Excalibur Carrier Board 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

The *Excalibur Carrier Board Software Tools* is a set of C language functions designed to aid users of the multimode, multiprotocol Excalibur 4000/8000 avionics test and simulation carrier board series to write their own applications. *Excalibur Carrier Board Software Tools* are for Excalibur standalone boards and for 4000/8000 family carrier boards. In addition to board level *Software Tools* described in this manual, each module has module level *Software Tools*.

The *Excalibur Carrier Board Software Tools* are for the following boards:

DAS-429[cc]PMC/RTx[D]
DAS-429ExCARD/RT[D]
DAS-429mPCI/RTx
DAS-429mPCIe/RTx
DAS-429UNET/RTx
DAS-429USB/RxTx
ES-1553RUNET/Px
ES-9810 RUNET II
EXC-1394PCI
EXC-1394PCIe
EXC-1553[c]PCI/MCH
EXC-1553[cc]PMC/MCH
EXC-1553[cc]PMC/Px
EXC-1553ccVME/Px
EXC-1553ExCARD/Px
EXC-1553mPCIe/P1
EXC-1553mPCIeD/P2
EXC-1553UNET/Px
EXC-1553USB/PM
EXC-2000PCI
EXC-2000PCIe
EXC-4000cPCI
EXC-4000P104plus
EXC-4000PCI
EXC-4000PCI/HC
EXC-4000PCIe
EXC-4000PCIe64
EXC-4000PCIeHC
EXC-4000VME
EXC-4000VXI
EXC-4500ccVPX
EXC-664PCIe
EXC-708ccPMC
EXC-708mPCIe
EXC-8000ccVPX
EXC-8000ccXMC
EXC-8000cPCI
EXC-8000PCIe
EXC-8000PCIe104
EXC-8000PCIeHC
EXC-DiscretePCI/Dx
EXC-DiscretePCIe/Dx
EXC-ETHPCIe
EXC-UNET2

Getting Started

The following sections describe the installation procedure. These sections are not relevant to the *UNET* family. For these products, refer to the Installation chapter of the appropriate *UNET* 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

The ExcConfig utility 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`**.

Software Tools DLLs

The *Excalibur Carrier Board Software Tools* functions are located in a separate, general purpose DLL. This DLL is automatically installed when you install the *Software Tools* for any module. Some of the board level functions in this DLL are accessed by the module level DLL.

Depending on the board the names of the files are:

Carrier Board	DLL Name
PCI[e] and PMC Boards and ExpressCards	`EXC4000MS.DLL`
VME and VXI Boards	`EXCV4000MS.DLL`

The *Excalibur Carrier Board Software Tools* functions apply to the carrier board and to all the modules on the board. The functions are not module or mode specific. To program individual modules, refer to the module's programmer's reference.

Updating to the Most Recent Version of the *Software Tools*

Excalibur Carrier Board Software Tools are periodically updated independently of the *Software Tools* for the M4K/M8K modules. To update the *Software Tools* to the latest version, download the latest version from our website (www.mil-1553.com/4000).

To install the latest source files and DLLs:

1. Download the file `4000pci-st-win32-win64.zip` from www.mil-1553.com/4000.
2. Unzip the file to:
   `C:\Excalibur\4000PCI Family Software Tools\Source`.
   The zip file contents should completely replace the existing folder and file structure.
3. In addition, copy the latest copies of `Exc4000Ms.dll` (`Win32/x64`, `Debug/Release`) to the appropriate folder(s) of your M4K/M8K module software tools.
   The 32-bit DLLs are located at:
   `C:\Excalibur\4000PCI Family Software Tools\Source\lib\excMSVisual\Win32` (`Debug/Release`)
   The 64-bit DLLs are located at:
   `C:\Excalibur\4000PCI Family Software Tools\Source\lib\excMSVisual\x64` (`Debug/Release`)
   The target folder for any M4K/M8K module starts at:
   `C:\Excalibur\ModuleName Software Tools\Source\lib`

Compiler Options

The DLL is compiled under Microsoft Visual Studio using **_cdecl** options. The driver functions in the *Excalibur Carrier Board 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 programs on the same module simultaneously.

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. For the values of error messages associated with *Software Tools* see **Appendix B Error Messages**.

Conventions Used in the Programmer's Reference

To help differentiate between different kinds of information, the following text Functions look like this.

`Parameters` look like this.

Variable types look like this.

`File names` look like this.

FLAGS look like this.

Note: WORD = unsigned short int
DWORD = unsigned int

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 Carrier Board Level Functions

This chapter lists the functions that apply to both PCI[e] and VME/VXI boards. These functions apply to the carrier board and to all the modules on the board. The functions are not module or mode specific. To write applications for individual modules, refer to the specific module's *Programmer's Reference*.

Get_4000Board_Type	
Get_4000Interface_Rev	
Get_4000Module_Type	(not available for the *EXC-1553ccVME/Px* carrier board; use Modules_Ready_CCVMEPX instead)
Get_Error_String_4000	
Get_Time_Tag_Source_4000	(not available for the *EXC-2000PCI[e]* carrier board)
Reset_Module_4000	
Reset_Timetags_On_All_Modules_4000	
Select_Time_Tag_Source_4000	(not available for the *EXC-2000PCI[e]* carrier board)

Get_4000Board_Type

Description Get_4000Board_Type checks the board type of the specified board.

Note: See also **IsPciExpress_4000** on page 3-8.

Syntax Get_4000Board_Type (WORD `device_num`, WORD `*boardtype`)

Input Parameters

`device_num` The device number assigned to the board using the **ExcConfig** utility, or the default device number.

You can use the default device number or the corresponding #define value when only one board with this default device number exists in the computer. For example, for a single *EXC-4000PCIe* board you can use the value 25 or the #define value **EXC_4000PCI**. For more information, see **Assigning a Device Number** on page 1-3.

For VME/VXI carrier boards, specify the logical address (0–255) according to the DIP switch settings (or jumper settings for *EXC-1553ccVME/Px* boards).

Output Parameters

`boardtype`

PCI[e] carrier boards:

EXC_BRDTYPE_1394PCI
: *EXC-1394PCI* board is present [1394 H]

EXC4000_BRDTYPE_PCI
: *EXC-4000PCI* board is present [4000 H]

EXC4000_BRDTYPE_CPCI
: *EXC-4000cPCI* board is present [4001 H]

EXC4000_BRDTYPE_MCH_PCI
: *EXC-1553PCI/MCH* board is present [4002 H]

EXC4000_BRDTYPE_MCH_CPCI
: *EXC-1553cPCI/MCH* board is present [4003 H]

EXC4000_BRDTYPE_MCH_PMC
: *EXC-1553[cc]PMC/MCH* board is present [4004 H]

EXC4000_BRDTYPE_429_PMC
: *DAS-429[cc]PMC/RTx[D]* board is present [4005 H]

EXC2000_BRDTYPE_PCI
: *EXC-2000PCI* board is present [4006 H]

EXC4000_BRDTYPE_P104P
: *EXC-4000P104plus* board is present [4007 H]

EXC4000_BRDTYPE_PCIHC
: *EXC-4000PCI/HC* board is present [4008 H]

EXC4000_BRDTYPE_708_PMC
: *EXC-708ccPMC* board is present [4009 H]

EXC4000_BRDTYPE_1553PX_PMC
: *EXC-1553[cc]PMC/Px* board is present [400A H]

Get_4000Board_Type (cont.)

EXC4000_BRDTYPE_DISCR_PCI
EXC-DiscretePCI/Dx board is present [400D H]
MINIPCI_BRDTYPE_429RTX
DAS-429mPCI/RTx card is present [4011 H]
EXC_BRDTYPE_UNET
EXC-1553UNET/Px or *DAS-429UNET/RTx* card is present [5502 H]
EXC_BRDTYPE_RNET
ES-1553RUNET/Px card is present [5505 H]
EXC_BRDTYPE_USB_INLINE
EXC-1553USB/PM or *DAS-429USB/RxTx* card is present [5507 H]
EXC_BRDTYPE_UNET2
EXC-UNET2 card is present [5508 H]
EXC_BRDTYPE_RNET2
ES-9810 RUNET II card is present [5509 H]
EXC8000_BRDTYPE_CPCI
EXC-8000cPCI board is present [8001 H]
EXC_BRDTYPE_1394PCIE
*EXC-1394PCI*e board is present [E394 H]
EXC4000_BRDTYPE_PCIE
EXC-4000PCIe board is present [E400 H]
EXCARD_BRDTYPE_1553PX
EXC-1553ExCARD/Px card is present [E401 H]
EXCARD_BRDTYPE_429RTX
DAS-429ExCARD/RT[D] card is present [E402 H]
MINIPCIE_BRDTYPE_1553PX
EXC-1553mPCIe/P1 card is present [E403 H]
MINIPCIE_BRDTYPE_429RTX
DAS-429mPCIe/RTx card is present [E404 H]
EXC2000_BRDTYPE_PCIE
EXC-2000PCIe board is present [E406 H]
EXC4000_BRDTYPE_PCIEHC
EXC-4000PCIeHC board is present [E408 H]
EXC4000_BRDTYPE_DISCR_PCIE
EXC-DiscretePCIe/Dx board is present [E40D H]
MINIPCIE_BRDTYPE_1553DPX
EXC-1553mPCIeD/P2 card is present [E423 H]
EXCARD_BRDTYPE_708MPCIE
EXC-708mPCIe card is present [E427 H]
EXC4500_BRDTYPE_PCIE_VPX
EXC-4500ccVPX board is present [E450 H]
EXC4000_BRDTYPE_PCIE64
EXC-4000PCIe64 board is present [E464 H]
EXC_BRDTYPE_664PCIE
EXC-664PCIe or *EXC-ETHPCIe* board is present [E664 H]
EXC8000_BRDTYPE_PCIe
EXC-8000PCIe board is present [E800 H]

Get_4000Board_Type (cont.)

EXC8000_BRDTYPE_PCIEHC
EXC-8000PCIeHC board is present [E801 H]
EXC_BRDTYPE_8000P104e
EXC-8000PCIe104 board is present [E807 H]
EXC8000_BRDTYPE_CCXMC
EXC-8000ccXMC board is present [E810 H]
EXC8000_BRDTYPE_PCIE_VPX
EXC-8000ccVPX board is present [E850 H]

VME/VXI carrier boards:

EXC4000_BOARDSIGVALUE
EXC-4000VME or *EXC-4000VXI* board is present [4000 H]
CCVMEPX_BOARDSIGVALUE
EXC-1553ccVME/Px board is present [4F00 H]

Return Values

emodnum	If an invalid module number was specified
eopenkernel	Cannot open kernel device; check the **ExcConfig** settings
ekernelcantmap	Kernel driver cannot map memory
ekernelnot4000card	If the designated board is not a 4000/8000 family board
einstr	If the device was not initialized
edevnum	If the value for `device_num` is invalid
0	If successful

Get_4000Interface_Rev

Description Get_4000Interface_Rev returns the host revision number of the specified board.

Syntax Get_4000Interface_Rev (WORD `device_num`, WORD `*interface_rev`)

Input Parameters

`device_num`	The device number assigned to the board using the **ExcConfig** utility, or the default device number. You can use the default device number or the corresponding #define value when only one board with this default device number exists in the computer. For example, for a single *EXC-4000PCI[e]* board you can use the value 25 or the #define value **EXC_4000PCI**. For more information, see **Assigning a Device Number** on page 1-3. For VME/VXI carrier boards, specify the logical address (0–255) according to the DIP switch settings (or jumper settings for *EXC-1553ccVME/Px* boards).

Output Parameters

`interface_rev`	A pointer to the host revision number of the board.

Return Values

emodnum	If an invalid module number was specified
eopenkernel	Cannot open kernel device; check the **ExcConfig** settings
ekernelcantmap	Kernel driver cannot map memory
ekernelnot4000card	If the designated board is not a 4000/8000 family board
einstr	If the device was not initialized
edevnum	If the value for `device_num` is invalid
0	If successful

Get_4000Module_Type

Description

Get_4000Module_Type checks which, if any, module is currently in the specified location.

This function is also called automatically from the Init_Module function of each module's specific *Software Tools* to ascertain that the type of module is that specific module.

The lower five bits of the output value are the module type. The 6th bit indicates whether the module is 16-bit or 32-bit module. A value of 1 indicates that it is a 32-bit module.

The values listed below in `modtype` are after masking off the upper 11 bits of the output value.

Syntax

Get_4000Module_Type (WORD `device_num`, WORD `module_num`, WORD `*modtype`)

Input Parameters

`device_num` The device number assigned to the board using the **ExcConfig** utility, or the default device number.

You can use the default device number or the corresponding #define value when only one board with this default device number exists in the computer. For example, for a single *EXC-4000PCI[e]* board you can use the value 25 or the #define value **EXC_4000PCI**. For more information, see **Assigning a Device Number** on page 1-3.

For VME/VXI carrier boards, specify the logical address (0–255) according to the DIP switch settings (or jumper settings for *EXC-1553ccVME/Px* boards).

`module_num` A module position number (0–7 for most boards).

Output Parameters

`modtype`

EXC4000_MODTYPE_SERIAL
: Serial module is present [0002 H]

EXC4000_MODTYPE_MCH
: MCH module is present [0003 H]

EXC4000_MODTYPE_RTX
: RT*x* module is present [0004 H]

EXC4000_MODTYPE_PX
: Px module is present [0005 H]

EXC4000_MODTYPE_MMSI
: MMSI module is present [0006 H]

EXC4000_MODTYPE_708
: 708 module is present [0007 H]

Get_4000Module_Type (cont.)

EXC4000_MODTYPE_H009
H009 module is present [0009 H]
EXC4000_MODTYPE_CAN
CAN module is present [000C H]
EXC4000_MODTYPE_DIO
Discrete module is present [000D H]
EXC4000_MODTYPE_SERIAL_PLUS
SerialPlus module is present [0012 H]
EXC4000_MODTYPE_AFDX_RX
ARINC 664 Part 7 (AFDX) receive module is present [001A H]
EXC4000_MODTYPE_ETHERNET
Ethernet module is present [001B H]
EXC4000_MODTYPE_AFDX_TX
ARINC 664 Part 7 (AFDX) transmit module is present [001C H]
EXC4000_MODTYPE_NONE
If no module is present [001F H]

Return Values

emodnum	If an invalid module number was specified
eopenkernel	Cannot open kernel device; check the **ExcConfig** settings
ekernelcantmap	Kernel driver cannot map memory
ekernelnot4000card	If the designated board is not a 4000/8000 family board
0	If successful

Get_Error_String_4000

Description

Get_Error_String_4000 accepts the error code return values from other board level functions. This function returns the string containing a corresponding error message.

Syntax

Get_Error_String_4000 (int `errcode`, int `errlen`, char `*errstring`)

Example:

```
#define ERRORLEN 255
char ErrorStr[ERRORLEN];

Get_Error_String_4000 (errorcode, ERRORLEN,
&Errorstr);
printf("error is: %s", ErrorStr);
```

Get_Error_String_4000 (cont.)

Input Parameters	`errcode`	The error code returned from a *Software Tools* call.
	`errlen`	Maximum length of string to be returned - the message string that contains the corresponding error message
Output Parameters	`errstring`	A pointer to an array of '`errlen`' characters, the message string that contains the corresponding error message. In case of bad input, this function returns a string denoting that.
Return Values	0	Always

Get_Time_Tag_Source_4000

Description Get_Time_Tag_Source_4000 returns the source of the Time Tag used in all modes and for all modules on the board. It is one setting for *all* the modules used.

Note: *EXC-1553ccVME/Px* boards and ExpressCards do not support this function.

Syntax Get_Time_Tag_Source_4000 (WORD `device_num`, WORD `*source`)

Input Parameters	`device_num`	The device number assigned to the board using the **ExcConfig** utility, or the default device number. You can use the default device number or the corresponding #define value when only one board with this default device number exists in the computer. For example, for a single *EXC-4000PCI[e]* board you can use the value 25 or the #define value **EXC_4000PCI**. For more information, see **Assigning a Device Number** on page 1-3. For VME/VXI carrier boards, specify the logical address (0–255) according to the DIP switch settings (or jumper settings for *EXC-1553ccVME/Px* boards).

Get_Time_Tag_Source_4000 (cont.)

Output Parameters	`source`	EXC4000_INTERNAL_CLOCK Uses the board's 4 µsec. timer (default) [0000 H]
		EXC4000_EXTERNAL_CLOCK [0001 H]
		See "External Signals Connector" in the **Mechanical and Electrical Specifications** chapter in the applicable hardware *User's Manual*
Return Values	ekernelnot4000card	If the designated board is not a 4000/8000 family board
	ekernelinitmodule	If error initializing kernel related data
	ekernelbadparam	If input parameter is invalid
	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
	eopenkernel	If there was an error opening a device
	ekernelbadpointer	If output parameter buffer is invalid
	ekerneldevicenotopen	If specified device has not been opened
	ekernelcantmap	If a pointer to memory cannot be obtained
	enotforexcard	ExCARD does not support this function
	0	If successful

Reset_Module_4000

Description Reset_Module_4000 resets the module via the global reset register on the carrier board.

Syntax Reset_Module_4000 (WORD `device_num`, WORD `module_num`)

Input parameters

`device_num`	The device number assigned to the board using the **ExcConfig** utility, or the default device number. You can use the default device number or the corresponding #define value when only one board with this default device number exists in the computer. For example, for a single *EXC-4000PCI[e]* board you can use the value 25 or the #define value **EXC_4000PCI**. For more information, see **Assigning a Device Number** on page 1-3. For VME/VXI carrier boards, specify the logical address (0–255) according to the DIP switch settings (or jumper settings for *EXC-1553ccVME/Px* boards).
`module_num`	A module position number (0–7 for most boards).

Output Parameters none

Return Values

emodnum	If an invalid module number was specified
ekernelnot4000card	If the designated board is not a 4000/8000 family board
0	If successful

Reset_Timetags_On_All_Modules_4000

Description Reset_Timetags_On_All_Modules_4000 resets the Time Tag in all modules on the board.

Syntax Reset_Timetags_On_All_Modules_4000 **(**WORD `device_num`**)**

Input Parameters

`device_num`	The device number assigned to the board using the **ExcConfig** utility, or the default device number. You can use the default device number or the corresponding #define value when only one board with this default device number exists in the computer. For example, for a single *EXC-4000PCI[e]* board you can use the value 25 or the #define value **EXC_4000PCI**. For more information, see **Assigning a Device Number** on page 1-3. For VME/VXI carrier boards, specify the logical address (0–255) according to the DIP switch settings (or jumper settings for *EXC-1553ccVME/Px* boards).

Output Parameters none

Return Values

eopenkernel	Cannot open kernel device; check the **ExcConfig** settings
ekernelcantmap	Kernel driver cannot map memory
0	If successful

Select_Time_Tag_Source_4000

Description Select_Time_Tag_Source_4000 selects the source of the Time Tag used in all modes and for all modules on the board. It is one setting for *all* the modules used.

If this function is not called, the internal clock is the default clock source.

Note: *EXC-1553ccVME/Px* boards and ExpressCards do not support this function.

Syntax Select_Time_Tag_Source_4000 (WORD `device_num`, WORD `source`)

Input Parameters

`device_num`	The device number assigned to the board using the **ExcConfig** utility, or the default device number. You can use the default device number or the corresponding #define value when only one board with this default device number exists in the computer.

Select_Time_Tag_Source_4000 (cont.)

		For example, for a single *EXC-4000PCI[e]* board you can use the value 25 or the #define value **EXC_4000PCI**. For more information, see **Assigning a Device Number** on page 1-3.
		For VME/VXI carrier boards, specify the logical address (0–255) according to the DIP switch settings (or jumper settings for *EXC-1553ccVME/Px* boards).
	`source`	EXC4000_INTERNAL_CLOCK Uses the board's 4 µsec. timer (default) [0000 H] EXC4000_EXTERNAL_CLOCK [0001 H]
		See External Signals Connector in the **Mechanical and Electrical Specifications** chapter in your carrier board's hardware *User's Manual*
Output Parameters	none	
Return Values	eopenkernel	Cannot open kernel device; check the **ExcConfig** settings
	ekernelcantmap	Kernel driver cannot map memory
	eclocksource	If an invalid clock source was specified
	enotforexcard	ExCARD does not support this function
	0	If successful

3 PCI[e] and UNET Carrier Board Level Functions

This chapter lists the functions that apply to the PCI[e] boards listed on the front cover of this manual. These functions apply to the carrier board and to all the modules on the board. The functions are not module or mode specific. To write applications for individual modules, refer to the specific module's *Programmer's Reference*.

General PCI[e] Functions	Get_4000Board_Name Get_4000Board_NumModulesSupported Get_4000Module_Info Get_4000Module_Timetag64 Get_Extended_UNet_Info Get_UniqueID_P4000 IsDmaSupported_4000 IsRepetitiveDMASupported_4000 IsUnetFamilyDevice_4000	See **General PCI[e] Functions** on page 3-2
IRIG B Functions **(not available for *EXC-2000PCI[e]*, *EXC-1553[c]PCI/MCH*, *EXC-1553ExCARD/Px and DAS-429ExCARD/RT[D]*)**	GetIrigControl_4000 GetIrigSeconds_4000 GetIrigTime_4000 GetIrigYear_4000 IsIrigTimeavail_4000 SetIrig_4000	Before using these functions, see **Using IRIG B Mode** on page 3-11 for instructions
Timer Functions **(not available for *EXC-2000PCI[e]* and *EXC-1553[c]PCI/MCH*)**	Init_Timers_4000 IsTimerRunning_4000 ReadTimerValue_4000 Release_Timers_4000 ResetWatchdogTimer_4000 StartTimer_4000 StopTimer_4000	Before using these functions, see **Using Timer Functions** on page 3-16 for instructions
Interrupt Functions	Get_Interrupt_Count_P4000 InitializeInterrupt_P4000 Wait_For_Interrupt_P4000	See **Using Interrupts Under Windows** on page 3-21

General PCI[e] Functions

Get_4000Board_Name

Description Get_4000Board_Name returns the board name as a string.

Syntax Get_4000Board_Name (WORD `device_num`, char `*pBoardName`)

Input Parameters

`device_num`	The device number assigned to the board using the **ExcConfig** utility, or the default device number. You can use the default device number or the corresponding #define value when only one board with this default device number exists in the computer. For example, for a single *EXC-4000PCI[e]* board you can use the value 25 or the #define value **EXC_4000PCI**. For more information, see **Assigning a Device Number** on page 1-3. For VME/VXI carrier boards, specify the logical address (0–255) according to the DIP switch settings (or jumper settings for *EXC-1553ccVME/Px* boards).

Output Parameters

`pBoardName`	A pointer to a string containing the board name (without slashes). A forward slash (/) in a board name is replaced by a dash (-).

Return Values

eunrecognizedboard	If unrecognized carrier board
<0	If other kernel level error
0	If successful

Get_4000Board_NumModulesSupported

Description Get_4000Board_NumModulesSupported returns the number of modules supported on the board.

Syntax Get_4000Board_NumModulesSupported (WORD `device_num`, int `*pNumModsSupported`)

Input Parameters

`device_num`	The device number assigned to the board using the **ExcConfig** utility, or the default device number. You can use the default device number or the corresponding #define value when only one board with this default device number exists in the computer. For example, for a single *EXC-4000PCI[e]* board you can use the value 25 or the #define value **EXC_4000PCI**. For more information, see **Assigning a Device Number** on page 1-3. For VME/VXI carrier boards, specify the logical address (0–255) according to the DIP switch settings (or jumper settings for *EXC-1553ccVME/Px* boards).

Output Parameters

`pNumModsSupported`	A pointer to the number of modules supported on the board

Return Values

eunrecognizedboard	If unrecognized carrier board
<0	If other kernel level error
0	If successful

Get_4000Module_Info

Description

Get_4000Module_Info returns information about the specified module on the board.

Syntax

Get_4000Module_Info (WORD `device_num`, WORD `module_num`, t_ModuleInfo `*pModuleInfo`)

Input Parameters

`device_num` — The device number assigned to the board using the **ExcConfig** utility, or the default device number.

You can use the default device number or the corresponding #define value when only one board with this default device number exists in the computer. For example, for a single *EXC-4000PCI[e]* board you can use the value 25 or the #define value **EXC_4000PCI**. For more information, see **Assigning a Device Number** on page 1-3.

For VME/VXI carrier boards, specify the logical address (0–255) according to the DIP switch settings (or jumper settings for *EXC-1553ccVME/Px* boards).

`module_num` — A module position number (0–7 for most boards).

Output Parameters

`pModuleInfo` — A pointer to a struct with module information:

```
struct t_ModuleInfo
{
WORD moduleType;          // e.g. 05 (Px), 04 (RTx)
WORD numOfBitsAccess;     // e.g. 16, 32, 64
WORD moduleFamily;        // e.g. 4, 8
char moduleNameStr [100]; // e.g. M4K1553Px
WORD timetagSize;         // e.g. 16, 32, 64
WORD reserved [12];
} t_ModuleInfo;
```

Return Values

emodnum	If an invalid module number was specified (greater than max. modules)
<0	If other kernel level error
0	If successful

Get_4000Module_Timetag64

Description

Get_4000Module_Timetag64 returns a module's current Time Tag value.

Note: The 64 at the end of the function name indicates that it returns a 64-bit value. Most modules have a 32-bit Time Tag, some have a 64-bit Time Tag and MCH modules have a 16-bit Time Tag.

Syntax

Get_4000Module_Timetag64 (WORD `device_num`, WORD `module_num`, __int64 `*pModuleTTAG`)

Input Parameters

`device_num`	The device number assigned to the board using the **ExcConfig** utility, or the default device number. You can use the default device number or the corresponding #define value when only one board with this default device number exists in the computer. For example, for a single *EXC-4000PCI[e]* board you can use the value 25 or the #define value **EXC_4000PCI**. For more information, see **Assigning a Device Number** on page 1-3. For VME/VXI carrier boards, specify the logical address (0–255) according to the DIP switch settings (or jumper settings for *EXC-1553ccVME/Px* boards).
`module_num`	A module position number (0–7 for most boards).

Output Parameters

`pModuleTTAG`	A pointer to a 64-bit register containing the module's Time Tag.

Return Values

<0	If other kernel level error
0	If successful

Get_Extended_UNet_Info

Description Get_Extended_UNet_Info returns a structure with extended UNET information.

Syntax Get_Extended_UNet_Info (WORD `device_num`, t_INSTANCE_EXTENDED_EXCUNETINFO *`pExtendedUnetInfo`)

Input Parameters

`device_num` — The device number assigned to the board using the **ExcConfig** utility, or the default device number.

You can use the default device number or the corresponding #define value when only one board with this default device number exists in the computer. For example, for a single *EXC-4000PCI[e]* board you can use the value 25 or the #define value **EXC_4000PCI**. For more information, see **Assigning a Device Number** on page 1-3.

For VME/VXI carrier boards, specify the logical address (0–255) according to the DIP switch settings (or jumper settings for *EXC-1553ccVME/Px* boards).

Output Parameters

`pExtendedUnetInfo` — A pointer to a struct with extended UNET information:

```
struct t_INSTANCE_EXTENDED_EXCUNETINFO
{
UINT CommOptions;   // bits 0-2 max header version; rest
                       are reserved)
WORD SerialNumber;
char UNetControlFwStr  [10];
WORD OptionsReg;  // bit 0 = supports
                       CLEAR_EOM_STATUS;
                       bit 1 = supports FIFO_RW;
                       bit 2 = supports 32BIT_RW;
                       rest are reserved
char MACaddrStr [20];// actual length is 18
char IPaddrStr [20];  // actual length is 16
char reserved [50];
} t_INSTANCE_EXTENDED_EXCUNETINFO;
```

Return Values

<0	If other UNET level error
0	If successful

Get_UniqueID_P4000

Description Get_UniqueID_P4000 returns the four bit board identification number, according to the DIP switch setting on the board (or jumper settings for *EXC-1553ccVME/Px* boards). See **Board Identification Number** in the *EXC-4000VME User's Manual*.

Syntax Get_UniqueID_P4000 (WORD `device_num`, WORD `*pwUniqueID`)

Input parameters

`device_num`	The device number assigned to the board using the **ExcConfig** utility, or the default device number. You can use the default device number or the corresponding #define value when only one board with this default device number exists in the computer. For example, for a single *EXC-4000PCI[e]* board you can use the value 25 or the #define value **EXC_4000PCI**. For more information, see **Assigning a Device Number** on page 1-3.

Output Parameters

`pwUniqueID`	A pointer to the four-bit board identification number, according to the DIP switch setting on the board (or jumper settings for *EXC-1553ccVME/Px* boards)

Return Values

ekernelinitmodule	If error initializing kernel related data
ekernelbadparam	If input parameter is invalid
eopenkernel	If cannot open kernel device; check the **ExcConfig** settings
ekernelfrontdeskload	If error loading **`frontdesk.dll`**
ekernelfrontdesk	If error opening kernel device; check **ExcConfig** settings
eallocresources	If there was an error allocating resources
regnotset	If board is not configured; reboot after **ExcConfig** is run and board is in slot
ekernelbadpointer	If output parameter buffer is invalid
ekerneldevicenotopen	If the specified device was not opened
ekernelcantmap	If a pointer to memory cannot be obtained
ekernelnot4000card	If the designated board is not a 4000/8000 family board
0	If successful

IsDmaSupported_4000

Description IsDmaSupported_4000 returns whether Direct Memory Access (DMA) is available. DMA enables the board, card or module to read blocks of 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.

Syntax IsDmaSupported_4000 (WORD `device_num`)

Input parameters

`device_num`	The device number assigned to the board using the **ExcConfig** utility, or the default device number. You can use the default device number or the corresponding #define value when only one board with this default device number exists in the computer. For example, for a single *EXC-4000PCI[e]* board you can use the value 25 or the #define value **EXC_4000PCI**. For more information, see **Assigning a Device Number** on page 1-3.

Output Parameters none

Return Values

1	DMA is supported
0	DMA is not supported

IsPciExpress_4000

Description IsPciExpress_4000 returns whether the board is a PCI Express board.

Note: See also **Get_4000Board_Type** on page 2-2.

Syntax IsRepetitiveDMASupported_4000 (WORD `wBoardType`)

Input parameters

`wBoardType`	The board type. See page 2 for the list of board types. **Note:** The board type can be retrieved by calling Get_4000Board_Type.

Output Parameters none

Return Values

TRUE	The board is a PCI Express board
FALSE	The board is not a PCI Express board

IsRepetitiveDMASupported_4000

Description

IsRepetitiveDMASupported_4000 checks whether Repetitive DMA (Repetitive Direct Memory Access) is available. DMA enables the board, card or module to read blocks of 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 using Repetitive DMA to read from memory, the board reads repeatedly from the same memory location and writes to one or more RAM memory locations. When using Repetitive DMA to write to memory, the board reads from one or more RAM memory locations and writes repeatedly to the same memory location.

This is required when reading from, or writing to, a FIFO, where the data is read from, or written to, a single memory location.

Syntax

IsRepetitiveDMASupported_4000 (WORD `device_num`)

Input parameters

`device_num`	The device number assigned to the board using the **ExcConfig** utility, or the default device number. You can use the default device number or the corresponding #define value when only one board with this default device number exists in the computer. For example, for a single *EXC-4000PCI[e]* board you can use the value 25 or the #define value **EXC_4000PCI**. For more information, see **Assigning a Device Number** on page 1-3.

Output Parameters

none

Return Values

TRUE	DMA is supported
FALSE	DMA is not supported

IsUnetFamilyDevice_4000

Description	IsUnetFamilyDevice_4000 reads the Windows Registry to determine whether the board is part of the UNET family of boards.	
Syntax	IsUnetFamilyDevice_4000 (WORD `device_num`)	
Input Parameters	`device_num`	The device number assigned to the board using the **ExcConfig** utility, or the default device number. You can use the default device number or the corresponding #define value when only one board with this default device number exists in the computer. For example, for a single *EXC-4000PCI[e]* board you can use the value 25 or the #define value **EXC_4000PCI**. For more information, see **Assigning a Device Number** on page 1-3. For VME/VXI carrier boards, specify the logical address (0–255) according to the DIP switch settings (or jumper settings for *EXC-1553ccVME/Px* boards).
Output Parameters	none	
Return Values	0	If board is not UNET family board
	1	If board is UNET family board

Using IRIG B Mode

To set up IRIG B mode:

1. Call **Init_Timers_4000** on page 3-17 before calls to carrier board Timer and IrigB functions. This function returns a `handle`, which is used as the first parameter in all IRIG B, Timer and Interrupt functions.
2. Call **SetIrig_4000** on page 3-15, to set up the IRIG B feature.
3. Wait, in a loop, until `availFlag` is TRUE. See **IsIrigTimeavail_4000** on page 3-15.
4. IRIG B time is then available either in seconds or a structure of 16-bit words in the form of days/hours/minutes/seconds. See **GetIrigSeconds_4000** on page 3-13 and **GetIrigTime_4000** on page 3-14.
5. To receive additional information about the Control field bits from the IRIG B bus see **GetIrigControl_4000** on page 3-13.
6. After completing your application, call **Release_Timers_4000** on page 3-18, to release Timer and IrigB functions.

The **`demo_irig.c`** program, on the *Excalibur Installation CD*, demonstrates the use of the IRIG B feature.

Note:

- The synchronization of IRIG B time can take up to two seconds. IRIG B functions are meant to used on an occasional basis, not on a constant basis.
- IRIG B time is not available for all boards. See **IRIG B Functions** on page 3-1.

IRIG B Time Tag Format

When using the IRIG B Time Tag, the Time Tag is made up of four 16-bit Words.

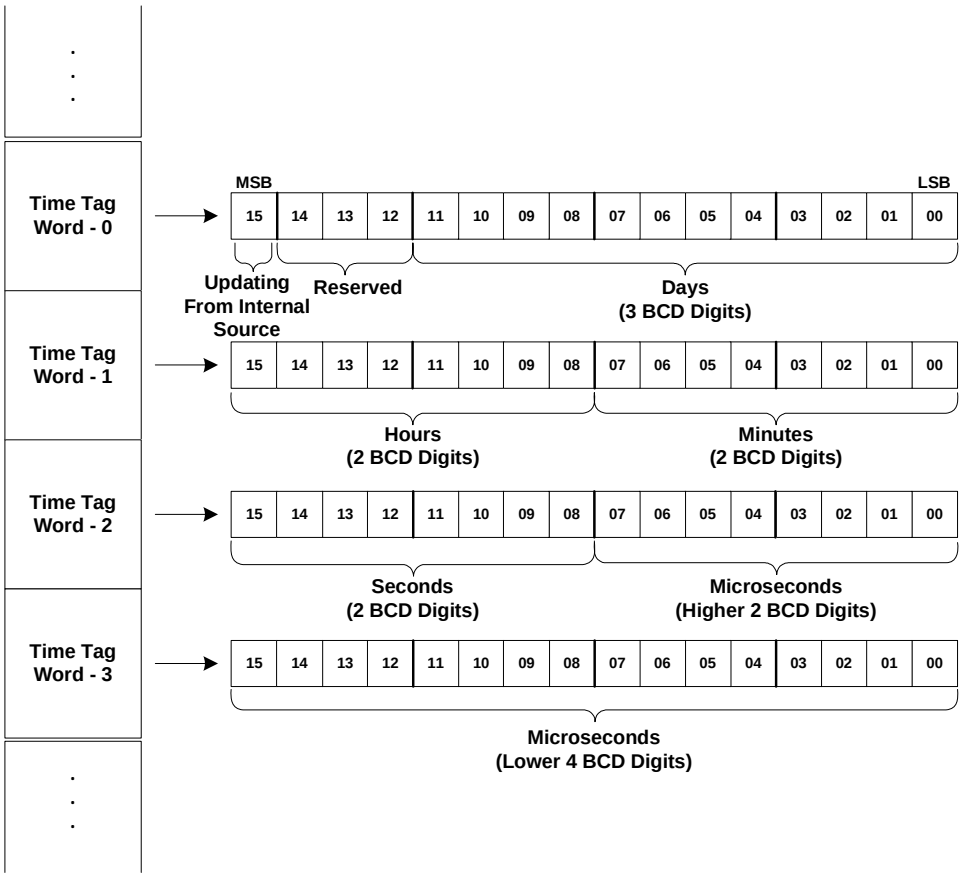

Figure 3-1 Time Tag Word Format in IRIG B Mode

The Updating from Internal Source bit indicates whether a valid 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.

GetIrigControl_4000

Description	GetIrigControl_4000 gets the 27 bits from the Control functions field in the IRIG B word.	
Syntax	GetIrigControl_4000 (int `handle`, unsigned long `*control`)	
Input Parameters	`handle`	The handle to the specified board, as returned by Init_Timers_4000
Output Parameters	`control`	A pointer to the 27 bits of the Control functions field received from the IRIG B bus.
Return Values	eopenkernel	Cannot open kernel device; check the **ExcConfig** settings
	ekernelcantmap	Kernel driver cannot map memory
	enotforexcard	ExCARD does not support this function
	0	If successful

GetIrigSeconds_4000

Description	Once IRIG B time is available, GetIrigSeconds_4000 returns the IRIG B time in seconds.	
Syntax	GetIrigSeconds_4000 (int `handle`, unsigned long `*seconds`)	
Input Parameters	`handle`	The handle to the specified board, as returned by Init_Timers_4000
Output Parameters	`seconds`	A pointer to the IRIG B time in seconds
Return Values	eopenkernel	Cannot open kernel device; check the **ExcConfig** settings
	ekernelcantmap	Kernel driver cannot map memory
	enotforexcard	ExCARD does not support this function
	0	If successful

GetIrigTime_4000

Description	Once IRIG B time is available, GetIrigTime_4000 returns a structure of 16-bit words in the form of days/hours/minutes/ seconds.	
Syntax	GetIrigTime_4000 (int `handle`, t_IrigTime `*IrigTime`)	
Input Parameters	`handle`	The handle to the specified board, as returned by Init_Timers_4000
Output Parameters	`IrigTime`	A pointer to the IRIG B days [1– 366, number of days since January 1]: hours: minutes: seconds
Return Values	eopenkernel	Cannot open kernel device; check the **ExcConfig** settings
	ekernelcantmap	Kernel driver cannot map memory
	enotforexcard	ExCARD does not support this function
	0	If successful

GetIrigYear_4000

Description	GetIrigYear_4000 returns the two-digit year component of the IRIG B time, when IRIG B time is available.	
Syntax	GetIrigYear_4000 (int `handle`, WORD `*year`)	
Input Parameters	`handle`	The handle to the specified board, as returned by Init_Timers_4000
Output Parameters	`year`	A pointer to the two-digit year value received by the IRIG B receiver
Return Values	ebaddevhandle	If invalid handle specified. Use value returned by Init_Timers_4000
	enotforexcard	ExCARD does not support this function
	0	If successful

IsIrigTimeavail_4000

Description	IsIrigTimeavail_4000 indicates if IRIG B time is available. Must be called after SetIrig_4000.	
Syntax	IsIrigTimeavail_4000 (int `handle`, int `*availFlag`)	
Input Parameters	`handle`	The handle to the specified board, as returned by Init_Timers_4000
Output Parameters	`availFlag`	1 IRIG B Time came in 0 IRIG B Time not yet available
Return Values	eopenkernel	Cannot open kernel device; check the **ExcConfig** settings
	ekernelcantmap	Kernel driver cannot map memory
	enotforexcard	ExCARD does not support this function
	0	If successful

SetIrig_4000

Description	SetIrig_4000 sets the board to receive IRIG B time.	
Syntax	SetIrig_4000 (int `handle`, WORD `flag`)	
Input Parameters	`handle`	The handle to the specified board, as returned by Init_Timers_4000
	`flag`	IRIG_TIME_AND_RESET Sets the carrier board to receive IRIG B time. In addition for synchronization purposes, Time Tags on all modules on the carrier board will be reset to 0 when the IRIG B time comes in [0100 H]. IRIG_TIME Sets the carrier board to get IRIG B time but does not do a Time Tag reset [0200 H].
Output Parameters	`none`	
Return Values	eopenkernel	Cannot open kernel device; check the **ExcConfig** settings
	ekernelcantmap	Kernel driver cannot map memory
	enotforexcard	ExCARD does not support this function
	0	If successful

Using Timer Functions

To use the Timer functions:

1. Call **Init_Timers_4000** on page 3-17 before calls to carrier board Timer and IrigB functions. This function returns a `handle`, which is used as the first parameter in all IRIG B, Timer and Interrupt functions.
2. Start Timer with the desired options. See **StartTimer_4000** on page 3-19.
3. *Either* keep checking until the Timer stops running. See **IsTimerRunning_4000** on page 3-17.

   or

4. Set an interrupt on Timer end and set up a thread to check if an interrupt occurred. See **Wait_For_Interrupt_P4000** on page 3-24. (Timer interrupts are not available for the *UNET* family.)
5. After completing your application, call **Release_Timers_4000** on page 3-18, to release Timer and IrigB functions.

The **`demo_timer.c`** program, on the *Excalibur Installation CD*, demonstrates the use of the Timer functionality.

Init_Timers_4000

Description Init_Timers_4000 must be called before calls to carrier board Timer and IrigB functions.

Syntax Init_Timers_4000 (WORD `device_num`, int `*handle`)

Input Parameters

`device_num`	The device number assigned to the board using the **ExcConfig** utility, or the default device number. You can use the default device number or the corresponding #define value when only one board with this default device number exists in the computer. For example, for a single *EXC-4000PCI[e]* board you can use the value 25 or the #define value **EXC_4000PCI**. For more information, see **Assigning a Device Number** on page 1-3.

Output Parameters

`handle`	A pointer to the handle to the specified board

Return Values

edevtoomany	If called too many boards
ekernelnot4000card	If the designated board is not a 4000/ 8000 family board
enotimersirig	If Timers and IrigB not supported on this version of carrier board
0	If successful

IsTimerRunning_4000

Description IsTimerRunning_4000 returns 1 if the Timer is running; 0 otherwise. This can be easily used in a loop, to wait for the Timer to end. (This is recommended for use when not in automatic restart mode.) This is also useful for multiple modules on the board, to check the availability of the Timer.

Syntax IsTimerRunning_4000 (int `handle`, int `*errorcondition`)

Input Parameters

`handle`	The handle to the specified board, as returned by Init_Timers_4000

Output Parameters

`errorconditon`	Error condition value:	
	einvalidOS	If invalid operating system
	ebaddevhandle	If handle is out of bounds or if no board has been allocated (this happens at the end of function call Init_Timers_4000)
	0	no error

IsTimerRunning_4000 (cont.)

Return Values	`1`	If Timer is running – Timer value is greater than 0
	`0`	If Timer is not running – Timer value is 0. (If value is 0, you can check the output value to see if there is an error condition.)

ReadTimerValue_4000

Description	ReadTimerValue_4000 returns how many µsecs. are left until the Timer completes.	
Syntax	ReadTimerValue_4000 (int `device_num`, DWORD `*timervalue`)	
Input Parameters	`device_num`	The device number assigned to the board using the **ExcConfig** utility, or the default device number. You can use the default device number or the corresponding #define value when only one board with this default device number exists in the computer. For example, for a single *EXC-4000PCI[e]* board you can use the value 25 or the #define value **EXC_4000PCI**. For more information, see **Assigning a Device Number** on page 1-3.
Output Parameters	`timervalue`	A pointer to the current value of the Timer in µsecs.
Return Values	eopenkernel	If cannot open kernel device; check **ExcConfig** settings
	ekernelcantmap	If kernel driver cannot map memory
	0	If successful

Release_Timers_4000

Description	Call Release_Timers_4000 to release Timer and IrigB functions.	
Syntax	Release_Timers_4000 (int `handle`)	
Input Parameters	`handle`	The handle to the specified board, as returned by Init_Timers_4000
Output Parameters	none	
Return Values	ebaddevhandle	If invalid handle specified. Use value returned by Init_Timers_4000
	0	If successful

ResetWatchdogTimer_4000

Description ResetWatchdogTimer_4000 stops and restarts the Timer with the same values as the last time StartTimer_4000 was called.

Syntax ResetWatchdogTimer_4000 (int handle)

Input Parameters

handle	The handle to the specified board, as returned by Init_Timers_4000

Output Parameters none

Return Values

etimmernotrunning	Timer not running when ResetWatchdogTimer_4000 was called
eopenkernel	If cannot open kernel device; check **ExcConfig** settings
ekernelcantmap	If kernel driver cannot map memory
0	If successful

StartTimer_4000

Description StartTimer_4000 starts the Timer on the carrier board.

Syntax StartTimer_4000 (int handle,
unsigned long microsecsToCount, int reload_flag,
int interrupt_flag, int globalreset_flag,
int *timeoffset)

Input Parameters

handle	The handle to the specified board, as returned by Init_Timers_4000
microsecsToCount	Time to count in µsecs.
reload_flag	TIMER_RELOAD To automatically restart the Timer when it completes [0001 H] TIMER_NO_RELOAD Do *not* automatically restart the Timer when it completes [0000 H]
interrupt_flag	TIMER_INTERRUPT To cause an interrupt when or each time the Timer completes [0001 H] (not available for the *UNET* family) TIMER_NO_INTERRUPT Do *not* cause an interrupt when or each time the Timer completes [0000 H]
globalreset_flag	TIMER_GLOBRESET To cause a global reset when or each time the Timer completes [0001 H] TIMER_NO_GLOBRESET Do *not* cause a global reset when or each time the Timer completes [0000 H]

StartTimer_4000 (cont.)

Output Parameters	`timeoffset`	`>0`	How much the actual time being counted is greater than the time to count requested
		`<0`	How much the actual time being counted is less than the time to count requested
		`0`	If the same
Return Values	eparmreload		Illegal parameter used for `reload_flag`
	eparminterrupt		Illegal parameter used for `interrupt_flag`
	eparmglobalreset		Illegal parameter used for `globalreset_flag`
	etimerrunning		Timer is already running when StartTimer_4000 was called
	0		If successful

StopTimer_4000

Description	StopTimer_4000 stops the Timer immediately.	
Syntax	StopTimer_4000 (int `handle`, unsigned long `*timervalue`)	
Input Parameters	`handle`	The handle to the specified board, as returned by Init_Timers_4000
Output Parameters	`timervalue`	A pointer to the last value of Timer, in µsecs.
Return Values	eopenkernel	If cannot open kernel device; check **ExcConfig** settings
	ekernelcantmap	If kernel driver cannot map memory
	0	If successful

Using Interrupts Under 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_P4000, in order to wait for the next interrupt. When the function returns, the interrupt is processed as needed. This method is demonstrated in the demo program **`demo_int.c`**, which is included with the *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_P4000 function may be used to determine if multiple interrupts have occurred. Conversely, it is possible that the Wait_For_Interrupt_P4000 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_P4000 function and the call to Get_Interrupt_Count_P4000. Once again, the Get_Interrupt_Count_P4000 function may be used to determine if the interrupt has already been processed.

The following functions are described below:

Get_Interrupt_Count_P4000
InitializeInterrupt_P4000
Wait_For_Interrupt_P4000

Note: The interrupts in this manual are Timer interrupts, and are not available for the *UNET* family.

Get_Interrupt_Count_P4000

Description

Get_Interrupt_Count_P4000 returns the total interrupt count for the specified device from the time the device was opened with OpenKernelDevice.

Note: This function is not available for the *UNET* family.

Syntax

Get_Interrupt_Count_P4000 (int `handle`, unsigned long `*pdwInterruptCount`)

Input Parameters

`handle`	The handle to the specified board, as returned by Init_Timers_4000

Output Parameters

`pdwInterruptCount`	A pointer to an unsigned long which receives the interrupt count

Return Values

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_P4000

Description

InitializeInterrupt_P4000 may be called to initialize interrupt handling for the given board. In general, it is not necessary to call this function since the Wait_For_Interrupt_P4000 function initializes the interrupt handling automatically when they are first called. However, in certain situations, such as a program in which the Wait_For_Interrupt_P4000 function is not run in a separate thread but is executed sequentially in the main thread, one may wish to initialize the interrupt handling before calling Wait_For_Interrupt_P4000. Thus, if an interrupt occurs after the initialization but before the call to Wait_For_Interrupt_P4000, the latter call will return immediately, reporting the interrupt which occurred previously.

Note: This function is not available for the *UNET* family.

Note

If a single module is specified use the InitializeInterrupt_[*module specific*] function.

Syntax

InitializeInterrupt_P4000 (int `handle`)

Input Parameters

`handle`	The handle to the specified board, as returned by Init_Timers_4000

Output Parameters

none

Return Values

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
0	If successful

Wait_For_Interrupt_P4000

Description

Wait_For_Interrupt_P4000 waits for an interrupt on the module/board. 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.

Note: This function is not available for the *UNET* family.

Syntax

Wait_For_Interrupt_P4000 (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_P4000 (device_num, 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…
     // Check total number of interrupts
    Get_Interrupt_Count_P4000 (device_num, &numints);
  }
}
```

Input Parameters

`handle`	The handle to the specified board, as returned by Init_Timers_4000
`timeout`	Timeout is specified in milliseconds, or INFINITE

Output Parameters

none

Return Values

	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	If successful

4 VME/VXI Carrier Board Level Functions

This chapter lists the functions that apply to the Excalibur VME/VXI boards listed on the front cover of this manual. These functions apply to the carrier board and to all the modules on the board. The functions are not module or mode specific. To write applications for individual modules, refer to the specific module's *Programmer's Reference*.

General VME/VXI Functions	Set_IRQ_Line_V4000	See **General VME/VXI Functions** on page 4-2
Using Interrupts Under VISA for VME/VXI Carrier Boards	Setup_Interrupt_Handler_V4000 Unset_Interrupt_Handler_V4000	See **Using Interrupts Under VISA for VME/VXI Carrier Boards** on page 4-2
***EXC-1553ccVME/Px*-Specific Functions**	Modules_Ready_CCVMEPX	See **EXC-1553ccVME/Px-Specific Functions** on page 4-5

General VME/VXI Functions

Set_IRQ_Line_V4000

Description	Set_IRQ_Line_V4000 sets the IRQ interrupt line.	
Syntax	Set_IRQ_Line_V4000 (WORD `device_num`, WORD `irq`)	
Input Parameters	`device_num`	The device number of the board. Specify the logical address (0–255) according to the DIP switch settings (or jumper settings for *EXC-1553ccVME/Px* boards).
	`irq`	The desired IRQ interrupt line.
Return Values	edevnum	If the value for `device_num` is invalid
	einstr	If the device was not initialized
	einstallhandler	viInstallHandler returned an error
	0	If successful

Using Interrupts Under VISA for VME/VXI Carrier Boards

When writing a program under VISA that processes interrupts, the system must be set up to enable signal processing events so the program can receive them into its event handler. The programmer provides this interrupt handler (or 'interrupt service function') as part of the application program. There the user can place any code that is to run in response to receiving an interrupt.

Two functions are provided in the dll **`EXCv4000.dll`**, (source code in **`EXCv4000.c`**) to set and cancel the user's interrupt handler:

Setup_Interrupt_Handler_V4000
Unset_Interrupt_Handler_V4000

The module's *Software Tools* include a test program **`demo_int.c`** that contains an event handler routine IHandler that is called whenever an interrupt event is received. It is a callback function, and it *must* be defined as specified in the demo program even though the parameters will not be used in the routine. The demo demonstrates the use of Setup_Interrupt_Handler_V4000 and Unset_Interrupt_Handler_V4000.

Setup_Interrupt_Handler_V4000

Description

Setup_Interrupt_Handler_V4000 assigns an interrupt handler callback routine to handle interrupts for the specified device.

Syntax

Setup_Interrupt_Handler_V4000 (WORD `device_num`, ViHndlr `IHandler`)

Input Parameters

`device_num`	The device number of the board. Specify the logical address (0–255) according to the DIP switch settings (or jumper settings for *EXC-1553ccVME/Px* boards).
`IHandler`	A pointer to an interrupt handler routine within the application program. Note that this routine must have the following prototype:

ViStatus _VI_FUNCH IHandler (ViSession `instr`, ViEventType `etype`, ViEvent `event`, ViAddr `userhandle`)

Output Parameters

none

Return Values

edevnum	If the value for `device_num` is invalid
einstr	If the device was not initialized
einstallhandler	viInstallHandler returned an error
eenableevent	viEnableEvent returned an error
0	If successful

Unset_Interrupt_Handler_V4000

Description Unset_Interrupt_Handler_V4000 uninstalls the handler that was previously installed to handle interrupt for the specified device. See **Setup_Interrupt_Handler_V4000** on page 4-3.

Syntax Unset_Interrupt_Handler_V4000 (WORD `device_num`, ViHndlr `IHandler`)

Input Parameters

`device_num`	The device number of the board. Specify the logical address (0–255) according to the DIP switch settings (or jumper settings for *EXC-1553ccVME/Px* boards).
`IHandler`	A pointer to the interrupt handler routine previously assigned (by Setup_Interrupt_Handler_V4000) to handle interrupts for the specified device.

Output Parameters none

Return Values

edevnum	If the value for `device_num` is invalid
einstr	If the device was not initialized
euninstallhandler	Error returned by viUninstallHandler
0	If successful

EXC-1553ccVME/Px-Specific Functions

The following functions are available only for the *EXC-1553ccVME/Px*. These functions apply to the carrier board and to all the modules on the board. The functions are not module or mode specific. To write applications for individual modules, refer to the *1553Px Family Software Tools Programmer's Reference*.

Modules_Ready_CCVMEPX

Description Modules_Ready_CCVMEPX checks the *EXC-1553ccVME/Px* board and returns a value indicating which modules on the board are present and operational.

Syntax Modules_Ready_CCVMEPX (WORD device_num, WORD `*modules_ready`)

Input Parameters

`device_num`	The device number of the board. Specify the logical address (0–255) according to the DIP switch settings (or jumper settings for *EXC-1553ccVME/Px* boards).

Output Parameters

`modules_ready`	Each bit refers to one module location: bit 0 refers to module 0; bit 15 refers to module 15. A bit is set to 1 when the corresponding module is present and operational.

Return Values

efuncinvalid	If the board is not an *EXC-1553ccVME/Px*
edevnum	If the value for `device_num` is invalid
eopendefaultrm	Error opening the default RM
eviopen	Error opening VISA
evimapaddress	Error mapping address
0	If successful

Appendix A *Excalibur Carrier Board Software Tools* Library

Appendix A includes a list of the files in the *Excalibur Carrier Board Software Tools* needed to write user-defined applications. The files are divided into three categories:

Source code and Header files for the *Excalibur Carrier Board 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 *Excalibur Carrier Board Software Tools.* They 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

***Excalibur Carrier Board Software Tools* Library for PCI[e] Carrier Board**

	File Name	Description
Source Code Files	**deviceio.c**	Functions relating to resources such as memory and interrupts – these files act on kernel drivers
	exc4000.c	Source code for Excalibur PCI[e] carrier board functions
Source Header files	**deviceio.h**	Header file for interaction with kernel driver
	error_devio.h	Header file containing error codes for the Excalibur module kernel drivers
	exc4000.h	Header file for Excalibur PCI[e] carrier board functions
Demo Programs	**demo_irig.c**	Demo program to demonstrate the use of IRIG B functionality
	demo_timer.c	Demo program to demonstrate the use of the Timer functionality
DLL and *.LIB files	The DLL and LIB files have the same filename except for the file extension. The filename is: **Exc4000Ms** In addition, a module level DLL is required for each module on the board for which you are writing applications. See the module's programmer's reference.	

***Excalibur Carrier Board Software Tools* Library for VME/VXI Carrier Board**

	File Name	Description
Source Code File	**excv4000.c**	Source code for Excalibur VME/VXI carrier board functions
Source Header files	**error_devio.h**	Header file containing error codes for the Excalibur module kernel drivers
	excv4000.h	Header file for Excalibur VME/VXI carrier board functions
DLL and *.LIB files	The DLL and LIB files have the same filename except for the file extension. The filename is: **ExcV4000Ms** In addition, a module level DLL is required for each module on the board for which you are writing applications. See the module's programmer's reference.	

Appendix B Error Messages

All functions in *Excalibur Carrier Board Software Tools* are written as C functions, i.e., they return values. A negative value signifies an error. Below is a list of all *Software Tools* error messages, the value of each, and an explanation of the error.

Error Code	Value	Explanation
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)
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 **readme.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

Error Code	Value	Explanation
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 board 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
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 a 4000/8000 board
enotimersirig	-1042	Timers and IrigB are not supported on this version of the 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

Error Code	Value	Explanation
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
einvalttagsource	-1087	Invalid Time Tag source specified
enotforexcard	-1088	ExCARD does not support this function
enotforunet	-1089	UNET devices do not support this function or option
eunsupportedfeature	-1090	This board does not support the requested feature
eiriggeneratorrunning	-1091	The attempted operation cannot be performed while the IRIG Generator is running
enotremotedevice	-1092	The specified device is not a remote (i.e., UNET family) device
enotmaccdevice	-1093	The specified device is not a MACC (ES-9800 MACC II) device
enotnetdevice	-1094	The specified device is not a NET-Connected device
enotusbdevice	-1095	The specified device is NOT a USB-Connected device
ereadunetdevice	-1096	Error reading from the UNET device
efunctnotforallunetdevnums	-1097	This function does not support UNET devices above device number 15
eNoZeroBit	-1098	Error reading IRIG B007 middle bit; not set to zero
eunrecognizedboard	-1099	Unrecognized carrier board
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

Error Code	Value	Explanation
evimapaddress	-1054	Error mapping address
evicommand	-1055	Error in VISA command
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

Function Index

April 2024, Rev B-3