header file for RT mode functions
	pxincl.h	Header file, include file for the *Px* DLL code, not for applications
Demo Programs	**demo_async.c**	Demo program to test asynchronous functions
	demo_bc1.c	Demo programs to test basic BC functions
	demo_bc2.c	Demo program to test advanced BC functions
	demo_2md.c	Multi-mode demo program to test Message contents and status
	demo_2crd.c	Demo program runs a BC on one module which sends messages and monitors those messages on the second module. Both modules can be on different EXC-4000/8000 family boards or on the same one.
	demo_bcmon.c	Demo program to test BC/ Internal Concurrent Monitor
	demo_int.c	Demo program to test interrupts
	demo_mon.c	Demo program to test Bus Monitor functions
	demo_loopback.c	Demo program to test External and Internal Loopback functions
	demo_rt.c	Demo program to test RT functions
	demo_minor.c	Demo program to test the use of Minor Frames
DLL and LIB files	The DLL and LIB files have the same filename except for the file extension. The module level DLL is required for the removable *Px* module, and for any board that has an integrated *Px* module. The filename is: **Exc1553PxMs**. In addition, a board level DLL accompanies the module for each of the boards and cards. The filename is: **Exc4000Ms**.	

Appendix E-2 Code Index

The *1553Px Software Tools* are C language functions designed to aid programmers of the *Px* module to write test applications. Below is an alphabetical list of all the functions and the name of the *1553Px Software Tools* file containing the programming code. The extension of all the files is *.c.

Functions	Code File Name
Alter_Cmd_Px	**bcrtmsg.c**
Alter_IMG_Px	**bcframe.c**
Alter_Message_Px	**bcrtmsg.c**
Alter_MsgSendTime_Px	**bcframe.c**
Assign_Blk_Px	**monlkup.c**
Assign_DB_Datablk_Px	**rtset.c**
Assign_RT_Data_Px	**rtset.c**
BC_Check_Alter_Msgentry_Px	**bcrtmsg.c**
Clear_Card_Px	**bcset.c**
Clear_Frame_Px	**bcset.c**
Clear_Msg_Blks_Px	**mon.c**
Clear_RT_Sync_Entry_Px	**rtset.c**
Clear_Timetag_Sync_Px	**gset.c**
Command_Word_Px	**bcrtmsg.c**
Convert_IrigTimetag_BCD_to_Decimal_Px	**gget.c**
Create_1553_Message_Px	**bcrtmsg.c**
Create_Frame_Px	**bcframe.c**
DelayMicroSec_Px	**initcard.c**
Enable_1553A_Support_Px	**gset.c**
Enable_Checksum_Error_Px	**bcframe.c**
Enable_Checksum_Px	**bcframe.c**
Enable_Mon_1553A_Support_Px	**monseq.c**
Enable_Lkup_Int_Px	**monlkup.c**
Enable_SRQ_Support_Px	**bcset.c**
External_Loopback_Px	**gget.c**
Get_BC_Msgentry_Px	**bcrtmsg.c**
Get_BC_Msgentry_Px	**bcrtmsg.c**

Functions	Code File Name
Get_BC_Status_Px	**bcget.c**
Get_Blknum_Px	**rtset.c**
Get_Board_Options_Hi_Px	**gget.c**
Get_Board_Status_Px	**gget.c**
Get_Board_Status_Px	**gget.c**
Get_Board_Status_Px	**gget.c**
Get_Card_Type_Px	**gget.c**
Get_Checksum_Blocks_Px	**bcframe.c**
Enable_Mon_1553A_Support_Px	**monseq.c**
Get_DmaAvailable_Px	**initcard.c**
Get_Error_String_Px	**error.c**
Get_Header_Exists_Px	**gget.c**
Get_Header_Value_Px	**gget.c**
Get_Id_Px	**gget.c**
Get_Interrupt_Count_Px	**deviceio.c**
Get_Last_Blknum_Px	**monlkup.c**
Get_Message_Px	**mon.c**
Get_Message_Count_Px	**gget.c**
Get_Message_Info_Px	**monseq.c**
Get_Minor_Frame_Time_Px	**bcget.c**
Get_Mode_Px	**gget.c**
Get_ModuleIrigTime_Px	**gget.c**
Get_ModuleTime_Px	**gget.c**
Get_MON_Status_Px	**mon.c**
Get_Msgentry_Status_Px	**bcget.c**
Get_Multibuf_Nextbuf_Px	**rtset.c**
Get_Next_Message_BCM_Px	**bcrtmsg.c**
Get_Next_Message_Px	**monseq.c**
Get_Next_Message_RTM_Px	**rtset.c**
Get_Next_Mon_IRIGttag_Message_Px	**monseq.c**
Get_Next_Mon_Message_Px	**gget.c**

Functions	Code File Name
Get_Next_RT_Message_Px	**rtset.c**
Get_Rev_Level_Px	**gget.c**
Get_SerialNumber_Px	**gget.c**
GetTimeTag_Px	**initcard.c**
Get_UseDmaIfAvailable_Px	**gget.c**
Ignore_Timetag_Overrun_Px	**monseq.c**
Init_Module_Px	**initcard.c**
Init_Module_Px for VME and VXI Boards	**initcard.c**
InitializeInterrupt_Px	**deviceio.c**
Insert_Msg_Err_Px	**bcset.c**
Internal_Loopback_Px	**gget.c**
Is_Enhanced_Mode_Supported_Px	**monseq.c**
Is_Expanded_Mode_Supported_Px	**gget.c**
Is_IrigSignal_Present_Px	**gget.c**
Is_Single_Function_Px	**gget.c**
Load_Datablk_Px	**rtset.c**
Load_Multiple_Datablk_Px	**rtset.c**
Load_RTid_Px	**rtset.c**
Parse_CommandWord_Px	**gget.c**
Preset_IrigTimeTag_Px	**gget.c**
Print_Error_Px	**error.c**
PrintIRIGtime_Px	**gget.c**
Re_Create_Message_Px	**bcrtmsg.c**
Read_Datablk_Px	**rtset.c**
Read_Message_Px	**bcrtmsg.c**
Read_RT_Status_Px	**rtset.c**
Read_RTid_Px	**rtset.c**
Read_SRQ_Message_Px	**bcrtmsg.c**
Read_Start_Reg_Px	**gget.c**
Release_Module_Px	**initcard.c**
Reset_BC_Status_Px	**bcset.c**

Functions	Code File Name
Reset_RTid_Multibuf_Px	**rtset.c**
Reset_Time_Tag_Px	**initcard.c**
Restart_Px	**gset.c**
RT_Id_Px	**rtset.c**
Run_BC_Px	**bcset.c**
Run_MON_Px	**mon.c**
Run_RT_Px	**rtset.c**
Select_Async_Frame_Px	**bcframe.c**
Send_Async_Frame_Px	**bcframe.c**
Send_Msg_Once_Px	**bcset.c**
Send_Timetag_Sync_Px	**bcset.c**
Set_1553Status_Px	**rtset.c**
Set_BC_Resp_Px	**bcset.c**
Set_Bit_Cnt_Px	**bcset.c**
Set_Broad_Ctl_Px	**mon.c**
Set_Broad_Interrupt_Px	**rtset.c**
Set_Bus_Px	**bcset.c**
Set_Checksum_Blocks_Px	**bcframe.c**
Set_Cnt_Trig_Px	**monseq.c**
Set_Continue_Px	**bcset.c**
Set_Enhanced_Mon_Px	**monseq.c**
Set_Error_Location_Px	**bcset.c**
Set_Expanded_Block_Mode_Px	**gset.c**
Set_Frame_Time_Px	**bcset.c**
Set_Frequency_Mode_Px	**gset.c**
Set_IRIG_TimeTag_Mode_Px	**gset.c**
Set_Halt_Px	**bcset.c**
Set_Header_Exists_Px	**gset.c**
Set_Header_Exists_Px	**gset.c**
Set_Header_Value_Px	**gset.c**
Set_Header_Value_Px	**gset.c**

Functions	Code File Name
Set_Interrupt_On_Msg_Px	**bcframe.c**
Set_Interrupt_Px	**bcset.c**
Set_Invalid_Data_Res_Px	**rtset.c**
Set_Jump_Px	**bcset.c**
Set_Message_Interval_Interrupt_Value_Px	**monseq.c**
Set_Minor_Frame_Time_Px	**bcset.c**
Set_Mode_Addr_Px	**rtset.c**
Set_Mode_Px	**gset.c**
Set_ModuleTime_Px	**gset.c**
Set_MON_Concurrent_Px	**mon.c**
Set_Mon_Response_Time_1553A_Px	**mon.c**
Set_Mon_Response_Time_Px	**mon.c**
Set_Replay_Px	**bcset.c**
Set_Restore_Px	**bcset.c**
Set_Retry_Px	**bcset.c**
Set_RT_Active_Bus_Px	**rtset.c**
Set_RT_Active_Px	**rtset.c**
Set_RT_Broadcast_Px	**initcard**
Set_RT_Errors_Px	**rtset.c**
Set_RT_Nonactive_Px	**rtset.c**
Set_RT_Resp_Time_Px	**rtset.c**
Set_RTid_Interrupt_Px	**rtset.c**
Set_RTid_Multibuf_Px	**rtset.c**
Set_RTid_Status_Px	**rtset.c**
Set_Skip_Px	**bcset.c**
Set_Stop_On_Error_Px	**bcset.c**
Set_Sync_Pattern_Px	**bcset.c**
Set_Timetag_Res_Px	**gset.c**
Set_Trigger_Mode_Px	**monseq.c**
Set_Trigger1_Px	**monseq.c**
Set_Trigger2_Px	**monseq.c**

Functions	Code File Name
Set_UseDmaIfAvailable_Px	**gset.c**
Set_Var_Amp_Px	**bcset.c**
Set_Vector_Px	**rtset.c**
Set_Wd_Cnt_Err_Px	**rtset.c**
Set_Word_Cnt_Px	**bcset.c**
Set_Zero_Cross_Px	**bcset.c**
Start_Frame_Px	**bcframe.c**
Stop_Px	**gset.c**
Wait_For_Interrupt_Px	**deviceio.c**
Wait_For_Multiple_Interrupts_Px	**deviceio.c**

Appendix E-3 Error Messages

All functions in the *1553Px Software Tools* are written as C functions, i.e., they return values. A negative value signifies an error. Below is a list of the error codes, their corresponding negative value and a definition of each error.

Note: In the following list, the error codes are sorted by error value, with kernel level errors for VME/VXI boards at the end of the list.

Error Code	Value	Definition
ebadid	-1	Undefined message ID used as input
einval	-2	Illegal value used as input
emode	-3	Mode specific command called when in the wrong mode
ebadchan	-4	Tried to set channel to illegal value
esim_no_mem	-5	Not enough memory available for simulation
msg2big	-6	Attempted to create a message with too many words
msgnospace	-7	Not enough space in message stack for this message
msg2many	-8	Exceeded maximum number of messages permitted (1660)
frm_badid	-9	Attempted to place an undefined message into a message frame
frm_nostacksp	-10	Not enough space in frame stack for this frame
frm_erange	-11	Frame id specified was not defined with Create_Frame
bcr_erange	-12	Run_BC_Px called with type > 255
frm_maxframe	-13	Exceeded maximum number of frames permitted (20)
frm_erangecnt	-14	Message entry greater than number of messages in the frame
eintr	-15	Attempt to set an undefined interrupt
etiming	-16	Attempt to change a message while module is accessing that message
estackempty	-17	Attempt to read command stack before any messages have been received
enomsg	-18	No new messages have been received

Error Code	Value	Definition
enoskip	-19	Attempt to restore a message that was not set to be skipped
enoasync	-20	The async frame contains fewer messages than the user wants to send asynchronously
etimeout	-21	Timed out waiting for BOARD_HALTED
einvamp	-22	Attempt to set invalid amplitude
eboardnotfound	-23	Too many modules initialized
eboardnotalloc	-24	Attempt to switch to segment which was not allocated in the initialization procedure
etimeoutreset	-26	Timed out waiting for reset
ewrngmodule	-27	Module specified on carrier board is not a *Px* module
enomodule	-28	No module is present at specified location
enotallowedreset	-29	Hardware is not yet ready for software reset; timed out after 100 msec (only for *EXC-1553ExCARD/Px* and *EXC-1553PCMCIA/Px*)
efrequencymodenotsupported	-30	Frequency mode is not supported by the module's firmware
efrequencymodemustbeon	-31	Frequency mode must be turned ON before calling this function; call Set_Frequency_Mode_Px first
ebadhandle	-33	An invalid handle was specified; should be value returned by Init_Module_Px
eboardtoomany	-36	Too many modules initialized
frm_nostack	-37	Start_Frame_Px was not yet called to set up the message stack
func_invalid	-49	Function invalid for this module
ecallmewhenstopped	-50	The function can be called only when the module is stopped
noirqset	-53	No interrupt allocated.
ilbfailure	-57	Internal loopback test failed
elbfailure	-58	External or onboard loopback test failed
rterror	-59	Illegal RT number selected
ebadcommandword	-60	If the T/R bit in the Command word is set incorrectly for the given cmdtype parameter

Error Code	Value	Definition
einit	-61	The module was not initialized
enoalter	-62	This message is being transmitted; it cannot be altered now
enobcmsg	-65	Attempted to read BC message when no new messages have been received
einbcmsg	-66	Error in specified BC message
ertbusy	-67	Current entry in the RT message stack is being written to; try again
ewrongprocessor	-68	The processor is not a NIOS II
esetmoduletime	-70	Could not set the Module Time
eoverrunTtag	-72	Cannot read monitored message. Time Tag of the message was overwritten while the message was being read
eoverrunEOMflag	-73	Cannot read monitored message. The End Of Message flag was overwritten while the message was being read
econcurrmonmodule	-81	PCI[e]: Modules 1 or 3 must be selected VME/VXI: Modules 1 or 3 or 5 or 7 must be selected
eoverrun	-82	Cannot read monitored message, messages were overrun
enoonboardloopback	-83	The onboard loopback feature is not available
eminorframe	-84	The message specified is of the type Minor frame
edbnotset	-85	Double buffering not set
ebadblknum	-86	Bad data block number assigned for double buffering
exreset	-87	Xilinx failed to reset
ercvfunc	-88	This function valid for receive RTid only
ebadttag	-89	Time Tag not working
ettagexternal	-90	Time Tag source is set to an external clock source
enosrq	-91	SRQ was disabled by the user

Error Code	Value	Definition
edoublebuf	-96	An attempt was made to set multiple buffers on an RTid that is already configured to use double buffering
eboundary	-97	If multibufferring attempts to cross an illegal data block boundary (see the note in the description of Set_RTid_Multibuf_Px)
emonmode	-98	Invalid mode; the card/module only operates in Monitor mode
ertvalue	-99	Invalid RT value; does not match single function module's RT value (for *PxS* only)
ebitlocked	-100	The single function RT Number register is locked (for *PxS* only)
ebiterror	-101	The single function RT Number register's error bit is lit (for *PxS* only)
esinglefunconlyerror	-102	This function is only for single function modules (*PxS*)
enotforsinglefuncerror	-103	This function is not for single function modules (*PxS*)
eresptimeerror	-104	Single function response time cannot be greater than 12,000 nanoseconds (for *PxS* only)
efeature_not_supported_PX	-105	The feature is not supported on this module
eIrigTimetagSet_PX	-111	This function was called when in IRIG B Time Tag mode (Set_IRIG_TimeTag_Mode_Px); use Get_Next_Mon_IRIGttag_Message_Px instead
Kernel Level Errors		
eopenkernel	-1001	Error opening kernel device; check **`ExcConfig`** settings
ekernelcantmap	-1002	Error mapping memory
ereleventhandle	-1003	Error releasing the event handle
egetintcount	-1004	Error getting interrupt count
egetchintcount	-1005	Error getting channel interrupt count
egetintchannels	-1006	Error getting interrupt channels
ewriteiobyte	-1007	Error writing I/O memory
ereadiobyte	-1008	Error reading I/O memory
egeteventhand1	-1009	Error getting event handle (in first stage)

Error Code	Value	Definition
egeteventhand2	-1010	Error getting event handle (in second stage)
eopenscmant	-1011	Error opening Service Control Manager (in startkerneldriver)
eopenservicet	-1012	Error opening kernel service (in startkerneldriver)
estartservice	-1013	Error starting kernel service (in startkerneldriver)
eopenscmanp	-1014	Error opening Service Control Manager (in stopkerneldriver)
eopenservicep	-1015	Error opening kernel service (in stopkerneldriver)
econtrolservice	-1016	Error in control service (in stopkerneldriver)
eunmapmem	-1017	Error unmapping memory
egetirq	-1018	Error getting IRQ number
eallocresources	-1019	Error allocating resources; see **`Installation Instructions.pdf`** for details on resource allocation problems
egetramsize	-1020	Error getting RAM size
ekernelwriteattrib	-1021	Error writing attribute memory
ekernelreadattrib	-1022	Error reading attribute memory
ekernelfrontdesk	-1023	Error opening kernel device; check **`ExcConfig`** set up
ekernelOscheck	-1024	Error determining operating system
ekernelfrontdeskload	-1025	Error loading **`frontdesk.dll`**
ekerneliswin2000compatible	-1026	Error determining Windows 2000 compatibility
ekernelbankramsize	-1027	Error determining memory size
ekernelgetcardtype	-1028	Error getting card type
emodnum	-1029	Invalid module number specified
regnotset	-1030	Board not configured; reboot after **`ExcConfig`** is run and board is in slot
ekernelbankphysaddr	-1031	Error getting physical memory address
ekernelclosedevice	-1032	Error closing kernel device
ekerneldevicenotopen	-1034	Error returned by kernel: device not open

Error Code	Value	Definition
ekernelinitmodule	-1035	Error initializing kernel
ekernelbadparam	-1036	Error returned by kernel: bad input parameter
ekernelbadpointer	-1037	Error returned by kernel: invalid pointer to output buffer
ekerneltimeout	-1038	Timeout expired before interrupt occurred
ekernelnotwin2000	-1039	Error returned by kernel: operating system is not Windows 2000 compatible
erequestnotification	-1040	Error requesting interrupt notification
ekernelnot4000card	-1041	Error returned by kernel: designated board is not EXC-4000/8000 family
enotimersirig	-1042	Timers and IrigB are not supported on this version of EXC-4000/8000 family board
eclocksource	-1059	Invalid clock source specified
eparmglobalreset	-1062	Illegal parameter used for globalreset_flag in the StartTimer_4000 function
etimernotrunning	-1063	Timer not running when function was called; did nothing
etimerrunning	-1064	Timer already running; did nothing
eparmreload	-1065	Illegal parameter used for reload_flag in StartTimer_4000
eparminterrupt	-1066	Illegal parameter used for interrupt_flag in StartTimer_4000
ebaddevhandle	-1067	Invalid handle specified; use value returned by Init_Timers_4000
edevtoomany	-1068	Init_Timers_4000 called for too many boards
einvalidOS	-1069	Invalid operating system
Kernel Level Errors for VME/VXI Boards		
eviclosedev	-1050	Error closing the device
evicloserm	-1051	Error closing the default RM
eopendefaultrm	-1052	Error opening the default RM
eviopen	-1053	Error opening VISA
evimapaddress	-1054	Error mapping address
evicommand	-1055	Error in VISA command

Error Code	Value	Definition
einstallhandler	-1056	Error installing VISA handler
eenableevent	-1057	Error enabling VISA event
eunistallhandler	-1058	Error uninstalling VISA handler
edevnum	-1060	Specified device number greater than 255
einstr	-1061	Error initializing module
efuncinvalid	-1082	Function invalid for this board
enotforunet	-1089	Function invalid for UNET

Function Index

February 2025, Rev C-6