429RTx & Discrete

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

4 ARINC 429 Transmit Channels Functions

5 Discrete Channels Functions

1 Introduction

Chapter 1 provides an overview of the *429RTx & Discrete Software Tools*. The topics covered in this chapter are:

Overview

The *429RTx & Discrete Software Tools* is a set of 'C' language functions designed to aid users of Excalibur's *429RTx[D]* cards, boards and modules to write test programs. These functions provide access to all of the *429RTx[D]* functions in a structured and straightforward programming environment.

429RTx & Discrete Software Tools is available for Windows and Linux. (See our website for additional software support.) The Windows functions were written and tested using Microsoft Visual C++.

Chapters 2 – 4 describe the functions, including syntax, flags and error messages, for the ARINC receive and transmit channels. Chapter 5 covers the functions unique to the Discrete channels on boards that have ARINC 429 and Discrete channels on the same module.

Note: The generic term *429RTx[D]* is used in the *Programmer's Reference* to refer to all of the hardware items.

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

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

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

The Unique Identifier is set by a DIP switch or jumpers on the board. (For more details, see your board's user's manual. In the user's manual, the Unique Identifier is called the Selected ID.)

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

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

An Overview of the 429RTx Data Communications Bus

429RTx & Discrete Software Tools enables the user to create custom diagnostic programs for any of the *429RTx[D]* family of products. To apply *Software Tools,* some familiarity with the functioning of the multi-protocol, test and simulation *429RTx[D]* hardware is necessary and will assist in understanding the rationale of *Software Tools* functions.

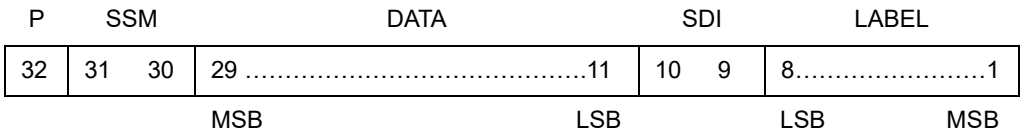
For the additional functionality of the Discrete channels on modules that have both ARINC 429 and Discrete channels, see **Discrete Channels** on page 1-6 and **Discrete Channels Functions**.

Aeronautical Radio INC (ARINC) is an organization composed of major airlines and airplane manufacturers, which promotes standardization within aircraft equipment. To facilitate this end ARINC puts out specifications describing

communications busses as well as certain types of avionics systems. One of the more popular busses used in commercial aircraft today is the ARINC 429 bus. ARINC can be reached at www.ARINC.com.

ARINC 429 is a two wire differential bus, which can connect a single transmitter or source to one or more receivers or sinks. Two speeds are defined, 12.5 kHz and 100kHz. The ARINC specification defines not only the physical layer and electrical characteristics but the data format of data sent over the bus as well.

All data sent over the ARINC bus is composed of 32 bit words. While a number of different formats are used the following diagram is typical for ARINC 429 data.



Bits are transmitted starting with bit 1, the final bit transmitted is the parity bit which is implemented with odd parity. The label is transmitted with the most significant bit first while the data is transmitted least significant bit first.

The label is a value from 1 to 255 representing a particular type of data. Most labels are defined in the specification though some are reserved for future needs. Many labels are multiply defined in the specification based on the type of equipment being used.

The SDI field is used when a transmitter is connected to multiple receivers but not all data is meant to be used by all the receivers. In this case each receiver will be assigned an SDI value and will look only at labels which match its SDI value. While the specification calls for SDI 00 to be universally accepted, a good deal of equipment appears to disregard this requirement.

The Data field contains the actual data to be sent. A number of data formats are defined in the specification. Binary Coded Decimal (BCD) format uses each 4 bits to contain a single decimal digit. BNR data is a binary coding. For both data types the specification calls out the units, the resolution, the range, the number of bits used and how frequently the label should be sent. A Discrete type has multiple single bit fields defined within a single label. A number of other formats are described in the specification.

The SSM, which is 2 bits long [bits 30 and 31] when using BCD numeric Data Words or 3 bits long [bits 29 – 31] when using BNR numeric Data Words, interprets the numeric value in the data field. Examples of SSM values might be Plus, North, East, Right, To or Above.

P is the parity bit. ARINC 429 calls for odd parity. The parity bit is the last bit sent over the bus.

Time Tag Word Formats

The default Time Tag format is a standard 32-bit Time Tag. You have the option to use an IRIG B Time Tag instead, by calling Set_IRIG_Timetag_Mode_RTx.

When using the standard 32-bit Time Tag, the function Select_Time_Tag_Source_4000 sets the Time Tag source for all the modules on the carrier board. The Time Tag source cannot be set for individual modules. This function selects either the internal or external Time Tag. For more details on this function, see the *Excalibur Carrier Board Software Tools Programmer's Reference*.

Standard 32-bit Time Tag Format

When using the standard 32-bit Time Tag, the Time Tag is made up of two 16-bit Words: Time Tag-Hi and Time Tag-Lo. The Time Tag resolution is 10 microseconds.

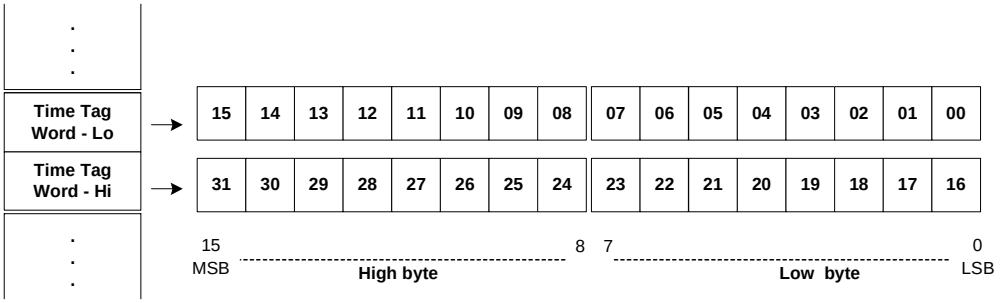


Figure 1-1 Standard 32-bit Time Tag Format

IRIG B Time Tag Format

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

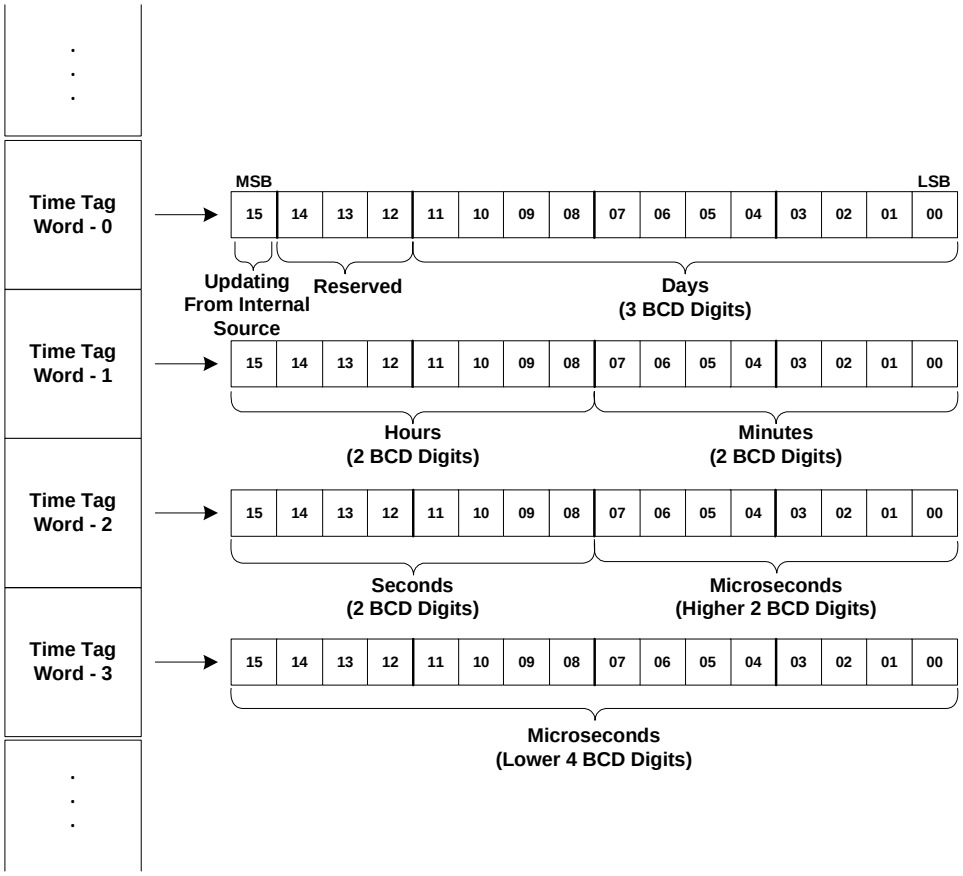


Figure 1-2 Time Tag Word Format in IRIG B Mode

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

Discrete Channels

In addition to ARINC 429 modules, some boards have ARINC 429 and Discrete channels on the same module. Discrete input channels are user-selectable to TTL (0–5V) or Avionics (0–32V) voltage levels. Output channels are open collector, capable of handling up to 32V with a maximum sink current of 100 milliamperes each.

The Discrete channels contain control I/O registers that are memory mapped and can be accessed in realtime.

429RTx & Discrete Software Tools for Modules on a Carrier Board

There are numerous modules available for use on the EXC-4000/8000 family carrier board for various communications protocols.

Various combinations of these modules may be present on the board at one time. One application may access any of the following configurations:

- one module
- multiple modules of the same type located on one board
- multiple modules of the same type located on separate boards
- multiple modules of different types located on one or separate boards

A board level DLL accompanies the module for each of the boards and cards. Depending on the board, the names of the files are:

Carrier Board	DLL Name
VME and VXI	**EXCV4000MS.DLL**
All Other Boards/Cards	**EXC4000MS.DLL**

The functions that operate at the carrier board level are contained in the **EXC4000.c** file for PCI carrier boards and the **EXCv4000.c** file for VME/VXI boards. The function, Get_4000Module_Type may be called for each board and module number, to check which, if any, modules are currently available at that location. (This function is also called automatically from Init_Module_RTx to ascertain that the type of module is a *429RTx* module.) These DLLs are installed automatically in the Windows/System folder when the *429RTx & Discrete Software Tools* are installed for each of the boards/modules.

For more information on board level functions, see the *Programmer's Reference* for your carrier board. For other functions that operate at the carrier board level see the applicable hardware *User's Manual*.

Compiler Options

The DLL is compiled under Microsoft Visual studio using the **_cdecl** calling convention. The driver functions in the *429RTx & Discrete 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.

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_RTx function. (**Appendix D: Error Messages**).

In Windows, all user-defined programs must include the file **proto_rtx.h.** (See **Appendix B: 429RTx & Discrete Software Tools Library**.) This file includes all the necessary header files and DLL function prototypes to operate *429RTx & Discrete Software Tools*.

Direct Memory Access (DMA)

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

When DMA is available, the *429RTx & Discrete Software Tools Software Tools* use DMA access for functions that read from, or write to, memory.

The following functions use DMA:

Read_Next_Data_RTx
Read_Next_Merge_Data_RTx

Conventions Used in This Manual

The following conventions are used in this manual:

Functions look like this.

Variable types look like this.

`Parameter` 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 software functions that are necessary to set up and operate the *429RTx[D]*. These functions are not mode specific: they can be used in at least two, or more, different modes.

Functions to handle interrupts in Windows are described at the end of this chapter, **Using Interrupts Under Windows** on page 2-19.

Note: For the *DAS-429[cc]PMC/RTx* board, the software relates to channels 0–9 as module 0; channels 10–19 as module 1.
For the *DAS-429[cc]PMC/RTxD* board, the software relates to channels 0–9 as module 0; channels 10–14 as module 1.

The following functions are described in this chapter:

Get_DmaAvailable_RTx
Get_Error_String_RTx
Get_Time_Tag_RTx
Get_UseDmaIfAvailable_RTx
Init_Module_RTx
Read_Board_Intr_Status_RTx
Read_Board_Status_RTx
Read_Chan_Config_Register_RTx
Read_Chan_Config_Stat_RTx
Read_Global_Start_RTx
Read_HwRevision_RTx
Read_Number_Of_Channels_RTx
Read_Revision_RTx
Read_SerialNumber_RTx
Release_Module_RTx
Reset_Channel_Configuration_RTx
Reset_Ttag_RTx
Set_IRIG_Timetag_Mode_RTx
Set_Prog_Bit_Rate_RTx
Set_UseDmaIfAvailable_RTx
Start_Channel_RTx
Start_Selected_Channels_RTx
Stop_Channel_RTx
Stop_Selected_Channels_RTx

For Interrupts on PCI[e] and PMC boards, see **Using Interrupts Under Windows** on page 2-19. The following functions are described:

Get_Interrupt_Count_RTx
InitializeInterrupt_RTx
Wait_For_Interrupt_RTx
Wait_For_Multiple_Interrupts_RTx

For Interrupts on VME/VXI boards, the interrupt functions are at the carrier board level. See **Using Interrupts Under VISA for VME Carrier Boards** on page 2-23.

Get_DmaAvailable_RTx

Description Get_DmaAvailable_RTx 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 **Set_UseDmaIfAvailable_RTx** on page 2-15.

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

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

Input Parameters	`handle`	The handle designated by Init_Module_RTx.
Output Parameters	`pDmaEnabled`	A pointer to whether DMA is available.
		ENABLE_RTX DMA is available [1 H]
		DISABLE_RTX DMA is not available [0 H]
Return Values	ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
	0	If successful

Get_Error_String_RTx

Description Get_Error_String_RTx accepts the error returns from other *429RTx & Discrete Software Tools* functions. This function returns the string containing a corresponding error message.

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

Example:

```
char ErrorStr[255];

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

Input Parameters	`errcode`	The error code returned from a Software Tools call.
Output Parameters	`errorstring`	An array of 255 characters, the message string that contains the corresponding error message. In case of bad input, this function returns a string denoting that.
Return Values	0	Always

Get_Time_Tag_RTx

Description Get_Time_Tag_RTx returns the running Time Tag of the module in RT and Monitor modes

Syntax Get_Time_Tag_RTx (int `handle`, unsigned int `*timetag`)

Input Parameters

`handle`	The handle designated by Init_Module_RTx.

Output Parameters

`timetag`	32-bit Time Tag.

Return Values

ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
0	If successful

Get_UseDmaIfAvailable_RTx

Description Get_UseDmaIfAvailable_RTx 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.

Syntax Get_UseDmaIfAvailable_RTx (int `handle`, BOOL `*pUseDmaFlag`)

Input Parameters

handle	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_RTX	DMA will be used if available [1 H]
	DISABLE_RTX	DMA will not be used [0 H]

Return Values

ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
0	If successful

Init_Module_RTx

Description

Init_Module_RTx enables the user to access a module. This is the first function the user must call for each module that is accessed in the application program.

All channels are initially disabled, unless specifically enabled. The default setting for all enabled channels is set to receive channel. Transmit channels must be specifically set by the programmer.

The function may be called with the SIMULATE [FFFF H] 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 and simulated modules in the same application.

Before exiting a program, call Release_Module_RTx for each module initialized with Init_Module_RTx.

Syntax

Init_Module_RTx (usint `device_num`, usint `module_num`, usint `enabled_channels`, usint `xmt_channels`)

Input Parameters

`device_num` The device number is the index number of the board set in `ExcConfig`: 0 – 15.

Note: If only one board is used, the value 25 or the #define value EXC_4000PCI 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.

or SIMULATE
Indicating no actual module on a board present [FFFF H]

`module_num` The module number of the *429RTx & Discrete Software Tools* module on the board: 0 – 7 (depending on the board). The value is ignored if `device_num` is SIMULATE.
You can run **`demo_information_429.exe`** to check the module number.

Init_Module_RTx (cont.)

`enabled_channels` One or more of the following flags OR'ed together (indicating which channels are enabled):

NO_CHANNELS Only Discrete channels enabled [0 H]

For RT5 modules	For RT10 modules
CHAN_0 [1 H]	CHAN_0 [1 H]
CHAN_1 [2 H]	CHAN_1 [2 H]
CHAN_2 [4 H]	CHAN_2 [4 H]
CHAN_3 [8 H]	CHAN_3 [8 H]
CHAN_4 [10 H]	CHAN_4 [10 H]
ALL_RT5_CHANNELS [1F H] All channels are enabled	CHAN_5 [20 H]
	CHAN_6 [40 H]
	CHAN_7 [80 H]
	CHAN_8 [100 H]
	CHAN_9 [200 H]
	ALL_RT10_CHANNELS [3FF H] All channels are enabled

Note: All channels are initially disabled, unless specifically enabled.

`xmt_channels` One or more of the following flags OR'ed together (indicating which channels are transmit):

NO_CHANNELS No channels set to transmit – only Discrete channels in use [0 H]

For RT5 modules	For RT10 modules
CHAN_0 [1 H]	CHAN_0 [1 H]
CHAN_1 [2 H]	CHAN_1 [2 H]
CHAN_2 [4 H]	CHAN_2 [4 H]
CHAN_3 [8 H]	CHAN_3 [8 H]
CHAN_4 [10 H]	CHAN_4 [10 H]
ALL_RT5_CHANNELS [1F H] All channels are set to transmit	CHAN_5 [20 H]
	CHAN_6 [40 H]
	CHAN_7 [80 H]
	CHAN_8 [100 H]
	CHAN_9 [200 H]
	ALL_RT10_CHANNELS [3FF H] All channels are set to transmit

Note: All channels are set to receive, unless specifically set to be transmit channels.

Init_Module_RTx (cont.)

Output Parameters none

Return values

emodnum	If an invalid module number was specified.
bad_chan	If an invalid channel was specified.
eboardtoomany	If too many modules were initialized.
sim_no_mem	If not enough memory available for simulation.
enomodule	If no module is present at specified location.
ewrngmodule	If module specified on the carrier board is not an RTx module.
init_failed	If failed to initialize the module/card.
init_failed_1	If failed to initialize the module/card after first reset.
init_failed_2	If failed to initialize the module/card after second reset.
init_failed_3	If failed to initialize the module/card.
fivechanmod	If tried to enable a channel number greater than 4 on a five channel RT*x* module. The module is not initialized.
sixchancard	If tried to enable a channel number greater than 5 on a six channel card. The card is not initialized.
sixtransmitchancard	If tried to set more than six channels to transmit on a card that does not support more than six transmit channels. The card is not initialized.
tenchanmod	If tried to enable a channel number greater than 9 on a ten channel RT*x* module. The module is not initialized.
ekernelcantmap	If a pointer to memory cannot be obtained.
ekernelnot4000card	If the board is not EXC-4000/8000 family.
edevtoomany	If Init_Timers_4000 called for too many boards.
ekernelinitmodule	If error initializing kernel related data.
ekernelbadparam	If input parameter is invalid.
eopenkernel	If there was an error opening a device.
ekernelfrontdeskload	If error loading **frontdesk.dll.**
ekernelfrontdesk	If error opening kernel device; check **ExcConfig** set up.
eallocresources	If there was an error allocating resources; see **Installation Instructions.pdf** for details on resource allocation problems.

Init_Module_RTx (cont.)

ekerneldevicenotopen	If specified device has not been opened.
regnotset	If board is not configured; reboot after **`ExcConfig`** is run and board is in slot.
ekernelbadpointer	If output parameter buffer is invalid.
<handle>	If successful, returns the handle to the specified module on the board. This handle is used as the first parameter in all module functions. A valid handle is an integer 0 or greater.

Read_Board_Intr_Status_RTx

Description Read_Board_Intr_Status_RTx returns and clears the module/card interrupt status. On an interrupt, this value indicates which channel used the interrupt.

Syntax Read_Board_Intr_Status_RTx (int `handle`)

Input Parameters

`handle`	The handle designated by Init_Module_RTx.

Output Parameters none

Return Values

ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
<interrupt status>	If successful, returns the module/card interrupt status. Each channel bit is 1, only if the respective channel has an interrupt since the last call to this function.

Bit	Description
10-15	0
00-09	Channel Interrupt Status bits

Read_Board_Status_RTx

Description Read_Board_Status_RTx performs a self-test, then returns the module/card status. The function indicates which channels are installed.

Note: You must call Init_Module_RTx before calling this function. Otherwise, it will fail the self-test.

Syntax Read_Board_Status_RTx (int `handle`)

Input Parameters

`handle`	The handle designated by Init_Module_RTx.

Output Parameters none

Read_Board_Status_RTx (cont.)

Return Values

ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
<status>	If successful, returns the module/card status.

Bit	Description
14-15	0
13	Extended Time Support 1 = Extended Time is supported 0 = Extended Time is not supported
12	1
11	Host Ready Timeout
10	Module/card memory OK
00-09	Channel Status Bits 1 = channel present (passed self-test) 0 = failed self-test or no channel present

Read_Chan_Config_Register_RTx

Description Read_Chan_Config_Register_RTx returns the Channel *x* Configuration register. For values, see the hardware *User's Manual* for your module or card.

Syntax Read_Chan_Config_Register_RTx (int `handle`, usint `channel`)

Input Parameters

`handle`	The handle designated by Init_Module_RTx.
`channel`	The channel value, 0 – 9, depending on the board or module. See Read_Number_Of_Channels_RTx on page 2-10.

Output Parameters none

Return Values

ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
param_Read_Chan_Config_Stat	If an invalid parameter was used as an input.
<Channel x Configuration register>	If successful, returns the contents of the Channel *x* Configuration register. For details, see the hardware *User's Manual* for your module or card.

Read_Chan_Config_Stat_RTx

Description Read_Chan_Config_Stat_RTx returns the Channel *x* Configuration Status register. For values, see the hardware *User's Manual* for your module or card.

Syntax Read_Chan_Config_Stat_RTx (int `handle`, usint `channel`)

Input Parameters

`handle`	The handle designated by Init_Module_RTx.
`channel`	The channel value, 0 – 9, depending on the board or module. See Read_Number_Of_Channels_RTx on page 2-10.

Output Parameters none

Return Values

ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
param_Read_Chan_Config_Stat	If an invalid parameter was used as an input.
<Channel x Configuration Status register>	If successful, returns the contents of the Channel *x* Configuration Status register. For details, see the hardware *User's Manual* for your module or card.

Read_Global_Start_RTx

Description Read_Global_Start_RTx returns the contents of the Global Start register. The function indicates which channels are started.

Syntax Read_Global_Start_RTx (int `handle`)

Input Parameters

`handle`	The handle designated by Init_Module_RTx.

Output Parameters none

Return Values

ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
<Global Start register>	If successful, returns the contents of the Global Start register.

Bit	Description
10-15	0
00-09	Channel Status Bits 1 = Channel is started 0 = Channel is stopped

Read_HwRevision_RTx

Description	Read_HwRevision_RTx returns the hardware revision number.	
Syntax	Read_HwRevision_RTx (int handle)	
Input Parameters	handle	The handle designated by Init_Module_RTx.
Output Parameters	none	
Return Values	ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
	<hardware revision number>	If successful, returns the hardware revision number as a 3-digit hexadecimal number. For example: If a value of 021 (H) is returned, this indicates a hardware revision of 2.1.

Read_Number_Of_Channels_RTx

Description	Read_Number_Of_Channels_RTx returns the number of channels that exist on the module. **Note:** For more details, such as how many channels can be receive and transmit, see **Board Features** or **Module Features** in the **Introduction** of the user's manual.	
Syntax	Read_Number_Of_Channels_RTx (int handle)	
Input Parameters	handle	The handle designated by Init_Module_RTx.
Output Parameters	none	
Return Values	ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
	<number of channels>	If successful, returns the number of channels on the module.

Read_Revision_RTx

Description	Read_Revision_RTx returns the firmware revision number.	
Syntax	Read_Revision_RTx (int handle)	
Input Parameters	handle	The handle designated by Init_Module_RTx.
Output Parameters	none	
Return Values	ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
	<firmware revision number>	If successful, returns the firmware revision number. For example: If a value of 0109 (H) is returned, this indicates a firmware revision of 1.09.

Read_SerialNumber_RTx

Description	Read_SerialNumber_RTx returns the module's serial number.	
Syntax	Read_SerialNumber_RTx (int `handle`)	
Input Parameters	`handle`	The handle designated by Init_Module_RTx.
Output Parameters	none	
Return Values	ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
	<module's serial number>	If successful, returns the module's serial number.

Release_Module_RTx

Description	Release_Module_RTx releases any grabbed resources on a specific module. Call this function for each module initialized with Init_Module_RTx, before exiting a program. To continue accessing the module, call Init_Module_RTx again.	
Syntax	Release_Module_RTx (int `handle`)	
Input Parameters	`handle`	The handle designated by Init_Module_RTx
Output Parameters	none	
Return Values	ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
	0	If successful

Reset_Channel_Configuration_RTx

Description

Reset_Channel_Configuration_RTx resets the channel configuration.

The Setup_Receive_Channel_RTx function can only be set when all the channels are turned off (via the Stop_Channel_RTx/ Stop_Discrete_RTD functions.

The card/module should be started via the Start_Channel_RTx/ Start_Discrete_RTD functions only after a minimum of 1 msec. from the time that the contents of the function have been changed.

Note: For boards with Discrete channels, the ARINC 429 channels are set independently from the Discrete channels. Therefore:

- Use Start/Stop_Channel_RTx when setting the ARINC channels.
- Use Start/Stop_Discrete_RTD when setting the Discrete channels.

Syntax

Reset_Channel_Configuration_RTx (int `handle`, usint `channel`, usint `config`)

Input Parameters

`handle`	The handle designated by Init_Module_RTx.	
`channel`	The channel value, 0 – 9, depending on the board or module. See Read_Number_Of_Channels_RTx on page 2-10.	
`config`	Combination of 1 flag from each of the relevant groups:	
`bit_rate`	EXC_BIT_RATE_LO	12.5 KHz [0 H]
	EXC_BIT_RATE_HI	100 KHz [1 H]
	BIT_RATE_PROG	Bit rate as set in Set_Prog_Bit_Rate_RTx [2 H]
`parity`	AR_PARITY_ODD	Odd parity [0 H]
	AR_PARITY_EVEN	Even parity [10 H]
	AR_PARITY_OFF	No parity [8 H]
Transmit channel only		
`rise`	RISE_HI_SPEED	Tx rise/fall time 1.5 +/- 0.5 µsec. (default) [0 H]
	RISE_LO_SPEED	Tx rise/fall time 10 +/- 0.5 µsec. [4 H]

Reset_Channel_Configuration_RTx (cont.)

Receive channel only

`wrap_around`	NO_WRAP_AROUND	Stop receiving when receiver buffer is full [40 H]
	WRAP_AROUND	Wrap around when receiver buffer is full [0 H]

Output Parameters none

Return Values

ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
bad_chan	If an invalid channel was specified
eboardstarted	If cannot change communication parameters
0	If successful

Reset_Ttag_RTx

Description Reset_Ttag_RTx resets the Time Tag to 0. The function can be called at any time.

Syntax Reset_Ttag_RTx (int `handle`)

Input Parameters

`handle`	The handle designated by Init_Module_RTx.

Output Parameters none

Return Values

ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
0	If successful

Set_IRIG_Timetag_Mode_RTx

Description Set_IRIG_Timetag_Mode_RTx sets the IRIG B mode for all receive channels. The function must be called before a call to Setup_Receive_Channel_RTx. For details on the IRIG B Time Tag format, see **IRIG B Time Tag Format** on page 1-6.

Note: This function is only available for *M4K429RTx* modules Rev. D or later and all *M8K429RT5* modules.

Syntax Set_IRIG_Timetag_Mode_RTx (int `handle`, usint `mode`)

Input Parameters

`handle`	The handle designated by Init_Module_RTx.

Set_IRIG_Timetag_Mode_RTx (cont.)

	mode	One of the following flags: IRIG_TTAG_MODE Each received Data Word is stored together with a 64-bit IRIG B Time Tag and a Status Word. [10 H] 0 Each received Data Word is stored together with a standard 32-bit Time Tag and a Status Word.
Output Parameters	none	
Return Values	ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
	eNoIrigSupportModule	If the module is not a Nios-based processor and cannot use IRIG-based functions
	param_Set_IRIG_Timetag_Mode	If an invalid mode was selected
	0	If successful

Set_Prog_Bit_Rate_RTx

Description	Set_Prog_Bit_Rate_RTx sets the programmable bit rate for all channels which have programmable bit rate chosen in their Configuration Register.	
Syntax	Set_Prog_Bit_Rate_RTx (int handle, unsigned long hz)	
Input Parameters	handle	The handle designated by Init_Module_RTx.
	hz	Minimum value: 2500 Maximum value: 5,000,000
Output Parameters	none	
Return Values	ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
	param_Set_Prog_Bit_Rate	If an invalid parameter is used as an input.
	0	If successful

Set_UseDmaIfAvailable_RTx

Description

Set_UseDmaIfAvailable_RTx 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_RTx** on page 2-2.

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

Syntax

Set_UseDmaIfAvailable_RTx (int `handle`, BOOL `useDmaFlag`)

Input Parameters

`handle`	The handle designated by Init_Module_RTx.	
`useDmaFlag`	Indicates whether DMA should be used for data transfer (if available):	
	ENABLE_RTX	Use DMA if available [1 H]
	DISABLE_RTX	Do not use DMA [0 H]

Output Parameters

none

Return Values

ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
0	If successful

Start_Selected_Channels_RTx

Description

Start_Selected_Channels_RTx simultaneously starts the specified channels. All channels must be set up before they are started.

Note: This function avoids the need to wait 500 microseconds between consecutive calls to Start_Channel_RTx (i.e., between successive writes to the Start register).

Syntax

Start_Selected_Channels_RTx (int `handle`, usint `selected_channels`, BOOL `keep_existing_as_is`)

Input Parameters

`handle`	The handle designated by Init_Module_RTx.	
`selected_channels`	One or more of the following flags OR'ed together (indicating which channels are to be started simultaneously):	
	For RT5 modules	**For RT10 modules**
	CHAN_0 [1 H] CHAN_1 [2 H] CHAN_2 [4 H] CHAN_3 [8 H] CHAN_4 [10 H] ALL_RT5_CHANNELS [1F H] All channels are to be started simultaneously	CHAN_0 [1 H] CHAN_1 [2 H] CHAN_2 [4 H] CHAN_3 [8 H] CHAN_4 [10 H] CHAN_5 [20 H] CHAN_6 [40 H] CHAN_7 [80 H] CHAN_8 [100 H] CHAN_9 [200 H] ALL_RT10_CHANNELS [3FF H] All channels are to be started simultaneously

Note: To find the number of channels on a module, see Read_Number_Of_Channels_RTx on page 2-10.

`keep_existing_as_is`	TRUE	leave all other channels in their current state [1 H]
	FALSE	start the specified channels and stop all other channels [0 H]

Output Parameters

none

Return Values

ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
bad_chan	If a bad input parameter for `selected_channels`
einval	If an invalid parameter is used as an input
0	If successful

Start_Channel_RTx

Description	Start_Channel_RTx starts the specific channel. All channels must be set up before they are started.	
Syntax	Start_Channel_RTx (int `handle`, usint `channel`)	
Input Parameters	`handle`	The handle designated by Init_Module_RTx.
	`channel`	The channel value, 0 – 9, depending on the board or module. See Read_Number_Of_Channels_RTx on page 2-10.
Output Parameters	none	
Return Values	ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
	param_Start_Channel	If an invalid parameter was used as an input
	0	If successful

Stop_Channel_RTx

Description	Stop_Channel_RTx stops the specific channel.	
Syntax	Stop_Channel_RTx (int `handle`, usint `channel`)	
Input Parameters	`handle`	The handle designated by Init_Module_RTx.
	`channel`	The channel value, 0 – 9, depending on the board or module. See Read_Number_Of_Channels_RTx on page 2-10. STOPALL To stop all channels [10 Dec]
Output Parameters	none	
Return Values	ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
	param_Start_Channel	If an invalid parameter was used as an input
	0	If successful

Stop_Selected_Channels_RTx

Description

Stop_Selected_Channels_RTx simultaneously stops the specified channels.

Syntax

Stop_Selected_Channels_RTx (int `handle`, usint `selected_channels`)

Input Parameters

`handle`	The handle designated by Init_Module_RTx.
`selected_channels`	One or more of the following flags OR'ed together (indicating which channels are to be started simultaneously):

For RT5 modules	For RT10 modules
CHAN_0 [1 H]	CHAN_0 [1 H]
CHAN_1 [2 H]	CHAN_1 [2 H]
CHAN_2 [4 H]	CHAN_2 [4 H]
CHAN_3 [8 H]	CHAN_3 [8 H]
CHAN_4 [10 H]	CHAN_4 [10 H]
ALL_RT5_CHANNELS [1F H]	CHAN_5 [20 H]
All channels are to be stopped simultaneously	CHAN_6 [40 H]
	CHAN_7 [80 H]
	CHAN_8 [100 H]
	CHAN_9 [200 H]
	ALL_RT10_CHANNELS [3FF H]
	All channels are to be stopped simultaneously

Note: To find the number of channels on a module, see Read_Number_Of_Channels_RTx on page 2-10.

Output Parameters

none

Return Values

ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
bad_chan	If a bad input parameter for `selected_channels`
0	If successful

Using Interrupts Under Windows

When writing a Windows program that processes interrupts, a separate thread is generally created to handle the interrupt processing. This thread calls Wait_For_Interrupt_RTx, in order to wait for the next interrupt. When the function returns, the interrupt is processed as needed. This method is demonstrated in the test programs **`demo_int.c`** and **`demo_intms.c`** included with the *429RTx & Discrete 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_RTx function may be used to determine if multiple interrupts have occurred. Conversely, it is possible that the Wait_For_Interrupt_RTx 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_RTx function and the call to Get_Interrupt_Count_RTx.) Once again, the Get_Interrupt_Count_RTx function may be used to determine if the interrupt has already been processed.

The following functions are described below:

Get_Interrupt_Count_RTx

InitializeInterrupt_RTx

Wait_For_Interrupt_RTx

Wait_For_Multiple_Interrupts_RTx

Get_Interrupt_Count_RTx

Description Get_Interrupt_Count_RTx returns the total interrupt count for the specified module from the time the module/card was initialized with Init_[Module/Card]_RTx.

Syntax Get_Interrupt_Count_RTx (int `handle`, unsigned long `*pdwInterruptCount`)

Input Parameters

`handle`	The handle designated by Init_Module_RTx.

Output Parameters

`pwdInterruptCount`	Pointer to an unsigned long which receives the interrrupt count

Return Values

ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
egetintcount	If there was a kernel error
ekernelinitmodule	If error initializing kernel related data
ekernelbadparameter	If input parameter is invalid
ekernelbadpointer	If output parameter buffer is invalid
ekerneldevicenotopen	If specified device has not been opened
0	If successful

InitializeInterrupt_RTx

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

Syntax InitializeInterrupt_RTx (int `handle`)

Input Parameters

`handle`	The handle designated by Init_Module_RTx.

Output Parameters none

Return Values

ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
0	If successful

Wait_For_Interrupt_RTx

Description

Wait_For_Interrupt_RTx waits for an interrupt on the module/card. 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 [FFFFFFFF H], then the call will return only upon receipt of the interrupt.

Syntax

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

Example

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

```
DWORD InterruptThread(int referenceParam)
{
  while (1)
  {
    int status;
    status = Wait_For_Interrupt_RTx (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_RTx (module_handle, &numints);
  }
}
```

Input Parameters

`handle`	The handle designated by Init_Module_RTx.
`timeout`	Timeout is specified in milliseconds, or INFINITE [FFFFFFFF H]

Output Parameters

none

Return Values

	ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
	egeteventhand1	If there is an error in kernel mGetEventHandle, first part
	egeteventhand2	If there is an error in kernel mGetEventHandle, second part
	ekernelinitmodule	If error initializing kernel related data
	ekernelbadparam	If input parameter is invalid
	ekerneldevicenotopen	If specified device was not opened
Successful if *either*	ekerneltimeout	The wait timed out without receiving an interrupt
or	0	If successful

Wait_For_Multiple_Interrupts_RTx

Description

Wait_For_Multiple_Interrupts_RTx waits for an interrupt on any of the specified modules/cards. This function can be used for more than one type of module/card simultaneously. For example an *M4K429RTx* module and a *UNET2*.

The function 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 [FFFFFFFF H], then the call will return only upon receipt of the interrupt.

Syntax

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

Input Parameters

`numints`	Number of entries 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`	Pointer to an unsigned long which receives a bit field indicating which of the modules/cards have interrupted (note that more than one module/card may have interrupted simultaneously). The modules/cards are distributed in the bit field such that the lowest bit corresponds to the first module in the handle_list, and so on.

Return Values

	egeteventhand1	If there is an error in kernel mGetEventHandleForModule, first part
	egeteventhand2	If there is an error in kernel mGetEventHandleForModule, second part
	ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
	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

Using Interrupts Under VISA for VME Carrier Boards

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

These functions are carrier board-level functions and are described in the *Excalibur Carrier Board Software Tools Programmer's Reference*.

3 ARINC 429 Receive Channels Functions

Chapter 3 describes the functions to set and operate ARINC receive channels on *429RTx[D]* modules and cards.

See **Receive Channel Overview** on page 3-2 and **Receive Channel Setup** on page 3-3 for information on how to set up a receive channel.

Note: For the *DAS-429PMC/RTx* board, the software relates to channels 0–9 as module 0 and channels 10–19 as module 1.
For the *DAS-429[cc]PMC/RTxD* board, the software relates to channels 0–9 as module 0; channels 10–14 as module 1.

The following functions are described in this chapter:

General Receive Functions

Set_Global_Storage_Mode_RTx
Setup_Receive_Channel_RTx

Translation Functions

Add_Translation_Entry_RTx
Alter_Translation_Entry_RTx
Enable_Translation_Table_RTx
Map_Label_To_Translation_Entry_RTx

Standard Storage Mode and Data Only Storage Mode Functions

Clear_Rcv_Word_Count_RTx
Clear_Rcvd_Error_Cnt_RTx
Enable_Filter_Table_RTx
Read_Channel_Status_RTx
Read_Next_Data_IRIG_RTx
Read_Next_Data_RTx
Read_Rcvd_Error_Cnt_RTx
Read_Rcvd_Data_Word_Cnt_RTx
Readback_Filter_Entry_RTx
Set_Filter_Entry_RTx
Set_Interrupt_Cond_RTx
Set_Label_Trigger_RTx
Set_Rcv_Interval_Trig_RTx
Set_Rcv_Word_Cnt_Trig_RTx
Set_Rcvd_Error_Cnt_RTx
Set_Trigger_Cond_RTx

Merge Storage Mode Functions

Clear_Merge_Word_Count_RTx
Enable_Merge_Filter_Table_RTx
Read_Merge_Error_Cnt_RTx
Read_Merge_Rcvd_Word_Cnt_RTx
Read_Merge_Status_RTx
Read_Next_Merge_Data_IRIG_RTx
Read_Next_Merge_Data_RTx
Set_Merge_Interval_Trig_RTx
Set_Merge_Intr_Cond_RTx
Set_Merge_Label_Trigger_RTx
Set_Merge_Trig_Cond_RTx
Set_Merge_Word_Cnt_Trig_RTx
Setup_Merge_Mode_RTx

Look-up Table Storage Mode Functions

Enable_Lkup_Table_RTx
Enter_Lkup_Entry_RTx
Read_Lkup_Current_Lbl_RTx
Read_Lkup_Data_RTx

Receive Channel Overview

This section describes each storage mode for receive channels. The next section, **Receive Channel Setup** on page 3-3, provides details on how to set up each mode.

There are three storage modes:

- **Standard Storage Mode** – Standard Storage Mode is the default storage mode. In this mode, data is stored per channel. That is, the data for each channel is stored in a separate area. Each Data Word is stored together with a Time Tag and Status Word.

  In Standard Storage Mode, you have the option of storing the data sequentially (default) or you can use the **Look-up Table Storage** submode. When using Look-up Table Storage Mode, an incoming Data Word is stored together with a Time Tag, Error count and Status Word at a location determined by a Look-up Table. You assign separate buffer areas for labels, and only the most recent data associated with each label is saved. Use this mode when only the most recent value of each label is required and the sequence of labels is not important.

- **Merge Storage Mode** – In this mode, data for all channels is stored sequentially in a common area with a Time Tag and Status Word for each Data Word. In Merge Storage Mode, data is stored sequentially; there is no option to use the Look-up Table Storage submode. If data from different channels is to be processed similarly, using Merge Storage Mode is more efficient than reading data from each channel separately.
- **Data Only Storage Mode** – In this mode, data is stored sequentially per channel. That is, the data for each channel is stored in a separate area. Time Tags and the Status Words are not stored. In Data Only Storage Mode, data is stored sequentially; there is no option to use the Look-up Table Storage submode.

Note: Translation functions are not related to any storage mode. When using translation, the module does not record data at all. Its purpose is to immediately retransmit anything it receives after applying the appropriate translation operators.

This section lists the steps for setting up a receive channel in each storage mode.

Memory Usage Limits

The total available buffer space is approximately 64,000 bytes for both receive and transmit. The buffer space can be can be divided up in any way between the module's channels.

When using a standard (non-IRIG B) Time Tag in Standard Storage Mode or Merge Storage Mode, the module's buffer can store about 6400 ARINC 429 words for all of the module's channels, for both receive and transmit. When using an IRIG B Time Tag, the buffer can store about 4600 ARINC 429 words. In Data Only Storage Mode, the buffer can store about 16,000 ARINC 429 words since the Time Tags and Status Words are not stored.

Transmit messages use the same buffer, but do not usually cause memory space issues unless the bus list is very long.

For more information, see **Appendix A: Data Configuration Returned by Read Data and Load Message Functions**.

Receive Channel Setup

To set up a channel to receive in Standard Storage Mode, call:

1.	Set_Global_Storage_Mode_RTx with the STANDARD_MODE argument. This is the default option.	page 3-5
2.	(Optional) Set_IRIG_Timetag_Mode_RTx, if IRIG B Time Tag format is required.	page 2-13
3.	Setup_Receive_Channel_RTx to set up a receive channel.	page 3-6
4.	(Optional) Set_Label_Trigger_RTx to start receiving stored data only upon reception of a specific label.	page 3-22
5.	(Optional) Set_Rcv_Interval_Trig_RTx and/or Set_Rcv_Word_Cnt_Trig_RTx to generate an interrupt after a certain number of words are received.	page 3-22 page 3-23
6.	(Optional) Set_Interrupt_Cond_RTx and Set_Trigger_Cond_RTx interrupts or triggers are needed.	page 3-21 page 3-24
7.	(Optional) Enable_Filter_Table_RTx, if working with a Filter Table, to allow selective storing of labels and interrupts on selected labels.	page 3-15
8.	(Optional) Enable_Lkup_Table_RTx to set the channel to work in Look-up Table Mode.	page 3-34
9.	(Optional) Enter_Lkup_Entry_RTx for each label that requires a pointer to be entered to the Look-up Table.	page 3-35
10.	Start_Channel_RTx, after all the channels are set up, to start each channel.	page 2-17
11.	To read received data, use: Read_Next_Data_RTx (when using standard Time Tags), Read_Next_Data_IRIG_RTx (when using IRIG B Time Tags) or Read_Lkup_Data_RTx (when using a Look-up Table Mode).	page 3-18 page 3-17 page 3-36

To set up channels to receive in Merge Storage Mode, call:

1.	Set_Global_Storage_Mode_RTx with the MERGE_MODE argument.	page 3-5
2.	(Optional) Set_IRIG_Timetag_Mode_RTx, if IRIG B Time Tag format is required.	page 2-13
3.	Setup_Receive_Channel_RTx to set up each receive channel.	page 3-6

4.	Setup_Merge_Mode_RTx to set the module to work in Merge Storage Mode.	page 3-33
5.	(Optional) Set_Merge_Interval_Trig_RTx and/or Set_Merge_Word_Cnt_Trig_RTx to generate an interrupt after a certain number of words are received.	page 3-30 page 3-32
6.	(Optional) Set_Merge_Intr_Cond_RTx and/or Set_Merge_Trig_Cond_RTx to work with interrupts or triggers.	page 3-30 page 3-31
7.	(Optional) Enable_Merge_Filter_Table_RTx , if working with a Filter Table, to allow selective storing of labels and interrupts on selected labels.	page 3-15
8.	Start_Channel_RTx after all the channels are set up, to start each channel.	page 2-17
9.	To read received data, use: Read_Next_Merge_Data_RTx (when using standard Time Tags) or Read_Next_Merge_Data_IRIG_RTx (when using IRIG B Time Tags).	page 3-29 page 3-28

To set up a channel to receive in Data Only Storage Mode, call:

1.	Set_Global_Storage_Mode_RTx with the DATA_ONLY_MODE argument.	page 3-5
2.	Setup_Receive_Channel_RTx to set up each receive channel.	page 3-6
3.	(Optional) Set_Label_Trigger_RTx to start receiving stored data only upon reception of a specific label.	page 3-22
4.	(Optional) Set_Rcv_Interval_Trig_RTx and/or Set_Rcv_Word_Cnt_Trig_RTx to generate an interrupt after a certain number of words are received.	page 3-22
5.	(Optional) Set_Interrupt_Cond_RTx and Set_Trigger_Cond_RTx interrupts or triggers are needed.	page 3-21 page 3-24
6.	(Optional) Enable_Filter_Table_RTx, if working with a Filter Table, to allow selective storing of labels and interrupts on selected labels.	page 3-15
7.	Start_Channel_RTx, after all the channels are set up, to start each channel.	page 2-17
8.	To read received data, use Read_Next_Data_RTx.	page 3-18

General Receive Functions

General receive functions can be used in any receive storage mode.

Set_Global_Storage_Mode_RTx

Description	Set_Global_Storage_Mode_RTx sets the storage mode for all receive channels. The function must be called before a call to Setup_Receive_Channel_RTx.	
Syntax	Set_Global_Storage_Mode_RTx (int `handle`, usint `mode`)	
Input Parameters	`handle`	The handle designated by Init_Module_RTx.
	`mode`	STANDARD_MODE Each received Data Word is stored together with a Time Tag and a Status Word, in a separate area for each channel. [0 H]
		DATA_ONLY_MODE Each Data Word is stored *without* a Time Tag and a Status Word. Look-up Tables cannot be used. [1 H]
		MERGE_MODE Data Words received by the ARINC channels are stored in a common area together with a Time Tag and a Status Word for each Data Word. [2 H]
Output Parameters	none	
Return Values	ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
	param_Set_Global_Storage_mode	If an invalid parameter is used as an input
	0	If successful

Setup_Receive_Channel_RTx

Description

Setup_Receive_Channel_RTx sets a receive channel to receive. Set_Global_Storage_Mode_RTx *must* be called before Setup_Receive_Channel_RTx in order to specify the storage mode. If the storage mode is set to Merge Mode, the last parameter is not relevant, but must be supplied.

The Setup_Receive_Channel_RTx function can only be set when all the channels are turned off via the Stop_Channel_RTx/ Stop_Discrete_RTD. The card/module should be started via the Start_Channel_RTx/Start_Discrete_RTD functions only after a minimum of 1 msec. from the time that the contents of the function have been changed.

Note: For boards with Discrete channels, the ARINC 429 channels are set independently from the Discrete channels. Therefore:

- Use Start/Stop_Channel_RTx when setting the ARINC channels.
- Use Start/Stop_Discrete_RTD when setting the Discrete channels.

Syntax

Setup_Receive_Channel_RTx (int `handle`, usint `channel`, usint `config`, usint `num_data_words`)

Input Parameters

`handle`	The handle designated by Init_Module_RTx.	
`channel`	The channel value, 0 – 9, depending on the board or module. See Read_Number_Of_Channels_RTx on page 2-10.	
`config`	Combination of 1 flag from each of the relevant groups:	
`bit_rate`	EXC_BIT_RATE_LO	12.5 KHz [0 H]
	EXC_BIT_RATE_HI	100 KHz [1 H]
	BIT_RATE_PROG	Bit rate as set in **Set_Prog_Bit_Rate_RTx** on page 2-14 [2 H]
`parity`	AR_PARITY_ODD	Odd parity [0 H]
	AR_PARITY_EVEN	Even parity [10 H]
	AR_PARITY_OFF	No parity [8 H]
`wrap_around`	NO_WRAP_AROUND	Stop receiving when receiver buffer is full [40 H]
	WRAP_AROUND	Wrap around when receiver buffer is full [0 H]

Setup_Receive_Channel_RTx (cont.)

	num_data_words	Number of 32-bit ARINC 429 Data Words for which to allocate storage. In: **Standard Mode**: Space is automatically allocated for the Time Tags and Status Word. **Merge/Look-up Table Mode:** Set to 0 **Note:** For information on storage limits, see **Memory Usage Limits** on page 3-2.
Output Parameters	none	
	ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
Return Values	param_Setup_Receive_Channel	If an invalid parameter was used as an input.
	bad_rcv_chan	If the channel is not an ARINC receive channel
	mem_Setup_Receive_Channel	If the memory allocation failed
	eboardstarted	If cannot change communication parameters
	0	If successful

Translation Functions

Translation functions are used to receive messages on one channel, modify the messages using one or more translation operators, then retransmit them on the next channel. These functions are not related to any storage mode since the module does not record the data.

Add_Translation_Entry_RTx

Description

Add_Translation_Entry_RTx adds a translation entry that can be applied to any label on any channel. A translation entry defines up to five operations to be performed on an incoming label. To perform an operation, supply a non-zero value as a parameter value. To ignore an operation, supply a 0 as a parameter value. When more than one operation is specified, the operations are performed in the order of this function's parameters.

For example, if non-zero values are supplied for the ANDvalue and ORvalue parameters, the 24 data bits of the incoming label are ANDed with the lowest 24 bits of the ANDvalue, then the result of the AND operation is ORed with the ORvalue. Except for the replace operations, all operations are performed only on the 24 data bits. Assigning a non-zero value to the `Skip` parameters causes the label not to be transmitted at all.

This function is used in conjunction with Enable_Translation_Table_RTx on page 3-12 and Map_Label_To_Translation_Entry_RTx on page 3-13.

You can modify a previously created translation entry by calling Alter_Translation_Entry_RTx on page 3-10.

Note: This function is only available with firmware revision 2.4 or later.

Syntax

Add_Translation_Entry_RTx (int `handle`, unsigned int `ANDvalue`, unsigned int `ORvalue`, unsigned int `XORvalue`, unsigned int `ADDvalue`, unsigned int `SUBTRACTvalue`, unsigned int `NANDvalue`, unsigned int `NORvalue`, unsigned int `REPLACEvalue`, unsigned int `REPLACElabelValue`, unsigned int `Skip`, unsigned int `*TranslationEntryHandle`)

Input Parameters

`handle`	The handle designated by Init_Module_RTx.
`ANDvalue`	A value to be ANDed with the incoming label's data (1 AND 1 == 1, 1 AND 0 == 0, 0 AND 1 == 0, 0 AND 0 == 0).
`ORvalue`	A value to be ORed with the incoming label's data (1 OR 1 == 1, 1 OR 0 == 1, 0 OR 1 == 1, 0 OR 0 == 0).

Add_Translation_Entry_RTx (cont.)

	`XORvalue`	A value to be XORed with the incoming label's data (1 XOR 1 == 0, 1 XOR 0 == 1, 0 XOR 1 == 1, 0 XOR 0 == 0).
	`ADDvalue`	A value to be added to the incoming label's data; this will not affect the label, but may carry into the SSM bits.
	`SUBTRACTvalue`	A value to be subtracted from the incoming label's data; this will not affect the label, but may alter the SDI bits.
	`NANDvalue`	A value to be NANDed with the incoming label's data (1 NAND 1 == 0, 1 NAND 0 == 1, 0 NAND 1 == 1, 0 NAND 0 == 1).
	`NORvalue`	A value to be NORed with the incoming label data (1 NOR 1 == 0, 1 NOR 0 == 0, 0 NOR 1 == 0, 0 NOR 0 == 1).
	`REPLACEvalue`	A value to be substituted for the incoming label and data; all 32 bits will be replaced.
	`REPLACElabelValue`	A value to be substituted for the incoming 8-bit label value; the data will be retransmitted as is.
	`Skip`	If this value is non-zero, the incoming label will not be retransmitted.
Output Parameters	`TranslationEntryHandle`	A handle to this translation entry to be used when calling Map_Label_To_Translation_Entry_RTx.
Return Values	ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
	mem_Enable_Translation_Entry	If there is insufficient memory for an new translation table entry or there are more than five translation operations defined in the translation entry
	0	If successful

Alter_Translation_Entry_RTx

Description

Alter_Translation_Entry_RTx modifies a previously added translation entry that was added by calling Add_Translation_Entry_RTx. A translation entry that can be applied to any label on any channel. A translation entry defines up to five operations to be performed on an incoming label. To perform an operation, supply a non-zero value as a parameter value. To ignore an operation, supply a 0 as a parameter value. When more than one operation is specified, the operations are performed in the order of this function's parameters.

For example, if non-zero values are supplied for the ANDvalue and ORvalue parameters, the 24 data bits of the incoming label are ANDed with the lowest 24 bits of the ANDvalue, then the result of the AND operation is ORed with the ORvalue. Except for the replace operations, all operations are performed only on the 24 data bits. Assigning a non-zero value to the `Skip` parameters causes the label not to be transmitted at all.

This function is used in conjunction with Enable_Translation_Table_RTx on page 3-12 and Map_Label_To_Translation_Entry_RTx on page 3-13.

Note: This function is only available with firmware revision 2.4 or later.

Syntax

Add_Translation_Entry_RTx (int `handle`, unsigned int `ANDvalue`, unsigned int `ORvalue`, unsigned int `XORvalue`, unsigned int `ADDvalue`, unsigned int `SUBTRACTvalue`, unsigned int `NANDvalue`, unsigned int `NORvalue`, unsigned int `REPLACEvalue`, unsigned int `REPLACElabelValue`, unsigned int `Skip`, unsigned int `*TranslationEntryHandle`)

Input Parameters

`handle`	The handle designated by Init_Module_RTx.
`ANDvalue`	A value to be ANDed with the incoming label's data (1 AND 1 == 1, 1 AND 0 == 0, 0 AND 1 == 0, 0 AND 0 == 0).
`ORvalue`	A value to be ORed with the incoming label's data (1 OR 1 == 1, 1 OR 0 == 1, 0 OR 1 == 1, 0 OR 0 == 0).
`XORvalue`	A value to be XORed with the incoming label's data (1 XOR 1 == 0, 1 XOR 0 == 1, 0 XOR 1 == 1, 0 XOR 0 == 0).
`ADDvalue`	A value to be added to the incoming label's data; this will not affect the label, but may carry into the SSM bits.

Alter_Translation_Entry_RTx (cont.)

	`SUBTRACTvalue`	A value to be subtracted from the incoming label's data; this will not affect the label, but may alter the SDI bits.
	`NANDvalue`	A value to be NANDed with the incoming label's data (1 NAND 1 == 0, 1 NAND 0 == 1, 0 NAND 1 == 1, 0 NAND 0 == 1).
	`NORvalue`	A value to be NORed with the incoming label data (1 NOR 1 == 0, 1 NOR 0 == 0, 0 NOR 1 == 0, 0 NOR 0 == 1).
	`REPLACEvalue`	A value to be substituted for the incoming label and data; all 32 bits will be replaced.
	`REPLACElabelValue`	A value to be substituted for the incoming 8-bit label value; the data will be retransmitted as is.
	`Skip`	If this value is non-zero, the incoming label will not be retransmitted.
	`TranslationEntryHandle`	The handle of the translation table entry. This was the output parameter of Add_Translation_Entry_RTx when the translation table entry was created.
Output Parameters	none	
Return Values	ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
	mem_Enable_Translation_Entry	If there is insufficient memory for an new translation table entry or there are more than five translation operations defined in the translation entry
	0	If successful

Enable_Translation_Table_RTx

Description

Enable_Translation_Table_RTx allocates space for a translation table, defines a channel as a translation receive channel and allocates the next channel (channel + 1) as the associated transmit channel over which all translated labels will be retransmitted. It must be called before any calls to Map_Label_To_Translation_Entry_RTx.

This function is used in conjunction with Add_Translation_Entry_RTx on page 3-8 and Map_Label_To_Translation_Entry_RTx on page 3-13.

Note: This function is only available with firmware revision 2.4 or later.

Syntax

Enable_Translation_Table_RTx (int `handle`, usint `channel`)

Input Parameters

`handle`	The handle designated by Init_Module_RTx.
`channel`	The channel to receive the labels, 0 – 8 or 0 – 4, depending on the board or module. See Read_Number_Of_Channels_RTx on page 2-10. The next channel will be used to transmit the translated labels. Therefore, the receive channel cannot be the highest channel on the module.

Output Parameters

none

Return Values

ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
bad_rcv_chan	If the channel is not an ARINC receive channel
bad_tx_chan	If not an ARINC transmit channel
mem_Enable_Translation_Table	If there is not enough memory available for a translation table
0	If successful

Map_Label_To_Translation_Entry_RTx

Description

Map_Label_To_Translation_Entry_RTx associates a channel/label combination with a translation table entry previously defined by calling Add_Translation_Entry_RTx.

Map_Label_To_Translation_Entry_RTx causes all incoming labels whose 8-bit label value matches the `label` parameter defined in this function, to be translated according to the operations defined in the translation table entry. The resulting message is then transmitted on the next channel (`channel` + 1). This function can only be called for channels for which Enable_Translation_Table_RTx has been called.

See **Add_Translation_Entry_RTx** on page 3-8 and **Enable_Translation_Table_RTx** on page 3-12.

Note: This function is only available with firmware revision 2.4 or later.

Syntax

Map_Label_To_Translation_Entry_RTx (int `handle`, usint `channel`, unsigned int `label`, unsigned int `TranslationEntryHandle`)

Input Parameters

`handle`	The handle designated by Init_Module_RTx.
`channel`	The channel to which to apply the translation table entry.
`label`	The 8-bit label to which to apply the translation table entry.
`TranslationEntryHandle`	The handle of the translation table entry. This was the output parameter of Add_Translation_Entry_RTx when the translation table entry was created.

Output Parameters

none

Return Values

ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
bad_rcv_chan	If the channel is not an ARINC receive channel
param_Enable_Translation_Table	If Enable_Translation_Table_RTx was not called for this channel
no_translation_table	If an attempt was made to map to a translation entry on a channel with no translation table
0	If successful

Standard Storage Mode and Data Only Storage Mode Functions

Clear_Rcv_Word_Count_RTx

Description Clear_Rcv_Word_Count_RTx resets the received word count to 0. It should be called only when the channel is inactive, that is, the start is bit set to 0.

Syntax Clear_Rcv_Word_Count_RTx (int `handle`, usint `channel`)

Input Parameters

`handle`	The handle designated by Init_Module_RTx.
`channel`	The channel value, 0 – 9, depending on the board or module. See Read_Number_Of_Channels_RTx on page 2-10.

Output Parameters none

Return Values

ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
param_Rcv_Word_Count	If an invalid parameter was used as an input.
0	If successful

Clear_Rcvd_Error_Cnt_RTx

Description Clear_Rcvd_Error_Cnt_RTx clears the register containing the number of errors encountered on a channel.

Syntax Clear_Rcvd_Error_Cnt_RTx (int `handle`, usint `channel`)

Input Parameters

`handle`	The handle designated by Init_Module_RTx.
`channel`	The channel value, 0 – 9, depending on the board or module. See Read_Number_Of_Channels_RTx on page 2-10.

Output Parameters none

Return Values

ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
param_Read_Rcvd_Error_Cnt	If channel number is out of the valid range
0	If successful

Enable_Filter_Table_RTx

Description

Enable_Filter_Table_RTx sets up a channel to receive only those Data Words that contain a label that has been enabled in the Filter Table. An interrupt is issued when a label is received that has the interrupt bit entered in the Filter Table.

Syntax

Enable_Filter_Table_RTx (int `handle`, usint `channel`, usint `*table_ptr`)

Example

To set up Filter Table if you want to store only label a5

```
usint MyFilterTable[256]={0};

MyFilterTable[0xa5] = 0x01;
Enable_Filter_Table(rcvchan, MyFilterTable);
```

Input Parameters

`handle` The handle designated by Init_Module_RTx.

`channel` The channel value, 0 – 9, depending on the board or module. See Read_Number_Of_Channels_RTx on page 2-10.

`table_ptr` The address of a 256-word array. Word 0 of the array contains the Control word for label 0, word 1 for label 1 etc.

Bit	Description
02-07	0
01	0 = Do not interrupt 1 = Interrupt
00	0 = Do not store 1 = Store Word

NO_FILTER_TABLE_RTX
Cancels a Filter Table operation
[FFFFFFFF H]

Output Parameters

none

Return Values

ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
param_Enable_Filter_Table	If an invalid parameter was used as an input.
mem_Enable_Filter_Table	If the memory allocation failed.
0	If successful

Read_Channel_Status_RTx

Description

Read_Channel_Status_RTx returns and clears the Interrupt status of the specified channel.

Input Parameters

Read_Channel_Status_RTx (int `handle`, int `channel`)

Syntax

`handle`	The handle designated by Init_Module_RTx.
`channel`	The channel value, 0 – 9, depending on the board or module. See Read_Number_Of_Channels_RTx on page 2-10.

Output Parameters

none

Return Values

ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
param_Read_ Channel_Status	If an invalid parameter was used as an input.

One or more of the following flags:

LABEL_RECEIVED
set upon reception of a label which was enabled for interrupt in a Look-up Table or Filter Table [4 H]

INTERVAL_CNT_TRIG
set when a received interval count trigger occurs [8 H] (see **Set_Rcv_Interval_Trig_RTx** on page 3-22)

WORD_CNT_TRIG
set when a received word count trigger occurs [10 H] (see **Set_Rcv_Word_Cnt_Trig_RTx** on page 3-23).

ERROR_RECEIVED
set when a receive error is detected [20 H]

STOP_ON_BUFFER_FULL
Set when reception is stopped due to full receive buffer [40 H]

0	If successful

Read_Next_Data_IRIG_RTx

Description

Read_Next_Data_IRIG_RTx copies received data from the module's/card's receive buffer to the array specified by `'msgptr'`. The first call to this function copies data beginning with the oldest word in the receive buffer. On subsequent calls Read_Next_Data_IRIG_RTx resumes copying data where the previous call left off.

If the entire buffer has been filled since the last time Read_Next_Data_IRIG_RTx was called, some unread data may be overwritten by the new data. This will be reported in the output parameter.

Note:

- This function should be used when using IRIG B Time Tags. Otherwise use Read_Next_Data_RTx. See **Read_Next_Data_RTx** on page 3-18.
- This function is only available for *M4K429RTx* modules Rev. D or later (and all *M8K429RT5* modules) on a carrier board that has IRIG B wired to the module.

Syntax

Read_Next_Data_IRIG_RTx (int `handle`, usint `channel`, usint `num_data_words`, usint `*msgptr`, int `*overwritten`)

Input Parameters

`handle`	The handle designated by Init_Module_RTx.
`channel`	The channel value, 0 – 9, depending on the board or module. See Read_Number_Of_Channels_RTx on page 2-10.
`num_data_words`	Number of 32-bit ARINC 429 Data Words to read.

Output Parameters

`msgptr`	Data returned See **Appendix A: Data Configuration Returned by Read Data and Load Message Functions**.
`overwritten`	DATA_OVERWRITTEN Old data overwritten by new data [-1] NOT_OVERWRITTEN No data was overwritten [0]

Return Values

ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
param_Read_Next_Data	If an invalid parameter was used as an input.
param_Read_Next_Data_IRIG	If this function was used in DATA_ONLY_MODE; DATA_ONLY_MODE is not legal in IRIG_TTAG_MODE.
eNoIrigSupportModule	If the module is not a Nios-based processor and cannot use IRIG-based functions

Read_Next_Data_IRIG_RTx (cont.)

ewrongmode	If not in IRIG_TTAG_MODE
<number of words read>	If successful, returns the number of words read

Read_Next_Data_RTx

Description

Read_Next_Data_RTx copies received data from the module's/ card's receive buffer to the array specified by `'msgptr'`. The first call to this function copies data beginning with the oldest word in the receive buffer. On subsequent calls Read_Next_Data_RTx resumes copying data where the previous call left off.

If the entire buffer has been filled since the last time Read_Next_Data_RTx was called, some unread data may be overwritten by the new data. This will be reported in the output parameter.

Note: When using IRIG B Time Tags, use Read_Next_Data_IRIG_RTx instead. See **Read_Next_Data_IRIG_RTx** on page 3-17.

Syntax

Read_Next_Data_RTx (int `handle`, usint `channel`, usint `num_data_words`, usint `*msgptr`, int `*overwritten`)

Input Parameters

`handle`	The handle designated by Init_Module_RTx.
`channel`	The channel value, 0 – 9, depending on the board or module. See Read_Number_Of_Channels_RTx on page 2-10.
`num_data_words`	Number of 32-bit ARINC 429 Data Words to read

Output Parameters

`msgptr`	Data returned See **Appendix A: Data Configuration Returned by Read Data and Load Message Functions**.
`overwritten`	DATA_OVERWRITTEN Old data overwritten by new data [-1] NOT_OVERWRITTEN No data was overwritten [0]

Return Values

ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
param_Read_Next_Data	If an invalid parameter was used as an input.
<number of words read>	If successful, returns the number of words read

Read_Rcvd_Error_Cnt_RTx

Description Read_Rcvd_Error_Cnt_RTx returns the number of receive errors detected by the specified channel, since the last call to Setup_Receive_Channel_RTx.

Note: This is a 16-bit counter, which returns to 0 after reaching 65,535.

Syntax Read_Rcvd_Error_Cnt_RTx (int `handle`, usint `channel`)

Input Parameters

`handle`	The handle designated by Init_Module_RTx.
`channel`	The channel value, 0 – 9, depending on the board or module. See Read_Number_Of_Channels_RTx on page 2-10.

Output Parameters none

Return Values

ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
param_Read_Rcvd_Error_Cnt	If an invalid parameter was used as an input
<number of receive errors>	If successful, returns the number of receive errors

Read_Rcvd_Data_Word_Cnt_RTx

Description Read_Rcvd_Data_Word_Cnt_RTx returns the number of received 32-bit ARINC 429 Data Words since the last call to Setup_Receive_Channel_RTx.

Note: This is a 16-bit counter, which returns to 0 after reaching 65,535.

Syntax Read_Rcvd_Data_Word_Cnt_RTx (int `handle`, usint `channel`)

Input Parameters

`handle`	The handle designated by Init_Module_RTx.
`channel`	The channel value, 0 – 9, depending on the board or module. See Read_Number_Of_Channels_RTx on page 2-10.

Output Parameters none

Return Values

ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
param_Read_Rcvd_Data_Word_Cnt	If an invalid parameter was used as an input
<number of words read>	If successful, returns the number of words read

Readback_Filter_Entry_RTx

Description Readback_Filter_Entry_RTx reads back one Filter Table entry of a receive channel. The function enables the programmer to read back data written to the dual-port RAM.

Syntax Readback_Filter_Entry_RTx (int `handle`, usint `channel`, uchar `label`)

Input Parameters

`handle`	The handle designated by Init_Module_RTx.
`channel`	The channel value, 0 – 9, depending on the board or module. See Read_Number_Of_Channels_RTx on page 2-10.
`label`	The label of the filter entry

Output Parameters none

Return Values

ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
bad_chan	If an invalid channel is specified
bad_rcv_chan	If the channel is not an ARINC receive channel
no_filter_table	If no Filter Table is defined
<Filter Table entry>	If successful, returns a Filter Table entry

Set_Filter_Entry_RTx

Description Set_Filter_Entry_RTx sets one Filter Table entry of a receive channel.

Syntax Set_Filter_Entry_RTx(int `handle`, usint `channel`, uchar `label`, usint `newentry`)

Input Parameters

`handle`	The handle designated by Init_Module_RTx.
`channel`	The channel value, 0 – 9, depending on the board or module. See Read_Number_Of_Channels_RTx on page 2-10.
`label`	The label of the filter entry
`newentry`	The Filter Table entry

Output Parameters none

Return Values

ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
bad_chan	If an invalid channel is specified
bad_rcv_chan	If the channel is not an ARINC receive channel
no_filter_table	If no Filter Table is defined
0	If successful

Set_Interrupt_Cond_RTx

Description

Set_Interrupt_Cond_RTx sets the condition or conditions under which the specified channel will issue an interrupt. The default is for no interrupts to be generated.

Interrupts can be cancelled by stopping the channel and calling Set_Interrupt_Cond_RTx with the argument 'condition' set to NO_INTERRUPTS.

Syntax

Set_Interrupt_Cond_RTx (int `handle`, usint `channel`, usint `condition`)

Input Parameters

`handle`	The handle designated by Init_Module_RTx.
`channel`	The channel value, 0 – 9, depending on the board or module. See Read_Number_Of_Channels_RTx on page 2-10.
`condition`	One or more of the following values: LABEL_RECEIVED Received a label enabled for interrupts in the Filter Table or Look-up Table [4 H] INTERVAL_CNT_TRIG The received word count is a multiple of the interval specified in **Set_Rcv_Interval_Trig_RTx** on page 3-22 [8 H] WORD_CNT_TRIG Received word count matches count specified in **Set_Rcv_Word_Cnt_Trig_RTx** on page 3-23 [10 H] ERROR_RECEIVED A word with an error injection was received [20 H] STOPPED_ON_BUFFER_FULL The channel is set to stop reception when the buffer is full. Only valid if NO_WRAP_AROUND Mode is selected when the receive channel is set up [40 H] NO_INTERRUPTS Cancels interrupts [0 H]

Output Parameters

none

Return Values

ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
param_Set_Interrupt_Cond	If an invalid parameter is used as an input
noirqset	If no interrupt is allocated
0	If successful

Set_Label_Trigger_RTx

Description Set_Label_Trigger_RTx sets up the channel to start storing received data only after a specified label is received. From this point onwards, all data received with this label is stored.

Syntax Set_Label_Trigger_RTx (int `handle`, usint `channel`, unsigned char `label`)

Input Parameters

`handle`	The handle designated by Init_Module_RTx.
`channel`	The channel value, 0 – 9, depending on the board or module. See Read_Number_Of_Channels_RTx on page 2-10.
`label`	The label which triggers data to be stored.

Output Parameters none

Return Values

ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
param_Set_Label_Trigger	If the parameter is out of range
0	If successful

Set_Rcv_Interval_Trig_RTx

Description Set_Rcv_Interval_Trig_RTx sets the value for the interval count trigger. This sets up the channel to set a bit in the Channel Status Register and to issue an interrupt and/or a trigger every time the received word count is a multiple of the specified interval.

This function must be called prior to calls to:
Set_Interrupt_Cond_RTx on page 3-21
or
Set_Trigger_Cond_RTx on page 3-24

Syntax Set_Rcv_Interval_Trig_RTx (int `handle`, usint `channel`, usint `interval`)

Input Parameters

`handle`	The handle designated by Init_Module_RTx.
`channel`	The channel value, 0 – 9, depending on the board or module. See Read_Number_Of_Channels_RTx on page 2-10.
`interval`	The size of the desired interval: A value 0 – 65535 NO_TRIGGER Cancels the trigger [0 H]

Output Parameters none

Set_Rcv_Interval_Trig_RTx (cont.)

Return Values	ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
	param_Set_Rcv_Interval_Trig	If an invalid parameter is used as an input
	0	If successful

Set_Rcv_Word_Cnt_Trig_RTx

Description Set_Rcv_Word_Cnt_Trig_RTx sets the value for the word count trigger. It must be called prior to calls to:

Set_Interrupt_Cond_RTx on page 3-21
or
Set_Trigger_Cond_RTx on page 3-24

Syntax Set_Rcv_Word_Cnt_Trig_RTx (int `handle`, usint `channel`, usint `count`)

Input Parameters	`handle`	The handle designated by Init_Module_RTx.
	`channel`	The channel value, 0 – 9, depending on the board or module. See Read_Number_Of_Channels_RTx on page 2-10.
	`count`	The word count which causes the trigger
Output Parameters	none	
Return Values	ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
	param_Set_Rcv_Word_Cnt_Trig	If an invalid parameter is used as an input
	0	If successful

Set_Rcvd_Error_Cnt_RTx

Description Set_Rcvd_Error_Cnt_RTx sets the register containing the number of errors encountered on a channel to a specified value.

Syntax Set_Rcvd_Error_Cnt_RTx (int `handle`, usint `channel`, usint `value`)

Input Parameters	`handle`	The handle designated by Init_Module_RTx.
	`channel`	The channel value, 0 – 9, depending on the board or module. See Read_Number_Of_Channels_RTx on page 2-10.
	`value`	The value to store in the register.
Output Parameters	none	
Return Values	ebadhandle	If handle other than the one returned by Init_Module_RTx is used.

Set_Rcvd_Error_Cnt_RTx (cont.)

param_Read_Rcvd_Error_Cnt	If channel number is out of the valid range
0	If successful

Set_Trigger_Cond_RTx

Description Set_Trigger_Cond_RTx sets the condition or conditions under which the specified channel issues a trigger. The default is for no triggers to be issued.

Syntax Set_Trigger_Cond_RTx (int `handle`, usint `channel`, usint `condition`)

Input Parameters

`handle`	The handle designated by Init_Module_RTx.
`channel`	The channel value, 0 – 9, depending on the board or module. See Read_Number_Of_Channels_RTx on page 2-10.
`condition`	One or more of the following values: LABEL_RECEIVED Received a label enabled for interrupt in the Filter Table or Look-up Table [4 H] INTERVAL_CNT_TRIG The received word count is a multiple of the interval specified in **Set_Rcv_Interval_Trig_RTx** on page 3-22 [8 H] WORD_CNT_TRIG Received word count matches count specified in **Set_Rcv_Word_Cnt_Trig_RTx** on page 3-23 [10 H] ERROR_RECEIVED A word with an error condition was received [20 H] STOP_ON_BUFFER_FULL The channel is set to stop reception when the buffer is full. Only valid if NO_WRAP_AROUND Mode is selected when the receive channel is set up [40 H]

Output Parameters none

Return Values

ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
param_Set_Trigger_Cond	If an invalid parameter is used as an input
0	If successful

Merge Storage Mode Functions

Clear_Merge_Word_Count_RTx

Description Clear_Merge_Word_Count_RTx sets the received word count to zero. The function should only be called if all the ARINC channels are inactive, otherwise, the received word count will be meaningless.

Syntax Clear_Merge_Word_Count_RTx (int `handle`)

Input Parameters

`handle`	The handle designated by Init_Module_RTx.

Output Parameters none

Return Values

ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
0	If successful

Enable_Merge_Filter_Table_RTx

Description Enable_Merge_Filter_Table_RTx sets Merge Mode to receive only those Data Words that contain a label that has been enabled in the Filter Table. An interrupt is issued upon reception of a label whose entry in the Filter Table so indicates.

Syntax Enable_Merge_Filter_Table_RTx (int `handle`, usint `*table_ptr`)

Example To set up a Filter Table, to store only label A5 [H]:

```
usint MyFilterTable[256]={0};

MyFilterTable[0xa5] = 0x01;
Enable_Filter_Table(rcvchan, MyFilterTable);
```

Input Parameters

`handle`	The handle designated by Init_Module_RTx.
`table_ptr`	The address of a 256-byte array. Byte 0 of the array contains the Control byte for label 0, byte 1 for label 1 etc.

Bit	Description
02-15	0
01	0 = Do not interrupt 1 = Interrupt
00	0 = Do not store 1 = Store Word

NO_FILTER_TABLE_RTX
Cancels a Filter Table operation. Stop all cards/modules before cancelling a Filter Table operation, otherwise the command will take effect only the next time the card/module is started [FFFFFFFF H]

Enable_Merge_Filter_Table_RTx (cont.)

Output Parameters	none	
Return Values	ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
	mem_Enable_Merge_Filter_Table	If the memory allocation failed
	0	If successful

Read_Merge_Error_Cnt_RTx

Description	Read_Merge_Error_Cnt_RTx returns the received error count in Merge Mode.	
	Note: This is a 16-bit counter, which returns to 0 after reaching 65,535.	
Syntax	Read_Merge_Error_Cnt_RTx (int `handle`)	
Input Parameters	`handle`	The handle designated by Init_Module_RTx.
Output Parameters	none	
Return Values	ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
	<global error count>	If successful, returns the global error count

Read_Merge_Rcvd_Word_Cnt_RTx

Description	Read_Merge_Rcvd_Word_Cnt_RTx returns the received word count in Merge Mode.	
	Note: This is a 16-bit counter, which returns to 0 after reaching 65,535.	
Syntax	Read_Merge_Rcvd_Word_Cnt_RTx (int `handle`)	
Input Parameters	`handle`	The handle designated by Init_Module_RTx.
Output Parameters	none	
Return Values	ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
	<number of received Data Words>	If successful, returns the number of received 32-bit ARINC 429 Data Words stored in Merge Mode since the last call to Setup_Merge_Mode_RTx.

Read_Merge_Status_RTx

Description	Read_Merge_Status_RTx returns and clears the Merge Interrupt Status.	
Syntax	Read_Merge_Status_RTx (int `handle`)	
Input Parameters	`handle`	The handle designated by Init_Module_RTx.
Output Parameters	none	
Return Values	ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
	<interrupt status>	If successful, returns the interrupt status as a combination of the following flags: LABEL_RECEIVED set upon reception of a label which was enabled for interrupt in a Look-up Table or Filter Table [4 H] INTERVAL_CNT_TRIG set when a received interval count trigger occurs [8 H] (see **Set_Merge_Interval_Trig_RTx** on page 3-30) WORD_CNT_TRIG set when a received word count trigger occurs [10 H] (see **Set_Merge_Word_Cnt_Trig_RTx** on page 3-32) ERROR_RECEIVED set when a receive error is detected [20 H] STOP_ON_BUFFER_FULL set when reception is stopped due to a full receive buffer [40 H]

Read_Next_Merge_Data_IRIG_RTx

Description

Read_Next_Merge_Data_IRIG_RTx copies received 32-bit ARINC 429 Data Words together with their associated Time Tags and Status Words from the module's/card's common storage area to the array pointed to by `msgptr`.

The first call to this function copies data starting with the oldest word in the common storage area. On the next and subsequent calls, Read_Next_Merge_Data_IRIG_RTx resumes copying data where the previous calls left off.

Note:

- This function should be used when using IRIG B Time Tags. Otherwise use Read_Next_Merge_Data_RTx. See **Read_Next_Merge_Data_RTx** on page 3-29.
- This function is only available for *M4K429RTx* modules Rev. D or later (and all *M8K429RT5* modules) on a carrier board that has IRIG B wired to the module.

Syntax

Read_Next_Merge_Data_IRIG_RTx (int `handle`, usint `num_data_words`, usint `*msgptr`, int `*overwritten`)

Input Parameters

`handle`	The handle designated by Init_Module_RTx.
`num_data_words`	Number of 32-bit ARINC 429 Data Words to read. The maximum number of Data Words to be copied is specified by `num_data_words.` The size of the specified array, where each element is a 16-bit word, must be at least 7× `num_data_words` 16-bit words in order to accommodate Data, Time Tag and Status Word.

Output Parameters

`msgptr`	Data returned See **Appendix A: Data Configuration Returned by Read Data and Load Message Functions**.
`overwritten`	DATA_OVERWRITTEN Old data was overwritten by new data [-1] NOT_OVERWRITTEN Data stored successfully; no data was overwritten [0]

Return Values

ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
eNoIrigSupportModule	If the module is not a Nios-based processor and cannot use IRIG-based functions
ewrongmode	If not in IRIG_TTAG_MODE
<number of words read>	If successful, returns the number of words read

Read_Next_Merge_Data_RTx

Description

Read_Next_Merge_Data_RTx copies received 32-bit ARINC 429 Data Words together with their associated Time Tags and Status Words from the module's/card's common storage area to the array pointed to by `msgptr`.

The first call to this function copies data starting with the oldest word in the common storage area. On the next and subsequent calls, Read_Next_Merge_Data_RTx resumes copying data where the previous calls left off.

Note: When using IRIG B Time Tags, use Read_Next_Merge_Data_IRIG_RTx instead. See **Read_Next_Merge_Data_IRIG_RTx** on page 3-28.

Syntax

Read_Next_Merge_Data_RTx (int `handle`, usint `num_data_words`, usint `*msgptr`, int `*overwritten`)

Input Parameters

`handle`	The handle designated by Init_Module_RTx.
`num_data_words`	Number of 32-bit ARINC 429 Data Words to read. The maximum number of Data Words to be copied is specified by `num_data_words.` The size of the specified array, where each element is a 16-bit word, must be at least 5× `num_data_words` 16-bit words in order to accommodate Data, Time Tag and Status Word.

Output Parameters

`msgptr`	Data returned See **Appendix A: Data Configuration Returned by Read Data and Load Message Functions**.
`overwritten`	DATA_OVERWRITTEN Old data was overwritten by new data [-1] NOT_OVERWRITTEN Data stored successfully; no data was overwritten [0]

Return Values

ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
<number of words read>	If successful, returns the number of words read

Set_Merge_Interval_Trig_RTx

Description Set_Merge_Interval_Trig_RTx sets the value of the interval count trigger. It must be set prior to calls to:
Set_Merge_Intr_Cond_RTx on page 3-30
or
Set_Merge_Trig_Cond_RTx on page 3-31

Syntax Set_Merge_Interval_Trig_RTx (int `handle`, usint `interval`)

Input Parameters

`handle`	The handle designated by Init_Module_RTx.
`interval`	The requested interval A value 0 – 65535

Output Parameters none

Return Values

ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
0	If successful

Set_Merge_Intr_Cond_RTx

Description Set_Merge_Intr_Cond_RTx sets the condition or conditions for Merge Mode to issue an interrupt. The default is for no interrupts to be issued.

Syntax Set_Merge_Intr_Cond_RTx (int `handle`, usint `condition`)

Input Parameters

`handle`	The handle designated by Init_Module_RTx.
`condition`	LABEL_RECEIVED Set upon reception of a label which was enabled for interrupt in a Look-up Table or Filter Table [4 H] INTERVAL_CNT_TRIG The received word count is a multiple of interval specified in **Set_Merge_Interval_Trig_RTx** on page 3-30 [8 H] WORD_CNT_TRIG Issue an interrupt every time the received Word count matches count specified in **Set_Merge_Word_Cnt_Trig_RTx** on page 3-32 [10 H] ERROR_RECEIVED A word with an error condition was received [20 H] STOP_ON_BUFFER_FULL The channel is set to stop reception when the buffer is full. Only valid if NO_WRAP_AROUND Mode is selected when the receive channel is set up [40 H]

Set_Merge_Intr_Cond_RTx (cont.)

		NO_INTERRUPTS Cancels interrupts [0 H]
Output Parameters	none	
Return Values	ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
	noirqset	No interrupt allocated
	0	If successful

Set_Merge_Label_Trigger_RTx

Description	Set_Merge_Label_Trigger_RTx sets up Merge Mode to start storing received data only on receiving a specified label the first time.	
Syntax	Set_Merge_Label_Trigger_RTx (int `handle`, unsigned char `label`)	
Input Parameters	`handle`	The handle designated by Init_Module_RTx.
	`label`	The first label to be stored
Output Parameters	none	
Return Values	ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
	0	If successful

Set_Merge_Trig_Cond_RTx

Description	Set_Merge_Trig_Cond_RTx sets the condition or conditions under which Merge Mode issues a trigger. The default is for no triggers to be issued.	
Syntax	Set_Merge_Trig_Cond_RTx (int `handle`, usint `condition`)	
Input Parameters	`handle`	The handle designated by Init_Module_RTx.
	`condition`	LABEL_RECEIVED Set upon reception of a label which was enabled for interrupt in a Look-up Table or Filter Table [4 H] INTERVAL_CNT_TRIG Set when the received word count is a multiple of interval specified in **Set_Merge_Interval_Trig_RTx** on page 3-30 [8 H] WORD_CNT_TRIG Issue an interrupt every time the received Word count matches count specified in **Set_Merge_Word_Cnt_Trig_RTx** on page 3-32 [10 H]

Set_Merge_Trig_Cond_RTx (cont.)

		ERROR_RECEIVED A word with an error condition was received [20 H] STOP_ON_BUFFER_FULL The channel is set to stop reception when the buffer is full. Only valid if NO_WRAP_AROUND Mode is selected when the receive channel is set up [40 H] NO_INTERRUPTS Cancels interrupts [0 H]
Output Parameters	none	
Return Values	ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
	0	If successful

Set_Merge_Word_Cnt_Trig_RTx

Description	Set_Merge_Word_Cnt_Trig_RTx sets the value for the Word Count Trigger in Merge Mode. It must be set prior to calls to: **Set_Merge_Intr_Cond_RTx** on page 3-30 *or* **Set_Merge_Trig_Cond_RTx** on page 3-31	
Syntax	Set_Merge_Word_Cnt_Trig_RTx (int `handle`, usint `count`)	
Input Parameters	`handle`	The handle designated by Init_Module_RTx.
	count	The count which causes the trigger
Output Parameters	none	
Return Values	ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
	0	If successful

Setup_Merge_Mode_RTx

Description

Setup_Merge_Mode_RTx sets the module to receive in Merge Storage Mode. Set_Merge_Mode_RTx must be called if the storage mode is set to Merge Storage Mode in Set_Global_Storage_Mode_RTx.

A common storage area is set up for all ARINC receive channels in the active module. The area is large enough to contain the number of 32-bit ARINC 429 Data Words specified by `num_data_words` together with the associated Time Tags and Status Words.

Note: Setup_Receive_Channel_RTx must be called for each of the individual active receive channels whose outputs are to be merged together.

Syntax

Setup_Merge_Mode_RTx (int `handle`, usint `num_data_words`, usint `wrap_around`)

Input Parameters

`handle`	The handle designated by Init_Module_RTx.
`num_data_words`	The number of 32-bit ARINC 429 Data Words the buffer must hold.
`wrap_around`	WRAP_AROUND If data should wrap around to the top of the buffer when the buffer has been filled [0 H] NO_WRAP_AROUND reception will stop when the receive area is full; the flag STOP_ON_BUFFER_FULL will be set in the Channel Status [40 H]

Output Parameters

none

Return Values

ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
mem_Setup_Merge_Mode	If the memory allocation failed
0	If successful

Look-up Table Storage Mode Functions

Look-up Table Storage Mode is a submode available in Standard Storage Mode. For more information, see **Receive Channel Overview** on page 3-2.

Enable_Lkup_Table_RTx

Description

Enable_Lkup_Table_RTx sets the channel to work in Look-up Table Mode. In this mode only the most recent data associated with each label selected by Enter_Lkup_Entry_RTx is kept.

This mode is used when the most recent value of each label is needed and the sequence of labels is unimportant. Following a call to this function, Enter_Lkup_Entry_RTx must be called to fill in the Look-up Table.

Syntax

Enable_Lkup_Table_RTx (int `handle`, usint `channel`)

Input Parameters

`handle`	The handle designated by Init_Module_RTx.
`channel`	The channel value, 0 – 9, depending on the board or module. See Read_Number_Of_Channels_RTx on page 2-10.

Output Parameters

none

Return Values

ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
param_Enable_Lkup_Table	If an invalid parameter was used as an input
mem_Enable_Lkup_Table	If the memory allocation failed
mode_Enable_Lkup_Table	If in receive Data Only Storage Mode
0	If successful

Enter_Lkup_Entry_RTx

Description Enter_Lkup_Entry_RTx allocates space for the selected label.

Syntax Enter_Lkup_Entry_RTx (int `handle`, usint `channel`, unsigned char `label`, usint `intr`)

Input Parameters

`handle`	The handle designated by Init_Module_RTx.
`channel`	The channel value, 0 – 9, depending on the board or module. See Read_Number_Of_Channels_RTx on page 2-10.
`label`	The label being set up
`intr`	INTERRUPT Causes an interrupt to be generated by the module when this label is received [8000 H] LABEL_RECEIVED Appears in the Channel Status when the specified label is received [4 H]; if Set_Interrupt_Cond_RTx, page 3-21 or Set_Trigger_Cond_RTx, page 3-24 are called with LABEL_RECEIVED, an interrupt is issued on this condition NO_INTERRUPT Cancels the interrupt process [0 H]

Output Parameters none

Return Values

ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
param_Enter_Lkup_Entry	If an invalid parameter was used as an input
mem_Enter_Lkup_Entry	If the memory allocation failed
0	If successful

Read_Lkup_Current_Lbl_RTx

Description Read_Lkup_Current_Lbl_RTx returns the Look-up Table Current Pointer.

Syntax Read_Lkup_Current_Lbl_RTx (int `handle`, usint `channel`, uchar `*label`)

Input Parameters

`handle`	The handle designated by Init_Module_RTx.
`channel`	The channel value, 0 – 9, depending on the board or module. See Read_Number_Of_Channels_RTx on page 2-10.

Output Parameters

`label`	The current label

Read_Lkup_Current_Lbl_RTx (cont.)

Return Values	ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
	param_Read_Lkup_Current_Label	If an invalid parameter was used as an input
	0	If successful

Read_Lkup_Data_RTx

Description

Read_Lkup_Data_RTx copies the contents of the Receiver Data Block for the specified channel and label to the array pointed to by a message pointer.

The Receiver Data Block consists of a:

- 32-bit ARINC 429 Data Word
- 32-bit Time Tag
- 16-bit received error count
- 16-bit Status Word

Syntax

Read_Lkup_Data_RTx (int `handle`, usint `channel`, unsigned char `label`, usint *`msgptr`)

Input Parameters	`handle`	The handle designated by Init_Module_RTx.
	`channel`	The channel value, 0 – 9, depending on the board or module. See Read_Number_Of_Channels_RTx on page 2-10.
	`label`	The label being read
Output Parameters	`msgptr`	The data returned. See **Appendix A: Data Configuration Returned by Read Data and Load Message Functions**.
Return Values	ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
	param_Read_Lkup_Data	If an invalid parameter was used as an input
	lkup_data_invalid	If the Look-up Table data is invalid
	0	If successful

4 ARINC 429 Transmit Channels Functions

Chapter 4 describes the functions to set and operate ARINC transmit channels on *429RTx[D]* modules and cards.

The following functions are described in this chapter:

General Transmit Functions

Allocate_Message_RTx
Load_FrequencyMessage_RTx
Load_Message_RTx
Read_Channel_Status_RTx
Read_Tx_Loop_Cnt_RTx
Read_Tx_Total_Words_Sent_RTx
Set_Interrupt_Cond_RTx
Set_Skip_Message_RTx
Set_Transmit_Loop_Counter_RTx
Set_Transmit_Message_Once_RTx
Set_Trigger_Cond_RTx
Setup_Frame_RTx
Setup_Transmit_Channel_RTx

FIFO Mode Functions

Allocate_MAX_Transmit_FIFOs_RTx
Allocate_Transmit_FIFO_RTx
Load_Transmit_FIFO_RTx
Set_Transmit_FIFO_Size_RTx

Data Reconstructor Mode (Time Tag Mode) Functions

Add_TTAG_Data_RTx
Setup_Ttag_Mode_RTx

Note: For the *DAS-429PMC/RTx* board, the software relates to channels 0–9 as module 0; channels 10–19 as module 1.
For the *DAS-429[cc]PMC/RTxD* board, the software relates to channels 0–9 as module 0; channels 10–14 as module 1.

Transmit Channel Overview

There are several modes that can be used for transmit channels that will determine the timing method with which the data will be transmitted. This section describes each mode. The next section, **Transmit Channel Setup** on page 4-4, provides details on how to set up each mode.

Three of the modes, Interblock Gap Mode, Data Rate Mode and FIFO Mode are selected via the `tx_mode` parameter of Setup_Transmit_Channel_RTx. The fourth mode, Data Reconstructor Mode (also called Time Tag Mode), is selected by calling Setup_Ttag_Mode_RTx after calling Setup_Transmit_Channel_RTx.

- **Interblock Gap Mode** – In this mode, you define blocks of ARINC 429 labels to be transmitted consecutively. You define the number of labels in each block, the number of bit times to wait between the transmission of labels within the block, and the amount of time to wait between the transmitting of one block to the next. All defined blocks form a frame, and the frame is transmitted the number of times specified in Setup_Transmit_Channel_RTx.
- **Data Rate Mode** – In this mode, you set up blocks of data and define the number of labels in each block and the number of bit times to wait between transmission of labels within the block. However, in this mode you select how often the block will be transmitted.

  For example, you can request Block 1 be sent every 20 msec. and block 2 every 50 msec. In this case, the blocks would be sent as follows:

Time in Milliseconds	Data Sent
0	Blocks 1 and 2
20	Block 1
40	Block 1
50	Block 2
60	Block 1
80	Block 1
100	Blocks 1 and 2

- **FIFO Mode** – In this mode, multiple buffers (called FIFOs) can be set up for each channel. Each buffer contains labels that will be transmitted at a selected data rate. For example, if a buffer containing ten labels is setup to be transmitted every 100 msec., the first label in the buffer will be transmitted following a call to Start_Channel_RTx, and the next label in the buffer will be sent after 100 msec., the third after 200 msec., and so on.

  When the end of the buffer is reached, the module will continue transmission with data from the beginning of the buffer. You should make the buffer large enough to guarantee the application will have time to refill the buffer before it is emptied. This mode is used when you know in advance what data is to be transmitted for an extended period of time. This can be useful for file transmission.

- **Data Reconstructor Mode** – Data Reconstructor Mode (also called Time Tag Mode) is for retransmitting previously recorded labels with the same timing as the original transmission. This mode is primarily for debug purposes. In this mode, no blocks are defined. The structure used to determine the order of transmission of labels is the same as the structure used for receive channels. That is, it is comprised of 5 16-bit words per label as follows:

Word	Description
Word 1	High word of data to be sent
Word 2	Low word of data to be sent
Word 3	High word of time tag to be sent
Word 4	Low word of Time Tag to be sent
Word 5	Status of word to be sent

In this mode, the module transmits the first label in memory when Start_Channel_RTx is called, the next label is transmitted based on the gap between the Time Tag of the first label and the Time Tag of the second label. For example, if the first label has a Time Tag of 1,705,000 and the second label has a Time Tag of 1,705,200 then the first label is sent out immediately upon start and the second label is sent 200 Time Tag units later, which is 2000 µsec. or 2 msec. Succeeding labels are sent based on their Time Tags in order to match the timing of the original transmission.

When the end of the buffer is reached, the firmware wraps around to the beginning of the buffer. The `flag` parameter of Setup_Ttag_Mode_RTx determines what the module does at this point.

If the recording is small enough to fit into the module's memory, you have the option of settings the `flag` parameter to TTAGBASEDLOOP, and the labels will be transmitted continuously in a loop.

If the previous recording is too long to fit into the module's memory, you should not use the TTAGBASEDLOOP value for the `flag` parameter. You should set the `flag` parameter to 0, and when the end of the buffer is reached, the firmware will wrap around to the beginning of the buffer and wait for the first label's Time Tag to be greater than the Time Tag of the previously transmitted label, i.e., the one at the end of the buffer. In this case, your program must call Add_TTAG_Data_RTx periodically to reload the module's memory with new labels as the old labels are transmitted.

Transmit Channel Setup

To set up transmit channels in Interblock Gap Mode or Data Rate Mode, call:

1.	Init_Module_RTx to initialize the module.	page 2-4
1.	Setup_Transmit_Channel_RTx to set up a transmit channel transmission.	page 4-19
2.	Allocate_Message_RTx to allocate a unique message number in the message stack.	page 4-6
3.	Load_Message_RTx or Load_FrequencyMessage_RTx to load data to be transmitted.	page 4-9
4.	(Optional) Set_Skip_Message_RTx to skip a specific message and transmit it only on demand.	page 4-14
5.	(Optional) Set_Interrupt_Cond_RTx or Set_Trigger_Cond_RTx to work with interrupts and/or external triggers.	page 4-13 and page 4-17
6.	Start_Channel_RTx or Start_Selected_Channels_RTx after all the channels are set up.	page 2-17
7.	(Optional) Set_Transmit_Message_Once_RTx to transmit a message once.	page 4-16

To set up transmit channels in FIFO Mode, call:

1.	Init_Module_RTx to initialize the module.	page 2-4
1.	Setup_Transmit_Channel_RTx to set up a transmit channel transmission.	page 4-19
2.	Allocate_MAX_Transmit_FIFOs_RTx to set maximum the number of different FIFOs to be used.	page 4-21
3.	Allocate_Transmit_FIFO_RTx to allocate space for each FIFO to be used.	page 4-22
4.	Set_Transmit_FIFO_Size_RTx to set the actual size of a particular FIFO.	page 4-24
5.	Load_Transmit_FIFO_RTx to load data into the FIFO.	page 4-23
6.	Start_Channel_RTx or Start_Selected_Channels_RTx after all the channels are set up.	page 2-17
7.	When there is too much data to fit into the FIFO, call Load_Transmit_FIFO_RTx periodically to load data while the data is being transmitted.	page 4-23

To set up transmit channels in Data Reconstructor Mode (Time Tag Mode), call:

1.	Init_Module_RTx and set the desired channels to transmit.	page 2-4
2.	Setup_Transmit_Channel_RTx to set up a transmit channel transmission.	page 4-19
3.	Setup_Ttag_Mode_RTx to set up Data Reconstructor Mode (Time Tag Mode).	page 4-26
4.	Add_TTAG_Data_RTx to load data to be transmitted.	page 4-25
5.	Start_Channel_RTx or Start_Selected_Channels_RTx after all the channels are set up.	page 2-17
6.	When there is too much data to fit into the module's memory, call Add_TTAG_Data_RTx periodically to load data while the data is being transmitted.	page 4-25

General Transmit Functions

General transmit functions are relevant in more than one transmit mode.

Allocate_Message_RTx

Description

Allocate_Message_RTx allocates a message in the message stack. A message consists of 1 – 255 ARINC words that are transmitted consecutively and with the same interword gap time.

Each message is given a message number. For each channel, the first message is message number 1. The message numbers increment by 1 for subsequent messages.

For each call to this function, a call must be made to **Load_Message_RTx** on page 4-9, using the allocated message number. You can allocate multiple messages by calling this function (and Load_Message_RTx) multiple times. For details on memory size, see **Memory Usage Limits** on page 3-2.

A call to **Setup_Transmit_Channel_RTx** on page 4-19, clears the message stack and message numbers.

Syntax

Allocate_Message_RTx (int `handle`, usint `channel`, usint `max_msg_size`)

Input Parameters

`handle`	The handle designated by Init_Module_RTx.
`channel`	The channel value, 0 – 9, depending on the board or module. See Read_Number_Of_Channels_RTx on page 2-10.
`max_msg_size`	Number of ARINC words in the message

Return Values

ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
param_Allocate_Message	If an invalid parameter was used as an input
mem_Allocate_Message	If the memory allocation failed
msg_Allocate_Message	If there are too many calls to this function; Too many calls to Allocate_Message_RTx; the number of messages cannot exceed `max_messages` in Setup_Transmit_Channel_RTx
<number of the new message allocated>	If successful, returns the number of the new message allocated

Load_FrequencyMessage_RTx

Description

Load_FrequencyMessage_RTx loads a message previously allocated by **Allocate_Message_RTx** on page 4-6. This function defines the number of labels in the message and its timing characteristics including how frequently the message will be transmitted, how much time between transmission of labels within the message and how long after the start of the channel will the first label in the message be transmitted.

Note: This function is only available with firmware revision 3.9 or later.

Syntax

Load_FrequencyMessage_RTx (int `handle`, usint `channel`, usint `msg_num`, usint `erj_inj`, usint `word_cnt`, usint `interword_dly`, usint `frequency`, usint `offset`, usint `*msgptr`)

Input Parameters

`handle`	The handle designated by Init_Module_RTx.
`channel`	The channel value, 0 – 9, depending on the board or module. See Read_Number_Of_Channels_RTx on page 2-10.
`msg_num`	Message number of a message created by Allocate_Message_RTx
`err_inj`	Error injection – one or more of the following flags: NO_ERROR_429 No error injection [0 H] PARITY_429_ERROR Incorrect parity inserted in every word message [1 H] NULL_BIT_ERROR null bit error inserted in ARINC bit 02 of every word in message [2 H] STRETCH_BIT_ERROR ARINC bit 02 of every word is stretched causing a Manchester coding error [4 H] BIT_CNT_HI_ERROR Every word in message transmitted with 33 bits [8 H] BIT_CNT_LO_ERROR Every word in message transmitted with 31 bits [10 H] SUPPRESS_PARITY_ERROR Forces no parity condition on every word in message [20 H]

Load_FrequencyMessage_RTx (cont.)

	word_cnt	Number of 32-bit ARINC 429 Data Words in a message. This number of words will be copied from the specified array to the module's/card's memory. This value must be less than or equal to the parameter max_msg_size passed to **Allocate_Message_RTx** on page 4-6.
	interword_dly	Number of bit times inserted between consecutive words within the message. Allowed values are 0 – 255. 4 is the minimum legal value according to the ARINC 429 specification.
	frequency	How much time to wait between transmissions of the message. If Setup_Transmit_Channel_RTx was called with EXTENDED_TIME_ENABLE, the frequency is in milliseconds with a range of 1 to 1310. Otherwise, it is in bit times with a range of 1 to 65,535. For example, to transmit a high speed message every 40 milliseconds, set this value to 4000.
	offset	How long to wait for the first transmission following start of the channel. To load balance, divide the labels from a single frequency into N parts and adjust the offset by frequency/N for each message. For example, if you need to transmit 20 labels every 40 milliseconds, you can have 4 messages of 5 labels, all having a frequency of 4000 and the offsets 0 for Message 0, 1000 for Message 1, 2000 for Message 2 and 3000 for Message 3.
	msgptr	This is the pointer to an integer array containing the 32-bit ARINC 429 Data Words to be loaded.
Output Parameters	none	
Return Values	ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
	esetuptxchan	If Setup_Transmit_Channel_RTx was not called for this channel
	eenhancedfrequencynotsupported	If the firmware on this module does not support this Enhanced Frequency function
	msg_num_Load_Message	If msg_num is invalid; was not returned by a call to Allocate_Message_RTx
	enotdatarate mode	If Setup_Transmit_Channel_RTx did not set this channel to be in DATA_RATE_MODE

Load_FrequencyMessage_RTx (cont.)

param_Load_Message	If the channel does not exist on the board
param_Load_Message1	If `word_cnt` is greater than 255 or 0
param_Load_Message2	If `interword_dly` is greater than 255
param_Load_Message3	If Setup_Transmit_Channel_RTx specifies EXTENDED_TIME_ENABLE and `frequency` is greater than 1310 or less than 1
param_Load_Message4	If Setup_Transmit_Channel_RTx did not specify EXTENDED_TIME_ENABLE and `frequency` is greater than 65535 (FFFF H) or less than 0
param_Load_Message5	If there is not enough space left on the module for `word_cnt` words
0	If successful

Load_Message_RTx

Description

Load_Message_RTx copies data from an array to a message in the message stack. This function must be called at least once for each message allocated by **Allocate_Message_RTx** on page 4-6. The function can be called periodically while the channel is running, to change a message in real time.

Note: In Data Rate Mode, you can also use Load_FrequencyMessage_RTx, which allows you to select the start of message transmission, relative to other Data Rate Mode messages. See **Load_FrequencyMessage_RTx** on page 4-7.

Syntax

Load_Message_RTx (int `handle`, usint `channel`, usint `msg_num`, usint `erj_inj`, usint `word_cnt`, usint `interword_dly`, usint `intermsg_rate`, usint `*msgptr`)

Input Parameters

`handle`	The handle designated by Init_Module_RTx.
`channel`	The channel value, 0 – 9, depending on the board or module. See Read_Number_Of_Channels_RTx on page 2-10.
`msg_num`	Message number of a message created by Allocate_Message_RTx
`err_inj`	Error injection – one or more of the following flags: NO_ERROR_429 No error injection [0 H]

Load_Message_RTx (cont.)

	PARITY_429_ERROR Incorrect parity inserted in every word message [1 H]
	NULL_BIT_ERROR null bit error inserted in ARINC bit 02 of every word in message [2 H]
	STRETCH_BIT_ERROR ARINC bit 02 of every word is stretched causing a Manchester coding error [4 H]
	BIT_CNT_HI_ERROR Every word in message transmitted with 33 bits [8 H]
	BIT_CNT_LO_ERROR Every word in message transmitted with 31 bits [10 H]
	SUPPRESS_PARITY_ERROR Forces no parity condition on every word in message [20 H]
`word_cnt`	Number of 32-bit ARINC 429 Data Words in a message. This number of words will be copied from the specified array to the module's/card's memory. This value must be less than or equal to the parameter max_msg_size passed to **Allocate_Message_RTx** on page 4-6.
`interword_dly`	Number of bit times inserted between consecutive words within the message. Allowed values are 0 – 255.
`intermsg_rate`	In INTERBLOCK_GAP_MODE, this specifies the number of bit times to be inserted between this message and the next one. Allowed values are 0-65535. If Extended Time is enabled on the channel, this value is in msec. and the allowed values are 1 – 1300 when using a high bit rate, and 8 – 10480 when using a low bit rate. See **Setup_Transmit_Channel_RTx** on page 4-19. In DATA_RATE_MODE, this specifies the period of the message in bit times (if the value is *n* the message is transmitted every *n* bit times).
`msgptr`	This is the pointer to an integer array containing the 32-bit ARINC 429 Data Words to be loaded.

Load_Message_RTx (cont.)

Output Parameters	none	
Return Values	ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
	esetuptxchan	If Setup_Transmit_Channel_RTx was not called for this channel
	msg_num_Load_Message	If `msg_num` is invalid; was not returned by a call to Allocate_Message_RTx
	param_Load_Message	If the channel does not exist on the board
	param_Load_Message1	If `word_cnt` is greater than 255 or 0
	param_Load_Message2	If `interword_dly` is greater than 255
	param_Load_Message3	If Setup_Transmit_Channel_RTx specifies EXTENDED_TIME_ENABLE and `intermsg_rate` is greater than 1310 or less than 1
	param_Load_Message4	If Setup_Transmit_Channel_RTx did not specify EXTENDED_TIME_ENABLE and `intermsg_rate` is greater than 65535 (FFFF H) or less than 0
	param_Load_Message5	If there is not enough space left on the module for `word_cnt` words
	0	If successful

Read_Channel_Status_RTx

Description	Read_Channel_Status_RTx returns and clears the status of the specific channel.	
Syntax	Read_Channel_Status_RTx (int `handle`, usint `channel`)	
Input Parameters	`handle`	The handle designated by Init_Module_RTx.
	`channel`	The channel value, 0 – 9, depending on the board or module. See Read_Number_Of_Channels_RTx on page 2-10.
Output Parameters	none	
Return Values	ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
	param_Read_Channel_Status	If an invalid parameter was used as an input

Read_Channel_Status_RTx (cont.)

	<status of the channel>	If successful, returns the status of the channel: TX_END_OF_BLOCK Set when a transmit channel has completed transmission of a block (message) [1 H] TX_END_OF_FRAME Set when a transmit channel has completed transmission of an entire stack of blocks (messages) [2 H]

Read_Tx_Loop_Cnt_RTx

Description	Read_Tx_Loop_Cnt_RTx reads the current transmit loop register. This quantity counts down from its initial value set in Setup_Transmit_Channel_RTx until it reaches 1.	
Syntax	Read_Tx_Loop_Cnt_RTx (int `handle`, usint `channel`)	
Input Parameters	`handle`	The handle designated by Init_Module_RTx.
	`channel`	The channel value, 0 – 9, depending on the board or module. See Read_Number_Of_Channels_RTx on page 2-10.
Output Parameters	none	
Return Values	ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
	param_Read_Tx_Loop_Cnt	If an invalid parameter was used as an input
	<number of loops left to send>	If successful, returns the number of loops left to send

Read_Tx_Total_Words_Sent_RTx

Description	Read_Tx_Total_Words_Sent_RTx reads the total number of words sent. **Note:** This is a 16-bit counter, which returns to 0 after reaching 65,535.	
Syntax	Read_Tx_Total_Words_Sent _RTx (int `handle`, usint `channel`)	
Input Parameters	`handle`	The handle designated by Init_Module_RTx.
	`channel`	The channel value, 0 – 9, depending on the board or module. See Read_Number_Of_Channels_RTx on page 2-10.
Output Parameters	none	

Read_Tx_Total_Words_Sent_RTx (cont.)

Return Values	ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
	param_Read_Tx_Total_Words_Sent	If an invalid parameter was used as an input
	<number of words sent>	If successful, returns the total number of words sent

Set_Interrupt_Cond_RTx

Description

Set_Interrupt_Cond_RTx sets the condition or conditions for which the specified channel issues an interrupt. The default is for no interrupts to be issued.

Syntax

Set_Interrupt_Cond_RTx (int `handle`, usint `channel`, usint `condition`)

Input Parameters	`handle`	The handle designated by Init_Module_RTx.
	`channel`	The channel value, 0 – 9, depending on the board or module. See Read_Number_Of_Channels_RTx on page 2-10.
	`condition`	TX_END_OF_BLOCK An entire block (message) has been transmitted [1 H]
		TX_END_OF_FRAME An entire frame (stack of messages) has been transmitted [2 H]
		NO_INTERRUPTS Cancels interrupts [0 H]
Output Parameters	none	
	ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
	param_Set_Interrupt_Cond	If an invalid parameter was used as an input
	0	If successful

Set_Skip_Message_RTx

Description

Set_Skip_Message_RTx causes a message that has been loaded by Load_Message_RTx not to be transmitted. All other messages will be transmitted as scheduled.

Note: This function is only available with firmware revision 1.47 or later.

Syntax

Set_Skip_Message_RTx (int `handle`, usint `channel`, usint `msg_num`, usint `skip_flag`)

Input Parameters

`handle`	The handle designated by Init_Module_RTx.
`channel`	The channel over which the message was supposed to be transmitted, 0 – 9, depending on the board or module. See Read_Number_Of_Channels_RTx on page 2-10.
`msg_num`	Message number used by Load_Message_RTx to load the message
`skip_flag`	One of the following: SKIP_MESSAGE Skip this message [40 H] RESTORE_MESSAGE Allow the skipped message to start transmitting again as scheduled [0 H]

Output Parameters

none

Return Values

ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
param_Skip_ Message	If an invalid `channel` was used
msg_num_Skip_ Message	If an invalid `msg_num` was used
0	If successful

Set_Transmit_Loop_Counter_RTx

Description

Set_Transmit_Loop_Counter_RTx sets the loop counter to how many times an instruction stack should be sent.

Note: The function should be called only when the channel is *not* running. If the function is called when the channel is running, it will not take effect. It will take effect the next time the transmit channel is started.

Syntax

Set_Transmit_Loop_Counter_RTx (int `handle`, usint `channel`, usint `loop_cntr`)

Input Parameters

`handle`	The handle designated by Init_Module_RTx.
`channel`	The channel value, 0 – 9, depending on the board or module. See Read_Number_Of_Channels_RTx on page 2-10.
`loop_cntr`	CONTINUOUS Transmit message stack continuously [0 H] 1 – 65535 Transmit message stack this number of times

Output Parameters

none

Return Values

ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
bad_tx_chan	If not an ARINC transmit channel
0	If successful

Set_Transmit_Message_Once_RTx

Description

Set_Transmit_Message_Once_RTx causes a message that was set to be skipped, to be transmitted once, after which the firmware returns it to be skipped. This allows you to send asynchronous messages.

To use this function, set up the message using Allocate_Message_RTx and Load_Message_RTx, set the message to be skipped using Set_Skip_Message_RTx, then call this function to transmit the message asynchronously. After the message is transmitted, the firmware sets the message back to being skipped.

When is this asynchronous message transmitted?

In Interblock Gap Mode, the message will be transmitted when the appropriate block is called (in sequence).

In Data Rate Mode, the message will be transmitted according to its frequency, relative to the last Time Tag of this message. Hence, this is the appropriate method for transmitting the message immediately. The frequency should be set to the smallest interval this message might be transmitted. For example, if this message will be sent at least 65 milliseconds apart, set its frequency to 65 milliseconds. If this message might be sent one millisecond apart, set its frequency to one millisecond.

Note: This function is only available with firmware revision 2.7 and later.

Syntax

Set_Transmit_Message_Once_RTx (int `handle`, usint `channel`, usint `msg_num`)

Input Parameters

`handle`	The handle designated by Init_Module_RTx.
`channel`	The channel value, 0 – 9, depending on the board or module. See Read_Number_Of_Channels_RTx on page 2-10.
`msg_num`	Message number of a message created by Allocate_Message_RTx

Output Parameters

none

Return Values

ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
efeature_not_supported_RTX	If feature is not supported on this module
param_Skip_Message	If an invalid `channel` was used

Set_Transmit_Message_Once_RTx (cont.)

esetuptxchan	If Setup_Transmit_Channel_RTx was not called for this channel
msg_num_ Skip_Message	If an invalid `msg_num` was used
0	If successful

Set_Trigger_Cond_RTx

Description Set_Trigger_Cond_RTx sets the condition or conditions under which the specified channel issues a trigger. The default is for no triggers to be issued.

Syntax Set_Trigger_Cond_RTx (int `handle`, usint `channel`, usint `condition`)

Input Parameters

`handle`	The handle designated by Init_Module_RTx.
`channel`	The channel value, 0 – 9, depending on the board or module. See Read_Number_Of_Channels_RTx on page 2-10.
`condition`	TX_END_OF_BLOCK An entire block (message) has been transmitted [1 H] TX_END_OF_FRAME An entire frame (stack of messages) has been transmitted [2 H] NO_TRIGGER Cancels triggers [0 H]

Output Parameters none

Return Values

ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
param_Set_ Trigger_Cond	If an invalid parameter was used as an input
0	If successful

Setup_Frame_RTx

Description

Setup_Frame_RTx sets up a frame. The user indicates which message to begin transmitting and how many messages in the stack to transmit.

Syntax

Setup_Frame_RTx (int `handle`, usint `channel`, usint `begin_message`, usint `end_message`)

Input Parameters

`handle`	The handle designated by Init_Module_RTx.
`channel`	The channel value, 0 – 9, depending on the board or module. See Read_Number_Of_Channels_RTx on page 2-10.
`begin_message`	Message number of first message to transmit Valid values: 1 – number of messages loaded *or* FIRST_MESSAGE Start from the first message allocated [1 H]
`end_message`	Message number of last message to transmit Valid values: 1 – number of messages loaded *or* LAST_MESSAGE Transmit until the last message allocated [0 H]

Output Parameters

none

Return Values

ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
bad_tx_chan	If not an ARINC transmit channel
bad_param	If an invalid parameter was used as an input
0	If successful

Setup_Transmit_Channel_RTx

Description

Setup_Transmit_Channel_RTx is the first function to call to set up and operate the ARINC transmit channels.

The Setup_Transmit_Channel_RTx function can only be set when all the channels are turned off via the Stop_Channel_RTx/ Stop_Discrete_RTD functions. The card/module should be started via the Start_Channel_RTx/Start_Discrete_RTD functions only after a minimum of 1 msec. from the time that the contents of the function have been changed.

Note: For boards with Discrete channels, the ARINC 429 channels are set independently from the Discrete channels. Therefore:

- Use Start/Stop_Channel_RTx when setting the ARINC channels.
- Use Start/Stop_Discrete_RTD when setting the Discrete channels.

Syntax

Setup_Transmit_Channel_RTx (int `handle`, usint `channel`, usint `config`, usint `max_messages`, usint `loop_cntr`)

Input Parameters

`handle`	The handle designated by Init_Module_RTx.	
`channel`	The channel value, 0 – 9, depending on the board or module. See Read_Number_Of_Channels_RTx on page 2-10.	
`config`	Combination of 1 flag from each of the relevant groups:	
`bit_rate`	EXC_BIT_RATE_LO	12.5 KHz [0 H]
	EXC_BIT_RATE_HI	100 KHz [1 H]
	BIT_RATE_PROG	bit rate as set in **Set_Prog_Bit_Rate_RTx** on page 2-14 [2 H]
`parity`	AR_PARITY_ODD	odd parity [0 H]
	AR_PARITY_EVEN	even parity [10 H]
	AR_PARITY_OFF	no parity [8 H]
`tx_mode`	INTERBLOCK_GAP_MODE Send each message with a specified delay between messages (default mode) [0 H] DATA_RATE_MODE Send each message according to the specified rate [100 H] FIFO_DATA_MODE Set up FIFO buffers for each channel [1000 H]	

Note: For Data Reconstructor Mode (Time Tag Mode), use any of the above values for `tx_mode`, then call Setup_Ttag_Mode_RTx.

Setup_Transmit_Channel_RTx (cont.)

	`tx_rise`	RISE_HI_SPEED tx rise/fall time 1.5+/-0.5 µsec. [0 H] (default value; for hi speed bit rate, a hi speed rise/fall time must be selected.) RISE_LO_SPEED tx rise/fall time 10+/-5 µsec. [4 H]
	`ext_time`	EXTENDED_TIME_ENABLE Extended Time is enabled on the channel [400 H]; this feature increases the transmit timing (see **Load_Message_RTx** on page 4-9) **Note:** This bit is only relevant when using INTERBLOCK_GAP_MODE for the `tx_mode` parameter, and Extended Time is supported by the firmware. See **Read_Board_Status_RTx** on page 2-7. EXTENDED_TIME_DISABLE Extended Time is disabled on the channel (default) [0 H]
	`max_messages`	The maximum number of messages to be defined in the instruction stack
	`loop_cntr`	Number of times to transmit an entire stack. CONTINUOUS Transmit message stack continuously [0 H] 1 – 65535 Transmit message stack this number of times **Note:** When using DATA_RATE_MODE or FIFO_DATA_MODE for the `tx_mode` parameter, the `loop_cntr` parameter is ignored and the message is transmitted continuously. See **`tx_mode` on page 4-19**.
Output Parameters	none	
Return Values	ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
	param_Setup_ Transmit_Channel	If the parameter is out of range
	mem_Setup_ Transmit_Channel	If the memory allocation failed
	bad_tx_chan	If not an ARINC transmit channel
	eboardstarted	If cannot change communication parameters
	no_extended_ time_support	If Extended Time is enabled on the channel but not supported by the firmware
	0	If successful

FIFO Mode Functions

Allocate_MAX_Transmit_FIFOs_RTx

Description

Allocate_MAX_Transmit_FIFOs_RTx allocates memory for a table of pointers to FIFOs for the selected channel. The memory allocation for the FIFOs themselves is done by calling Allocate_Transmit_FIFO_RTx.

Allocate_MAX_Transmit_FIFOs_RTx must be called before any calls to Allocate_Transmit_FIFO_RTx. The `numFIFOs` parameter determines the maximum number of times Allocate_Transmit_FIFO_RTx can be called for that channel.

Note: This function is only available with firmware revision 2.4 or later.

Syntax

Allocate_MAX_Transmit_FIFOs_RTx (int `handle`, usint `channel`, usint `numFIFOs`)

Input Parameters

`handle`	The handle designated by Init_Module_RTx.
`channel`	The channel value, 0 – 9, depending on the board or module. See Read_Number_Of_Channels_RTx on page 2-10.
`numFIFOs`	The maximum number of FIFOs that can be allocated for this channel.

Output Parameters

none

Return Values

ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
0	If successful

Allocate_Transmit_FIFO_RTx

Description

Allocate_Transmit_FIFO_RTx allocates memory for a FIFO for the selected channel. Labels associated with a particular FIFO will be transmitted at a rate of one label every '`frequency`' µsec.

The `FIFOmaxSize` parameter sets the maximum number of labels that can be stored in the FIFO using Load_Transmit_FIFO_RTx.

For example, if Allocate_Transmit_FIFO_RTx is called with `FIFOmaxSize` = 10 and `frequency` = 50000, you can then load 10 labels using Load_Transmit_FIFO_RTx and they will be sent out at times 0, 50000 µsec., 100000 µsec., etc., until the final label is sent at 450,000 µsec. (450 msec.).

You can call Load_Transmit_FIFO_RTx during this transmission to top off the FIFO, and the transmission will continue as long as fresh data is available.

Note:

- Allocate_MAX_Transmit_FIFOs_RTx must be called before calling this function.
- This function is only available with firmware revision 2.4 or later.

Syntax

Allocate_Transmit_FIFO_RTx (int `handle`, usint `channel`, int `FIFOnumber`, int `FIFOmaxSize`, int `frequency`)

Input Parameters

`handle`	The handle designated by Init_Module_RTx.
`channel`	The channel value, 0 – 9, depending on the board or module. See Read_Number_Of_Channels_RTx on page 2-10.
`FIFOnumber`	The ID number to assign to this FIFO. This number is used when loading data into the FIFO via Load_Transmit_FIFO_RTx. The valid range is 0 to `numFIFOs`–1. (`numFIFOs` is the last parameter in Allocate_MAX_Transmit_FIFOs_RTx.)
`FIFOmaxSize`	The maximum number of labels that can be stored in this FIFO.
`frequency`	The number of µsec. between the start of one label's transmission until the start of the next label's transmission.

Return Values

ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
0	If successful

Load_Transmit_FIFO_RTx

Description

Load_Transmit_FIFO_RTx loads one or more words into a FIFO previously allocated by calling Allocate_Transmit_FIFO_RTx.

If your application is constantly loading data to the FIFO, you generally do not want the application to wait for the FIFO to be empty before loading data, since this would harm the transmission timing. To help avoid the FIFO being empty when loading data, this function has two output parameters, `actualLoaded` and `actualStartIndex`, that enable you to calculate how full the FIFO was at the time the data was loaded.

Note:

- You must call Set_Transmit_FIFO_Size_RTx before calling this function.
- This function is only available with firmware revision 2.4 or later.

Syntax

Load_Transmit_FIFO_RTx (int `handle`, usint `channel`, int `FIFOnumber`, struct FIFO_DATA_CONTROL *`data`, int `numWords`, int *`actualLoaded`, int *`actualStartIndex`)

Input Parameters

`handle`	The handle designated by Init_Module_RTx.
`channel`	The channel value, 0 – 9, depending on the board or module. See Read_Number_Of_Channels_RTx on page 2-10.
`FIFOnumber`	The ID number of the desired FIFO. This number was assigned when calling Allocate_Transmit_FIFO_RTx.
`data`	A pointer to a structure of type:

```
struct FIFO_DATA_CONTROL {
   int control; //error injection associated with the 'data' in this
                  //entry; see note below
   int data; //ARINC 429 Data Word
};
```

Note: For error injection options, see the `err_inj` parameter of **Load_Message_RTx** on page 4-9.

`numWords`	The number of entries within the structure pointed to by `data` that the user has filled in.

Output Parameters

`actualLoaded`	A pointer to an integer that will be filled in by this function with the actual number of words loaded into the FIFO. This may be fewer than `numWords` if `numWords` is greater than the `FIFOsize` parameter of Set_Transmit_FIFO_Size_RTx or if the FIFO still contains unused data and does not have enough room for `numWords` additional words.

Load_Transmit_FIFO_RTx (cont.)

	`actualStartIndex`	A pointer to an integer that will be filled in by this function. The integer is the index in the FIFO that was loaded with the first Data Word in `data` structure.
Return Values	ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
	0	If successful

Set_Transmit_FIFO_Size_RTx

Description

Set_Transmit_FIFO_Size_RTx determines the number of labels to be stored in the FIFO. This number cannot be greater than the `FIFOsize` parameter that was set when Allocate_Transmit_FIFO_RTx was called.

When this number of labels is loaded into the FIFO, Load_Transmit_FIFO_RTx wraps around to the beginning of the FIFO.

This functions allows you to allocate the maximum size when creating the FIFO (Allocate_Transmit_FIFO_RTx) and decide later how large of a FIFO you want to actually use. You can call Set_Transmit_FIFO_Size_RTx later in your program to change the size of FIFO, as long as it is not larger than `FIFOsize`.

Note: This function is only available with firmware revision 2.4 or later.

Syntax

Set_Transmit_FIFO_Size_RTx (int `handle`, usint `channel`, int `FIFOnumber`, int `FIFOsize`)

Input Parameters	`handle`	The handle designated by Init_Module_RTx.
	`channel`	The channel value, 0 – 9, depending on the board or module. See Read_Number_Of_Channels_RTx on page 2-10.
	`FIFOnumber`	The ID number of the desired FIFO. This number was assigned when calling Allocate_Transmit_FIFO_RTx.
	`FIFOsize`	The actual number of labels to be stored in the FIFO before Load_Transmit_FIFO_RTx wraps around to the beginning of the FIFO.
Output Parameters	none	
Return Values	ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
	0	If successful

Data Reconstructor Mode Functions

The following functions are used in Data Reconstructor Mode (also called Time Tag Mode). This mode reconstructs received data and transmits it on the ARINC 429 bus. The data transmission is synchronized based on the Time Tags of the received messages. You can also use the *Data Reconstructor 429* utility provided by Excalibur.

See **Transmit Channel Overview** on page 4-2 for details on using this mode and see **Transmit Channel Setup** on page 4-4 for the correct order to call these functions. For a sample application using these functions, see the demo program **`demo_ttagxmt.c`**.

Add_TTAG_Data_RTx

Description

Add_TTAG_Data_RTx adds one or more labels to transmit. Each label is accompanied by a Time Tag and a Status Word.

The Time Tag determines when to transmit; the Status Word is used for data injection and to select the channel on which to transmit.

Note: This function is only available with firmware revision 1.47 or later.

Syntax

Add_TTAG_Data_RTx (int `handle`, struct SEQ_DATA `*pSEQdata`, int `numlabels`, int `*labelsAdded`)

Input Parameters

`handle` The handle designated by Init_Module_RTx.

`pSEQdata` A pointer to the list of labels and data to be added to the list. The struct is as follows:

```
struct SEQ_DATA{
   usint DataHi;
   usint DataLo;
   usint TimeTagHi;
   usint TimeTagLo;
   usint Status;
};
```

Note: Data Words, Time Tags and Status Word are described in **Appendix A: Data Configuration Returned by Read Data and Load Message Functions**.

`numlabels` The number of labels to add to the list

Output Parameters

`labelsAdded` The number of labels added to the list. This will be less than `numlabels` if there is not enough room for `numlabels` new entires in the buffer. This function does not overwrite labels that have not been sent yet.

Return Values

ebadhandle If handle other than the one returned by Init_Module_RTx is used.

0 If successful

Setup_Ttag_Mode_RTx

Description

Setup_Ttag_Mode_RTx sets up the module or card to transmit previously recorded data. The user fills memory with data recorded in Merge Storage Mode of a previous run of the module/card. The format of the data is exactly the same as it was when it was recorded.The `pMap` parameter is an array of ten unsigned short integers used to map input to output channels. For example, to transmit all data that was received by Channel 0 over Channel 2 and to transmit all data that was received on Channel 1 over Channel 7, fill the first entries pointed to by `pMap` follows:

```
pMap[0]=2;
pMap[1]=7;
```

Note: This function is only available with firmware revision 1.47 or later.

Syntax

Setup_Ttag_Mode_RTx (int `handle`, usint `flag`, usint *`pMap`)

Input Parameters

`handle`	The handle designated by Init_Module_RTx.
`flag`	One of the following flags: TTAGBASEDLOOP The labels will be transmitted continuously in a loop [8 H] 0 When the end of the buffer is reached, the firmware will wrap around to the beginning of the buffer and wait for the first label's Time Tag to be greater than the Time Tag of the previously transmitted label, i.e., the one at the end of the buffer.
`pMap`	A pointer to array containing the input to output channel mapping.

Output Parameters

none

Return Values

ebadhandle	If handle other than the one returned by Init_Module_RTx is used.
0	If successful

5 Discrete Channels Functions

Chapter 5 contains descriptions of all the functions necessary to write test programs for the Discrete channels [0–3] on modules that have both ARINC 429 and Discrete channels on the **same** module.

Note: This chapter does not apply to products that have separate ARINC 429 and Discrete modules. If you are not sure if your product has both ARINC 429 and Discrete on the same module, contact Technical Support. See **Technical Support** on page 1-8. (For working with Discretes on a separate module, see *Discrete Software Tools Programmer's Reference*.)

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.

For card level functions, Init_Module_RT*x*, Release_Module_RT*x* and Get_Error_String_RT*x* see **Chapter 2 General Functions**.

To handle interrupts see **Using Interrupts Under Windows** on page 2-19.

The following functions are described in this chapter:

Get_HWid_RTD	Set_Threshold_RTD
Read_All_Input_Discretes_RTD	Set_Trigger_RTD
Read_Input_Discrete_RTD	Set_Trigger_Destination_RTD
Read_Output_Discrete_RTD	Set_Trigger_Mask_RTD
Read_Pending_RTD	Set_Trigger_Value_RTD
Read_Pending_Register_RTD	Start_Discrete_RTD
Reset_RTD	Stop_Discrete_RTD
Reset_Pending_RTD	Write_Output_Discrete_RTD
Reset_Pending_Register_RTD	

Get_HWid_RTD

Description	Get_HWid_RTD returns the hardware ID	
Syntax	Get_HWid_RTD _RTD (int `handle`, usint `*hwid`)	
Input Parameters	`handle`	The handle designated by Init_Card_RTx.
Output Parameters	`hwid`	Returns the current version of the module
Return Values	ebadhandle	If handle other than the one returned by Init_Card_RTx is used
	enodiscretes	If no module is present with both ARINC 429 and Discrete channels
	0	If successful

Read_All_Input_Discretes_RTD

Description	Read_All_Input_Discretes_RTD reads all input channels.	
Syntax	Read_All_Input_Discretes_RTD (int `handle`, usint `*readbuf`)	
Input Parameters	`handle`	The handle designated by Init_Card_RTx
Output Parameters	readbuf	Each bit represents value of an input Discrete for a channel: bit 0 = channel 0 bit 1 = channel 1 bit 2 = channel 2 bit 3 = channel 3
Return Values	ebadhandle	If handle other than the one returned by Init_Card_RTx is used
	enodiscretes	If no module is present with both ARINC 429 and Discrete channels
	0	If successful

Read_Input_Discrete_RTD

Description Read_Input_Discrete_RTD reads a specific input channel.

Syntax Read_Input_Discrete_RTD (int `handle`, usint `inchannel`, usint `*value`)

Input Parameters

`handle`	The handle designated by Init_Card_RTx
`inchannel`	The input channel: a value 0 – 3

Output Parameters

value	0 low 1 high

Return Values

ebadhandle	If handle other than the one returned by Init_Card_RTx is used
ebaddischan	If an illegal channel was specified
enodiscretes	If no module is present with both ARINC 429 and Discrete channels
0	If successful

Read_Output_Discrete_RTD

Description Read_Output_Discrete_RTD reads output for a specific channel.

Syntax Read_Output_Discrete_RTD (int `handle`, usint `outchannel`, usint `*value`)

Input Parameters

`handle`	The handle designated by Init_Card_RTx
`outchannel`	The output channel: a value 0 – 3

Output Parameters

value	0 low 1 high

Return Values

ebadhandle	If handle other than the one returned by Init_Card_RTx is used
ebaddischan	If an illegal channel was specified
enodiscretes	If no module is present with both ARINC 429 and Discrete channels
0	If successful

Read_Pending_RTD

Description Read_Pending_RTD reads the pending value for a specific channel, from the Interrupt Pending Register

Syntax Read_Pending_RTD (int `handle`, usint `inchannnel`, usint `*pending_val`)

Input Parameters

`handle`	The handle designated by Init_Card_RTx
`inchannel`	The input channel: a value 0 – 3

Output Parameters

pending_val	0 input channel was not triggered 1 input channel was triggered

Return Values

ebadhandle	If handle other than the one returned by Init_Card_RTx is used
ebaddischan	If an illegal channel was specified
enodiscretes	If no module is present with both ARINC 429 and Discrete channels
0	If successful

Read_Pending_Register_RTD

Description Read_Pending_Register_RTD reads the Pending Interrupt Register.

Syntax Read_Pending_Register_RTD (int `handle`, usint `*pending_reg`)

Input Parameters

`handle`	The handle designated by Init_Card_RTx

Output Parameters

`pending_reg`	For each bit (0 – 3) 0 corresponding input channel was not triggered 1 corresponding input channel was triggered

Return Values

ebadhandle	If handle other than the one returned by Init_Card_RTx is used
ebaddischan	If an illegal channel was specified
enodiscretes	If no module is present with both ARINC 429 and Discrete channels
0	If successful

Reset_RTD

Description Reset_RTD resets the Discrete I/O channel to its default values. Discrete I/O is stopped.

Syntax Reset_RTD (int `handle`)

Input Parameters	`handle`	The handle designated by Init_Card_RTx
Output Parameters	none	
Return Values	ebadhandle	If handle other than the one returned by Init_Card_RTx is used
	enodiscretes	If no module is present with both ARINC 429 and Discrete channels
	0	If successful

Reset_Pending_RTD

Description Reset_Pending_RTD resets the bit of the Pending Register corresponding to the specific input channel.

Syntax Read_Input_Discrete_RTD (int `handle`, usint `inchannel`)

Input Parameters	`handle`	The handle designated by Init_Card_RTx
	`inchannel`	The input channel: a value 0 – 3
Output Parameters	none	
Return Values	ebadhandle	If handle other than the one returned by Init_Card_RTx is used
	ebaddischan	If an illegal channel was specified
	enodiscretes	If no module is present with both ARINC 429 and Discrete channels
	0	If successful

Reset_Pending_Register_RTD

Description Reset_Pending_Register_RTD resets the specified bits of the Interrupt Pending Register.

Syntax Reset_Pending_Register_RTD (int `handle`, usint `pending_reg`)

Input Parameters

`handle`	The handle designated by Init_Card_RTx
pending_reg	One or more of the following flags O'Red together (indicating which channels should be reset): RESET_CHAN_0 [1 H] RESET_CHAN_1 [2 H] RESET_CHAN_2 [4 H] RESET_CHAN_3 [8 H]

Output Parameters none

Return Values

ebadhandle	If handle other than the one returned by Init_Card_RTx is used
ebaddischan	If an illegal channel was specified
enodiscretes	If no module is present with both ARINC 429 and Discrete channels
0	If successful

Set_Threshold_RTD

Description Set_Threshold_RTD sets the threshold voltage range.

Syntax Set_Threshold_RTD (int `handle`, usint `threshold`)

Input Parameters

`handle`	The handle designated by Init_Card_RTx
`threshold`	THRESH_TTL 0 – 5V (default) [1 H] THRESH_AV 0 – 32V [0 H]

Output Parameters none

Return Values

ebadhandle	If handle other than the one returned by Init_Card_RTx is used
einval	If an illegal threshold was specified
enodiscretes	If no module is present with both ARINC 429 and Discrete channels
0	If successful

Set_Trigger_RTD

Description Set_Trigger_RTD sets triggering on/off and if the trigger/ interrupt will be activated by a rising or a falling edge input for a specific channel.

Syntax Set_Trigger_RTD (int `handle`, usint `inchannel`, int `triggerval`)

Input Parameters

`handle`	The handle designated by Init_Card_RTx
`inchannel`	The input channel: a value 0 – 3
triggerval	TRIGGER_OFF no trigger [2 H] TRIGGER_FALLING trigger falling edge [1 H] TRIGGER_RISING trigger rising edge [0 H]

Output Parameters none

Return Values

ebadhandle	If handle other than the one returned by Init_Card_RTx is used
einval	If an illegal trigger value was specified
enodiscretes	If no module is present with both ARINC 429 and Discrete channels
0	If successful

Set_Trigger_Destination_RTD

Description If a trigger occurs Set_Trigger_Destination_RTD sets whether or not an interrupt will occur.

Syntax Set_Trigger_Destination_RTD (int `handle`, usint `dest`)

Input Parameters

`handle`	The handle designated by Init_Card_RTx
`dest`	DEST_INTERRUPT Sets an interrupt on trigger [1 H] DEST_NONE No interrupt on trigger [0 H]

Output Parameters none

Return Values

ebadhandle	If handle other than the one returned by Init_Card_RTx is used
einval	If an invalid value was specified as in input
enodiscretes	If no module is present with both ARINC 429 and Discrete channels
0	If successful

Set_Trigger_Mask_RTD

Description

Set_Trigger_Mask_RTD sets the triggering on/off for each input channel. It specifies which channel should be monitored for the activation of the interrupt line and or the external trigger.

Syntax

Set_Trigger_Mask_RTD (int `handle`, usint `mask`)

Input Parameters

`handle`	The handle designated by Init_Card_RTx
mask	For each input channel to trigger on, the corresponding bit should be set to 1: TRIG_CHAN_0 [1 H] TRIG_CHAN_1 [2 H] TRIG_CHAN_2 [4 H] TRIG_CHAN_3 [8 H] TRIG_ALL_CHANS [7FFF H]

Output Parameters

none

Return Values

ebadhandle	If handle other than the one returned by Init_Card_RTx is used
einval	If an invalid value was specified as in input
enodiscretes	If no module is present with both ARINC 429 and Discrete channels
0	If successful

Set_Trigger_Value_RTD

Description Set_Trigger_Value_RTD specifies for each input channel if the trigger/interrupt will be activated by a rising edge or a falling edge input. Rising edge is the default value.

Syntax Set_Trigger_Mask_RTD (int `handle`, usint `value`)

Input Parameters

`handle`	The handle designated by Init_Card_RTx
value	One or more of these flags OR'ed together may be used to specify which channels should trigger on falling edge: CHAN_0_FALL [1 H] CHAN_1_FALL [2 H] CHAN_2_FALL [4 H] CHAN_3_FALL [8 H] ALL_CHANS_FALL [7FFF H]

Output Parameters none

Return Values

ebadhandle	If handle other than the one returned by Init_Card_RTx is used
einval	If an invalid value was specified as in input
enodiscretes	If no module is present with both ARINC 429 and Discrete channels
0	If successful

Start_Discrete_RTD

Description Start_Discrete_RTD starts the Discrete input / output channels. All channels must be set up before they are started

Syntax Start_Discrete_RTD (int `handle`)

Input Parameters

`handle`	The handle designated by Init_Card_RTx

Output Parameters none

Return Values

ebadhandle	If handle other than the one returned by Init_Card_RTx is used
enodiscretes	If no module is present with both ARINC 429 and Discrete channels
0	If successful

Stop_Discrete_RTD

Description Stop_Discrete_RTD stops the Discrete input / output channels.

Syntax Stop_Discrete_RTD (int `handle`)

Input Parameters

`handle`	The handle designated by Init_Card_RTx

Output Parameters none

Return Values

ebadhandle	If handle other than the one returned by Init_Card_RTx is used
enodiscretes	If no module is present with both ARINC 429 and Discrete channels
0	If successful

Write_Output_Discrete_RTD

Description Write_Output_Discrete_RTD writes the output Discrete for a specific channel.

Syntax Write_Output_Discrete_RTD (int `handle`, usint `outchannel`, usint `value`)

Input Parameters

`handle`	The handle designated by Init_Card_RTx
`outchannel`	The output channel: a value 0 – 3
value	0 low 1 high

Output Parameters none

Return Values

ebadhandle	If handle other than the one returned by Init_Card_RTx is used
ebadchan	If an illegal channel was specified
enodiscretes	If no module is present with both ARINC 429 and Discrete channels
0	If successful

Appendix A Data Configuration Returned by Read Data and Load Message Functions

The data returned by Read_Next_Data_RTx and Read_Next_Merge_Data_RTx consists of 5-word blocks.

The data returned by Read_Next_Data_IRIG_RTx and Read_Next_Merge_Data_IRIG_RTx consists of 7-word blocks.

Each block has the following configuration:

For Non-IRIG B Time

Word 0	DATA WORD HI
Word 1	DATA WORD LO
Word 2	TIME TAG HI
Word 3	TIME TAG LO
Word 4	STATUS

For IRIG B Time

Word 0	DATA WORD HI
Word 1	DATA WORD LO
Word 2	TIME TAG HI
Word 3	TIME TAG SECOND
Word 4	TIME TAG THIRD
	TIME TAG FOURTH
	STATUS

When not using IRIG B time, the Time Tag precision is 10 µsec. When using IRIG B time, the Time Tag precision is 1 µsec. The IRIG B Time Tag is described on page A-2.

The Status Word is a combination of the following flags:

WORD_RECEIVED	Indicates that a Data Word has been written
HI_BIT_CNT_ERR	Hi bit count error
LO_BIT_CNT_ERR	Lo bit count error
PARITY_ERR	Parity error
INVALID_CODING_ERR	Bit level decoding error
GAP_TIME_ERR	Gap time error between words (less than 4 bit times)
VALID_WORD	Received ARINC word was valid in all respects

In Merge Mode, the Status Word contains the number of the receiving channel in bits 08 – 11 (where the LSB is bit 08).

The data returned by the function Read_Lkup_Data_RTx consists of a single 6-word block when not using IRIG B time, or 8-word blocks when using IRIG B time, where each block has the following configuration:

For Non-IRIG B Time

Word 0	DATA WORD HI
Word 1	DATA WORD LO
Word 2	TIME TAG HI
Word 3	TIME TAG LO
Word 4	ERROR COUNT
Word 5	STATUS

For IRIG B Time

Word 0	DATA WORD HI
Word 1	DATA WORD LO
Word 2	TIME TAG HI
Word 3	TIME TAG 2nd
Word 4	TIME TAG 3rd
Word 5	TIME TAG 4th
Word 6	ERROR COUNT
Word 7	STATUS

When not using IRIG B time, the Time Tag precision is 10 µsec. When using IRIG B time, the Time Tag precision is 1 µsec. The error count is per label.

The following figure shows the IRIG B Time Tag.

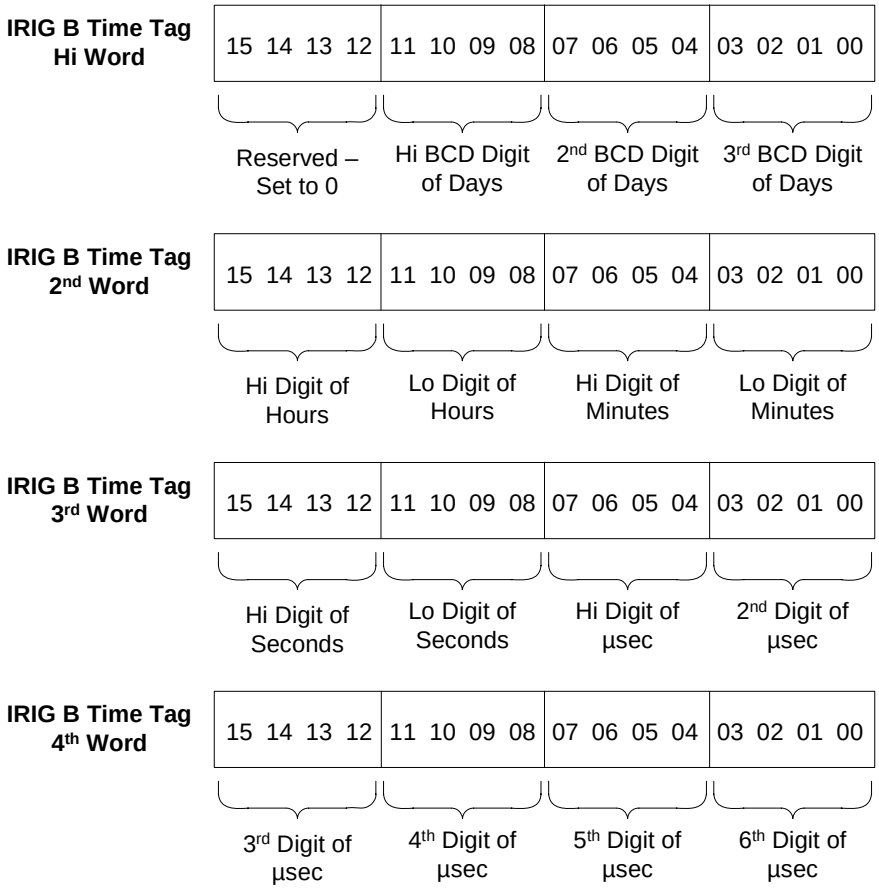


Figure A-1 IRIG B Time Tag

The Status Word is a combination of the flags:

WORD_RECEIVED	Indicates that a Data Word has been written
HI_BIT_CNT_ERR	Hi bit count error
LO_BIT_CNT_ERR	Lo bit count error
PARITY_ERR	Parity error
INVALID_CODING_ERR	Bit level decoding error
GAP_TIME_ERR	Gap time error between words (less than 4 bit times)
VALID_WORD	Received ARINC word was valid in all respects

Figure A-2 illustrates the memory configuration of the 32-bit data read or loaded using:

Output – Read	Read_Next_Data_RTx
	Read_Next_Merge_Data_RTx
	Read_Lkup_Data_RTx
Input – Load	Load_Message_RTx

The numbers shown represent the ARINC bit locations within the 32-bit word.

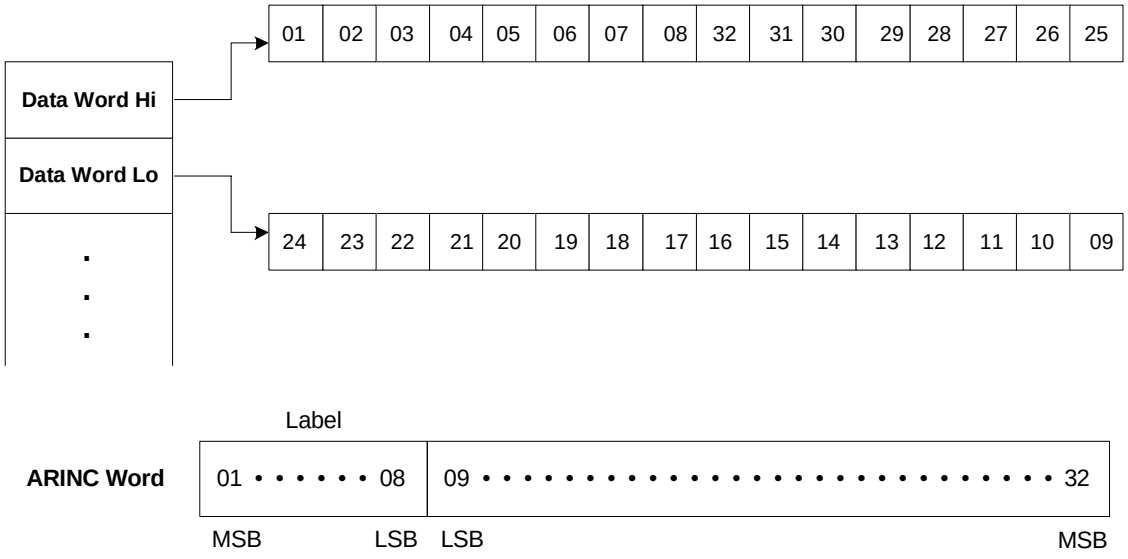


Figure A-2 Configuration in Memory of 32-bit Data Memory

Note: The bits 09 through 32 are ordered from LSB to MSB (the opposite from the Label field that is organized from MSB to LSB). For this reason, the Hi-Word is followed by the Lo-Word in the data block: the Label and the ARINC field 32 through 25 in the Hi-Word and bits 24 through 09 in the Lo-Word.

Appendix B *429RTx & Discrete Software Tools* Library

Appendix B includes a list of the files in the Excalibur *429RTx & Discrete Software Tools* needed to write user-defined applications. The files are divided into three categories:

Source code and Header files for the *429RTx & Discrete 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.dll`	Index file used to create applications using Microsoft DLL functions

Demo Programs are examples of programs using *429RTx & Discrete 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	**api_rtd.c**	Functions for writing test applications for Discrete channels
	deviceio.c	Functions relating to resources such as memory and interrupts – these functions act on kernel drivers
	deviceio_rtx.c	Functions for interaction with kernel driver for Windows operating systems
	EXC4000.c	For *M4K429RTx/M8K429RT5* modules on PCI carrier boards – functions for extracting information associated with the carrier board.
	EXCv4000.c	For *M4K429RTx/M8K429RT5* modules on VME/XVI carrier boards – functions for extracting information associated with the carrier board.
	initcard.c	Initialization and Release module/card functions
	Libdbg.c	Functions to display error messages
	Libgen.c	General functions to setup RT*x* modules/ cards
	Libmer.c	Functions to setup and operate ARINC receive channels in Merge Storage Mode
	Librcv.c	Functions to setup and operate ARINC receive channels
	Libtx.c	Functions to setup and operate ARINC transmit channels
Source Header files	**deviceio.h**	Header file for interaction with kernel driver
	error_devio.h	Header file containing error codes for the Excalibur kernel drivers
	error_rtx.h	Header file containing error message codes
	exc4000.h	For *M4K429RTx/M8K429RT5* on PCI carrier boards – Header file containing prototypes for functions associated with the carrier board

	File Name	Description
	EXCv4000.h	For *M4K429RTx/M8K429RT5* on VME/ VXI carrier boards – Header file containing prototypes for functions associated with the carrier board.
	excdef.h	Header file to differentiate between different Excalibur products
	Excsysio.h	Header file which defines control codes in kernel drivers and defines strings for module/card names for use in Init_Module_RTx/Init_Card_RTx calls to kernel drivers
	Flags_rtx.h	Header file containing flags for *RTx*, for applications
	rtxIncl.h	Header file, include file for the *RTx* DLL code, not to include for applications
	Instance_rtx.h	Header file for global module structure
	proto_rtx.h	Header file, prototypes of all functions. This will include all the header files needed for applications
Demo Programs	**demo_async.exe**	Async loopback demo
	demo_data_only_ mode.exe	Loopback demo using Data Only Storage Mode
	demo_dio.exe	Demonstrates Discrete input /output channels
	demo_dio_int.exe	Demonstrates Discrete input /output channels with interrupts
	demo_enhanced_ freq.exe	Demo using the Enhanced Frequency function Load_FrequencyMessage_RTx
	demo_FIFO.exe	FIFO Mode demo
	demo_filter.exe	Filter Table demo
	demo_information _429.exe	Demo that returns 429RTx board type information
	demo_int.exe	Interrupt demo using a loopback (test) cable
	demo_IRIG_ttag .exe	Loopback with IRIG Time Tag demo

	File Name	Description
	demo_IRIG_ttag_merge.exe	Loopback with IRIG Time Tag Merge Mode demo
	demo_lookup.exe	Look-up Table Storage Mode demo
	demo_loopback.exe	Loopback demo
	demo_merge.exe	Merge Mode demo
	demo_parse_429_data.exe	Demo parsing a 32-bit ARINC 429 Data Word
	demo_RX_only.exe	Receive (Monitor) demo
	demo_skip.exe	Demo with skipped messages
	demo_translate.exe	Demo that the translation function
	demo_translate_other_side.exe	Demo to run opposite **demo_translate**
	demo_ttagxmt.exe	Data Reconstructor demo
	demo_TX_only.exe	Transmit demo
DLL and LIB files	The DLL and LIB files have the same filename except for the file extension. The filename is: **Exc429RtxMs** A board level DLL accompanies the module for each of the boards and cards. Depending on the board, the names of the files are:	
	Carrier Board	**DLL Name**
	VME and VXI	**ExcV4000Ms.dll**
	All Other Boards/Cards	**Exc4000Ms.dll**

Appendix C Code Index

The *429RTx & Discrete Software Tools* is a set of C language functions designed to aid users of the *429RTx[D]* to write test programs. Below is an alphabetical listing of all the functions and the name of the *Software Tools* file which contains its programming code.

Driver Functions	Code File Name
Add_Translation_Entry_RTx¤	**librcv.c**
Add_TTAG_Data_RTx¤	**libxmtTtag.c**
Allocate_MAX_Transmit_FIFOs_RTx¤	**libtx.c**
Allocate_Message_RTx	**libtx.c**
Allocate_Transmit_FIFO_RTx	**libtx.c**
Alter_Translation_Entry_RTx¤	**librcv.c**
Clear_Merge_Word_Count_RTx	**libmer.c**
Clear_Rcv_Word_Count_RTx	**librcv.c**
Clear_Rcvd_Error_Cnt_RTx	**librcv.c**
Enable_Filter_Table_RTx	**librcv.c**
Enable_Lkup_Table_RTx	**librcv.c**
Enable_Merge_Filter_Table_RTx	**libmer.c**
Enable_Translation_Table_RTx	**librcv.c**
Enter_Lkup_Entry_RTx	**librcv.c**
Get_DmaAvailable_RTx	**initcard.c**
Get_Error_String_RTx	**libdbg.c**
Get_HWid_RTD	**api_rtd.c**
Get_Interrupt_Count_RTx	**deviceio_rtx.c**
Get_Time_Tag_RTx	**libgen.c**
Get_UseDmaIfAvailable_RTx	**initcard.c**
Init_Module_RTx	**initcard.c**
InitializeInterrupt_RTx	**deviceio_rtx.c**
Load_FrequencyMessage_RTx	**libtx.c**
Load_Message_RTx	**libtx.c**
Load_Transmit_FIFO_RTx¤	**libtx.c**

Driver Functions	Code File Name
Map_Label_To_Translation_Entry_RTx¤	**librcv.c**
Read_All_Input_Discretes_RTD	**api_rtd.c**
Read_Board_Intr_Status_RTx	**libgen.c**
Read_Board_Status_RTx	**libgen.c**
Read_Chan_Config_Register_RTx	**libgen.c**
Read_Chan_Config_Stat_RTx	**libgen.c**
Read_Channel_Status_RTx	**librcv.c**
Read_Global_Start_RTx	**libgen.c**
Read_Input_Discrete_RTD	**api_rtd.c**
Read_HwRevision_RTx	**libgen.c**
Read_Lkup_Current_Lbl_RTx	**librcv.c**
Read_Lkup_Data_RTx	**librcv.c**
Read_Merge_Error_Cnt_RTx	**libmer.c**
Read_Merge_Rcvd_Word_Cnt_RTx	**libmer.c**
Read_Merge_Status_RTx	**libmer.c**
Read_Next_Data_IRIG_RTx	**librcv.c**
Read_Next_Data_RTx	**librcv.c**
Read_Next_Merge_Data_IRIG_RTx	**libmer.c**
Read_Next_Merge_Data_RTx	**libmer.c**
Read_Number_Of_Channels_RTx	**libgen.c**
Read_Output_Discrete_RTD	**api_rtd.c**
Read_Pending_Register_RTD	**api_rtd.c**
Read_Pending_RTD	**api_rtd.c**
Read_Rcvd_Data_Word_Cnt_RTx	**librcv.c**
Read_Rcvd_Error_Cnt_RTx	**librcv.c**
Read_Revision_RTx	**libgen.c**
Read_SerialNumber_RTx	**libgen.c**
Read_Tx_Loop_Cnt_RTx	**libtx.c**
Read_Tx_Total_Words_Sent_RTx	**libtx.c**

Driver Functions	Code File Name
Readback_Filter_Entry_RTx	**librcv.c**
Release_Module_RTx	**initcard.c**
Reset_Channel_Configuration_RTx	**libgen.c**
Reset_Pending_Register_RTD	**api_rtd.c**
Reset_Pending_RTD	**api_rtd.c**
Reset_RTD	**api_rtd.c**
Reset_Ttag_RTx	**libgen.c**
Set_Filter_Entry_RTx	**librcv.c**
Set_Global_Storage_Mode_RTx	**libgen.c**
Set_Interrupt_Cond_RTx	**librcv.c**
Set_IRIG_Timetag_Mode_RTx	**libgen.c**
Set_Label_Trigger_RTx	**librcv.c**
Set_Merge_Interval_Trig_RTx	**libmer.c**
Set_Merge_Intr_Cond_RTx	**libmer.c**
Set_Merge_Label_Trigger_RTx	**libmer.c**
Set_Merge_Trig_Cond_RTx	**libmer.c**
Set_Merge_Word_Cnt_Trig_RTx	**libmer.c**
Set_Prog_Bit_Rate_RTx	**libgen.c**
Set_Rcv_Interval_Trig_RTx	**librcv.c**
Set_Rcv_Word_Cnt_Trig_RTx	**librcv.c**
Set_Rcvd_Error_Cnt_RTx	**librcv.c**
Set_Skip_Message_RTx	**libtx.c**
Set_Threshold_RTD	**api_rtd.c**
Set_Transmit_FIFO_Size_RTx¤	**libtx.c**
Set_Transmit_Loop_Counter_RTx¤	**libtx.c**
Set_Transmit_Message_Once_RTx	**libtx.c**
Set_Trigger_Cond_RTx	**librcv.c**
Set_Trigger_Cond_RTx¤	**librcv.c**
Set_Trigger_Destination_RTD	**api_rtd.c**

Driver Functions	Code File Name
Set_Trigger_Mask_RTD	**api_rtd.c**
Set_Trigger_RTD	**api_rtd.c**
Set_Trigger_Value_RTD	**api_rtd.c**
Set_UseDmaIfAvailable_RTx	**libtx.c**
Setup_Frame_RTx	**libtx.c**
Setup_Merge_Mode_RTx	**libmer.c**
Setup_Receive_Channel_RTx	**librcv.c**
Setup_Transmit_Channel_RTx	**libtx.c**
Setup_Ttag_Mode_RTx¤	**libxmtTtag.c**
Start_Channel_RTx	**libgen.c**
Start_Discrete_RTD	**api_rtd.c**
Start_Selected_Channels_RTx	**libgen.c**
Stop_Channel_RTx	**libgen.c**
Stop_Discrete_RTD	**api_rtd.c**
Stop_Selected_Channels_RTx	**libgen.c**
Wait_For_Interrupt_RTx	**deviceio_rtx.c**
Wait_For_Multiple_Interrupts_RTx	**deviceio_rtx.c**
Write_Output_Discrete_RTD	**api_rtd.c**

Appendix D Error Messages

All functions in *429RTx & Discrete Software Tools* 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_RTx function. Below is a list of all *Software Tools* error messages, the negative value of each, and an explanation of the error.

Error Code	Value	Explanation
einval	-2	Illegal value used as an input
sim_no_mem	-5	Not enough memory available for simulation
ewrngmodule	-27	Module specified on carrier board is not an RTx module
enomodule	-28	No module is present at the specified location
ebadhandle	-33	Invalid handle specified; should be value returned by Init_[Module/Card]_RTx
eboardtoomany	-36	Too many modules initialized
noirqset	-53	No interrupt allocated.
sixchancard	-400	Tried to enable a channel number greater than 5 on a six channel card. The card is not initialized.
sixtransmitchancard	-401	Only six 429 transmit channels available on card
init_failed	-402	Failed to initialize the module/card
param_Read_Chan_Config_Stat	-403	Bad parameter in Read_Chan_Config_Stat_RTx
param_Set_Global_Storage_Mode	-404	Bad parameter in Set_Global_Storage_Mode_RTx
param_Start_Channel	-405	Bad parameter in Start_Channel_RTx
param_Stop_Channel	-406	Bad parameter in Stop_Channel_RTx
param_Set_Prog_Bit_Rate	-407	Bad parameter in Set_Prog_Bit_Rate_RTx
param_Setup_Receive_Channel	-408	Bad parameter in Setup_Receive_Channel_RTx
bad_rcv_chan	-409	Not a receive channel
mem_Setup_Receive_Channel	-410	Not enough memory to set up a receive channel
param_Enable_Lkup_Table	-411	Bad parameter in Enable_Lkup_Table_RTx
mem_Enable_Lkup_Table	-412	Not enough memory
mode_Enable_Lkup_Table	-413	Bad receive mode
param_Enter_Lkup_Entry	-414	Bad parameter in Enter_Lkup_Entry_RTx

Error Code	Value	Explanation
mem_Enter_Lkup_Entry	-415	Not enough memory for a new Look-up Table entry
param_Read_Lkup_Data	-416	Bad parameter in Read_Lkup_Data_RTx
param_Enable_Filter_Table	-418	Bad parameter in Enable_Filter_Table_RTx
mem_Enable_Filter_Table	-419	Not enough memory to enable a Filter Table
param_Read_Rcvd_Data_Word_Cnt	-420	Bad parameter in Read_Rcvd_Data_Word_Cnt_RTx
param_Read_Rcvd_Error_Cnt	-421	Bad parameter in Read_Rcvd_Error_Cnt_RTx
param_Read_Lkup_Current_Lbl	-422	Bad parameter in Read_Lkup_Current_Lbl_RTx
param_Read_Channel_Status	-423	Bad parameter in Read_Channel_Status_RTx
param_Set_Label_Trigger	-424	Bad parameter in Set_Label_Trigger_RTx
param_Set_Rcv_Interval_Trig	-425	Bad parameter in Set_Rcv_Interval_Trig_RTx
param_Set_Rcv_Word_Cnt_Trig	-426	Bad parameter in Set_Rcv_Word_Cnt_Trig_RTx
bad_chan	-427	An invalid channel was specified
bad_param	-428	An invalid parameter was specified
fivechanmod	-429	Tried to enable a channel number greater than 4 on a five channel RT*x* module. The module is not initialized.
no_filter_table	-430	No Filter Table is defined
esetuptxchan	-431	Illegal call to Allocate_Message_RTx or Load_Message_RTx before Setup_Transmit_Channel_RTx
ewrngcard	-432	If no module is present with both ARINC 429 and Discrete channels
eboardnotstopped	-433	Cannot change communication parameters
param_Set_Interrupt_Cond	-434	Bad parameter in Set_Interrupt_Cond_RTx
param_Set_Trigger_Cond	-435	Bad parameter in Set_Trigger_Cond_RTx
param_Setup_Transmit_Channel	-436	Bad parameter in Setup_Transmit_Channel_RTx
mem_Setup_Transmit_Channel	-437	Not enough memory to set up a transmit channel
bad_tx_chan	-438	Not a transmit channel
param_Load_Message	-439	Bad parameter in Load_Message_RTx
msg_num_Load_Message	-440	Bad message number

Error Code	Value	Explanation
lkup_data_invalid	-441	Look-up Table data is invalid
mem_Setup_Merge_Mode	-442	Not enough memory
mem_Enable_Merge_Filter_Table	-443	Not enough memory
param_Read_Tx_Loop_Cnt	-444	Bad parameter in Read_Tx_Loop_Cnt_RTx
param_Read_Next_Data	-445	Bad parameter in Read_Next_Data_RTx
param_Allocate_Message	-446	Bad parameter in Allocate_Message_RTx
mem_Allocate_Message	-447	Not enough memory to allocate the message
msg_Allocate_Message	-448	Too many calls to Allocate_Message_RTx; the number of messages cannot exceed `max_messages` in Setup_Transmit_Channel_RTx
param_Clear_Rcv_Word_Count	-449	Bad parameter in Clear_Rcv_Word_Count_RTx
param_Read_Tx_Total_Words_Sent	-450	Bad parameter in Read_Tx_Total_Words_Sent_RTx
param_Enable_Translation_Table	-451	Enable_Translation_Table_RTx was not called for this channel
param_Skip_Message	-452	An invalid channel was specified in Set_Skip_Message_RTx
msg_num_Skip_Message	-453	If an invalid message number was specified in Set_Skip_Message_RTx
init_failed_1	-461	Failed to initialize the module/card
init_failed_2	-462	Failed to initialize the module/card
init_failed_3	-463	Failed to initialize the module/card
no_extended_time_support	-464	Extended Time is enabled on the channel but not supported by the firmware
tenchanmod	-465	Tried to enable a channel number greater than 9 on a ten channel RT*x* module. The module is not initialized.
param_Set_IRIG_Timetag_Mode	-467	Illegal `mode` selected in Set_IRIG_Timetag_Mode
param_Read_Next_Data_IRIG	-468	Cannot use Read_Next_Data_IRIG_RTx in DATA_ONLY_MODE
ewrongmode	-469	Incorrect mode for using this function
eNoIrigSupportModule	-470	Module is not a Nios-based processor, so it cannot use IRIG-based functions

Error Code	Value	Explanation
no_translation_table	-472	Attempt to map to a translation entry on a channel with no translation table
mem_Enable_Translation_Table	-473	Insufficient memory for a translation table
mem_Enable_Translation_Entry	-474	Insufficient memory for an new translation table entry or there are more than five translation operations defined in the translation entry
efeature_not_supported_RTX	-475	Feature is not supported on this module
param_Load_Message1	-481	Bad parameter in Load_Message_RTx or Load_FrequencyMessage_RTx; word_cnt is greater than 255
param_Load_Message2	-482	Bad parameter in Load_Message_RTx or Load_FrequencyMessage_RTx; `interword_dly` is greater than 255
param_Load_Message3	-483	Bad parameter in Load_Message_RTx or Load_FrequencyMessage_RTx; Setup_Transmit_Channel_RTx specifies EXTENDED_TIME_ENABLE and `intermsg_rate` or `frequency` is greater than 1310 or less than 1
param_Load_Message4	-484	Bad parameter in Load_Message_RTx or Load_FrequencyMessage_RTx; Setup_Transmit_Channel_RTx did not specify EXTENDED_TIME_ENABLE and `intermsg_rate` or `frequency` is greater than 65535 (FFFF H) or less than 0
param_Load_Message5	-485	Bad parameter in Load_Message_RTx or Load_FrequencyMessage_RTx; there is no space left on the module for `word_cnt` words
eenhancedfrequencynotsupported	-487	If the firmware on this module does not support the Enhanced Frequency function Load_FrequencyMessage_RTx
enotdataratemode	-488	If Setup_Transmit_Channel_RTx did not set this channel to be in DATA_RATE_MODE
For Discrete Channels		
ebaddischan	-500	Tried to set channel to illegal value
enodiscretes	-501	If no module is present with both ARINC 429 and Discrete channels

Error Code	Value	Explanation
For all Boards *EXCEPT* VME/VXI Boards		
eopenkernel	-1001	Error opening kernel device; check **`ExcConfig`** settings
ekernelcantmap	-1002	Error mapping memory
ereleventhandle	-1003	Error releasing the event handle
egetintcount	-1004	Error getting interrupt count
egetchintcount	-1005	Error getting channel interrupt count
egetintchannels	-1006	Error getting interrupt channels
ewriteiobyte	-1007	Error writing I/O memory
ereadiobyte	-1008	Error reading I/O memory
egeteventhand1	-1009	Error getting event handle (in first stage)
egeteventhand2	-1010	Error getting event handle (in second stage)
eopenscmant	-1011	Error opening Service Control Manager (in startkerneldriver)
eopenservicet	-1012	Error opening kernel service (in startkerneldriver)
estartservice	-1013	Error starting kernel service (in startkerneldriver)
eopenscmanp	-1014	Error opening Service Control Manager (in stopkerneldriver)
eopenservicep	-1015	Error opening kernel service (in stopkerneldriver)
econtrolservice	-1016	Error in control service (in stopkerneldriver)
eunmapmem	-1017	Error unmapping memory
egetirq	-1018	Error getting IRQ number
eallocresources	-1019	Error allocating resources; see **`Installation Instructions.pdf`** for details on resource allocation problems
egetramsize	-1020	Error getting RAM size
ekernelwriteattrib	-1021	Error writing attribute memory
ekernelreadattrib	-1022	Error reading attribute memory
ekernelfrontdesk	-1023	Error opening kernel device; check **`ExcConfig`** set up
ekernelOscheck	-1024	Error determining operating system
ekernelfrontdeskload	-1025	Error loading **`frontdesk.dll`**

Error Code	Value	Explanation
ekerneliswin2000compatible	-1026	Error determining Windows 2000 compatibility
ekernelbankramsize	-1027	Error determining memory size
ekernelgetcardtype	-1028	Error getting card type
emodnum	-1029	Invalid module number specified
regnotset	-1030	Board not configured; reboot after **`ExcConfig`** is run and board is in slot
ekernelbankphysaddr	-1031	Error getting physical memory address
ekernelclosedevice	-1032	Error closing kernel device
ekerneldevicenotopen	-1034	Error returned by kernel: device not open
ekernelinitmodule	-1035	Error initializing kernel
ekernelbadparam	-1036	Error returned by kernel: bad input parameter
ekernelbadpointer	-1037	Error returned by kernel: invalid pointer to output buffer
ekerneltimeout	-1038	Timeout expired before interrupt occurred
ekernelnotwin2000	-1039	Error returned by kernel: operating system is not Windows 2000 compatible
erequestnotification	-1040	Error requesting interrupt notification
ekernelnot4000card	-1041	Error returned by kernel: designated board is not EXC-4000/8000 family
enotimersirig	-1042	Timers and IrigB are not supported on this version of EXC-4000/8000 family board
eclocksource	-1059	Invalid clock source specified
eparmglobalreset	-1062	Illegal parameter used for globalreset_flag in the StartTimer_4000 function
etimernotrunning	-1063	Timer not running when function was called; did nothing
etimerrunning	-1064	Timer already running; did nothing
eparmreload	-1065	Illegal parameter used for reload_flag in StartTimer_4000
eparminterrupt	-1066	Illegal parameter used for interrupt_flag in StartTimer_4000
ebaddevhandle	-1067	Invalid handle specified; use value returned by Init_Timers_4000

Error Code	Value	Explanation
edevtoomany	-1068	Init_Timers_4000 called for too many boards
einvalidOS	-1069	Invalid operating system
enotforunet	-1089	Function invalid for UNET
For VME/VXI Boards Only		
eviclosedev	-1050	Error closing the device
evicloserm	-1051	Error closing the default RM
eopendefaultrm	-1052	Error opening the default RM
eviopen	-1053	Error opening VISA
evimapaddress	-1054	Error mapping address
evicommand	-1055	Error in VISA command
einstallhandler	-1056	Error installing VISA handler
eenableevent	-1057	Error enabling VISA event
eunistallhandler	-1058	Error uninstalling VISA handler
edevnum	-1060	Specified device number greater than 255
einstr	-1061	Error initializing module

Function Index

March 2025, Rev B-3