Serial
Software Tools

Programmer's Reference

EXCALIBUR SYSTEMS

311 Meacham Avenue ◆ 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 Excalibur's *Serial Software Tools*.

The following topics are covered:

Overview

Serial Software Tools enables the user to create custom diagnostic programs for the *Serial* module.

The *Serial* module is an interface module for the multimode, multiprotocol Excalibur EXC-4000/8000 family of carrier boards. This module supports up to 4 independent channels of serial communications, each of which can be selected as RS485, RS422 or RS232. The on-board FIFOs offer a simple interface to the module. The module operates independently of the host computer, reducing the need for host intervention.

Each of the *M8KSerial* and *M4KSerialPlus* modules' channels contain a Transmit FIFO capable of holding up to 16,384 data bytes; 8,192 for *M4KSerial*.

The limit of the Receive FIFO varies depending on the module and the module's settings.

For the *M8KSerial* and *M4KSerialPlus* modules, the Receive FIFO options are:

- 32,768 input data bytes without their associated Status Words or Time Tags
- 16,384 data bytes with their associated Status Words
- 5461 data bytes with their associated 32-bit Time Tags
- 5461 data bytes with their associated 32-bit Time Tags and Status Words
- 3276 data bytes with their associated 64-bit IRIG B Time Tags and Status Words (*M8KSerial* only)

For the *M4KSerial* module, the Receive FIFO options are:

- 16,384 input data bytes without their associated Status Words or Time Tags
- 8,192 data bytes with their associated Status Words
- 3276 data bytes with their associated 32-bit Time Tags
- 2730 data bytes with their associated 32-bit Time Tags and Status Words

These large buffers allow application code to transfer large quantities of data with a single non-blocking subroutine call. The 32-bit Time Tag on receive channels is

very useful for synchronizing serial data with data from other modules on the carrier board or on other interface boards.

Getting Started

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	None – the board's device number must be set via **ExcConfig**	N/A
VME, VPX	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
MCH	29 (dec)	EXC_1553PCIMCH
All Other Boards in this Manual	25 (dec)	EXC_4000PCI

Serial Software Tools

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

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

Board-Level Software Tools

This manual describes the module-level software functions. Board-level software functions are described in the *EXC-4000 Family Carrier Boards Software Tools Programmer's Reference*.

Compiler Options

The DLL is compiled under Microsoft Visual Studio using **_cdecl** calling convention. The driver functions in the *Serial 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, therefore multiple programs cannot access the same module simultaneously.

Each function is presented with its formal definition, including data types of all input and output variables, including flag values. 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_SER function. (See **Appendix C: Error Messages**.)

In Windows all user-defined programs must include the file **`proto_ser.h.`** this file includes all the necessary header files and DLL functions prototypes to operate *Serial Software Tools* for Serial modules.

Direct Memory Access

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

When DMA is available, the *Serial Software Tools* use DMA access for functions that read from and write to module memory.

The following functions use DMA reads:

TransmitString_SER (when sending 256 bytes or more in a single function call)

NumBytesWaiting_SER (when reading 16 bytes or more in a single function call)

See also:

SetUseDmaIfAvailable_SER

GetUseDmaIfAvailable_SER

Conventions Used in this Manual

To help differentiate between different kinds of information, the following text styles are used in the *Serial Software Tools*:

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 Initialization Functions

Chapter 2 contains descriptions of the initialization functions necessary to write test programs for the *Serial* module. Each function is presented with its formal definition, data types of all input parameters, including legal values where applicable, and output variables.

The following functions are described in this chapter:

Get_Error_String_SER

Description Get_Error_String_SER accepts the error returns from other *Serial Software Tools* functions. This function returns the string containing a corresponding error message. See **Appendix C: Error Messages**.

Syntax Get_Error_String_SER (`int` errcode, `int` errlen, `char` *errstring)

Example

```
char Errorstr [255] ;
Get_ErrorString_SER (errorcode, ERRORLEN, &Errorstr) ;
printf("error is: %s", ErrorStr) ;
```

Input Parameters

errorcode	The error code returned from a *Serial Software Tools* function.
errlen	The maximum length of a string to be returned.

Note: If the actual error is not longer than the length specified, the error string returned will end with a 'new line' character.

Output Parameters

errorstring	A string of characters, with the corresponding error message. In case of bad input, a string denoting that.

Return Values

0	Always

GetHWid_SER

Description	GetHWid_SER returns the module revision as an integer value.	
Syntax	GetHWid_SER (`int` devhandle, `int` *hwid)	
Input Parameters	devhandle	The handle designated by Init_Module_SER
Output Parameters	hwid	Returns the current revision of the module
Return Values	ebadhandle	If an invalid handle was specified; should be the value returned by Init_Module_SER
	0	If successful

GetHWidStr_SER

Description	GetHWidStr_SER returns the complete module revision as a string (array of chars).	
Syntax	GetHWidStr_SER (`int` devhandle, `char const` **ppHwidStr)	
Input Parameters	devhandle	The handle designated by Init_Module_SER
Output Parameters	pHwidStr	A pointer to a pointer to the module revision
Return Values	ebadhandle	If an invalid handle was specified; should be the value returned by Init_Module_SER
	ewrngmodule	If module specified on the board is not a *Serial* module
	0	If successful

GetModuleTimetag_SER

Description	Use GetModuleTimetag_SER to get the current value of the running Time Tag on the specified module.	
Syntax	GetModuleTimetag_SER (`int` devhandle, `DWORD` timetag)	
Input Parameters	devhandle	The handle designated by Init_Module_SER
Output Parameters	timetag	The current value of the running Time Tag
Return Values	ebadhandle	If an invalid handle was specified; should be the value returned by Init_Module_SER
	0	If s<$chapnumuccessful

Init_Module_SER

Description

Init_Module_SER is the first function the user must call for each *Serial* module on each device that is accessed in the application program.

Before exiting a program, call Release_Module_SER.

On **PCI** carrier boards, Init_Module_SER enables the user to access up to four modules on a single board or any combination of up to 16 modules on four separate boards.

On **VME/VXI** carrier boards, Init_Module_SER enables the user to access up to eight modules on a single board or any combination of up to 32 modules on four separate boards.

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.

On **PCI** carrier boards up to 17 real or SIMULATED modules may be initialized.

On **VME/VXI** carrier boards up to 33 or SIMULATED modules may be initialized.

More than one module may be SIMULATED simultaneously.

Syntax

Init_Module_SER (int `device_num`, int `module_num`)

Input parameters

`device_num`

For **PCI**:
The device number is the number assigned to the board in **`ExcConfig`**:
0 – 15
or
SIMULATE Indicating a virtual module [FFFF H]

For **VME/VXI:**
The device number is the number assigned to the board via the board's DIP Switches: 0 – 255
or
SIMULATE Indicating a virtual module [FFFF H]

`module_num`

The module number of the *Serial* module on the board: 0 - 7 (depending on the board)

Init_Module_SER (cont.)

Note: If only one board is used, the define value EXC_4000PCI (default device number = 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.

Output parameters none

Return Values

eopenkernel	If there was an error opening a device
emodnum	Invalid module number specified
enomodule	If no module is present at specified location
ewrngmodule	If module specified on the board is not a *Serial* module
etimeoutreset	If timed out waiting for reset
eboardtoomany	If too many boards initialized
sim_no_mem	If not enough memory available for simulation
devhandle	If successful, the handle to the specified module on the board. For **PCI:** A valid handle is a positive number ranging from 0 – 16. For **VME/VXI:** A valid handle is a positive number ranging from 0 – 32.

ModuleTimetagReset_SER

Description ModuleTimetagReset_SER resets the running Time Tag on the specified module.

Note: The same Time Tag is used for all channels on the module.

Syntax ModuleTimetagReset_SER (`int` devhandle)

Input Parameters

devhandle	The handle designated by Init_Module_SER

Output Parameters none

Return Values

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

Release_Module_SER

Description

Release_Module_SER releases any grabbed resources on a specific module.

Before exiting a program, call this function for each module initialized with Init_Module_SER.

To continue accessing the module, call Init_Module_SER.

Syntax

Release_Module_SER (`int` devhandle)

Input Parameters

devhandle	The handle designated by Init_Module_SER.

Output Parameters

none

Return Values

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

3 Configuring Serial Channels

Chapter 3 contains descriptions of the functions necessary to configure the serial channels to write test programs for the *Serial* module. Each function is presented with its formal definition, data types of all input parameters, including legal values where applicable, and output variables.

The following functions are described in this chapter:

AlterChannelParams_SER
ChannelErrstatEnable_SER
ChannelTimetagEnable_SER
ClearRcvCountTrigFlag_SER
ClearRcvPatternTrigFlag_SER
GetDIPVal_SER
GetDIPValStr_SER
GetHiFreqOscMhz_SER
GetModuleTimetagResolution_SER
GetUseDmaIfAvailable_SER
IsHiFreqOscKnown_SER
NumChannels_SER
ResetChannel_SER
SetHiFreqOscMhz_SER
SetModuleTimetagResolution_SER
SetRcvCountTrigger_SER
SetRcvPatternTrigger_SER
SetRS485Channel_SER
SetTransmitGap_SER
SetupChannel_SER
SetUseDmaIfAvailable_SER

AlterChannelParams_SER

Description

AlterChannelParams_SER alters the parameters previously set by SetupChannel_SER.

Syntax

AlterChannelParams_SER (`int` chanhandle, `unsigned int` baud, `enum Parity` parity, `enum DataBits` databits, `enum StopBits` stopbits, `int` *bauderr)

Input Parameters

chanhandle — The handle designated by SetupChannel_SER

baud — Greater than 0, up to 1,500,000 bps

Note: For **RS-232**, maximum recommended baud rate is 1,000,000 bps

or

BAUD_NO_CHANGE
For channel to continue using the same baud rate set in SetupChannel_SER [0000 H]

parity — Specifies the parity scheme to be used:

NoParity	none
OddParity	odd
EvenParity	even
MarkParity	mark
SpaceParity	space
ParityNoChange	No change in parity

databits — Specifies the number of databits to be used:

FiveDataBits	5 databits
SixDataBits	6 databits
SevenDataBits	7 databits
EightDataBits	8 databits
DataBitsNoChange	No change in databits

stopbits — Specifies the stopbits to be used

OneStopBit	1 stopbit
NotOneStopBit	Stopbits depend on databits: Set 1.5 stopbits if databits = 5 Set 2 stopbits otherwise
StopBitsNoChange	No change in stopbits

AlterChannelParams_SER (cont.)

Output Parameters	bauderr	Not all the baud rates can be attained exactly by the oscillators on the module. The bauderr is the percent error from the specified baud rate to the actual baud rate. For more details, see **Programmable Baud Rate Generator** in the module's user's manual.
Return Values	ebadchanhandle	If an invalid handle was specified; should be the value returned by SetupChannel_SER
	einval	If an invalid parameter was used as an input
	ebauderr	If the requested baud rate could not be generated exactly; the actual baud rate was off by <bauderr>%
	0	If successful

ChannelErrstatEnable_SER

Description	When ChannelErrstatEnable_SER is enabled, each byte received by the specified channel has the error status field filled in with the current byte's status.	
Syntax	ChannelErrstatEnable_SER (`int` chanhandle, `BOOL` enableflag)	
Input Parameters	chanhandle	The handle designated by SetupChannel_SER
	enableflag	Legal flags allowed: ENABLE To enable the function [0001 H] DISABLE To disable the function [0000 H]
Output Parameters	none	
Return Values	ebadchanhandle	If the specified handle is not a channel handle that was allocated by SetupChannel_SER
	0	If successful

ChannelTimetagEnable_SER

Description When ChannelTimetagEnable_SER is enabled, each byte received by the specified channel has the Time Tag field filled in with the current Time Tag.

Syntax ChannelTimetagEnable_SER (`int` chanhandle, `BOOL` enableflag)

Input Parameters

chanhandle	The handle designated by SetupChannel_SER
enableflag	Legal flags allowed: ENABLE To enable the function [0001 H] DISABLE To disable the function [0000 H]

Output Parameters none

Return Values

ebadchanhandle	If an invalid handle was specified; should be the value returned by SetupChannel_SER
0	If successful

ClearRcvCountTrigFlag_SER

Description ClearRcvCountTrigFlag_SER clears the flag that indicates that the receive count was reached.

Note: This function is only available for *M8KSerial* and *M4KSerialPlus* modules.

Syntax ClearRcvCountTrigFlag_SER (`int` chanhandle)

Input Parameters

chanhandle	The handle designated by SetupChannel_SER

Output Parameters none

Return Values

ebadchanhandle	If an invalid handle was specified; should be the value returned by SetupChannel_SER
eunsupportedfeature	If the installed module does not support this feature
0	If successful

ClearRcvPatternTrigFlag_SER

Description ClearRcvPatternTrigFlag_SER clears the flag that indicates that the pattern match was reached.

Note: This function is only available for *M8KSerial* and *M4KSerialPlus* modules.

Syntax ClearRcvPatternTrigFlag_SER (`int` chanhandle)

Input Parameters	chanhandle	The handle designated by SetupChannel_SER
Output Parameters	none	
Return Values	ebadchanhandle	If an invalid handle was specified; should be the value returned by SetupChannel_SER
	eunsupportedfeature	If the installed module does not support this feature
	0	If successful

GetDIPVal_SER

Description GetDIPVal_SER returns the protocol value that the DIP switches are set to.

Syntax GetDIPVal_SER (`int` chanhandle, `WORD` *DIPval)

Input Parameters	chanhandle	The handle designated by SetupChannel_SER	
Output Parameters	DIPval	DIPVAL_RS232	[0x0000]
		DIPVAL_RS422	[0x0800]
		DIPVAL_RS485	[0x1800]
		DIPVAL_INVALID	[0x1000]
Return Values	ebadchanhandle	If an invalid handle was specified; should be the value returned by SetupChannel_SER	
	func_invalid	Hardware version earlier than revision 4	
	0	If successful	

GetDIPValStr_SER

Description GetDIPValStr_SER returns the protocol as a string that the DIP switches are set to.

Syntax GetDIPValStr_SER (`int` chanhandle, `char const` **ppDipValStr)

Input Parameters	chanhandle	The handle designated by SetupChannel_SER
Output Parameters	ppDipValStr	A pointer to a pointer to the DIP switch value; one of the following values: RS-485, RS-422, RS-232 or Invalid DIP setting
Return Values	ebadchanhandle	If an invalid handle was specified; should be the value returned by SetupChannel_SER
	0	If successful

GetHiFreqOscMhz_SER

Description GetHiFreqOscMhz_SER retrieves the High Oscillator's frequency value that will be used in baud rate calculations. See also SetHiFreqOscMhz_SER on page 3-9.

Syntax GetHiFreqOscMhz_SER (`int` devhandle, `double` *pHiOscFreqMhz)

Input Parameters	devhandle	The handle returned by Init_Module_SER
Output Parameters	pHiOscFreqMhz	The High Oscillator frequency in MHz: FREQ40 40 MHz [40.0] FREQ32 32.0 MHz [32.0] FREQ24 24 MHz [24.0] FREQ16 16 MHz [16.0] FREQ14 14.7456 MHz [14.7456] FREQNONE Oscillator not available [0] **Note:** As of the time of this writing, these are the available high oscillator values.
Return Values	ebadhandle	If an invalid handle was specified; should be the value returned by Init_Module_SER
	0	If successful

GetModuleTimetagResolution_SER

Description GetModuleTimetagResolution_SER returns the module's Time Tag resolution.

Syntax GetModuleTimetagResolution_SER (`int` devhandle, `int` pTimetagResolution)

Input Parameters

devhandle	The handle returned by Init_Module_SER

Output Parameters

pTimetagResolution	A pointer to the Time Tag resolution: TICKTIME_10_MICROSEC 100 KHz (10 µsec resolution) [0000 H] TICKTIME_1_MICROSEC 1 MHz (1 µsec resolution) [0001 H]

Return Values

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

GetUseDmaIfAvailable_SER

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

Syntax GetUseDmaIfAvailable_SER (`int` devhandle, `int` *pUseDmaFlag)

Input Parameters

devhandle	The handle returned by Init_Module_SER

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]

Return Values

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

IsHiFreqOscKnown_SER

Description	IsHiFreqOscKnown_SER checks whether the High Oscillator's frequency value is known (based on the module version or carrier board), or whether it should be set by the user via SetHiFreqOscMhz_SER. For more information, see SetHiFreqOscMhz_SER on page 3-9.	
Syntax	IsHiFreqOscKnown_SER (`int` devhandle)	
Input Parameters	devhandle	The handle returned by Init_Module_SER
Output Parameters	none	
Return Values	ebadhandle	If an invalid handle was specified; should be the value returned by Init_Module_SER
	1	If the High Oscillator's frequency value is known
	0	If the High Oscillator's frequency value is not known

NumChannels_SER

Description	NumChannels_SER returns the number of Serial channels on the module.	
Syntax	NumChannels_SER (`int` devhandle, `int` *numchannels)	
Input Parameters	devhandle	The handle designated by Init_Module_SER
Output Parameters	numchannels	The number of channels: 2 or 4
Return Values	ebadhandle	If an invalid handle was specified; should be the value returned by Init_Module_SER
	0	If successful

ResetChannel_SER

Description	ResetChannel_SER resets a channel on the module.	
Syntax	ResetChannel_SER (`int` chanhandle)	
Input Parameters	chanhandle	The handle designated by SetupChannel_SER
Output Parameters	none	
Return Values	ebadchanhandle	If an invalid handle was specified; should be the value returned by SetupChannel_SER
	0	If successful

SetHiFreqOscMhz_SER

Description

SetHiFreqOscMhz_SER sets the value used for the High Oscillator in baud rate calculations. This function is only relevant to *M4KSerial* modules with module revisions 5–8. On these revision, the High Oscillator can either be 24.0 MHz or 32.0 MHz.

When you set a channel's baud rate, the *Software Tools* use the highest oscillator that matches the desired baud rate with the least amount of error.

When a module with revisions 5–8 is installed on a PCI carrier board, the value of the High Oscillator can either be 24.0 MHz or 32.0 MHz. When calculating the baud rate for these modules, the *Software Tools* use 24.0 MHz as the High Oscillator value unless this function is called to set it to 32.0 MHz.

When using a module with revisions 5–8 that has a 32.0 MHz High Oscillator installed on a PCI carrier board, use this function to ensure correct baud rate calculations. When a module with revisions 5–8 is installed on a PCIe carrier board, the High Oscillator must be 32.0 MHz and this function is not relevant.

In all other cases, current *Software Tools* (revision 2.7 or later) automatically determine the High Oscillator value based on the module revision and carrier board.

Note: This function is not available for *M8KSerial* and *M4KSerialPlus* modules.

Syntax

SetHiFreqOscMhz_SER (`int` devhandle, `double` *pHiOscFreqMhz)

Input Parameters

devhandle	The handle returned by Init_Module_SER	
pHiOscFreqMhz	The High Oscillator frequency in MHz:	
	FREQ32	32.0 MHz [32.0]
	FREQ24	24 MHz [24.0]

Output Parameters

none

Return Values

ebadhandle	If an invalid handle was specified; should be the value returned by Init_Module_SER
einvalidhioscfreq	If an invalid High Oscillator frequency was specified
efreqalreadyknown	If the High Oscillator frequency has already been determined based on the module revision or carrier board
eunsupportedfeature	If the installed module does not support this feature
0	If successful

SetModuleTimetagResolution_SER

Description

SetModuleTimetagResolution_SER sets the Time Tag clock frequency: 100 KHz (10 µsec resolution) or 1 MHz (1 µsec resolution).

Note: This function is only available for *M8KSerial* and *M4KSerialPlus* modules. The *M4KSerial* module is fixed at 100KHz (10 µsec resolution).

Syntax

SetModuleTimetagResolution_SER (`int` devhandle, `int` timetagResolution)

Input Parameters

devhandle	The handle returned by Init_Module_SER
timetagResolution	The Time Tag resolution: TICKTIME_10_MICROSEC 100 KHz (10 µsec resolution) [0000 H] TICKTIME_1_MICROSEC 1 MHz (1 µsec resolution) [0001 H]

Output Parameters

none

Return Values

ebadhandle	If an invalid handle was specified; should be the value returned by Init_Module_SER
eInvalidTtagRes	If an invalid Time Tag resolution specified
eIncompatibleTtagRes	If the module is an *M4KSerial* module
eunsupportedfeature	If the installed module does not support this feature
0	If successful

SetRcvCountTrigger_SER

Description

SetRcvCountTrigger_SER sets up the receive count feature so that an interrupt or external trigger can be sent when the specified receive count is reached. For information on using interrupts, see **Chapter 5: Using Interrupt Functions**.

When the specified receive count is reached, Bit 09 of the Channel Status Register is set.

To disable the receive count feature, use a rcvCount value of 0.

Note: This function is only available for *M8KSerial* and *M4KSerialPlus* modules.

Syntax

SetRcvCountTrigger_SER (`int` chanhandle, `unsigned int` rcvCount, `int` trigDest)

Input Parameters

chanhandle	The handle designated by SetupChannel_SER	
rcvCount	The desired receive count in bytes (1–65535); a value of 0 disables the receive count feature	
trigDest	The desired trigger destination; one or more of the following flags ORed together:	
	DEST_EXT_TRIG	Send an external trigger [0002 H]
	DEST_INTERRUPT	Send an interrupt [0001 H]
	DEST_NONE	Do not send anything [0000 H]

Output Parameters

none

Return Values

ebadchanhandle	If an invalid handle was specified; should be the value returned by SetupChannel_SER
emaxrcvcount	If the specified receive count exceeds the maximum
einvalidtrigdest	If the specified trigger destination is invalid
eunsupportedfeature	If the installed module does not support this feature
0	If successful

SetRcvPatternTrigger_SER

Description

SetRcvPatternTrigger_SER sets up the pattern match feature so that an interrupt or external trigger can be sent when a specified bit pattern is received a specified amount of times (not necessarily consecutively). For information on using interrupts, see **Chapter 5: Using Interrupt Functions**.

When the specified pattern is received the specified amount of times, Bit 08 of the Channel Status Register is set.

To disable the pattern match feature, use a patternCount value of 0.

Note: This function is only available for *M8KSerial* and *M4KSerialPlus* modules.

Syntax

SetRcvPatternTrigger_SER(`int` chanhandle, `unsigned char` pattern, `unsigned int` patternCount, `int` trigDest)

Input Parameters

chanhandle	The handle designated by SetupChannel_SER
pattern	The desired bit pattern (0–255)
patternCount	The desired pattern count in bytes (1–255); a value of 0 disables the pattern match feature
trigDest	The desired trigger destination; one or more of the following flags ORed together:

DEST_EXT_TRIG	Send an external trigger [0002 H]
DEST_INTERRUPT	Send an interrupt [0001 H]
DEST_NONE	Do not send anything [0000 H]

Output Parameters

none

Return Values

ebadchanhandle	If an invalid handle was specified; should be the value returned by SetupChannel_SER
emaxpatrncount	If the specified pattern-match count exceeds the maximum
einvalidtrigdest	If the specified trigger destination is invalid
eunsupportedfeature	If the installed module does not support this feature
0	If successful

SetRS485Channel_SER

Description SetRS485Channel_SER sets up a channel to transmit and receive via RS485 protocol.

Note: This function is obsolete for module revision 4 and later. This is done automatically when RS-485 protocol is used.

Syntax SetRS485Channel_SER (`int` chanhandle)

Input Parameters

chanhandle	The handle designated by SetupChannel_SER

Output Parameters none

Return Values

ebadchanhandle	If an invalid handle was specified; should be the value returned by SetupChannel_SER
0	If successful

SetTransmitGap_SER

Description SetTransmitGap_SER sets the gap time between transmitted bytes. By default, the gap time is 0.

Note: This function is only available for *M8KSerial* and *M4KSerialPlus* modules.

Syntax SetTransmitGap_SER (`int` chanhandle, `unsigned int` gapTimeMicroSec)

Input Parameters

chanhandle	The handle designated by SetupChannel_SER
gapTimeMicroSec	The desired gap time in microseconds (1–FFFF FFFF H); when 0 is used, the data is transmitted with no gap time

Output Parameters none

Return Values

ebadchanhandle	If an invalid handle was specified; should be the value returned by SetupChannel_SER
eunsupportedfeature	If the installed module does not support this feature
0	If successful

SetupChannel_SER

Description

SetupChannel_SER sets a specific channel of the module to receive and transmit.

Syntax

SetupChannel_SER (`int` devhandle, `int` channel, `unsigned int` baud, `enum Parity` parity, `enum DataBits` databits, `enum StopBits` stopbits, `int` protocol, `int` *chanhandle, `int` *bauderr)

Input Parameters

devhandle — The handle returned by Init_Module_SER

channel — The channel on the module:

0 – 1	2 channel module
0 or 2	2 channel module with isolated channels
0 – 3	4 channel module

baud — The baud rate in bits per second (bps), 1 – maximum baud rate. The maximum baud rate varies depending on the module. See the standard data rates table in your module's user's manual.

The *Serial Software Tools* automatically select the best oscillator for the selected baud rate.

Note: For RS-232, maximum recommended baud rate is 1,000,000 bps

parity — Specifies the parity scheme to be used:

NoParity	none
OddParity	odd
EvenParity	even
MarkParity	mark
SpaceParity	space

databits — Specifies the number of databits to be used:

FiveDataBits	5 databits
SixDataBits	6 databits
SevenDataBits	7 databits
EightDataBits	8 databits

stopbits — Specifies the stopbits to be used

OneStopBit	1 stopbit
NotOneStopBit	Stopbits depend on databits: Set 1.5 stopbits if databits = 5 Set 2 stopbits otherwise

SetupChannel_SER (cont.)

	protocol	Specifies the communication type.
		PROTOCOL_HWSET Select communication type according to DIP switch settings [0082 H]
		PROTOCOL_RS232 Ignore DIP switches and use RS-232 [0080 H]
		PROTOCOL_RS422 Ignore DIP switches and use RS-422 [0081 H]
		PROTOCOL_RS485 Ignore DIP switches and use RS-485 [0083 H]
		Note: This parameter is only relevant for module revision 4 and later. Earlier versions always use the DIP switches.
Output Parameters	chanhandle	The handle to the specified channel on the module. This handle is the first parameter in all channel-specific functions.
	bauderr	Not all the baud rates can be attained exactly by the oscillators on the module. The bauderr is the percent error from the specified baud rate to the actual baud rate. For more details, see **Programmable Baud Rate Generator** in the module's user's manual.
Return Values	ebadhandle	If an invalid handle was specified; should be the value returned by Init_Module_SER
	einvalchan	If tried to set channel to illegal value: i.e. 2 or 3 on a 2-channel module
	einval	If an invalid parameter was used as an input
	echantoomany	If too many channels have been set up in this application
	ebauderr	If the requested baud rate could not be generated exactly; the actual baud rate was off by <bauderr>%
	0	If successful

SetUseDmaIfAvailable_SER

Description

SetUseDmaIfAvailable_SER sets whether DMA should be used for data transfer between the host memory and the *Serial* module memory, if DMA is available. (DMA use is enabled by default.)

Syntax

SetUseDmaIfAvailable_SER (`int` devhandle, `int` useDmaFlag)

Input Parameters

devhandle	The handle returned by Init_Module_SER
useDmaFlag	A flag indicating DMA memory access. Legal flags allowed: ENABLE To enable DMA [0001 H] DISABLE To disable DMA [0000 H]

Output Parameters

none

Return Values

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

4 Communication over Serial Channels

Chapter 4 contains descriptions of the receive and transmit functions necessary to write test programs for the *Serial* module. Each function is presented with its formal definition, data types of all input parameters, including legal values where applicable, and output variables.

The following functions are described in this chapter:

General Communication Functions

- ClearChannel_SER
- IsReceiveFifoEmpty_SER
- IsTransmitFifoEmpty_SER
- SetTransmitFifoEmptyInterrupt_SER

Transmit Functions

- GetTransmitFifoCount_SER
- GetTransmitTotalCount_SER
- ResetTransmitTotalCount_SER
- TransmitByte_SER
- TransmitString_SER

Receive Functions

- GetReceiveFifoCount_SER
- GetReceiveOverrunStatus_SER
- GetReceiveTotalCount_SER
- NumBytesWaiting_SER
- ReceiveByte_SER
- ReceiveByteImmediate_SER
- ReceiveString_SER
- ResetReceiveTotalCount_SER

General Communication Functions

ClearChannel_SER

Description ClearChannel_SER clears Receive and Transmit FIFOs.

Syntax ClearChannel_SER (`int` chanhandle, `int` Clearflag)

Input Parameters

chanhandle	The handle designated by SetupChannel_SER	
Clearflag	Legal flags allowed:	
	CLEAR_RECEIVE	Clears Receive FIFO [0001 H]
	CLEAR_TRANSMIT	Clears Transmit FIFO [0002 H]
	CLEAR_BOTH	Clears Receive and Transmit FIFO [0003 H]

Output Parameters none

Return Values

ebadchanhandle	If an invalid handle was specified; should be the value returned by SetupChannel_SER
einval	If an invalid parameter was used as an input
0	If successful

IsReceiveFifoEmpty_SER

Description IsReceiveFifoEmpty_SER can be used to check that the Receive FIFO is empty, *before* clearing the receive channel.

Syntax IsReceiveFifoEmpty_SER (`int` chanhandle, `BOOL` *Emptyflag)

Input Parameters chanhandle — The handle designated by SetupChannel_SER

Output Parameters

Emptyflag	Legal flags allowed:	
	ENABLE	Receive FIFO is empty [0001 H]
	DISABLE	Receive FIFO is *not* empty [0000 H]

Return Values

ebadchanhandle	If an invalid handle was specified; should be the value returned by SetupChannel_SER
0	If successful

IsTransmitFifoEmpty_SER

Description	IsTransmitFifoEmpty_SER can be used to check that the Transmit FIFO is empty, *before* clearing the transmit channel.	
Syntax	IsTransmitFifoEmpty_SER (`int` chanhandle, `BOOL` *Emptyflag)	
Input Parameters	chanhandle	The handle designated by SetupChannel_SER
Output Parameters	Emptyflag	Legal flags allowed: ENABLE Transmit FIFO is empty [0001 H] DISABLE Transmit FIFO is *not* empty [0000 H]
Return Values	ebadchanhandle	If an invalid handle was specified; should be the value returned by SetupChannel_SER
	0	If successful

SetTransmitFifoEmptyInterrupt_SER

Description	SetTransmitFifoEmptyInterrupt_SER can be used to generate an interrupt when the Transmit FIFO empties.	
Syntax	SetTransmitFifoEmptyInterrupt_SER (`int` chanhandle, `BOOL` enableflag)	
Input Parameters	chanhandle	The handle designated by SetupChannel_SER
Output Parameters	enableflag	Legal flags allowed: ENABLE To enable an interrupt when Transmit FIFO is empty [0001 H] DISABLE To disable an interrupt when Transmit FIFO is empty [0000 H]
Return Values	ebadchanhandle	If an invalid handle was specified; should be the value returned by SetupChannel_SER
	0	If successful

Transmit Functions

GetTransmitFifoCount_SER

Description GetTransmitFifoCount_SER returns the total number of bytes in the FIFO waiting to be transmitted.

Syntax GetTransmitFifoCount_SER (`int` chanhandle, `int` *pXmtFifoCount)

Input Parameters

chanhandle	The handle designated by SetupChannel_SER

Output Parameters

pXmtFifoCount	A pointer to the number of bytes in the FIFO waiting to be transmitted

Return Values

ebadchanhandle	If an invalid handle was specified; should be the value returned by SetupChannel_SER
0	If successful

GetTransmitTotalCount_SER

Description GetTransmitTotalCount_SER returns the total number of bytes transmitted since the counter was reset. (See also **ResetTransmitTotalCount_SER** on page 1-5.)

Note: This function is only available with module revision 8 or later.

Syntax GetTransmitTotalCount_SER (`int` chanhandle, `unsigned int` *pXmtTotalCount)

Input Parameters

chanhandle	The handle designated by SetupChannel_SER

Output Parameters

pXmtTotalCount	A pointer to the number of bytes transmitted since the counter was reset

Return Values

ebadchanhandle	If an invalid handle was specified; should be the value returned by SetupChannel_SER
eunsupportedfeature	If the installed module does not support this feature
0	If successful

ResetTransmitTotalCount_SER

Description	ResetTransmitTotalCount_SER resets the transmit counter. (See also **GetTransmitTotalCount_SER** on page 1-4.)	
	Note: This function is only available with module revision 8 or later.	
Syntax	ResetTransmitTotalCount_SER (`int` chanhandle)	
Input Parameters	chanhandle	The handle designated by SetupChannel_SER
Output Parameters	none	
Return Values	ebadchanhandle	If an invalid handle was specified; should be the value returned by SetupChannel_SER
	eunsupportedfeature	If the installed module does not support this feature
	0	If successful

TransmitByte_SER

Description	TransmitByte_SER transmits one byte over a channel.	
Syntax	TransmitByte_SER (`int` chanhandle, `const BYTE` cChar)	
Input Parameters	chanhandle	The handle designated by SetupChannel_SER
	cChar	The byte to be transmitted
Output Parameters	none	
Return Values	ebadchanhandle	If an invalid handle was specified; should be the value returned by SetupChannel_SER
	0	If successful

TransmitString_SER

Description	TransmitString_SER transmits a string of bytes.	
Syntax	TransmitString_SER (`int` chanhandle, `const BYTE*` lpBuf, `int` dwCount)	
Input Parameters	chanhandle	The handle designated by SetupChannel_SER
	lpBuf	The pointer to the string to transmit
	dwCount	The number of bytes to transmit
Output Parameters	none	
Return Values	ebadchanhandle	If an invalid handle was specified; should be the value returned by SetupChannel_SER
	0	If successful

Receive Functions

Receive Overview

When data is received by the module's UART, it is immediately transfered to the module's Receive FIFO. You move the data from the Receive FIFO to the internal Software Tools Buffer by calling NumBytesWaiting_SER. NumBytesWaiting_SER returns the total number of data bytes waiting to be read from the Software Tools Buffer. You extract the data from the Software Tools Buffer by calling ReceiveByte_SER, ReceiveByteImmediate_SER or ReceiveString_SER.

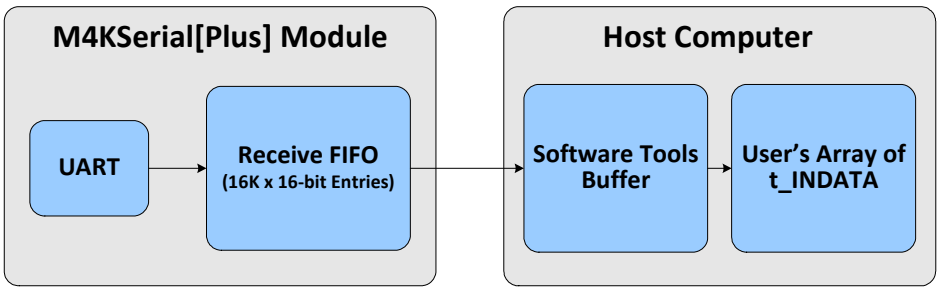

Figure 4-1 Receive Data Flow

GetReceiveFifoCount_SER

Description

GetReceiveFifoCount_SER returns the total number of (16-bit) entries that are still in the Receive FIFO. (The correlation between the number of entries and the number of received data bytes depends on whether you have selected to save Status and/or Time Tags, and whether the module is an *M8KSerial*, *M4KSerial* or *M4KSerialPlus*.)

See **Introduction** on page 1-1 and **Receive Block Structure in FIFO Mode** in the module's user's manual.

Syntax

GetReceiveFifoCount_SER (`int` chanhandle, `int` *pRcvFifoCount)

Input Parameters

chanhandle	The handle designated by SetupChannel_SER

Output Parameters

pRcvFifoCount	A pointer to the number of entries in the Receive FIFO

Return Values

ebadchanhandle	If an invalid handle was specified; should be the value returned by SetupChannel_SER
0	If successful

GetReceiveOverrunStatus_SER

Description	GetReceiveOverrunStatus_SER returns the receive overrun status. Calling the function resets the status to '0'.	
Syntax	GetReceiveOverrunStatus_SER (`int` chanhandle, `BOOL` *overrun)	
Input Parameters	chanhandle	The handle designated by SetupChannel_SER
Output Parameters	overrun	Returns the overrun status
Return Values	ebadchanhandle	If an invalid handle was specified; should be the value returned by SetupChannel_SER
	0	If successful

GetReceiveTotalCount_SER

Description	GetReceiveTotalCount_SER returns the total number of bytes received by the module since the counter was reset. (See also **ResetReceiveTotalCount_SER** on page 1-10.)	
	Note: This function is only available with module revision 8 or later.	
Syntax	GetReceiveTotalCount_SER (`int` chanhandle, `unsigned int` *pRcvTotalCount)	
Input Parameters	chanhandle	The handle designated by SetupChannel_SER
Output Parameters	pRcvTotalCount	A pointer to the total number of bytes received by the module since the counter was reset
Return Values	ebadchanhandle	If an invalid handle was specified; should be the value returned by SetupChannel_SER
	eunsupportedfeature	If the installed module does not support this feature
	0	If successful

NumBytesWaiting_SER

Description

NumBytesWaiting_SER is used before ReceiveByte_SER or ReceiveString_SER to transfer data from the Receive FIFO to the Software Tools Buffer. (See **Receive Overview** on page 1-6.)

The function returns how many data bytes are waiting to be received.

Note: When using a baud rate greater than 500,000 bps and using DMA to read data from the Receive FIFO, this function only transfers one DMA-buffer full (4 KB) per function call. In this case, NumBytesWaiting_SER must be called repeatedly to transfer all the data from the Receive FIFO to the Software Tools Buffer.

Syntax

NumBytesWaiting_SER (`int` chanhandle, `int` *NumBytes)

Input Parameters

chanhandle	The handle designated by SetupChannel_SER

Output Parameters

NumBytes	Number of data bytes currently waiting to be read (that is, the total number of data bytes in the Software Tools Buffer). Use this value as the dwCount in ReceiveString_SER.

Return Values

ebadchanhandle	If an invalid handle was specified; should be the value returned by SetupChannel_SER
0	If successful

ReceiveByte_SER

Description

ReceiveByte_SER is used after NumBytesWaiting_SER to receive a data byte from the Software Tools Buffer. If there are no data bytes in the Software Tools Buffer or the Receive FIFO, this function will wait indefinitely until a byte is received. (See **Receive Overview** on page 1-6.)

Syntax

ReceiveByte_SER (`int` chanhandle, `t_INDATA` *cChar)

Input Parameters

chanhandle	The handle designated by SetupChannel_SER

Output Parameters

cChar	A pointer to the byte received

Return Values

ebadchanhandle	If an invalid handle was specified; should be the value returned by SetupChannel_SER
0	If successful

ReceiveByteImmediate_SER

Description ReceiveByteImmediate_SER receives (or returns) one data byte from the Software Tools Buffer. If there are no data bytes in the Software Tools Buffer, it checks the Receive FIFO. If no data is available, it returns immediately. (See **Receive Overview** on page 1-6.)

Syntax ReceiveByteImmediate_SER (`int` chanhandle, `t_INDATA` *cChar, `int` *readflag)

Input Parameters	chanhandle	The handle designated by SetupChannel_SER
Output Parameters	cChar	The byte to be received, if any is available
	readflag	1 If byte was received 0 If no byte available to receive
Return Values	ebadchanhandle	If an invalid handle was specified; should be the value returned by SetupChannel_SER
	0	If successful

ReceiveString_SER

Description ReceiveString_SER is used after a call to NumBytesWaiting_SER to receive a specific number of data bytes from the Software Tools Buffer. If the requested number of bytes is not found, it takes additional bytes from the Receive FIFO. This function will wait indefinitely until the requested number of data bytes is received. (See **Receive Overview** on page 1-6.)

Syntax ReceiveString_SER (`int` chanhandle, `t_INDATA` *inbuf, `int` dwCount)

Input Parameters	chanhandle	The handle designated by SetupChannel_SER
	dwCount	The number of bytes to receive
Output Parameters	inbuf	A pointer to the next data
Return Values	ebadchanhandle	If an invalid handle was specified; should be the value returned by SetupChannel_SER
	0	If successful

ResetReceiveTotalCount_SER

Description

ResetReceiveTotalCount_SER resets the receive counter. (See also **GetReceiveTotalCount_SER** on page 1-7.)

Note: This function is only available with module revision 8 or later.

Syntax

ResetReceiveTotalCount_SER (`int` chanhandle)

Input Parameters

chanhandle	The handle designated by SetupChannel_SER

Output Parameters

none

Return Values

ebadchanhandle	If an invalid handle was specified; should be the value returned by SetupChannel_SER
eunsupportedfeature	If the installed module does not support this feature
0	If successful

5 Using Interrupt Functions

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_SER, 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 program **`demo_int.c`** which is included with the *Serial 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_SER function may be used to determine if multiple interrupts have occurred. Conversely, it is possible that the Wait_For_Interrupt_SER 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_SER function and the call to Get_Interrupt_Count_SER.) Once again, the Get_Interrupt_Count_SER function may be used to determine if the interrupt has already been processed.

Each function is presented with its formal definition, data types of all input parameters, including legal values where applicable, and output variables. The following functions are described in this chapter:

Get_Interrupt_Count_SER

GetInterruptStatus_SER

InitializeInterrupt_SER

SetDataReceivedInterrupt_SER

SetReceiveErrorInterrupt_SER

Wait_For_Interrupt_SER

Wait_For_Multiple_Interrupts_SER

Get_Interrupt_Count_SER

Description Get_Interrupt_Count_SER returns the total interrupt count for the specified module from the time the module was initialized with Init_Module_SER.

Syntax Get_Interrupt_Count_SER (`int` devhandle, `unsigned long` *pdwInterruptCount)

Input Parameters	devhandle	The handle designated by Init_Module_SER
Output Parameters	pwdInterruptCount	Pointer to an unsigned long which receives the interrrupt count
Return Values	ebadhandle	If an invalid handle was specified; should be the value returned by Init_Module_SER
	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

GetInterruptStatus_SER

Description GetInterruptStatus_SER returns the channel interrupt status.

Syntax GetInterruptStatus_SER (`int` chanhandle, `int` *InterruptStatus)

Input Parameters	chanhandle	The handle designated by SetupChannel_SER.
Output Parameters	InterruptStatus	Combination of 0 or more of the following flags:
		PARITY_ERR_INT — Parity error [0001 H]
		STOPBITS_ERR_INT — Stopbit error [0002 H]
		0 If the interrupt was caused by data received with no errors.
Return Values	ebadchanhandle	If an invalid handle was specified; should be the value returned by SetupChannel_SER
	0	If successful

InitializeInterrupt_SER

Description

InitializeInterrupt_SER instructs the Excalibur kernel driver to keep track of interrupts which occur on the *Serial* 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_SER function). When the application later calls Wait_For_Interrupt_SER, 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 is called to close the session.
2. The use of this function is not absolutely necessary for interrupt processing. If Wait_For_Interrupt_SER is called before InitializeInterrupt_SER has been called, then the initialization will be performed automatically, at that point. The use of InitializeInterrupt_SER is necessary only if the user requires interrupt tracking prior to the first call to Wait_For_Interrupt_SER.

Syntax

InitializeInterrupt_SER (`int` devhandle)

Input Parameters

devhandle	The handle designated by Init_Module_SER

Output Parameters

none

Return Values

ebadhandle	If an invalid handle was specified; should be the value returned by Init_Module_SER
egeteventhandle1	If there is an error in kernel mGetEventHandle, first part
egeteventhandle2	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

SetDataReceivedInterrupt_SER

Description	Call SetDataReceivedInterrupt_SER to receive an interrupt when data is received.	
Syntax	SetDataReceivedInterrupt_SER (`int` chanhandle, `BOOL` enableflag)	
Input Parameters	chanhandle	The handle designated by SetupChannel_SER
	enableflag	Legal flags allowed: ENABLE To enable the function [0001 H] DISABLE To disable the function [0000 H]
Output Parameters	none	
Return Values	ebadchanhandle	If an invalid handle was specified; should be the value returned by SetupChannel_SER
	0	If successful

SetReceiveErrorInterrupt_SER

Description	Call SetReceiveErrorInterrupt_SER to receive an interrupt when there is an error with received data.	
Syntax	SetReceiveErrorInterrupt_SER (`int` chanhandle, `BOOL` enableflag)	
Input Parameters	chanhandle	The handle designated by SetupChannel_SER
	enableflag	Legal flags allowed: ENABLE To enable the function [0001 H] DISABLE To disable the function [0000 H]
Output Parameters	none	
Return Values	ebadchanhandle	If an invalid handle was specified; should be the value returned by SetupChannel_SER
	0	If successful

Wait_For_Interrupt_SER

Description

Wait_For_Interrupt_SER 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_SER (`int` devhandle, `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_SER(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…
     // Check total number of interrupts
    Get_Interrupt_Count_SER(module_handle, &numints);
  }
}
```

Input Parameters

devhandle	The handle returned by Init_Module_SER.
timeout	Timeout is specified in milliseconds, or INFINITE

Output Parameters

none

Return Values

ebadhandle	If an invalid handle was specified; should be the value returned by Init_Module_SER
egeteventhandle1	If there is an error in kernel mGetEventHandle, first part
egeteventhandle2	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

Wait_For_Multiple_Interrupts_SER

Description

Wait_For_Multiple_Interrupts_SER 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_SER (`int` *handle_list, `int` num_modules, `unsigned int` timeout, `unsigned long` *pdw_interrupt_bitfield)

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

Output Parameters

pwd_interrupt_bitfield	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 mGetEventHandle, first part
egeteventhand2	If there is an error in kernel mGetEventHandle, second part
ebadhandle	If an invalid handle was specified; should be the value returned by Init_Module_SER
ekernelinitmodule	If error initializing kernel related data
ekernelbadparam	If input parameter is invalid
ekerneldevicenotopen	If the specified device was not opened
ekernelbadpointer	If output parameter buffer is invalid
	Successful if *either*:
ekerneltimeout	The wait timed out without receiving an interrupt *or*
0	If successful

Appendix A *Serial Software Tools* Library

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

Source code and Header files for the *Serial 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 *Serial 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

	File Name	Description
Source Code Files	**commapi**	Functions for receiving and transmitting communications over the channels
	configapi	Function for configuring the channel
	deviceio	Functions for interacting with the Excalibur kernel drivers for all Windows operating systems
	deviceio_ser	Functions for interacting with Excalibur device driver for the Serial module
	error_ser	Function for returning error messages
	initcard	Functions for initializing the device and module
Source Header files	**To be included by applications**	
	error_devio	Header file containing error codes for the Excalibur module kernel drivers
	error_ser	Header file containing error message codes
	excsysio	Header file which defines shared files between DLL and kernel driver
	Flags_ser	Header file containing flags for Serial module
	Proto_ser	Header file, prototypes of all functions
	exc4000	Header file containing functions for communicating with Excalibur kernel drivers
	NOT for applications	
	deviceio	Header file for interaction with kernel driver
	excdef	Header file to differentiate between different Excalibur products
	Instance_ser	Header file for global module structure
	Reg_ser	Header file, address of registers on the Serial module and UARTs
	serflags	Header file containing flags for Serial module
	serIncl	Header file, include file for the DLL code

	File Name	Description
Demo Programs (partial list)	**demo_ser**	Communication demo program for testing the Serial module using a loopback cable (a specially designed cable that wraps selected transmit channel back to receive channel on the module)
	demo_int	Demo program illustrating use of interrupts
DLL and *.LIB files	Our current software tools release contains: • A single set of source code files for the API functions. • One set of DLLs for use with 32-bit applications, found under the installation folder at: **\Source\lib\excMSVisual\Win32** in subfolders **Debug** and **Release**. • One set of DLLs for use with 64-bit applications, found under the installation folder at: **\Source\lib\excMSVisual\x64** in subfolders **Debug** and **Release**. • A single set of source code for the demo programs, found under the installation folder at: **\Source\demos_*ProtocolName*** with a separate folder for each demo program. • One set of demo program EXEs compiled for 32-bits, found under the installation folder at: **\Source\demos_*ProtocolName*\bin** • One set of demo program EXEs compiled for 64-bits, found under the installation folder at: **\Source\demos_*ProtocolName*\bin64** **Note:** The current software tools installation folder is **C:\Excalibur*ProtocolName* Software Tools**. You need to take the full set of DLLs found in the appropriate subfolder of the **LIB** folder for your compiler (**Win32** or **x64** and **Debug** or **Release**), and copy them to the folder where your EXE sits, or to C**:\Windows\SysWOW64** (for 64-bit Windows). The DLLs include a module level DLL (**Exc*Protocol*Ms.dll**), a board level DLL (**Exc4000Ms.dll**), and other DLLs for lower-level functions, to access the registers on the module. The DLL **frontdesk.dll** is provided for 32-bit applications running on 64-bit systems.	

Appendix B Code Index

The *Serial Software Tools* is a set of C language functions designed to aid programmers of the *Serial* module to write test applications. Below is an alphabetical list of all the functions and the name of the *Serial Software Tools* file containing the programming code. The extension of all the files is *.c

Functions	Code File Name
AlterChannelParams_SER	**configapi**
ChannelErrstatEnable_SER	**configapi**
ChannelTimetagEnable_SER	**configapi**
ClearChannel_SER	**commapi**
Get_Error_String_SER	**error**
GetHiFreqOscMhz_SER	**configapi**
Get_Interrupt_Count_SER	**deviceio**
GetUseDmaIfAvailable_SER	**configapi**
GetDIPVal_SER	**configapi**
GetHWid_SER	**initcard**
GetInterruptStatus_SER	**commapi**
GetModuleTimetag_SER	**deviceio**
GetReceiveOverrunStatus_SER	**commapi**
Init_Module_SER	**initcard**
InitializeInterrupt_SER	**deviceio**
IsReceiveFifoEmpty_SER	**commapi**
IsTransmitFifoEmpty_SER	**commapi**
ModuleTimetagReset_SER	**initcard**
NumBytesWaiting_SER	**commapi**
NumChannels_SER	**configapi**
ReceiveByte_SER	**commapi**
ReceiveByteImmediate_SER	**commapi**
ReceiveString_SER	**commapi**
Release_Module_SER	**initcard**
ResetChannel_SER	**configapi**

Functions	Code File Name
SetHiFreqOscMhz_SER	**configapi**
SetUseDmaIfAvailable_SER	**configapi**
SetDataReceivedInterrupt_SER	**commapi**
SetReceiveErrorInterrupt_SER	**commapi**
SetRS485Channel_SER	**configapi**
SetTransmitFifoEmptyInterrupt_SER	**commapi**
SetupChannel_SER	**configapi**
TransmitByte_SER	**commapi**
TransmitString_SER	**commapi**
Wait_For_Interrupt_SER	**deviceio**
Wait_For_Multiple_Interrupts_SER	**deviceio**

Appendix C Error Messages

All functions in the *Serial 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.

Error Code	Value	Definition
einval	-2	Illegal value used as input
sim_no_mem	-5	If not enough memory available for simulation
etimeoutreset	-26	Timed out waiting for reset
ewrngmodule	-27	Module specified on the board is not a Serial module
enomodule	-28	No module present at specified location
ebadhandle	-33	Invalid handle specified; should be the value returned by Init_Module_SER
eboardtoomany	-36	Too many modules initialized
func_invalid	-49	Function invalid for this card
einvalchan	-200	Tried to set channel to illegal value
enochans	-201	No channels found on module
echantoomany	-202	Too many channels have been set up in this application
ebadchanhandle	-203	Invalid handle specified; should be the value returned by SetupChannel_SER
ebauderr	-204	The requested baud rate cannot be generated exactly; the actual baud rate will be off by <bauderr>%
einvalidttagres	-205	Invalid Time Tag resolution
eincompatiblettagres	-206	Time Tag resolution is incompatible with this module
einvalidcarrierboard	-207	This version of the board does not support the *M4KSerialPlus* module
einvalidconfig	-208	Invalid module configuration; contact Excalibur support
eunsupportedfeature	-209	The installed module does not support this feature

Error Code	Value	Definition
eunsupportedmod	-210	This module is not supported by this revison of the *Software Tools*
einvalidtrigdest	-211	Invalid trigger destination
emaxrcvcount	-212	Specified receive count exceeds the maximum
emaxpatrncount	-213	Specified pattern-match count exceeds the maximum
einvalidhioscfreq	-214	An invalid High Oscillator frequency was specified
efreqalreadyknown	-215	The High Oscillator frequency has already been determined based on the module revision or carrier board

For PCI Carrier Boards Only

eopenkernel	-1001	Cannot open kernel device; check **`ExcConfig`** settings
ekernelcantmap	-1002	Kernel driver cannot map memory
ereleventhandle	-1003	Error in kernel Release_Event_Handle
egetintcount	-1004	Error in kernel Get_Interrupt_Count
egetchintcount	-1005	Kernel error in Get_Channel_Interrupt_Count when attempting to retrieve interrupt count
egetintchannels	-1006	Kernel error in Get_Interrupt_Channels when attempting to retrieve interrupt count
ewriteiobyte	-1007	Error in kernel WriteIOByte
ereadiobyte	-1008	Error in kernel ReadIOByte
egeteventhand1	-1009	Error in kernel mGetEventHandle, first part
egeteventhand2	-1010	Error in kernel mGetEventHandle, second part
eopenscmant	-1011	Error in openSCManager in startkerneldriver
eopenservicet	-1012	Error in openservice in startkerneldriver
estartservice	-1013	Error in startservice in startkerneldriver
eopenscmanp	-1014	Error in openSCManager in stopkerneldriver
eopenservicep	-1015	Error in openservice in stopkerneldriver
econtrolservice	-1016	Error in controlservice in stopkerneldriver
eunmapmem	-1017	Error in kernel unmapmemory

Error Code	Value	Definition
egetirq	-1018	Error in Get_IRQ_Number
eallocresources	-1019	Error allocating resources. See **ReadMe.pdf** for details on resource allocation problems.
egetramsize	-1020	Error in kernel GetRAMSize
ekernelwriteattrib	-1021	Error in kernel WriteAttributeMemory
ekernelreadattrib	-1022	Error in kernel ReadAttributeMemory
ekernelfrontdesk	-1023	Kernel frontdesk error
ekernelOscheck	-1024	Kernel Oscheck error
ekernelfrontdeskload	-1025	Kernel frontdeskload error
ekerneliswin2000compatible	-1026	Kernel is win2000 compatible error
ekernelbankramsize	-1027	Kernel banksize error
ekernelgetcardtype	-1028	Error in kernel GetCardType
emodnum	-1029	Invalid module number specified
regnotset	-1030	Module not configured. Reboot after **ExcConfig** is run and board is in slot
ekernelbankphysaddr	-1031	Kernel GetBankPhysAddr error
ekernelclosedevice	-1032	Kernel CloseKernelDevice error
ekerneldevicenotopen	-1034	Kernel error: device not open
ekernelinitmodule	-1035	Kernel initialization error
ekernelbadparam	-1036	Kernel error: bad input parameter
ekernelbadpointer	-1037	Kernel error: invalid pointer to output buffer
ekerneltimeout	-1038	A Wait_For_Interrupt function returned with timeout
ekernelnotwin2000	-1039	Operating System is not Windows 2000
erequestnotification	-1040	Request_Interrupt_Notification error
ekernelnot4000card	-1041	Designated board is not a 4000/8000 board
enotimersirig	-1042	Timers and IrigB not supported on this version of board
eclocksource	-1059	If an invalid clock source was specified
eparmglobalreset	-1062	Illegal parameter used for `globalreset_flag` in StartTimer

Error Code	Value	Definition
etimernotrunning	-1063	Timer not running when ResetWatchdogTimer was called; did nothing
etimerrunning	-1064	Timer already running when StartTimer was called; did nothing
eparmreload	-1065	Illegal parameter used for `reload_flag` in StartTimer
eparminterrupt	-1066	Illegal parameter used for `interrupt_flag` in StartTimer
ebaddevhandle	-1067	Invalid handle specified. Use value returned by Init_Timers
edevtoomany	-1068	Init_Timers called for too many boards
einvalidOS	-1069	If an invalid operating system is used

For VME/VXI Boards Only

eviclosedev	-1050	Error in ViClose device
evicloserm	-1051	Error in ViClose DefaultRM
eopendefaultrm	-1052	Error in ViOpenDefaultRM
eviopen	-1053	Error in ViOpen
evimapaddress	-1054	Error in viMapAddress
evicommand	-1055	Error in VISA command
einstallhandler	-1056	Error in VISA viInstallHandler
eenableevent	-1057	Error in VISA viEnableEvent
eunistallhandler	-1058	Error in VISA viUnistallHandler
edevnum	-1060	If a device number greater than 255 was used
einstr	-1061	If a device was not successfully initialized by Init_Module_SER

Functions Index

January 2021, Rev B-2