ARINC 717

Software Tools

Programmer's Reference

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

Table of Contents

1 Introduction

Chapter 1 provides an overview of the *ARINC 717* module. The following topics are covered:

Overview

ARINC 717 Software Tools are 'C' language functions designed to aid users of the Excalibur's *ARINC 717* module in writing application and diagnostic programs. These functions provide access to all of the module functions in a structured and straightforward programming environment.

The *ARINC 717 Software Tools* distributed with the product are compatible with the Windows® operating system. The Windows drivers were written and tested using Microsoft Visual C++.

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

The module supports two channels, each of which can transmit and receive messages simultaneously. Both HPB and BPRZ codings are supported. Each channel supports various bit rates (from 32 to 4096 words per second).

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

ARINC 717 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

Programmers *must* use the **_cdecl** calling options. The driver functions in the *ARINC 717 Software Tools* are supplied both in source form and linked as a DLL. When writing application-programs, keep in mind that the module is a physical resource, and therefore you cannot run multiple copies of the program simultaneously, accessing the same module.

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

Functions are written as 'C' functions, i.e., they return values. A negative value signifies an error. Full error messages may be printed using the Get_Error_String_717 function.

In Windows all user-defined programs must include the file **`proto_717.h`**. This file includes all the necessary header files and DLL functions to operate the *ARINC 717 Software Tools* for *ARINC 717* modules.

DMA Support

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

When DMA is available, the *Software Tools* use DMA access for functions that read from memory.

The following functions use DMA:

- Get_Next_IRIG_SubFrame_717
- Get_Next_SubFrame_717
- Load_Data_717

Conventions Used in this Manual

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

Functions look like this.

Parameters look like this.

`File names` look like this.

FLAGS look like this.

Technical Support

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

2 General Functions

Chapter 2 describes the *ARINC 717* functions that are not related to transmit or receive modes. The following functions are described in this chapter:

Initialization Functions

Init_Module_717

Description

Init_Module_717 is the first function the user must call for each *ARINC 717* module on each device that is accessed in the application program. Before exiting a program, call Release_Module_717 for each module initialized with Init_Module_717.

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

Syntax

Init_Module_717 (`t_Word` device_num, `t_Word` module_num, `int` *pModuleHandle)

Input Parameters

device_num	The device number is the index number of the board set in `ExcConfig`: 0 – 15
module_num	The module number of the *ARINC 717* module on the board: 0 - 7 (depending on the board)

Output Parameters

pModuleHandle	A pointer to the handle of the specified module; this handle is used as the first parameter in all module functions

Return Values

eBoardTooMany_717	If there are no more available handles to support the connection to this module
eTimeoutReset_717	If timed out waiting for reset
eallocresources	If there was an error allocating resources
eboardtoomany	If too many modules initialized; this module not initialized
ekernelbadparam	If input parameter is invalid
ekernelbadpointer	If output parameter buffer is invalid
ekernelcantmap	If a pointer to memory cannot be obtained
ekerneldevicenotopen	If specified device has not been opened
ekernelfrontdesk	If there was an error opening kernel device; check **`ExcConfig`** set up
ekernelfrontdeskload	It there was an error loading **`frontdesk.dll`**
ekernelinitmodule	If error initializing kernel related data
ekernelnot4000card	If board is not from EXC-4000/8000 family
emodnum	Invalid module number specified
enomodule	If no module is present at specified location
eopenkernel	If there was an error opening a device

Init_Module_717 (cont.)

	etimeoutreset	If timed out waiting for reset
	ewrngmodule	If module specified on the board is not an *ARINC 717* module
	regnotset	If the board is not configured; reboot after running **`ExcConfig`** and board is in slot
	Successful_ Completion_Exc	If successful

Release_Module_717

Description

Release_Module_717 must be called for each module initialized with Init_Module_717, before exiting a program.

Following a call to this function, the user must call Init_Module_717 before using the module again.

Syntax

Release_Module_717 (`int` modHandle)

Input Parameters

	modHandle	The handle designated by Init_Module_717

Output Parameters

none

Return Values

	eBadModHandle_717	If an invalid handle was specified; should be the value returned by Init_Module_717
	Successful_ Completion_Exc	If successful

DMA Functions

Get_DmaAvailable_717

Description Get_DmaAvailable_717 returns an indicator as to whether Direct Memory Access (DMA) is available for the transfer of data between the host memory and the module memory. See also Get_UseDmaIfAvailable_717 on page 2-4.

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

Syntax Get_DmaAvailable_717 (`int` modHandle, `BOOL` *pDmaEnabled)

Input Parameters

handle	The handle designated by Init_Module_717

Output Parameters

pDmaEnabled	A pointer to whether DMA is available:
	ENABLE DMA is available [1 H]
	DISABLE DMA is not available [0 H]

Return Values

eBadModHandle_717	If an invalid handle was specified; should be the value returned by Init_Module_717
Successful_ Completion_Exc	If successful

Get_UseDmaIfAvailable_717

Description Get_UseDmaIfAvailable_717 returns an indicator as to whether DMA will be used for data transfer between the host memory and the module memory, if DMA is available.

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

Syntax Get_UseDmaIfAvailable_717 (`int` modHandle, `BOOL` *pUseDmaFlag)

Input Parameters

modHandle	The handle designated by Init_Module_RTx

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 [1 H]
	DISABLE DMA will not be used [0 H]

Return Values

ebadhandle	If handle other than the one returned by Init_Module_RTx is used
Successful_ Completion_Exc	If successful

Set_UseDmaIfAvailable_717

Description Set_UseDmaIfAvailable_717 sets whether DMA should be used for data transfer between the host memory and the module memory, if DMA is available. (DMA is enabled by default.) See also Get_DmaAvailable_717 on page 2-4.

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

Syntax Set_UseDmaIfAvailable_717 (`int` modHandle, `BOOL` useDmaFlag)

Input Parameters

modHandle	The handle designated by Init_Module_717	
useDmaFlag	Indicates whether DMA should be used for data transfer (if available):	
	ENABLE	Use DMA if available [1 H]
	DISABLE	Do not use DMA [0 H]

Output Parameters none

Return Values

eBadModHandle_717	If an invalid handle was specified; should be the value returned by Init_Module_717
Successful_ Completion_Exc	If successful

Module Information Functions

Get_FirmwareRev_717

Description	Get_FirmwareRev_717 returns the firmware revision of the *ARINC 717* module.	
Syntax	Get_FirmwareRev_717 (`int` modHandle, `t_Word` *pFwMajorRev, t_Word * pFwMinorRev)	
Input Parameters	modHandle	The handle designated by Init_Module_717
Output Parameters	pFwMajorRev	A pointer to the upper 16 bits of the firmware revision, which is the major revision number
	pFwMinorRev	A pointer to the lower 16 bits of the firmware revision, which is the minor revision number
Return Values	eBadModHandle_717	If an invalid handle was specified; should be the value returned by Init_Module_717
	Successful_ Completion_Exc	If successful

Get_HardwareRev_717

Description	Get_HardwareRev_717 returns the hardware revision of the *ARINC 717* module.	
Syntax	Get_HardwareRev_717 (`int` modHandle, `t_Word` *pHwMajorRev, `t_Word` *pHwMinorRev)	
Input Parameters	modHandle	The handle designated by Init_Module_717
Output Parameters	pHwMajorRev	A pointer to the upper 16 bits of the hardware revision, which is the major revision number
	pHwMinorRev	A pointer to the lower 16 bits of the hardware revision, which is the minor revision number
Return Values	eBadModHandle_717	If an invalid handle was specified; should be the value returned by Init_Module_717
	Successful_ Completion_Exc	If successful

Get_NumChannels_717

Description	Get_NumChannels_717 returns the hardware revision of the *ARINC 717* module.	
Syntax	Get_NumChannels_717 (`int` modHandle, `t_Dword` *pNumChannels)	
Input Parameters	modHandle	The handle designated by Init_Module_717
Output Parameters	pNumChannels	A pointer to the number of channels available on the module
Return Values	eBadModHandle_717	If an invalid handle was specified; should be the value returned by Init_Module_717
	Successful_ Completion_Exc	If successful

Get_PCBRev_717

Description	Get_PCBRev_717 returns the printed circuit board revision of the *ARINC 717* module.	
Syntax	Get_PCBRev_717 (`int` modHandle, `t_Word` *pPCBRev)	
Input Parameters	modHandle	The handle designated by Init_Module_717
Output Parameters	pPCBRev	A pointer to the revision of the module's printed circuit board; valid values are A [H] – F [H]
Return Values	eBadModHandle_717	If an invalid handle was specified; should be the value returned by Init_Module_717
	Successful_ Completion_Exc	If successful

Get_SerialNumber_717

Description	Get_SerialNumber_717 returns the serial number of the *ARINC 717* module.	
Syntax	Get_SerialNumber_717 (`int` modHandle, `t_Dword` *pSerialNumber)	
Input Parameters	modHandle	The handle designated by Init_Module_717
Output Parameters	pSerialNumber	A pointer to the serial number of the module
Return Values	eBadModHandle_717	If an invalid handle was specified; should be the value returned by Init_Module_717
	Successful_ Completion_Exc	If successful

Channel Setup and Operation Functions

This section describes channel setup and general operation functions, not related to transmit or receive. (Transmit and receive operation are described in chapters 3 and 4.)

Get_Channel_Counters_717

Description	Get_Channel_Counters_717 returns the channel transmit/receive counters of the *ARINC 717* module.	
Syntax	Get_Channel_Counters_717 (`int` modHandle, `t_Dword` *pXmtedWordCount, `t_Dword` *pRcvedFrameCount, `t_Dword` *pRcvedSubFrameCount, `t_Dword` *pRcvedErrorCount)	
Input Parameters	chanHandle	The handle designated by Setup_Channel_717
Output Parameters	pXmtedWordCount	A pointer to the number of words transmitted by the transmitter portion of the channel
	pRcvedFrameCount	A pointer to the number of frames received by the receiver portion of the channel
	pRcvedSubFrameCount	A pointer to the number of subframes received by the receiver portion of the channel
	pRcvedErrorCount	A pointer to the number of errors encountered by the receiver portion of the channel
Return Values	eBadChanHandle_717	If an invalid handle was specified; should be the value returned by Setup_Channel_717
	Successful_ Completion_Exc	If successful

Setup_Channel_717

Description

Setup_Channel_717 configures an *ARINC 717* channel for operation.

Note: This function cannot be called when the channel is running.

Syntax

Setup_Channel_717 (`int` modHandle, `t_Word` channel, `te_DataRateWPS` dataRateWPS, `int` *pchanHandle)

Input Parameters

modHandle	The handle designated by Init_Module_717
channel	The number of the desired channel (0 or 1)
dataRateWPS	The desired data rate (words per second)

Output Parameters

pchanHandle	A pointer to the handle of the specified channel pair; for example, transmit channel 0 and receive channel 0 will receive the same channel handle

Return Values

eBadModHandle_717	If an invalid handle was specified; should be the value returned by Init_Module_717
eInvalidChan_717	If an invalid channel number was specified
eChanTooMany_717	If there are no more available channel handles to support the connection to this channel
eCantSetParameter WhileRunning_717	If the channel is running (transmit or receive); you cannot setup a channel while the channel is running
Successful_ Completion_Exc	If successful

Time Tag and IRIG B Functions

Get_ModuleIrigTime_717

Description Get_ModuleIrigTime_717 returns the current IRIG B time on the module.

Syntax Get_ModuleIrigTime_717 (`int` modHandle, `t_IrigOnModuleTime` *pIrigTime)

Input Parameters

modHandle	The handle designated by Init_Module_717

Output Parameters

pIrigTime	A pointer to a `t_IrigOnModuleTime` struct containing the current IRIG B time

```
typedef struct t_IrigOnModuleTime {
   unsigned short days; // 1–366, where 1 = Jan 1
   unsigned short hours;
   unsigned short minutes;
   unsigned short seconds;
   unsigned int microsecs;
} t_IrigOnModuleTime;
```

Return Values

eBadModHandle_717	If an invalid handle was specified; should be the value returned by Init_Module_717
Successful_ Completion_Exc	If successful

Get_TimeTag_717

Description Get_TimeTag_717 returns the running Time Tag of the module (non-IRIG B).

Syntax Get_TimeTag_717 (`int` modHandle, `t_DDword` *pTtag)

Input Parameters

modHandle	The handle designated by Init_Module_717

Output Parameters

pTtag	A pointer to the running Time Tag of the module, as a 64-bit unsigned integer

Return Values

eBadModHandle_717	If an invalid handle was specified; should be the value returned by Init_Module_717
Successful_ Completion_Exc	If successful

Is_IrigSignal_Present_717

Description Is_IrigSignal_Present_717 checks whether there is an incoming IRIG B signal. If there is no incoming IRIG B signal, the module updates the IRIG B time based on the module's internal clock. See also Preset_IrigTimeTag_717 on page 2-11.

Syntax Is_IrigSignal_Present_717 (`int` modHandle, `BOOL` *irigSignalPresentFlag)

Input Parameters

modHandle	The handle designated by Init_Module_717

Output Parameters

irigSignalPresentFlag	A pointer to whether there is an incoming IRIG B signal:
TRUE	There is an incoming IRIG B signal [1 H]
FALSE	There is no incoming IRIG B signal [0 H]

Return Values

eBadModHandle_717	If an invalid handle was specified; should be the value returned by Init_Module_717
Successful_ Completion_Exc	If successful

Preset_IrigTimeTag_717

Description Preset_IrigTimeTag_717 loads time values (day, hours, minutes, seconds) into the IRIG B Preload registers. These registers provide starting values for the IRIG B time registers for when the module is not receiving a signal from an external source. See Is_IrigSignal_Present_717 on page 2-11.

Syntax Preset_IrigTimeTag_717 (`int` modHandle, `t_Word` day, `t_Word` hour, `t_Word` minute, `t_Word` second)

Input Parameters

modHandle	The handle designated by Init_Module_717
day	Initial value to preset the number of days since the beginning of the year, 1–366, where 1 = Jan 1
hour	Initial value to preset the number of hours in the day
minute	Initial value to preset the number of minutes in the hour
second	Initial value to preset the number of seconds in the minute

Output Parameters none

Preset_IrigTimeTag_717 (cont.)

Return Values	eBadModHandle_717	If an invalid handle was specified; should be the value returned by Init_Module_717
	Successful_ Completion_Exc	If successful

PrintIRIGtime_717

Description PrintIRIGtime_717 prints (sprintf) the IRIG B Time Tag as a string. This function takes the `t_IrigOnModuleTime` struct returned by Get_ModuleIrigTime_717 and Get_Next_IRIG_SubFrame_717. In Get_Next_IRIG_SubFrame_717, the `t_IrigOnModuleTime` struct is part of the `t_IrigSubFrameHeader_717` struct.

Syntax PrintIRIGtime_717 (`t_IrigOnModuleTime` irigTime, `char` *outString)

Input Parameters

irigTime	The IRIG B time provided as a `t_IrigOnModuleTime` struct

```
typedef struct t_IrigOnModuleTime {
   unsigned short days; // 1–366, where 1 = Jan 1
   unsigned short hours;
   unsigned short minutes;
   unsigned short seconds;
   unsigned int microsecs;
} t_IrigOnModuleTime;
```

Output Parameters

outString	A string containing the visual display of the IRIG B Time Tag value, in the format: **Day Month hour:minute:second.microseconds**

Return Values none

Reset_TimeTag_717

Description Reset_TimeTag_717 resets the running Time Tag on the *ARINC 717* module (non-IRIG B).

Syntax Reset_TimeTag_717 (`int` modHandle)

Input Parameters modHandle — The handle designated by Init_Module_717

Output Parameters none

Return Values

eBadModHandle_717	If an invalid handle was specified; should be the value returned by Init_Module_717
Successful_ Completion_Exc	If successful

Interrupt and Trigger Functions

Interrupt/Trigger Bits

Several of the interrupt and trigger function have interrupt/trigger bits as one of the parameter. These bits are from the Interrupt Status register. The lower 16 bits of the module's Interrupt Status register correspond to Channel 0's status; the upper 16 bits correspond to Channel 1's status. For functions that affect only one channel (such as Clear_Channel_IntTrig_Status_717) only the Channel 0 bits are used even when Channel 1 is specified. When Channel 1 is specified, the function shifts the bits 16 bit places to apply to Channel 1.

Channel 0 Bits

INTTRIG_XMT_ERROR (100 H) – a transmit error occurred
INTTRIG_XMT_SEND_WC (200 H) – the specified number of words (Word Count) were transmitted
INTTRIG_XMT_WORDS_TO_SEND_COMPLETE (400 H) – the channel transmitted all of the words that it was instructed to send
INTTRIG_RCV_ERROR (01 H) – a receive error occurred
INTTRIG_RCV_END_OF_SF (02 H) – a subframe was received
INTTRIG_RCV_END_OF_FRAME (04 H) – a frame was received
INTTRIG_RCV_START_OF_SF1 (08 H) – a Subframe 1 is being received
INTTRIG_RCV_START_OF_SF2 (10 H) – a Subframe 2 is being received
INTTRIG_RCV_START_OF_SF3 (20 H) – a Subframe 3 is being received
INTTRIG_RCV_START_OF_SF4 (40 H) – a Subframe 4 is being received
INTTRIG_RCV_START_RCV (80 H) – reception of an ARINC717 subframe has started (that is, the receiver has achieved sync)

Channel 1 Bits

10000 H – a transmit error occurred
20000 H – the specified number of words (Word Count) were transmitted
40000 H – the channel transmitted all of the words that it was instructed to send
100 H – a receive error occurred
200 H – a subframe was received
400 H – a frame was received
800 H – a Subframe 1 is being received
1000 H – a Subframe 2 is being received
2000 H – a Subframe 3 is being received
4000 H – a Subframe 4 is being received
8000 H – reception of an ARINC717 subframe has started (that is, the receiver has achieved sync)

Clear_Channel_IntTrig_Status_717

Description

Clear_Channel_IntTrig_Status_717 clears the specified bits in one channel's portion of the module's Interrupt Status register. These bits affect both interrupts and triggers.

The lower 16 bits of the module's Interrupt Status register correspond to Channel 0's status; the upper 16 bits correspond to Channel 1's status. The bits specified for the channelIntTrigStatus parameter are within the lower 16 bits of the register (Channel 0). When you specify Channel 1, this function shifts the bits 16 bit places to the left, to apply to Channel 1.

Syntax

Clear_Channel_IntTrig_Status_717 (`int` chanHandle, `t_Dword` channelIntTrigStatus)

Input Parameters

chanHandle	The handle designated by Setup_Channel_717
channelIntTrigStatus	The channel's interrupt/trigger status bits to clear; one or more bits ORed together. The interrupt/trigger status bits are listed in Channel 0 Bits on page 2-13.

Output Parameters

none

Return Values

eBadChanHandle_717	If an invalid handle was specified; should be the value returned by Setup_Channel_717
Successful_ Completion_Exc	If successful

Clear_Module_IntTrig_Status_717

Description Clear_Module_IntTrig_Status_717 clears the specified bits of the module's Interrupt Status register. These bits affect both interrupts and triggers. The lower 16 bits of the module's Interrupt Status register correspond to Channel 0's status; the upper 16 bits correspond to Channel 1's status. See also Clear_Channel_IntTrig_Status_717 on page 2-14.

Syntax Clear_Module_IntTrig_Status_717 (`int` modHandle, `t_Dword` moduleIntTrigStatus)

Input Parameters

modHandle	The handle designated by Init_Module_717
moduleIntTrigStatus	The module's interrupt/trigger status bits to clear; one or more bits ORed together. The interrupt/trigger status bits are listed in **Channel 0 Bits** and **Channel 1 Bits** on page **2-13**.

Output Parameters none

Return Values

eBadModHandle_717	If an invalid handle was specified; should be the value returned by Init_Module_717
Successful_ Completion_Exc	If successful

Get_Channel_IntTrig_Status_717

Description Get_Channel_IntTrig_Status_717 returns one channel's portion of the module Interrupt Status register. The lower 16 bits of the module's Interrupt Status register correspond to Channel 0's status; the upper 16 bits correspond to Channel 1's status. The bits returned in pChannelIntTrigStatus are within the lower 16 bits of the register (Channel 0). When you specify Channel 1, this function shifts the bits 16 bit places to the right. See also Clear_Channel_IntTrig_Status_717 on page 2-14.

Syntax Get_Channel_IntTrig_Status_717 (`int` chanHandle, `t_Dword` *pChannelIntTrigStatus)

Input Parameters

chanHandle	The handle designated by Setup_Channel_717
pChannelIntTrigStatus	The channel's interrupt/trigger status bits; one or more bits ORed together. The interrupt/trigger status bits are listed in Channel 0 Bits on page 2-13.

Output Parameters none

Get_Channel_IntTrig_Status_717 (cont.)

Return Values	eBadChanHandle_717	If an invalid handle was specified; should be the value returned by Setup_Channel_717
	Successful_ Completion_Exc	If successful

Get_Interrupt_Count_717

Description	Get_Interrupt_Count_717 returns the total interrupt count for the specified module from the time the module was initialized with Init_Module_717.	
Syntax	Get_Interrupt_Count_717 (`int` modHandle, `t_Dword` *Sys_Interrupts_Ptr)	
Input Parameters	irigTime	The IRIG B time provided as a `t_IrigOnModuleTime` struct
Output Parameters	outString	A string containing the visual display of the IRIG B Time Tag value, in the format: **Day Month hour:minute:second.microseconds**
Return Values	none	

Get_Module_IntTrig_Status_717

Description	Get_Module_IntTrig_Status_717 returns the module's Interrupt Status register. The lower 16 bits of the module's Interrupt Status register correspond to Channel 0's status; the upper 16 bits correspond to Channel 1's status. See also Clear_Module_IntTrig_Status_717 on page 2-15.	
Syntax	Get_Module_IntTrig_Status_717 (`int` modHandle, `t_Dword` *pModuleIntTrigStatus)	
Input Parameters	modHandle	The handle designated by Init_Module_717
	pModuleIntTrigStatus	The module's interrupt/trigger status bits; one or more bits ORed together. The interrupt/trigger status bits are listed in **Channel 0 Bits** and **Channel 1 Bits** on page **2-13**.
Output Parameters	none	
Return Values	eBadModHandle_717	If an invalid handle was specified; should be the value returned by Init_Module_717
	Successful_ Completion_Exc	If successful

InitializeInterrupt_717

Description

InitializeInterrupt_717 instructs the Excalibur kernel driver to keep track of interrupts which occur on the *ARINC 717* 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_717 function). When the application later calls Wait_For_Interrupt_717, the function will return immediately, indicating an interrupt has occurred in the meantime.

Note: The use of this function is not absolutely necessary for interrupt processing. If Wait_For_Interrupt_717 is called before InitializeInterrupt_717 has been called, then the initialization will be performed automatically, at that point. The use of InitializeInterrupt_717 is necessary only if the user requires interrupt tracking prior to the first call to Wait_For_Interrupt_717.

Syntax

InitializeInterrupt_717 (`int` modHandle)

Input Parameters

modHandle	The handle designated by Init_Module_717

Output Parameters

none

Return Values

eBadModHandle_717	If an invalid handle was specified; should be the value returned by Init_Module_717
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
ekernelbadpointer	If output parameter buffer is invalid
Successful_ Completion_Exc	If successful

Set_External_Trigger_Conditions_717

Description Set_External_Trigger_Conditions_717 sets the trigger mask for the specified channel. The module then sets an external trigger when the mask conditions are met. The lower 16 bits of the module's Interrupt Status register correspond to Channel 0's status; the upper 16 bits correspond to Channel 1's status. The bits in trigerEvents are within the lower 16 bits of the register (Channel 0). When you specify Channel 1, this function shifts the bits 16 bit places to the left. See also Clear_Channel_IntTrig_Status_717 on page 2-14.

Syntax Set_External_Trigger_Conditions_717 (`int` modHandle, `t_Dword` trigerEvents)

Input Parameters

modHandle	The handle designated by Init_Module_717
trigerEvents	The channel's trigger conditions; one or more bits ORed together. The interrupt/ trigger status bits are listed in Channel 0 Bits on page 2-13.

Output Parameters none

Return Values

eBadModHandle_717	If an invalid handle was specified; should be the value returned by Init_Module_717
Successful_ Completion_Exc	If successful

Set_External_Trigger_OnWordsXmitted_717

Description Set_External_Trigger_OnWordsXmitted_717 sets the Word Count register **and** sets the INTTRIG_XMT_SEND_WC trigger bit for the specified channel. See Channel 0 Bits on page 2-13.

Note: This function must be called when the module is stopped.

Syntax Set_External_Trigger_OnWordsXmitted_717 (`int` chanHandle, `t_Dword` wordCount)

Input Parameters

chanHandle	The handle designated by Setup_Channel_717
wordCount	The number of words to transmit before generating an external trigger

Output Parameters none

Return Values

eBadChanHandle_717	If an invalid handle was specified; should be the value returned by Setup_Channel_717
Successful_ Completion_Exc	If successful

Set_Interrupt_Conditions_717

Description Set_Interrupt_Conditions_717 sets the interrupt mask for the specified channel, which causes the module to generate an interrupt when the mask conditions are met.

Syntax Set_Interrupt_Conditions_717 (`int` chanHandle, `t_Dword` interruptEvents)

Input Parameters

chanHandle	The handle designated by Setup_Channel_717
interruptEvents	The channel's interrupt conditions; one or more bits ORed together. The interrupt/trigger status bits are listed in Channel 0 Bits on page 2-13.

Output Parameters none

Return Values

eBadChanHandle_717	If an invalid handle was specified; should be the value returned by Setup_Channel_717
Successful_ Completion_Exc	If successful

Set_Interrupt_OnWordsXmitted_717

Description Set_Interrupt_OnWordsXmitted_717 sets the Word Count register **and** sets the INTTRIG_XMT_SEND_WC trigger bit for the specified channel. See Channel 0 Bits on page 2-13.

Note: This function must be called when the module is stopped.

Syntax Set_Interrupt_OnWordsXmitted_717 (`int` chanHandle, `t_Dword` wordCount)

Input Parameters

chanHandle	The handle designated by Setup_Channel_717
wordCount	The number of words to transmit before generating an interrupt

Output Parameters none

Return Values

eBadChanHandle_717	If an invalid handle was specified; should be the value returned by Setup_Channel_717
Successful_ Completion_Exc	If successful

Wait_For_Interrupt_717

Description Wait_For_Interrupt_717 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_717 (`int` modHandle, `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:

Wait_For_Interrupt_717 (cont.)

```
DWORD Module717Isr(t_InterruptThreadParams *pIntThreadParams)
{
   while (1)
   {
      iError = Wait_For_Interrupt_717(pIntThreadParams->modHandle717, INFINITE);
      if ((iError != ekerneltimeout) && (iError < 0))
      {
         Get_Error_String_717(iError, ErrorStr);
         printf("Module717Isr::Wait_For_Interrupt_717 returned: %s\n", ErrorStr);
         ExitThread (EXIT_FAILURE);
      }
      // See what event generated the interrupt; peel off just the transmit status
      // of interest
      Get_Module_IntTrig_Status_717(pIntThreadParams->modHandle717,
         &moduleIntStatus);

      // We're looking for Xmt Interrupt status on channel
      // pIntThreadParams-xmtChannelNum
      if (pIntThreadParams->xmtChannelNum == CHANNEL_0)
      {
         channelIntStatus = (moduleIntStatus & INTTRIG_MASK_CHANNEL_0) >>
            INTTRIG_SHIFT_CHANNEL_0;
         Clear_Module_IntTrig_Status_717(pIntThreadParams->modHandle717,
            (moduleIntStatus & INTTRIG_MASK_CHANNEL_0));
      }
      else if (pIntThreadParams->xmtChannelNum == CHANNEL_1)
      {
         channelIntStatus = (moduleIntStatus & INTTRIG_MASK_CHANNEL_1) >>
            INTTRIG_SHIFT_CHANNEL_1;
         Clear_Module_IntTrig_Status_717(pIntThreadParams->modHandle717,
            (moduleIntStatus & INTTRIG_MASK_CHANNEL_1));
      }
      // This choice is in error - invalid channel
      else
         return(1);

      // interrupt due to ...
      if ((channelIntStatus & INTTRIG_XMT_SEND_WC) == INTTRIG_XMT_SEND_WC)
         word_count_xmitted++;

      if ((channelIntStatus & INTTRIG_XMT_WORDS_TO_SEND_COMPLETE) ==
         INTTRIG_XMT_WORDS_TO_SEND_COMPLETE)
         words_to_send_xmitted++;

      // More interrupt conditions....

      // keep track of the number of times we processed interrupts
      numIntCalls++;

      // And update the interrupt counts from the board
      prev_totalInts = pIntThreadParams->currentIntCount;
      Get_Interrupt_Count_717(pIntThreadParams->modHandle717, &
         pIntThreadParams->currentIntCount);

      // printf("Number of new Interrupts: %u \r",
      // (pIntThreadParams->currentIntCount - prev_totalInts));
   }
   return(EXIT_SUCCESS);
}
```

Wait_For_Interrupt_717 (cont.)

Input Parameters	modHandle	The handle designated by Init_Module_717
	timeout	Timeout is specified in milliseconds *or* INFINITE [FFFFFFFF H]
Output Parameters	none	
Return Values	eBadModHandle_717	If an invalid handle was specified; should be the value returned by Init_Module_717
	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	If the wait timed out without receiving an interrupt
	or	
	Successful_ Completion_Exc	

Wait_For_Multiple_Interrupts_717

Description

Wait_For_Multiple_Interrupts_717 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_717 (`int` numints, `int` *handle_array, `unsigned int` timeout, `unsigned long` *Interrupt_Bitfield)

Input Parameters

numints	Number of modules in the handle_array
handle_array	An array of module handles
timeout	Timeout is specified in milliseconds *or* INFINITE [FFFFFFFF H]

Output Parameters

Interrupt_Bitfield	A 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_array, and so on.

Return Values

eBadModHandle_717	If an invalid handle was specified; should be the value returned by Init_Module_717
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
ekernelbadpointer	If output parameter buffer is invalid
Successful if *either*:	
ekerneltimeout	If the wait timed out without receiving an interrupt
or	
Successful_ Completion_Exc	

Errors Functions

Get_Error_String_717

Description	Get_Error_String_717 accepts the error returned by a software function and returns the corresponding error message.	
Syntax	Get_Error_String_717 (`int` errcode, `t_char` *errstring)	
Input Parameters	errcode	The error code returned by a software function
Output Parameters	errstring	The error message corresponding to the error code. In case of bad input, this routine returns a string indicating that.
Return Values	eBadChanHandle_717	If an invalid handle was specified; should be the value returned by Setup_Channel_717
	Successful_ Completion_Exc	Always

3 Transmit Functions

Chapter 3 describes the *ARINC 717* transmit functions. The following functions are described in this chapter:

Get_Available_XmtSpace_717

Description	Get_Available_XmtSpace_717 returns the amount of unused space in a channel's transmitter buffer.	
Syntax	Get_Available_XmtSpace_717 (`int` chanHandle, `t_Dword` *pNumAvailWords)	
Input Parameters	chanHandle	The handle designated by Setup_Channel_717
Output Parameters	pNumAvailWords	The number of words that can be added to the channel's DPRAM buffer without overwriting previously loaded unsent data.
Return Values	eBadChanHandle_717	If an invalid handle was specified; should be the value returned by Setup_Channel_717
	eNoMemoryAllocated_717	If no memory was allocated for the data to transmit
	Successful_ Completion_Exc	If successful

Is_XmtChannel_Running_717

Description	Is_XmtChannel_Running_717 returns whether the specified transmit channel is running, and sets the running flag output parameter accordingly.	
Syntax	Is_XmtChannel_Running_717 (`int` modHandle, `t_Word` channel, `BOOL` *pXmtChanRunningFlag)	
Input Parameters	modHandle	The handle designated by Init_Module_717
	channel	The number of the desired channel (0 or 1)
Output Parameters	pXmtChanRunningFlag	A flag indicating the channels running status
Return Values	eBadModHandle_717	If an invalid handle was specified; should be the value returned by Init_Module_717
	eInvalidChan_717	If an invalid channel number was specified
	Successful_ Completion_Exc	If successful

Load_Data_717

Description

Load_Data_717 loads the channel's transmit buffer with data. (If the channel is already running, and the number of words to transmit was set to XMTCOUNT_ALL_LOADED_DATA, then the number of words to transmit will be increased by the number of words loaded as a result of calling Load_Data_717.)

Note: When DMA is used, Load_Data_717 is limited to 4096 bytes per call.

Syntax

Load_Data_717 (`int` chanHandle, `t_Dword` numWordsToLoad, `t_Word` *pWord717Buffer, `t_Dword` *pNumWordsLoaded)

Input Parameters

chanHandle	The handle designated by Setup_Channel_717
numWordsToLoad	The number of words to load
pWord717Buffer	A pointer to the block of memory to load into the channel's transmit DPRAM buffer

Output Parameters

pNumWordsLoaded	The number of words actually loaded into the channel's transmit DPRAM buffer

Return Values

eBadChanHandle_717	If an invalid handle was specified; should be the value returned by Setup_Channel_717
eNoMemoryAllocated_717	If no memory was allocated for the data to transmit
eXmtChanStoppedBeforeLoadComplete_717	If the running transmit channel was stopped before the new data was completely loaded into the transmit channel buffer
Successful_Completion_Exc	If successful

Set_XmtChannel_SlewRate_717

Description

Set_XmtChannel_SlewRate_717 allows you to set a specific slew rate, instead of using the default.

Note: This function cannot be called when the transmit channel is running.

Syntax

Set_XmtChannel_SlewRate_717 (`int` chanHandle, `te_XmtSlewRate` xmtSlewRate)

Input Parameters

chanHandle	The handle designated by Setup_Channel_717
xmtSlewRate	The desired slew rate. Valid values are: XMT_SLEW_RATE_075US (0) – 7.5 microseconds XMT_SLEW_RATE_105US (1) or XMT_SLEW_RATE_105US_ALT (2) – 10.5 microseconds (both values yield 10.5 microseconds) XMT_SLEW_RATE_015US (3) – 1.5 microseconds (default when Setup_Channel_717 is called)

Output Parameters

none

Return Values

eBadChanHandle_717	If an invalid handle was specified; should be the value returned by Setup_Channel_717
eInvalidInput Parameter_717	If an invalid parameter was specified in the function call
eCantSetParameter WhileRunning_717	If the channel is running; cannot change the specified parameter while the channel is running
eBadModHandle_717	If an invalid handle was specified; should be the value returned by Init_Module_717; internal error from a call to Is_XmtChannel_Running_717
eInvalidChan_717	If an invalid channel number was specified; internal error from a call to Is_XmtChannel_Running_717
Successful_ Completion_Exc	If successful

Start_XmtChannel_717

Description	Start_XmtChannel_717 starts the specified channel's transmitter.	
Syntax	Start_XmtChannel_717 (`int` chanHandle, `te_XmtCount` numWordsToXmit)	
Input Parameters	chanHandle	The handle designated by Setup_Channel_717
	numWordsToXmit	The number of words to transmit, after which the channel will automatically stop. Valid values are: 1 to FFFFFFFE H – the number of words to transmit XMTCOUNT_CONTINUOUS (0) – transmit continuously XMTCOUNT_ALL_LOADED_DATA (FFFFFFFF H) – transmit all data loaded into the transmit buffer
Output Parameters	none	
Return Values	eBadChanHandle_717	If an invalid handle was specified; should be the value returned by Setup_Channel_717
	eNoMemoryAllocated_717	If no memory was allocated for the data to transmit
	Successful_ Completion_Exc	If successful

Stop_XmtChannel_717

Description	Stop_XmtChannel_717 stops the specified channel's transmitter.	
Syntax	Stop_XmtChannel_717 (`int` chanHandle)	
Input Parameters	chanHandle	The handle designated by Setup_Channel_717
Output Parameters	none	
Return Values	eBadChanHandle_717	If an invalid handle was specified; should be the value returned by Setup_Channel_717
	Successful_ Completion_Exc	If successful

4 Receive Functions

Chapter 4 describes the *ARINC 717* receive functions. The following functions are described in this chapter:

Get_Next_IRIG_SubFrame_717

Description

Get_Next_IRIG_SubFrame_717 returns the receive channel's next received subframe. This function is for subframes captured with IRIG B Time Tags. For subframes captured with standard Time Tags, use Get_Next_SubFrame_717.

Syntax

Get_Next_IRIG_SubFrame_717 (`int` chanHandle, `t_IrigSubFrameHeader_717` *pIrigSubframeHeader, `t_Word` *pSubframeBuffer)

Input Parameters

chanHandle	The handle designated by Setup_Channel_717

Output Parameters

pIrigSubframeHeader	A pointer to a `t_SubFrameHeader_717` struct containing status, time stamp, and counter information about the next received subframe

```
typedef struct t_SubFrameHeader_717 {
   t_Word  Sentinel; // SUBFRAME_SENTINEL
   t_Word  SubFrameIndex; // SF1,SF2,SF3 or SF4
   t_Word  Status; // Subframe complete, errors
   t_Word  RcvWordCount;
   t_Dword TimeTagLo; // Low order 32 bits of Time Tag
      // for subframe
   t_Dword TimeTagHi; // High order 32 bits of Time
      // Tag for subframe
   t_Dword RcvFrameCount; // Running frame counter of
      // all frames recorded by the board
   t_Dword RcvSubframeCount; // Running subframe
      // counter of all subframes recorded by the
      // board
   t_Dword Reserved[2];
} t_SubFrameHeader_717;
```

pSubframeBuffer	A pointer to an array of words containing that actual received subframe data

Return Values

eBadChanHandle_717	If an invalid handle was specified; should be the value returned by Setup_Channel_717
eStdTTagType_717	If the module is using standard Time Tags (TTAGTYPE_STD_TTAG), but the IRIG B Time Tag function (Get_Next_IRIG_SubFrame_717) was called
eNoMemoryAllocated_717	If no memory was allocated for the received subframes
eNoNewSubframe_717	If no new subframe was received
Successful_Completion_Exc	If successful

Get_Next_SubFrame_717

Description

Get_Next_SubFrame_717 returns the receive channel's next received subframe. This function is for subframes captured with standard Time Tags. For subframes captured with IRIG B Time Tags, use Get_Next_IRIG_SubFrame_717.

Syntax

Get_Next_SubFrame_717 (`int` chanHandle, `t_IrigSubFrameHeader_717` *pIrigSubframeHeader, `t_Word` *pSubframeBuffer)

Input Parameters

chanHandle	The handle designated by Setup_Channel_717

Output Parameters

pIrigSubFrameHeader	A pointer to a `t_IrigSubFrameHeader_717` struct containing status, time stamp, and counter information about the next received subframe

```
typedef struct t_IrigSubFrameHeader_717 {
   t_Word  Sentinel; // SUBFRAME_SENTINEL
   t_Word  SubFrameIndex; // SF1,SF2,SF3 or SF4
   t_Word  Status; // Subframe complete, errors
   t_Word  RcvWordCount;
   t_IrigOnModuleTime irigTime;
   t_Dword RcvFrameCount; // Running frame counter of
      // all frames recorded by the board
   t_Dword RcvSubFrameCount; // Running subframe
      // counter of all Subframes recorded by the
board
} t_IrigSubFrameHeader_717;
```

The `t_IrigOnModuleTime` struct is:

```
typedef struct t_IrigOnModuleTime {
   unsigned short days; // 1–366, where 1 = Jan 1
   unsigned short hours;
   unsigned short minutes;
   unsigned short seconds;
   unsigned int microsecs;
} t_IrigOnModuleTime;
```

pSubframeBuffer	A pointer to an array of words containing that actual received subframe data

Return Values

eBadChanHandle_717	If an invalid handle was specified; should be the value returned by Setup_Channel_717
eStdTTagType_717	If the module is using standard Time Tags (TTAGTYPE_STD_TTAG), but the IRIG B Time Tag function (Get_Next_IRIG_SubFrame_717) was called

Get_Next_SubFrame_717 (cont.)

	eNoMemoryAllocated_717	If no memory was allocated for the received subframes
	eNoNewSubframe_717	If no new subframe was received
	Successful_Completion_Exc	If successful

Is_RcvChannel_Running_717

Description	Is_RcvChannel_Running_717 checks if the specified receive channel is running, and sets the running flag (output parameter) accordingly.	
Syntax	Is_RcvChannel_Running_717 (`int` modHandle, `t_Word` channel, `BOOL` *pRcvChanRunningFlag)	
Input Parameters	modHandle	The handle designated by Init_Module_717
	channel	The number of the desired channel (0 or 1)
Output Parameters	pRcvChanRunningFlag	A pointer to a flag indicating the receive channel's running status
Return Values	eBadModHandle_717	If an invalid handle was specified; should be the value returned by Init_Module_717
	eInvalidChan_717	If an invalid channel number was specified
	Successful_Completion_Exc	If successful

Set_Rcv_CodingMode_717

Description Set_Rcv_CodingMode_717 allows you to specify the coding mode, instead of using the default.

Note: Only the receive coding mode is set via software. The transmit coding mode is set via a DIP switch.

Syntax Set_Rcv_CodingMode_717 (`int` chanHandle, `te_RcvCodingMode` rcvCodingMode)

Input Parameters

chanHandle	The handle designated by Setup_Channel_717
rcvCodingMode	The desired coding mode. Valid values are: RCV_CODING_MODE_HBP (0) – Harvard Bi-Phase (HBP) RCV_CODING_MODE_BPRZ (1) – Bi-Polar Return-to-Zero (BPRZ)

Output Parameters none

Return Values

eBadChanHandle_717	If an invalid handle was specified; should be the value returned by Setup_Channel_717
eInvalidInput Parameter_717	If an invalid parameter was specified in the function call
Successful_ Completion_Exc	If successful

Set_RcvTimetagType_717

Description Set_RcvTimetagType_717 sets the Time Tag type used when recording the time that each message is received.

Note: This function cannot be called when the transmit channel is running.

Syntax Set_RcvTimetagType_717 (`int` modHandle, `te_TTagType` ttagType)

Input Parameters

modHandle	The handle designated by Init_Module_717
ttagType	The type of Time Tag to use when time stamping each subframe as it is received. Valid values are: TTAGTYPE_STD_TTAG (0) – time stamp uses standard Time Tag TTAGTYPE_IRIG_TTAG (1) – time stamp uses IRIG B time

Output Parameters none

Set_RcvTimetagType_717 (cont.)

Return Values	eBadModHandle_717	If an invalid handle was specified; should be the value returned by Init_Module_717
	eInvalidInput Parameter_717	If an invalid parameter was specified in the function call
	eCantSetParameter WhileRunning_717	If the channel is running (transmit or receive); you cannot setup a channel while the channel is running
	eInvalidChan_717	If an invalid channel number was specified
	Successful_ Completion_Exc	If successful

Start_RcvChannel_717

Description	Start_RcvChannel_717 starts the specified channel's receiver.	
Syntax	Start_RcvChannel_717 (`int` chanHandle)	
Input Parameters	chanHandle	The handle designated by Setup_Channel_717
Output Parameters	none	
Return Values	eBadChanHandle_717	If an invalid handle was specified; should be the value returned by Setup_Channel_717
	eNoMemoryAllocated_ 717	If no memory was allocated for the received subframes
	Successful_ Completion_Exc	If successful

Stop_RcvChannel_717

Description	Stop_RcvChannel_717 stops the specified channel's receiver.	
Syntax	Stop_RcvChannel_717 (`int` chanHandle)	
Input Parameters	chanHandle	The handle designated by Setup_Channel_717
Output Parameters	none	
Return Values	eBadChanHandle_717	If an invalid handle was specified; should be the value returned by Setup_Channel_717
	Successful_ Completion_Exc	If successful

Appendix A Source Code Reference

A-1 *ARINC 717 Software Tools* File Types

The *ARINC 717 Software Tools* are divided into three categories:

- Source code and header files
- DLL and LIB files
- Demo programs

Source Code and Header Files

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 compiler DLL
`*Ms.lib`	Index file used to create applications using Microsoft DLL functions

Demo Programs

Demo programs are examples of programs using the *ARINC 717 Software Tools*, which can be used as a basis for user-defined programs. Demo programs include the following types of files:

File Extension	Description
`*.c, *.h`	Demo source code
`*.exe`	Demo executable files
`*.sln` `*.suo`	Microsoft project solution files

A-2 Files Required for Module-Level Applications

The following table lists the files required when compiling application that contain module-level software functions. The DLL must be in the execution path of the application.

	File Name	Description
Header Files	**error_717.h**	Header file containing error message codes
	flags_717.h	Header file containing flags
	proto_717.h	Header file containing prototypes of all functions
	types_717.h	Data type definitions
DLL File	**Exc717Ms.dll**	DLL file for module level functions
LIB File	**Exc717Ms.lib**	LIB file for module level functions

A-3 Files Required for Board-Level Applications

The following table lists the files required when compiling application that contain board-level software functions. The DLL must be in the execution path of the application. For more information on board-level software functions, see the *EXC-4000 Family Carrier Boards Software Tools Programmer's Reference.*

	File Name	Description
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 board level functions
	excsysio.h	Header file which defines shared codes and structures between DLL and kernel driver
DLL File	**exc4000ms.dll**	DLL file for board level functions
LIB File	**exc4000ms.lib**	LIB file for board level functions

A-4 Demo Programs

File Name	Description
demo_717.c	Loopback program that demonstrates basic transmit and receive functions
demo_717_int.c	Program that demonstrates interrupt functions
demo_717_rcv.c	Program that demonstrates receive mode functions using standard Time Tags
demo_717_rcv_irig.c	Program that demonstrates receive mode functions using IRIG B Time Tags
demo_717_xmt.c	Program that demonstrates transmit mode functions

Function Index

December 2020, Rev A-2