M4K1553MA
Software Tools
Programmer's Reference
with
DDC MicroAce Chip Specifications

EXCALIBUR SYSTEMS

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

5 Bus Controller [BC] Functions

Appendices

Functions Index

1 Introduction

The *M4K1553MA* module is an intelligent, MIL-STD-1553 interface module for the multimode, multiprotocol *EXC-4000P104plus* carrier board. The *M4K1553MA* provides a user-friendly MIL-STD-1553 interface for remote terminals, monitors and bus controllers. It is based on the DDC MicroAce chip. For an outline of how the chip has been implemented on the module, see **Appendix B: M4K1553MA Module Specific Implementation of the DDC MicroAce Chip**.

Chapter 1 provides an overview of the Excalibur *M4K1553MA* module for the *EXC-4000P104plus* carrier board.

The topics covered are:

Getting Started

Welcome to *M4K1553MA Software Tools*, your tool for creating custom application programs for the *M4K1553MA* module.

Before starting to write applications:

1. Install the *EXC-4000P104plus* carrier board.
   Note: The *M4K1553MA* module is integrated directly on the *EXC-4000P104plus* carrier board.
2. Copy *M4K1553MA Software Tools* from the *Excalibur Installation CD* on to your hard drive. You cannot run the program from the CD.
3. Follow the on-screen installation instructions. For detailed information, see the **`readme.pdf`** for the *EXC-4000P104plus* carrier board on the CD or **Appendix D: M4K1553MA Software Tools Installation Instructions,** on page D-1 in the *M4K1553MA Software Tools*: *Programmer's Reference.*
4. *[Optional]* Make a copy of the *M4K1553MA Software Tools*: *Programmer's Reference*.
5. Fill out the registration card and return it to Excalibur Systems.

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

The Excalibur Systems Website

More copies of the *M4K1553MA Software Tools*: *Programmer's Reference* can be downloaded from our website: www.mil-1553.com. In addition, information about our full range of products can be found at this site.

The software provided with Excalibur hardware is for Windows operating systems. The latest versions of the software can be downloaded from our website.

For further assistance, write to excalibur@mil-1553.com.

The Excalibur Configuration Utility Program

This is a configuration setup program for PCI hardware. Use the utility program to specify which hardware in a machine is being used. M4K1553MA Software Tools supports the use of up to four PCI boards. Only if more than one board or card is used, is it necessary to run the utility program.

The configuration program allows the user to set or change settings on the hardware itself. Excalibur hardware will work only after device values are properly entered and saved in the **`ExcConfig`** utility. Instructions for entering values in the utility program are in:

- **Appendix E: Multiple Board Support,** on page E-1 and
- In the **`readme.pdf`** file, on the *Excalibur Installation CD*, you received with your Excalibur hardware.

An Overview of the Data Communications Bus

M4K1553MA Software Tools enables the user to create custom programs for the *M4K1553MA* module. To use the *Software Tools*, some familiarity with the functioning of the 1553 data communications bus and an understanding of the hardware is required. This overview is an introduction to the 1553 bus. It will assist in understanding the rationale of the software functions for the hardware.

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

Each **Remote Terminal** can transfer data to or receive data from the Bus Controller or another Remote Terminal in response to a command from the Bus Controller. Remote Terminals for their part never initiate communications but merely react to the Bus Controller.

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

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

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

M4K1553MA Software Tools Functions

M4K1553MA Software Tools is a set of C language functions designed to aid users to write programs. These functions provide access to all of the module's functions in a structured and straightforward programming environment.

M4K1553MA Software Tools is available for Windows 9*x*, Windows NT and Windows 2000/XP. They were written and tested using Microsoft Visual C++.

Software Tools for the EXC-4000P104plus Carrier Board

The *EXC-4000P104plus* is a multiprotocol *PC/104-Plus* interface board. Each board holds up to four independent modules. Three are integrated directly on the board: two *M4K1553MA* modules based on the 1553 DDC MicroAce chip and one *M4K429RTx*. [The second *M4K1553MA* module is optional.] The fourth module may be any one of from Excalibur's range of M4K modules. To date the following modules are available: P*x*, 1553MCH, Discrete, Serial, 708, CAN, MMSI and *M4K429RTx*.

A general-purpose dll **`EXC4000.DLL`** (and its Microsoft-compiled version **`EXC4000MS.DLL`**) accompanies the *M4K[module] Software Tools* for each of the *EXC-4000P104plus* modules. 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_MA to ascertain that the type of module matches the module *Software Tools* package being used). These DLLs are installed automatically in the Windows/System folder when the *Software Tools* is installed for each of the modules.

Compiler Options

Programmers *must* use the Microsoft DLL **_cdecl** compiler option.

The driver functions in *M4K1553MA Software Tools* are supplied both in source form and linked as a DLL. When writing application-programs, keep in mind that the module is a physical resource, and therefore you cannot run multiple copies of the program simultaneously, accessing the same module.

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

Functions are written as 'C' functions, i.e., they return values of the enumerated type EXC_MA_DRIVER_STATUS. A negative value signifies an error. Full error messages may be printed using the Get_Error_String_MA function.
See **Appendix F-3: Error Messages,** on page F-4.)

In Windows all user-defined programs must include the file **`MA_proto.h`**. This file includes all the necessary header files and DLL functions to operate *M4K1553MA Software Tools* for the *M4K1553MA* module.

Font Conventions Used in *M4K1553MA Software Tools*

To help differentiate between different kinds of information, the following text styles are used in the *Programmer's Reference*:

Functions look like this.

`Variables` look like this: `unsigned short int = WORD`

Parameters look like this.

`File names` look like this.

FLAGS look like this

2 General Functions

Chapter 2 contains the software functions which are not mode specific and can be used in at least two modes.

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

The following functions are described in this chapter:

Disable_Selected_Transmitter_MA
Enable_Selected_Transmitter_MA
Get_Board_Status_MA
Get_Id_MA
Get_Error_String_MA
Get_Mode_MA
Get_Time_Tag_MA
Init_Module_MA
Print_Error_MA
Release_Module_MA
Reset_Time_Tag_MA
Set_Interrupt_MA
Set_Mode_MA
Set_Timetag_Res_MA
Stop_MA

To handle interrupts in Windows see **Using Interrupts in Windows** on page 2-13.
The functions described are:

Get_Interrupt_Count_MA
Wait_For_Interrupt_MA

Disable_Selected_Transmitter_MA

Description Calling Disable_Selected_Transmitter_MA disables transmission on the selected bus.

Syntax Disable_Selected_Transmitter_MA (`int` handle, `EXC_MA_TRANSMITTER` whichBus)

Input Parameters

handle	The handle designated by Init_Module_MA	
whichBus	`EXC_MA_BUS_A_TRANSMITTER` Disable transmission on Bus A [0000 H]	
	`EXC_MA_BUS_B_TRANSMITTER` Disable transmission on Bus B [0001 H]	

Output Parameters none

Return Values

EXC_MA_ERROR_ BAD_HANDLE	If an invalid handle passed in; should be value returned by Init_Module_MA
EXC_MA_ERROR _ILLEGAL_INPUT	If illegal value used as input
EXC_MA_STATUS_OK	No errors

Enable_Selected_Transmitter_MA

Description

Calling Enable_Selected_Transmitter_MA allows transmission on the selected bus. The default is to allow transmission.

Note: Call Enable_Selected_Transmitter_MA only if transmission was previously disabled.

Syntax

Enable_Selected_Transmitter_MA (`int` handle, `EXC_MA_TRANSMITTER` whichBus)

Input Parameters

handle	The handle designated by Init_Module_MA
whichBus	`EXC_MA_BUS_A_TRANSMITTER` Enable transmission on Bus A [0000 H]
	`EXC_MA_BUS_B_TRANSMITTER` Enable transmission on Bus B [0001 H]

Output Parameters

none

Return Values

EXC_MA_ERROR_ BAD_HANDLE	If an invalid handle passed in; should be value returned by Init_Module_MA
EXC_MA_ERROR _ILLEGAL_INPUT	If illegal value used as input
EXC_MA_STATUS_OK	No errors

Get_Board_Status_MA

Description Get_Board_Status_MA indicates the status of the specified module.

Syntax Get_Board_Status_MA (`int` handle, `WORD` *pStatus)

Input Parameters	handle	The handle designated by Init_Module_MA
Output Parameters	pStatus	0 or more of the following flags: EXC_MA_BOARD_READY Module has completed reset sequence [0001 H] EXC_MA_RAM_OK Module has passed internal memory self-test [0002 H] EXC_MA_MODE_OK Module is in a legal mode [0004 H] EXC_MA_SELF_TEST_OK Module has passed internal protocol self-test [0008 H] EXC_MA_BOARD_HALTED Module is stopped [0010 H]
Output Parameters		
Return Values	EXC_MA_ERROR_ BAD_HANDLE	If an invalid handle passed in; should be value returned by Init_Module_MA
	EXC_MA_ERROR_ ILLEGAL_INPUT	If an invalid value was used as an input

Note: Bits 07 is always set to 1

Get_Id_MA

Description Get_Id_MA returns the module ID of the specified module.

Syntax Get_Id_MA (`int` handle, `WORD` *pId)

Input Parameters	handle	The handle designated by Init_Module_MA
Output Parameters	pId	Module ID: E [0045 H]; any other value indicates hardware problem
Return Values	EXC_MA_ERROR_ BAD_HANDLE	If an invalid handle was specified; should be value returned by Init_Module_MA
	EXC_MA_STATUS_OK	No errors

Get_Error_String_MA

Description

Get_Error_String_MA accepts the error returns from other *M4K1553MA SoftwareTools* functions. This function returns the string containing a corresponding error message.

See **Appendix F-3 Error Messages** on page F-4.

Syntax

Get_Error_String_MA (`EXC_MA_DRIVER_STATUS` status, `char` *szStatusString, `int` maxStringLength

Example

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

Input Parameters

status	The error code returned from a *M4K1553MA SoftwareTools* function.
maxStringLength	Specifies the maximum length string that the buffer is large enough to contain. If it is smaller than the error string, the error string will be truncated to fit.

Output Parameters

szStatusString	A string of characters, with the corresponding error message. In case of bad input, a string denoting that.In case of good status (no error), a string indicating that there was no error.

Return Values

EXC_MA_STATUS_GETSTRING_TRUNCATED
If buffer was too small and string was truncated

EXC_MA_STATUS_OK No Errors

Get_Mode_MA

Description

Get_Mode_MA indicates the current mode of the specified module reflecting the last call to Set_Mode_MA.

Syntax

Get_Mode_MA (`int` handle, `int` *pmode)

Input Parameters

handle	The handle designated by Init_Module_MA

Output Parameters

pmode	One of the following flags: EXC_MA_RT_MODE Module set up as a Remote Terminal [0002 H] EXC_MA_BC_MODE Module set up as BC [0004 H] EXC_MA_MON_BLOCK Module set up as a Monitor in Sequential Block submode [0008 H]

Return Values

EXC_MA_ERROR_ BAD_HANDLE	If an invalid handle was specified; should be value returned by Init_Module_MA
EXC_MA_ERROR_ ILLEGAL_INPUT	If an invalid value was used as an input
EXC_MA_INVALID_ MODE	If the module is not properly initialized
EXC_MA_STATUS_OK	No errors

Get_Time_Tag_MA

Description Get_Time_Tag_MA returns the running Time tag of the module.

Syntax Get_Time_Tag_MA (`int` handle, `WORD` *pTimeTag)

Input Parameters	handle	The handle designated by Init_Module_MA
Output Parameters	pTimeTag	16-bit Time tag
Return Values	EXC_MA_ERROR_ BAD_HANDLE	If an invalid handle was specified; should be value returned by Init_Module_MA
	EXC_MA_STATUS_OK	No Errors

Init_Module_MA

Description Init_Module_MA is the first function the user must call for each MicroAce module, on each board that is accessed.

Init_Module_MA enables the user to access up to two modules on a single *EXC-4000P104plus*, or any combination of up to 8 modules on four separate boards.

The function may be called with the SIMULATE argument. If the SIMULATE argument is used, a portion of the memory equal to the size of the board's dual-port RAM is set aside.

Both real and simulated modules may be used in one application. Up to 10 real or simulated modules may be initialized. More than one module may be simulated simultaneously.

Note: Before exiting a program, call Release_Module_MA for each module initialized with Init_Module_MA.

Syntax Init_Module_MA (`WORD` deviceNum, `WORD` moduleNum, `EXC_MA_MODULE_HANDLE` *pHandle)

Input Parameters	deviceNum	The device number is the index of the entry value set in `ExcConfig`: 0 - 3 *or* SIMULATE [FFFF H]
	moduleNum	The module number of the MicroAce module on the board specified by device_num: 0 – 1

Note: If only one board is used, the define value EXC_4000PCI can be used instead of a device number. To set the device number, if more than one 4000 family board is used, run the **`ExcConfig`** utility.

Init_Module_MA (cont.)

Output parameters	pHandle	The handle to the specified module on the board. This handle is used as the first parameter in all module functions.
Return Values	EXC_MA_ERROR_NO_MEM_FOR_SIM_MODE Simulate mode failed due to inability to allocate sufficient memory on the heap	
	eopenkernel	If there was an error opening a device
	ekernelcantmap	If a pointer to memory cannot be obtained
	emodnum	Invalid module number specified
	enomodule	If no EXC-4000 module present at specified location
	ewrngmodule	If module specified on *EXC-4000P104plus* board is not a *M4K1553MA* module
	enoid	If init could not find a board with the given device number
	etimeoutreset	If timed out waiting for reset
	ealocresources	If there was an error allocating resources
	EXC_MA_STATUS_OK	No errors in initialization; the handle returned in the output parameter is valid.

Print_Error_MA

Description	Print_Error_MA accepts the error code returned from other *M4K1553MA SoftwareTools* functions. This function returns a const char* (See **Appendix F-3 Error Messages** on page F-4 for a complete list of error messages.)	
Syntax	const char * Print_Error (EXC_MA_DRIVER_STATUS errorcode)	
Input Parameters	errorcode	The error code returned from a *M4K1553MA SoftwareTools* call
Output Parameters	none	
Return Values	a pointer	Pointer to a message string that contains a corresponding error message. In case of bad input this function points to a string denoting "No such message".

Release_Module_MA

Description	Release_Module_MA must be called for each module initialized with Init_Module_MA, before exiting a program. Following a call to this function, the user must call Init_Module_MA in order to select another module.	
Syntax	Release_Module_MA (int handle)	
Input Parameters	handle	The handle designated by Init_Module_MA
Output Parameters	none	
Return Values	EXC_MA_ERROR_ BAD_HANDLE	If an invalid handle passed in; should be value returned by Init_Module_MA
	EXC_MA_STATUS_OK	No errors

Reset_Time_Tag_MA

Description	Reset_Time_Tag_MA resets the module's Time tag register.	
Syntax	Reset_Time_Tag_MA (int handle)	
Input Parameters	handle	The handle designated by Init_Module_MA
Output Parameters	none	
Return Values	EXC_MA_ERROR_ BAD_HANDLE	If an invalid handle passed in; should be value returned by Init_Module_MA
	EXC_MA_STATUS_OK	No errors

Set_Interrupt_MA

Description Set_Interrupt_MA describes the conditions under which the module is to generate an interrupt or disables interrupts.

Syntax Set_Interrupt_MA (`int` handle, `unsigned int` flag)

Input Parameters

	handle	The handle designated by Init_Module_MA
	flag	Conditions must be appropriate to the current mode. Multiple conditions (flags) maybe ORed together
	0	Do not set interrupts of any kind
BC mode	EXC_MA_MSG_CMPLT	Message completed [0002 H]
	EXC_MA_END OF FRAME	Complete frame of messages sent [0004 H]
	EXC_MA_MSG_ERR	Error occurred [0008 H]
	EXC_MA_END_MINOR_FRAME	Minor Frame completed [0010 H]
RT mode	EXC_MA_MSG_CMPLT	Message completed [0002 H]
	EXC_MA_MSG_ERR	Error occurred [0008 H]
Monitor mode	EXC_MA_MSG_CMPLT	Message completed [0002 H]
	EXC_MA_MSG_ERR	Error occurred [0008 H]

Output Parameters none

Return Values

EXC_MA_ERROR_BAD_HANDLE	If an invalid handle passed in; should be value returned by Init_Module_MA
EXC_MA_INVALID_INTERRUPT	If attempted to set an undefined interrupt
EXC_MA_STATUS_OK	No errors

Set_Mode_MA

Description

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

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

Note: Set_Mode_MA may be called just for reset purposes.

Syntax

Set_Mode_MA (`int` handle, `int` mode)

Input Parameters

handle	The handle designated by Init_Module_MA
mode	
EXC_MA_BC_MODE	Set up module as a BC [0004 H]
EXC_MA_MON_BLOCK	Set up module as Monitor [0008 H]
EXC_MA_RT_ MODE	Set up module as RT with Concurrent monitor [0010 H]

Output Parameters

none

Return Values

EXC_MA_ERROR_ BAD_HANDLE	If an invalid handle passed in; should be value returned by Init_Module_MA
EXC_MA_ERROR_ ILLEGAL_INPUT	If an invalid parameter was used as an input
EXC_MA_RESET_ TIMEOUT	If timed out waiting for reset or self-test
EXC_MA_STATUS_OK	No errors

Set_Timetag_Res_MA

Description

Set_Timetag_Res_MA sets the resolution of the Time tag used in the RT and Monitor Mode. There are six Time tag resolutions. [Default = 4µs.]

There is also an option to take the Time tag from an external clock source (which is connected to the clock on the carrier board, which in turn may be connected to a truly external Time tag.)

See section **B.1.3 External Time Tag Clock Register** on page B-2

Note: Set Set_Timetag_Res_MA before starting RT or Monitor mode.

Syntax

Set_Timetag_Res_MA (`int` handle,
`EXC_MA_TIMETAG_RES_OPTION` resOption)

Input Parameters

handle	The handle designated by Init_Module_MA
resOption	One of the following options EXC_MA_TTAGRES_64us 64µs EXC_MA_TTAGRES_32us 32µs EXC_MA_TTAGRES_16us 16µs EXC_MA_TTAGRES_8us 8µs EXC_MA_TTAGRES_4us 4µs EXC_MA_TTAGRES_2us 2µs EXC_MA_TTAGRES_EXTERNAL Take Time tag from an external source. See *EXC-4000P104plus User's Manual*

Output Parameters none

Return Values

EXC_MA_ERROR_ BAD_HANDLE	If an invalid handle was specified; should be value returned by Init_Module_MA
EXC_MA_ERROR_ ILLEGAL_INPUT	If an invalid parameter was used as an input
EXC_MA_STATUS_OK	No errors

Stop_MA

Description	Stop_MA stops the current operation of the module in all modes.	
Syntax	Stop_MA (`int` handle)	
Input Parameters	handle	The handle designated by Init_Module_MA
Output Parameters	none	
Return Values	EXC_MA_ERROR_BAD_HANDLE	If an invalid handle passed in; should be value returned by Init_Module_MA
	EXC_MA_STOP_TIMEOUT	If timed out waiting for module to respond to a stop request. Wait (checking EXC_MA_BOARD_HALTED in Get_Board_Status_MA), try again or reset.
	EXC_MA_STATUS_OK	No errors

Using Interrupts in Windows

When writing a Windows program that processes interrupts, a separate thread is generally created to handle the interrupt processing. This thread calls Wait_For_Interrupt_MA, (see page-2-14) 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 *M4K1553MA SoftwareTools.*

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_MA function may be used to determine if multiple interrupts have occurred. Conversely, it is possible that the Wait_For_Interrupt_MA 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_MA function and the call to Get_Interrupt_Count_MA.) Once again, the Get_Interrupt_Count_MA function may be used to determine if the interrupt has already been processed.

The following functions are described below:

Get_Interrupt_Count_MA
Wait_For_Interrupt_MA

Get_Interrupt_Count_MA

Description Get_Interrupt_Count_MA returns the total interrupt count for the specified module from the time the module was initialized with Init_Module_MA.

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

Input Parameters

handle	The handle designated by Init_Module_MA

Output Parameters

pdwInterruptCount	Pointer to an unsigned long which receives the interrupt count

Return Values

EXC_MA_ERROR_BAD_HANDLE
If an invalid handle was specified; should be value returned by Init_Module_MA

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
EXC_MA_STATUS_OK	No errors

Wait_For_Interrupt_MA

Description

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

Syntax

Wait_For_Interrupt_MA (`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)
  {
    EXC_MA_DRIVER_STATUS status;
    status = Wait_For_Interrupt_MA(module_handle, INFINITE);
    if (status !=EXC_MA_STATUS_OK)
    {
      // We don't check for ekerneltimeout since we passed
      // in a timeout value of INFINITE.
      // All other return values indicate error.
      // Process error…
       ExitThread(1);
    }
     // Process interrupt…
     // Get the status register to determine the
     //cause of this interrupt
     status = Get_BC_Status(handle,&bc_stat;)
     // add code here to check status and handle errors
     Reset_BC_Status();

     if (bc_stat & EXC_MA_END_OF_FRAME)
     {
     // interrupt due to end of frame
     // process…
     }
     if (bc_stat & EXC_MA_MSG_CMPLT)
     {
     // interrupt due to message completed
     // process…
     }
     if (bc_stat & EXC_MA_MSG_ERR)
     {
     // interrupt due to message error
     // process…
     }
     // Check total number of interrupts
    Get_Interrupt_Count_MA(module_handle, &numints);
  }
}
```

Wait_For_Interrupt_MA (cont.)

Input Parameters	handle	The handle designated by Init_Module_MA
	timeout	Timeout is specified in milliseconds *or* INFINITE [FFFF H]
Output Parameters	none	
Return Values	EXC_MA_ERROR_BAD_HANDLE	If an invalid handle passed in; should be value returned by Init_Module_MA
	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
	No errors if *either*:	
	ekerneltimeout	The wait timed out without receiving an interrupt
	or	
	EXC_MA_STATUS_OK	

3 Remote Terminal Functions

Chapter 3 describes the Excalibur *M4K1553MA* module in Remote Terminal (RT) mode. The RT module is used to act as one Remote Terminal. The user can:

- Control which RT will be activated.
- Determine which data should be sent for each subaddress combination.

The functions described in this chapter are:

Get_Next_RT_Message_MA	Set_Bit_Source_MA
Load_RT_Data_MA	Set_RT_Active_MA
Read_RT_Data_MA	Set_RT_Broadcast_MA
Read_RT_Status_MA	Set_RT_Buffering_Mode_MA
Run_RT_MA	Set_SA_Interrupt_MA
Set_Bit_MA	Set_Vector_MA

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

Functions by Category

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

Init_Module_MA	To initialize the module
Set_Mode_MA	To place the module into RT mode
Set_RT_Active_MA	To select which RT(s) to activate
Run_RT_MA	To start active RT
Stop_MA	To stop the RT
Release_Module_MA	To release resources assigned to the module

To control the data being transmitted by the active RT or read the data received by the RT:

Load_RT_Data_MA	To enter data to be transmitted
Read_RT_Status_MA	To read data received

To use messages in the Message Stack:

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

Get_Next_RT_Message_MA	To read an entry from the message stack
Set_Interrupt_MA	To describe the conditions under which the module is to generate an interrupt or disable an interrupts.

RT Functions

Get_Next_RT_Message_MA

Description

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

Note: If the user falls 512 entries behind, it will be impossible to know if the message returned is newer or older than the previous message returned.

Syntax

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

Input Parameters

handle — The handle designated by Init_Module_MA

Output Parameters

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

struct EXC_MA_CMDENTRYRT *cmdstruct

struct EXC_MA_CMDENTRYRT{

WORD command; 1553 Command word

WORD timetag; Time stamp of the message

WORD status; Excalibur Status word for this command containing one or more of these flags:

EXC_MA_END_OF_MSG	Indicates end of message [8000 H]
EXC_MA_BUS_A	Indicates bus A [4000 H]
EXC_MA_TX_TIME_OUT	In RT-to-RT no response [0400 H]
EXC_MA_SELFTEST_ERR	MicroAce chip failed internal test [0100 H]
EXC_MA_INVALID_WORD	At least one invalid 1553 Word received [0080 H]
EXC_MA_BAD_WD_CNT	Wrong number of words received[0020 H]
EXC_MA_BAD_SYNC	Sync of either the Command or the Data word(s) is incorrect [0008 H]
EXC_MA_RT2RTMSG	RT-to-RT message was received[[0002 H]
EXC_MA_MSG_ERROR	Error occurred - defined in other flags [0001 H]

Get_Next_RT_Message_MA (cont.)

EXC_MA_SUPERSEDED_MSG	Message was superseded [0040 H]
EXC_MA_RT2RT_ERROR	Receiving RT and one of the following [1000 H]: • Transmitting RT responds in under 4 µs • Incorrect sync type, encoding, bit count and/or parity error in the transmitting RT Status word • The RT address field of the transmitting RT Status word does not match the RT address in the transmit command word

}

Return Values

EXC_MA_ERROR_ BAD_HANDLE	If an invalid handle passed in; should be value returned by Init_Module_MA
EXC_MA_ERROR _WRONG_MODE	If module is not in RT mode
EXC_MA_STATUS_NO _MORE_MESSAGES	If no unread messages are available to return
EXC_MA_STATUS_OK	No errors

Load_RT_Data_MA

Description

Load_RT_Data_MA loads data to the buffer for the requested subaddress. Handles double-buffering: loads data to the inactive buffer, then, when finished, makes that buffer the active buffer (if double-buffering is enabled for this subaddress.)

Calling Load_RT_Data_MA can only load data to a transmit message, thus there is no buffer type parameter.

This function does not work for Mode codes; use Mode code specific functions: Set_Bit_MA and Set_Vector_MA.

Syntax

Load_RT_Data_MA (`int` handle, `int` subAddress, `WORD` *pWords)

Input Parameters

handle	The handle designated by Init_Module_MA
subAddress	The new Subaddress number Valid values: 0 – 31

Output Parameters

pWords	Pointer to an array of up to 32 data words to be placed in the block

Return Values

EXC_MA_ERROR_BAD_HANDLE	If an invalid handle passed in; should be value returned by Init_Module_MA
EXC_MA_ERROR_ILLEGAL_INPUT	If illegal value used as input
EXC_MA_ERROR_WRONG_MODE	If module is not in RT mode
EXC_MA_STATUS_OK	No errors

Read_RT_Data_MA

Description

Read_RT_Data_MA accepts all buffer types. The function reads the most recent data in the user's buffer. Handles double-buffering automatically – if the requested subaddress/buffer type combination is set to double-buffering, then data is read from the last completed set of data; incoming data (if it is a receive or broadcast message) or data set with Load_RT_Data_MA (if it is a transmit message) will go to the alternate buffer.

Syntax

Read_RT_Data_MA (`int` handle, `int` subAddress, `int` bufferType, `WORD` *pWords)

Input Parameters

handle	The handle designated by Init_Module_MA
subAddress	The new Subaddress number Valid values: 0 – 31
bufferType	EXC_MA_TRANSMIT Transmit command [0001 H] EXC_MA_RECEIVE Receive command [0000 H] EXC_MA_BROADCAST Broadcast command [0002 H]

Output Parameters

pWords	Pointer to an array of up to 32 data words to be placed in the block

Return Values

EXC_MA_ERROR_ BAD_HANDLE	If an invalid handle passed in; should be value returned by Init_Module_MA
EXC_MA_ERROR _ILLEGAL_INPUT	If illegal value used as input
EXC_MA_ERROR _WRONG_MODE	If module was not in RT mode
EXC_MA_STATUS_OK	No errors

Read_RT_Status_MA

Description	Read_RT_Status_MA returns 1 if a 1553 Message has been received since the last time Read_RT_Status_MA was called.	
Syntax	Read_RT_Status_MA (`int` handle, `unsigned char` *pStatus)	
Input Parameters	handle	The handle designated by Init_Module_MA
Output Parameters	pStatus	1 Message was received 0 No message was received
Return Values	EXC_MA_ERROR _BAD_HANDLE	If an invalid handle passed in; should be value returned by Init_Module_MA
	EXC_MA_ERROR __WRONG_MODE	If module is not in RT mode
	EXC_MA_STATUS_OK	No errors

Run_RT_MA

Description	Run_RT_MA starts the RT activated by Set_RT_Active_MA.	
Syntax	Run_RT_MA (`int` handle)	
Input Parameters	handle	The handle designated by Init_Module_MA
Output Parameters	none	
Return Values	EXC_MA_ERROR_ BAD_HANDLE	If an invalid handle passed in; should be value returned by Init_Module_MA
	EXC_MA_ERROR _WRONG_MODE	If module is not in RT mode
	EXC_MA_STATUS_OK	No errors

Set_Bit_MA

Description

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

Note:

1. The normal RT mode data access function cannot be used for loading or retrieving the data for Mode Code messages.
2. Set_Bit_MA automatically changes the BIT source to be external if the user has not done so already.

Syntax

Set_Bit_MA (`int` handle, `int` bitvalue)

Input Parameters

handle	The handle designated by Init_Module_MA
bitvalue	BIT response value – the value to be set

Output Parameters

none

Return Values

EXC_MA_ERROR_ BAD_HANDLE	If an invalid handle was specified; should be value returned by Init_Module_MA
EXC_MA_ERROR_ WRONG MODE	If module is not in RT mode
EXC_MA_ERROR_ ILLEGAL_INPUT	If parameter is out of range
EXC_MA_STATUS_OK	No errors

Set_Bit_Source_MA

Description

Set_Bit_Source_MA selects the source for the value to be returned in response to Transmit BIT (Built-in-Test) Word mode command – it can be:

- *Either* internal (created by the chip based on the internal state and results of the internal self-test. See Table 131 in the *Enhanced Miniature Advance Communications Engine (Enhanced Mini-Ace Series) User's Guide Volume 1: Architectural Reference*, section "RT Built-in-Test (BIT) Word"
- *Or* external – set by the user; see Set_Bit_MA on page 3-7.

Syntax

Set_Bit_Source_MA (`int` handle, `EXC_MA_BIT_SOURCE_OPTION` sourceOption)

Input Parameters

handle	The handle designated by Init_Module_MA
sourceOption	`EXC_MA_BITSOURCE_INTERNAL` or `EXC_MA_BITSOURCE_EXTERNAL`

Output Parameters

none

Set_Bit_Source_MA (cont.)

Return Values	EXC_MA_ERROR_ BAD_HANDLE	If an invalid handle passed in; should be value returned by Init_Module_MA
	EXC_MA_ERROR_ WRONG_MODE	If module is not in RT mode
	EXC_MA_ERROR_ ILLEGAL_INPUT	If parameter is out of range
	EXC_MA_STATUS_OK	No errors

Set_RT_Active_MA

Description	Set_RT_Active_MA selects the RT address. The *M4K1553MA* can only act as one RT. Therefore calling Set_RT_Active_MA activates an RT and deactivates any previously activated RT.	
Syntax	Set_RT_Active_MA (`int` handle, `int` rtnum)	
Input Parameters	handle	The handle designated by Init_Module_MA
	rtnum	Address of the RT Valid values: 0 – 31
Output Parameters	none	
Return Values	EXC_MA_ERROR_ BAD_HANDLE	If an invalid handle passed in; should be value returned by Init_Module_MA
	EXC_MA_ERROR_ WRONG_MODE	If module is not in RT mode
	EXC_MA_ERROR_ ILLEGAL_INPUT	If parameter is out of range
	EXC_MA_STATUS_OK	No errors

Set_RT_Broadcast_MA

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

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

Input Parameters

handle	The handle designated by Init_Module_MA
toggle	EXC_MA_ENABLE Enable RT31 as the broadcast address [0001 H] EXC_MA_DISABLE Disable RT31 from being the broadcast address [0000 H]

Output Parameters none

Return Values

EXC_MA_ERROR_BAD_HANDLE	If an invalid handle passed in; should be value returned by Init_Module_MA
EXC_MA_ERROR_WRONG_MODE	If module is not in RT mode
EXC_MA_STATUS_OK	No errors

Set_RT_Buffering_Mode_MA

Description Set_RT_Buffering_Mode_MA sets single or double-buffering for each subaddress/direction combination. Setting double-buffering for receive messages will affect both receive and broadcast messages.

Syntax Set_RT_Buffering_Mode_MA (`int` handle, `int` subAddress, `int` direction, `enum` EXC_MA_RT_BUFFERING_MODE direction)

Input Parameters

handle	The handle designated by Init_Module_MA
subAddress	The new Subaddress number Valid values: 0 – 31
direction	EXC_MA_TRANSMIT Transmit command [0001 H] EXC_MA_RECEIVE Receive command [0000 H]
EXC_MA_RT_BUFFERING_MODE	EXC_MA_RT_SINGLE_BUFFER Single buffering EXC_MA_RT_DOUBLE_BUFFER Double buffering

Output Parameters none

Set_RT_Buffering_Mode_MA (cont.)

Return Values	EXC_MA_ERROR _BAD_HANDLE	If an invalid handle passed in; should be value returned by Init_Module_MA
	EXC_MA_ERROR _ILLEGAL_INPUT	If illegal value used as input
	EXC_MA_ERROR _WRONG_MODE	If module is not in RT mode
	EXC_MA_STATUS_OK	No errors

Set_SA_Interrupt_MA

Description	Set_SA_Interrupt_MA enables or disables interrupt on End of Message for messages to specific subaddress [0 – 31].	
Syntax	Set_SA_Interrupt_MA (`int` handle, `int` subAddress, `int` bufferType, `int` enableFlag)	
Input Parameters	handle	The handle designated by Init_Module_MA
	subAddress	The new Subaddress number Valid values: 0 – 31
	bufferType	EXC_MA_TRANSMIT Transmit command [0001 H] EXC_MA_RECEIVE Receive command [0000 H] EXC_MA_BROADCAST Broadcast command [0002 H]
	enableFlag	EXC_MA_ENABLE Enable specific SA [0001 H] EXC_MA_DISABLE Disable specific SA [0000 H]
Output Parameters	none	
Return Values	EXC_MA_ERROR _BAD_HANDLE	If an invalid handle passed in; should be value returned by Init_Module_MA
	EXC_MA_ERROR _ILLEGAL_INPUT	If illegal value used as input
	EXC_MA_ERROR _WRONG_MODE	If module is not in RT mode
	EXC_MA_STATUS_OK	No errors

Set_Vector_MA

Description

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

Note: The normal RT mode data access function cannot be used for loading or retrieving the data for Mode Code messages.

Syntax

Set_Vector_MA (`int` handle, `int` vecValue)

Input Parameters

handle	The handle designated by Init_Module_MA
vecValue	Service Request vector response value to be set

Output Parameters

none

Return Values

EXC_MA_ERROR_ BAD_HANDLE	If an invalid handle was specified; should be value returned by Init_Module_MA
EXC_MA_ERROR_ WRONG MODE	If module is not in RT mode
EXC_MA_ERROR_ ILLEGAL_INPUT	If parameter is out of range
EXC_MA_STATUS_OK	No errors

4 Bus Monitor Functions

Chapter 4 describes the *M4K1553MA* module in Bus Monitor mode.

The functions described in this chapter are:

Get_MON_Status_MA

Run_MON_MA

Set_Broad_Ctl_MA

Get_Next_Message_MA

Get_MON_Status_MA

Description	Get_MON_Status_MA returns the status of the Monitor. The function is useful to determine what condition triggered an interrupt. **Note:** The status is automatically reset each time it is read.	
Syntax	Get_MON_Status_MA (`int` handle, `unsigned char` *pStatus)	
Input Parameters	handle	The handle designated by Init_Module_MA
Output Parameters	pStatus	EXC_MA_MSG_IN_PROGRESS A message is in the process of being received [0002 H]
Return Values	EXC_MA_ERROR_ BAD_HANDLE	If an invalid handle passed in; should be value returned by Init_Module_MA
	EXC_MA_ERROR_ WRONG_MODE	If module is not in Monitor mode
	EXC_MA_STATUS_OK	No errors

Run_MON_MA

Description	Run_MON_MA causes the Monitor module to start.	
Syntax	Run_MON_MA (`int` handle)	
Input Parameters	handle	The handle designated by Init_Module_MA
Output Parameters	none	
Return Values	EXC_MA_ERROR_ BAD_HANDLE	If an invalid handle passed in; should be value returned by Init_Module_MA
	EXC_MA_ERROR_ WRONG_MODE	If module is not in Monitor mode
	EXC_MA_STATUS_OK	No errors

Set_Broad_Ctl_MA

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

Syntax Set_Broad_Ctl_MA (`int` handle, `int` flag)

Input Parameters

handle	The handle designated by Init_Module_MA
flag	EXC_MA_ENABLE Sets RT31 as the broadcast address [0001 H] EXC_MA_DISABLE Sets RT31 as a regular RT [0000 H]

Output Parameters none

Return Values

EXC_MA_ERROR_ BAD_HANDLE	If an invalid handle passed in; should be value returned by Init_Module_MA
EXC_MA_ERROR_ WRONG_MODE	If module is not in Monitor mode
EXC_MA_ERROR_ ILLEGAL_INPUT	If an invalid parameter was used as an input
EXC_MA_STATUS_OK	No errors

Get_Next_Message_MA

Description

Get_Next_Message_MA reads the command stack entry following the entry to Get_Next_Message_MA and the data corresponding to that message. The first call to Get_Next_Message_MA will return the first entry in the command stack.

Note: The monitor has room for 4096 messages and 32768 data words. Since the message control information and the data are kept separately, the command stack (message stack) and the data can wrap around at different points.

Syntax

Get_Next_Message_MA (`int` handle, `struct EXC_MA_MONMSG` *msgptr)

Input Parameters

handle	The handle designated by Init_Module_MA

Output Parameters

msgptr	Pointer to the structure defined below in which to return the message

struct EXC_MA_MONMSG {

unsigned short msgstatus;	Status word containing the following flags:
EXC_MA_END_OF_MSG	Indicates end of message [8000 H]
EXC_MA_RT2RT_ERROR	Receiving RT and one of the following [1000 H]: • Transmitting RT responds in under 4 μs • Incorrect sync type, encoding, bit count and/or parity error in the transmitting RT Status word • The RT address field of the transmitting RT Status word does not match the RT address in the transmit command word
EXC_MA_INVALID_WORD	Bad bit count, Manchester or parity [0080 H]
EXC_MA_BAD_WD_CNT	Wrong number of words received [0020 H]
EXC_MA_BAD_SYNC	Sync of either the Command or the Data word(s) is incorrect [0008 H]
EXC_MA_MSG_ERROR	Error occurred, defined in other flags [0001 H]
EXC_MA_INVALID_MSG	Word count or Sync error occurred [0400 H]
EXC_MA_RT2RT_MSG	Message was RT-to-RT transfer [2000 H]
EXC_MA_BUS_A_XFER	Message was transferred on bus A [0100 H]
EXC_MA_BAD_RT_ADDR	1553 Status word contained wrong RT address [0010 H]
EXC_MA_MON_LATE_RESP	Response time error occurred in the message [0002 H]
unsigned short reserved;	This is a placeholder to resolve structure packing issues between compilers.

Get_Next_Message_MA (cont.)

	unsigned long elapsedtime;	The 16-bit Time tag associated with the message.
	unsigned short words [36];	A pointer to an array of 1553 words in the sequence they were received over the bus.
	}	
Return Values	EXC_MA_ERROR_ BAD_HANDLE	If an invalid handle passed in; should be value returned by Init_Module_MA
	EXC_MA_ERROR_ WRONG_MODE	If module is not in Monitor mode
	EXC_MA_STATUS_NO _MORE_MESSAGES	If there is no message to return
	EXC_MA_STATUS_OK	No errors

5 Bus Controller [BC] Functions

Chapter 5 describes the *M4K1553MA Software Tools* functions for operating the *M4K1553MA* module in Bus Controller [BC] mode. The user can define 1553 messages and group them into frames and has control over which commands and data are transmitted over the bus as well as the timing between messages and frames. To set the module in BC mode, see Set_Mode_MA on page 2-10.

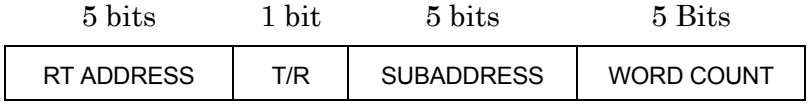
The following functions are described in this chapter:

Alter_Cmd_MA
Alter_Message_MA
BC_Check_Alter_Msg_MA
Command_Word_MA
Create_Frame_MA
Create_1553_Message_MA
Enable_Retries_For_Message_MA
Get_BC_Msgentry_MA
Get_BC_Status_MA
Get_Minor_Frame_Time_MA
Get_Msgentry_Status_MA
Read_Message_MA
Reset_BC_Status_MA
Run_BC_MA
Set_Bus_MA
Set_Frame_Time_MA
Set_Interrupt_On_Msg_MA
Set_Minor_Frame_Time_MA
Set_Retry_Options_MA
Start_Frame_MA

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

Command Word Calculation

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



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

The Command_Word_MA function, on page 5-6, is provided to carry out this calculation.

Functions by Category

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

Sending out a fixed number of messages in a loop

Init_Module_MA	To initialize the module/card (page 2-6)
Set_Mode_MA	To place the module in BC mode (page 2-10)
Create_1553_Message_MA	To create the messages to send
Create_Frame_MA	To select the order in which to transmit the messages
Set_Frame_Time_MA	To determine the time between transmission of frames
Start_Frame_MA	To choose the frame to transmit
Run_BC_MA	To start the module running
Stop_MA	To stop the BC
Release_Module_MA	To release resources assigned to the module (page 2-8)

Additional frame functions:

Set_Minor_Frame_Time_MA	To set the minor frame time for a Minor frame type of message

Obtaining data and status information during transmission

Read_Message_MA	To get data and 1553 status information
Get_BC_Msgentry_MA	To get both data and 1553 status information *and* status information for a selected message
Get_BC_Status_MA	To get module level status information
Reset_BC_Status_MA	To clear the BC status
Get_Msgentry_Status_MA	To return a Status word recorded by the Bus Controller associated with a selected message in a selected frame.

Changing message parameters in realtime

Alter_Cmd_MA	Changes the Command word of a previously defined message.
Alter_Message_MA	Changes the data of a previously defined message.
BC_Check_Alter_Msg_MA	Verifies that a message that is to be altered is not in the process of being transmitted or received.

BC Functions

Alter_Cmd_MA

Description

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

Syntax

Alter_Cmd_MA (`int` handle, `int` msgId, `WORD` rt, `WORD` sa)

Input Parameters

handle	The handle designated by Init_Module_MA
msgId	Message identifier returned from a prior call to Create_1553_Message_MA
rt	The new RT number Valid values: 0 – 31 *or* EXC_MA_SAME_RT Do not change the RT of the Command [0020 H]
sa	The new Subaddress number Valid values: 0 – 31 *or* EXC_MA_SAME_SA Do not change the subaddress of the Command [0020 H]

Output Parameters

none

Return Values

EXC_MA_ERROR_ BAD_HANDLE	If an invalid handle was specified; should be value returned by Init_Module_MA
EXC_MA_ERROR_ WRONG_MODE	If not in BC mode
EXC_MA_ERROR_CANT _ALTER_MODE_CODE	Cannot use Alter_Cmd_MA on a mode code message.
EXC_MA_ERROR_ ILLEGAL_INPUT	If illegal value used as input
EXC_MA_ERROR_ BAD_MSG_ID	If attempted to alter an undefined message
EXC_MA_STATUS_OK	No errors

Alter_Message_MA

Description

Alter_Message_MA changes the data of a previously defined message. The function receives as input a pointer to an array of Data words and copies these words into the original message. The Command word cannot be changed and therefore the number of Data words cannot be changed. Only the values of the Data words can be changed.

To ensure data integrity, call BC_Check_Alter_Msg_MA before a call to this function.

Note: See **Appendix C: MIL-STD-1553 Formats.**

Syntax

Alter_Message_MA (`int` handle, `int` msgId, `WORD` *data)

Input Parameters

handle	The handle designated by Init_Module_MA
msgId	Message identifier returned from a prior call to Create_1553_Message_MA
data	Pointer to an array of up to 32 words of new Data words for message msgId.

Output Parameters

none

Return Values

EXC_MA_ERROR_ BAD_HANDLE	If an invalid handle was specified; should be value returned by Init_Module_MA
EXC_MA_ERROR_ BAD_MSG_ID	If requested an undefined message
EXC_MA_ERROR_ WRONG_MODE	If not in BC mode
EXC_MA_ERROR_ MINORFRAMEMSG	If message is Minor Frame type
EXC_MA_STATUS_OK	No errors

BC_Check_Alter_Msg_MA

Description

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

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

Syntax

BC_Check_Alter_Msgentry_MA (`int` handle, `int` msgId)

Input Parameters

handle	The handle designated by Init_Module_MA
msgId	Message identifier returned from a prior call to Create_1553_Message_MA

Output Parameters

none

Return Values

EXC_MA_ERROR_ BAD_HANDLE	If an invalid handle was specified; should be value returned by Init_Module_MA
EXC_MA_ERROR_ WRONG_MODE	If module is not in BC mode.
EXC_MA_STATUS_ NOALTER	If this message is being transmitted
EXC_MA_ERROR_ BAD_MSG_ID	If attempted to access an undefined message
EXC_MA_STATUS_OK	No errors

Command_Word_MA

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

Syntax Command_Word_MA (`int` rtnum, `int` type, `int` subaddr, `int` wordcount, `WORD` *commandword)

Input Parameters

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

Output Parameters

commandword	The calculated Command word

Return Values

EXC_MA_ERROR_ ILLEGAL_INPUT	If an invalid parameter was used as an input.
EXC_MA_STATUS_OK	No errors

Create_Frame_MA

Description

Create_Frame_MA creates a frame of previously defined messages to be sent by the BC. Each element in the frame structure consists of a message id (the value returned from a call to Create_1553_Message_MA), and a gap time indicating the time between the transmission of the message and the following one. The frame structure must contain a final entry with id '0', indicating the end of the frame.

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

Syntax

Create_Frame_MA (`int` handle, `struct EXC_MA_FRAME` * pFrame, `int` *pFrameId)

Input Parameters

handle	The handle designated by Init_Module_MA

EXC_MA_FRAME [ ];
pFRAME; an array of frame entry structures, described below

struct EXC_MA_FRAME
{

long id;	Message identifier returned by Create_1553_Message_MA or 0 for last message
long gaptime;	Gap Time between this message and following message in microseconds

};

Output Parameters

pFrameId	A pointer to the frame identifier of a frame just created. The frameid is used in many of the functions in BC mode. Valid values 0 – 19

Return Values

EXC_MA_ERROR_ BAD_HANDLE	If an invalid handle was specified; should be value returned by Init_Module_MA
EXC_MA_ERROR_ BAD_MESSAGE_ID	If attempted to place an undefined message into a message frame
EXC_MA_ERROR_ NO_SPACE_FOR_ FRAME	If there is not enough space in frame stack for this frame
EXC_MA_TOO_ MANY_FRAMES	If exceeded maximum number of frames permitted (20)
EXC_MA_ERROR_ WRONG_MODE	If not in BC mode
EXC_MA_STATUS_OK	No errors

Create_1553_Message_MA

Description

Create_1553_Message_MA creates the basic building block of the 1553 protocol – the message. A message contains a minimum of one, and a maximum of two (for RT-to-RT messages), Command words and 32 Data words, depending on the type of message. Each message created with this function returns a unique id which is used to set or read characteristics of the message or to build a frame (i.e., a collection of messages) for subsequent transmission.

Note: See **Appendix C: MIL-STD-1553 Formats.**

Syntax

Create_1553_Message_MA (`int` handle, `WORD` cmdType, `WORD` data[ ], `short int` *msgId)

Input Parameters

handle	The handle designated by Init_Module_MA
cmdType	One of the following flags:
EXC_MA_RT2BC	Send a transmit message [0000 H]
EXC_MA_BC2RT	Send a Receive message [0001 H]
EXC_MA_RT2RT	Send an RT to RT transfer message [0002 H]
EXC_MA_MODE	Mode Command[0003 H]
EXC_MA_BRD_RCV	Broadcast Receive message [0004 H]
EXC_MA_BRD_RT2RT	Broadcast RT to RT transfer message [0005 H]
EXC_MA_BRD_MODE	Broadcast Mode Command [0006 H]
EXC_MA_MINOR_FRAME	Minor frame message - delay [000F H]
data[ ]	A pointer to an array of command + data

Output Parameters

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

Return Values

EXC_MA_ERROR_BAD_HANDLE	If an invalid handle passed in; should be value returned by Init_Module_MA
EXC_MA_ERROR_WRONG_MODE	If not in BC mode
EXC_MA_ERROR_BAD_COMMAND_WORD	If the T/R bit in the Command word is set incorrectly for the given cmdtype parameter
EXC_MA_ERROR_ILLEGAL_INPUT	If an invalid parameter was used as an input
EXC_MA_ERROR_NO_SPACE_FOR_MESSAGE	If there is not enough space in message stack for this message
EXC_MA_ERROR_TOO_MANY_MESSAGES	If exceeded maximum number of messages permitted (1363)
EXC_MA_STATUS_OK	No errors

Enable_Retries_For_Message_MA

Description

If Enable_Retries_For_Message_MA is enabled, the selected message, if it has an error, will be retried according to the options set with Set_Retry_Options_MA. If disabled, the selected message will not be retried, no matter was options are selected with Set_Retry_Options_MA.

Syntax

Enable_Retries_For_Message_MA (int handle, int msgId, int bool enable)

Input Parameters

handle	The handle designated by Init_Module_MA	
msgId	Message identifier returned from a prior call to Create_1553_Message_MA.	
bool enable	True	Enable retries
	False	Disable retries

Output Parameters

none

Return Values

EXC_MA_ERROR_ BAD_HANDLE	If an invalid handle passed in; should be value returned by Init_Module_MA
EXC_MA_ERROR_ BAD_MESSAGE_ID	If attempted to change the setting for an undefined message
EXC_MA_STATUS_OK	No errors

Get_BC_Msgentry_MA

Description Get_BC_Msgentry_MA returns the status and the contents of a message.

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

Input Parameters

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

Output Parameters

msgptr	Pointer to the structure defined below in which to return the data. Space should always be allocated for 36 words of data to accommodate the maximum case of RT-to-RT transmission of 32 Data words plus 2 Status words and 2 Command words.

struct EXC_MA_BCMSG {

WORD msgstatus	Status word containing the msg status flags:
EXC_MA_END_OF_MSG	Indicates end of message [8000 H]
EXC_MA_WAS RETRIED	Transmission failed and was retried [4000 H]
EXC_MA_ME_SET	Message Error bit in the RT Status word [1000 H]
EXC_MA_BAD_STATUS	Bit other than ME set in 1553 Status word [0800 H]
EXC_MA_INVALID_MSG	Word count or Sync error occurred [0400 H]
EXC_MA_LATE_RESP	RT responded late [0200 H]
EXC_MA_INVALID_WORD	Bad bit count or Manchester error or parity error [0080 H]
EXC_MA_BAD_WD_CNT	RT transmitted wrong number of words [0020 H]
EXC_MA_BAD_RT_ADDR	1553 Status word contained wrong RT address [0010 H]
EXC_MA_BAD_SYNC	Status or Data sync were wrong [0008 H]
EXC_MA_MSG_ERROR	Error occurred; defined in other flags [0001 H]
EXC_MA_SELFTEST_ERR	MicroAce chip failed internal test [0100 H]

Get_BC_Msgentry_MA (cont.)

	unsigned int words [36]	A pointer to an array of 1553 words.
	}	

Return Values

EXC_MA_ERROR_ BAD_HANDLE	If an invalid handle passed in; should be value returned by Init_Module_MA
EXC_MA_ERROR_ WRONG_MODE	If not in BC mode
EXC_MA_ERROR_ BAD_FRM_MSGENTRY	If invalid index of message in frame
EXC_MA_ERROR_ BAD_FRAME_ID	If undefined frame id
EXC_MA_STATUS_OK	No errors

Note: For a given message *either* call Get_Msgentry_Status_MA and Read_Message_MA separately or use this function.

Do *not* call Get_Msgentry_Status_MA before calling this function. An error message EXC_MA_NO_BC_MSG will be returned.

Get_BC_Status_MA

Description

Get_BC_Status_MA indicates what condition triggered an interrupt. The function returns status flags generated since the last call to Reset_BC_Status_MA.

The function may be used for polling applications.

Note: Reset this status with the Reset_BC_Status_MA command.

Syntax

Get_BC_Status_MA (`int` handle, `unsigned char` *pStatus)

Input Parameters

handle	The handle designated by Init_Module_MA

Output Parameters

pStatus	0 or more of the following flags: EXC_MA_MSG_ERR An error in some message [0008 H] EXC_MA_END_OF_FRAME Complete frame of messages sent [0004 H] EXC_MA_MSG_CMPLT Message sent [0002 H] EXC_MA_WAIT_FOR_CONTINUE The bus controller stopped due to a halt instruction [0001 H]

Return Values

EXC_MA_ERROR_ BAD_HANDLE	If an invalid handle was specified; should be value returned by Init_Module_MA
EXC_MA_ERROR_ WRONG MODE	If not in BC mode
EXC_MA_STATUS_OK	No errors

Get_Minor_Frame_Time_MA

Description

Get_Minor_Frame_Time_MA gets the previously stored minor frame time in microseconds. This value was set in a call to Set_Minor_Frame_Time_MA.

Syntax

Get_Minor_Frame_Time_MA (`int` handle, `long` *frame_time)

Input Parameters

handle	The handle designated by Init_Module_MA

Output Parameters

frame_time	The minor frame time in microseconds

Return Values

EXC_MA_ERROR_ BAD_HANDLE	If an invalid handle was specified; should be value returned by Init_Module_MA
EXC_MA_ERROR_ WRONG MODE	If not in BC mode
EXC_MA_STATUS_OK	No errors

Get_Msgentry_Status_MA

Description

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

Syntax

Get_Msgentry_Status_MA (`int` handle, `int` frameid, `int` msgentry, `WORD` *msgstatusword)

Input Parameters

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

Output Parameters

msgstatusword	Status word containing the msg status flags:

EXC_MA_END_OF_MSG	Indicates end of message [8000 H]
EXC_MA_WAS_RETRIED	Transmission failed and was retried [4000 H]
EXC_MA_ME_SET	Message Error bit in the 1553 Status word [1000 H]
EXC_MA_BAD_STATUS	Bit other than ME set in 1553 Status word [0800 H]
EXC_MA_INVALID_MSG	Word count or Sync error occurred [0400 H]
EXC_MA_LATE_RESP	RT responded late [0200 H]
EXC_MA_INVALID_WORD	Bad bit count or Manchester error or parity error [0080 H]
EXC_MA_BAD_WD_CNT	RT transmitted wrong number of words [0020 H]
EXC_MA_BAD_RT_ADDR	1553 Status word contained wrong RT address [0010 H]
EXC_MA_BAD_SYNC	Status or Data sync were wrong [0008 H]
EXC_MA_MSG_ERROR	Error occurred; defined in other flags [0001 H]
EXC_MA_SELFTEST_ERR	MicroAce chip failed internal test [0100 H]

Get_Msgentry_Status_MA (cont.)

Return Values	EXC_MA_ERROR_ BAD_HANDLE	If an invalid handle passed in; should be value returned by Init_Module_MA
	EXC_MA_ERROR_ WRONG_MODE	If not in BC mode
	EXC_MA_ERROR_ ILLEGAL_INPUT	If an invalid parameter was used as an input
	EXC_MA_ERROR_BAD _FRM_MSGENTRY	If invalid index of message in frame
	EXC_MA_STATUS_OK	No errors

Read_Message_MA

Description	Read_Message_MA allows the user to read back data associated with a given message. This is used for reading data returned from a Transmit command or for reading the 1553 Status word returned by an RT for all message types. The output consists of the entire message in the same sequence as it appeared on the bus including Command words, Status words and data. **Note:** See **Appendix C: MIL-STD-1553 Formats**	
Syntax	Read_Message_MA (`int` handle, `int` msgId, `WORD` *pWords)	
Input Parameters	handle	The handle designated by Init_Module_MA
	msgId	Message identifier returned from a prior call to Create_1553_Message_MA
Output Parameters	pWords	An array of up to 36 words containing Command, Status and Data words associated with the message id
Return Values	EXC_MA_ERROR_ BAD_HANDLE	If an invalid handle was specified; should be value returned by Init_Module_MA
	EXC_MA_ERROR_ WRONG_MODE	If not in BC mode
	EXC_MA_ERROR_ BAD_MSG_ID	If id is not a valid message id
	EXC_MA_ERROR_ MINORFRAMEMSG	If message is a Minor Frame type
	EXC_MA_STATUS_OK	No errors

Reset_BC_Status_MA

Description Reset_BC_Status_MA clears the BC status so that the next call to Get_BC_Status_MA is meaningful.

Syntax Reset_BC_Status_MA (`int` handle)

Input Parameters

handle	The handle designated by Init_Module_MA

Output Parameters none

Return Values

EXC_MA_ERROR_ BAD_HANDLE	If an invalid handle was specified; should be value returned by Init_Module_MA
EXC_MA_ERROR_ WRONG_MODE	If not in BC mode
EXC_MA_STATUS_OK	No errors

Run_BC_MA

Description Run_BC_MA causes the Bus Controller to start transmitting messages. All data structures must be set up before calling this function. See Create_1553_Message_MA, Create_Frame_MA, Start_Frame_MA (when necessary) and Set_Frame_Time_MA (if not using minor frames.)

Syntax Run_BC_MA (`int` handle, `int` type)

handle	The handle designated by Init_Module_MA
type	EXC_MA_RUN_CONTINUOUS Continuous operation [0000 H]
	EXC_MA_RUN_ONCE Execute frame once [0001 H]

Output Parameters none

Return Values

EXC_MA_ERROR_ BAD_HANDLE	If an invalid handle was specified; should be value returned by Init_Module_MA
EXC_MA_ERROR_NO_ FRAME_DEFINED	If tried to run the BC without defining a frame
EXC_MA_ERROR_ WRONG_MODE	If not in BC mode
EXC_MA_ERROR_ ALREADY_RUNNING	If user *either* did not call Stop_MA after the last continuous run *or* previous (non-continuous) run has not completed
EXC_MA_STATUS_OK	No errors

Set_Bus_MA

Description	Set_Bus_MA sets the bus over which a particular message will be transmitted.	
Syntax	Set_Bus_MA (`int` handle, `int` msgId, `int` bus)	
Input Parameters	handle	The handle designated by Init_Module_MA
	msgId	Message identifier returned from a prior call to Create_1553_Message_MA
	bus	EXC_MA_XMT_BUS_A Transmit message over Bus A [0080 H]
		EXC_MA_XMT_BUS_B Transmit message over Bus B [0000 H]
Output Parameters	none	
Return Values	EXC_MA_ERROR_ BAD_HANDLE	If an invalid handle was specified; should be value returned by Init_Module_MA
	EXC_MA_ERROR_ WRONG_MODE	If not in BC mode
	EXC_MA_ERROR_ BAD_MSG_ID	If id is not a valid message id
	EXC_MA_ERROR_ BAD_CHANNEL	If tried to set bus to illegal value
	EXC_MA_STATUS_OK	No errors

Set_Frame_Time_MA

Description

Set_Frame_Time_MA determines the time each frame will take when the frame is to be executed more than once. (See Run_BC_MA on page 5-15).

Note: Do not:

1. Set the frame time **smaller** than the time it will take for the defined frame to go out. The results are undefined.
2. Use Set_Frame_Time_MA and Set_Minor_Frame_Time_MA together. The architecture of the MicroAce assumes that the frame timer is used *either* for timing minor frames *or* for timing major frame but not both.

Syntax

Set_Frame_Time_MA (`int` handle, `long` time)

Input Parameters

handle	The handle designated by Init_Module_MA
time	Frame time in microseconds

Output Parameters

none

Return Values

EXC_MA_ERROR_ BAD_HANDLE	If an invalid handle was specified; should be value returned by Init_Module_MA
EXC_MA_ERROR_ WRONG_MODE	If not in BC mode
EXC_MA_STATUS_OK	No errors

Set_Interrupt_On_Msg_MA

Description Set_Interrupt_On_Msg_MA causes an interrupt to be generated when a chosen message is sent.

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

Input Parameters

handle	The handle designated by Init_Module_MA
frameid	frame_id returned from a call to Create_Frame_MA
msgentry	Message number within the frame
enable	EXC_MA_ENABLE Generate an interrupt [0001 H] EXC_MA_DISABLE Do *not* generate an interrupt [0000 H]

Output Parameters none

Return Values

EXC_MA_ERROR _BAD_HANDLE	If an invalid handle was specified; should be value returned by Init_Module_MA
EXC_MA_ERROR_ BAD_FRAME_ID	If undefined frame id
EXC_MA_ERROR _BAD_FRM_MSGENTRY	If invalid index of message in frame
EXC_MA_ERROR_ WRONG_MODE	If not in BC mode
EXC_MA_STATUS_OK	No errors

Set_Minor_Frame_Time_MA

Description

Set_Minor_Frame_Time_MA sets time for a Minor Frame message type. This message functions as a "delay time" message. The time given is from the beginning of a Minor Frame to the beginning of the next Minor Frame.

Note: Do not use Set_Frame_Time_MA and Set_Minor_Frame_Time_MA together. The architecture of the MicroAce assumes that the frame timer is used *either* for timing minor frames *or* for timing major frame but not both.

Syntax

Set_Minor_Frame_Time_MA (`int` handle, `long` micro_second)

Input Parameters

handle	The handle designated by Init_Module_MA
micro_second	Minor Frame time in microseconds

Output Parameters

none

Return Values

EXC_MA_ERROR_ BAD_HANDLE	If an invalid handle was specified; should be value returned by Init_Module_MA
EXC_MA_ERROR_ WRONG_MODE	If not in BC mode
EXC_MA_STATUS_OK	No errors

Set_Retry_Options_MA

Description

Call Set_Retry_Options_MA to set the global retry options. Once this function has been called, any message can have retries enabled by calling Enable_Retries_For_Message_MA.

Note: All messages with retries enabled will use the **same** options, as set with Set_Retry_Options_MA.

Syntax

Set_Retry_Options_MA (`int` handle, `MA_RETRY_OPTION` firstRetryOption, `MA_RETRY_OPTION` secondRetryOption

Input Parameters

handle	The handle designated by Init_Module_MA
firstRetryOption	MA_RETRY_PRIMARY_BUS Retry on primary bus MA_RETRY_ALTERNATE_BUS Retry on alternative bus MA_RETRY_NONE No retry
secondRetryOption	MA_RETRY_PRIMARY_BUS Retry on primary bus MA_RETRY_ALTERNATE_BUS Retry on alternative bus MA_RETRY_NONE No retry

Example

Assume that the original message is sent out on Bus A:

Set_Retry_Option_MA (MA_RETRY_PRIMARY_BUS, MA_RETRY_NONE)
One retry on Bus A

Set_Retry_Option_MA (MA_RETRY_PRIMARY_BUS, MA_RETRY_PRIMARY_BUS)
Two retries, both on Bus A

Set_Retry_Option_MA (MA_RETRY_PRIMARY_BUS, MA_RETRY_ALTERNATE_BUS)
Two retries: first on Bus A, second on Bus B

Set_Retry_Option_MA (MA_RETRY_ALTERNATE_BUS, MA_RETRY_PRIMARY_BUS)
Two retries: first on bus B, second on Bus A

Output Parameters

none

Return Values

EXC_MA_ERROR_ BAD_HANDLE	If an invalid handle passed in; should be value returned by Init_Module_MA
EXC_MA_ERROR_ WRONG_MODE	If not in BC mode
EXC_MA_ERROR_ ILLEGAL_INPUT	If an invalid parameter was used as an input
EXC_MA_STATUS_OK	No errors

Start_Frame_MA

Description

Start_Frame_MA selects which of the multiple frames created by calls to Create_Frame_MA are to be run when Run_BC_MA is called.

Note: This function is called internally by Create_Frame_MA. Thus, if you call Run_BC_MA without calling Start_Frame_MA, the last created frame will be run. If the module is stopped, either by calling Stop_MA, or because all requested messages have been transmitted, the module may be restarted by calling Start_Frame_MA and then Run_BC_MA

Syntax

Start_Frame_MA (`int` handle, `int` frameid, `int` numMessages)

Input Parameters

handle	The handle designated by Init_Module_MA
pFrameId	Frame identifier returned from a prior call to Create_Frame_MA
numMessages	EXC_MA_FULLFRAME Execute the entire frame [0000 H]

Output Parameters

none

Return Values

EXC_MA_ERROR_ BAD_HANDLE	If an invalid handle was specified; should be value returned by Init_Module_MA
EXC_MA_ERROR_ BAD_FRAME_ID	If undefined frame id
EXC_MA_ERROR_BAD_ FRM_MSGENTRY	If invalid index of message in frame
EXC_MA_ERROR_ WRONG_MODE	If not in BC mode
EXC_MA_STATUS_OK	No errors

Appendix A DDC MicroAce / Px API Comparison

	M4K1553MA	M4K1553PxII
Modes	Bus Controller, Monitor and RT/Concurrent monitor	BC/Concurrent-RT, Monitor (including Look-up and Sequential submodes) and RT/Concurrent monitor
Number of Remote devices (RT) supported	One RT	Up to 32 RTs
Unique function handle	_MA	_Px
Flags and types	Added prefix "EXC_MA_"	no prefix
Header file including all other header files	**`MA_Proto.h`**	**`galahad.h`**
Error names	New error names - old names can be used for most functions (retained the same negative values). A backward compatible header file can be provided.	
Return type	enum EXC_MA_DRIVER_STATUS	int
Get_Error_String syntax	Takes a length parameter: [`EXC_MA_DRIVER_STATUS` status, `char` *szStatusString, `int` max StringLength]	No length parameter: [`int` errocode, `char` *errstring]
Notation	`WORD` **Note:** Or make your own typedef or #define for `usint`	`usint`
Function output: Get_Board_Status Get_Mode Get_Id Get_Mon_Status Init_Module Create_Frame	Output parameters	Return values
Set_RT_Active	No interrupt parameter	`int` intrpt
MIscellaneous	Minor changes in individual functions such as that module status does not have timer test results as there is not timer test - for compatibility the bit is always set to 1, therefore appearing as if the timer test passed.	

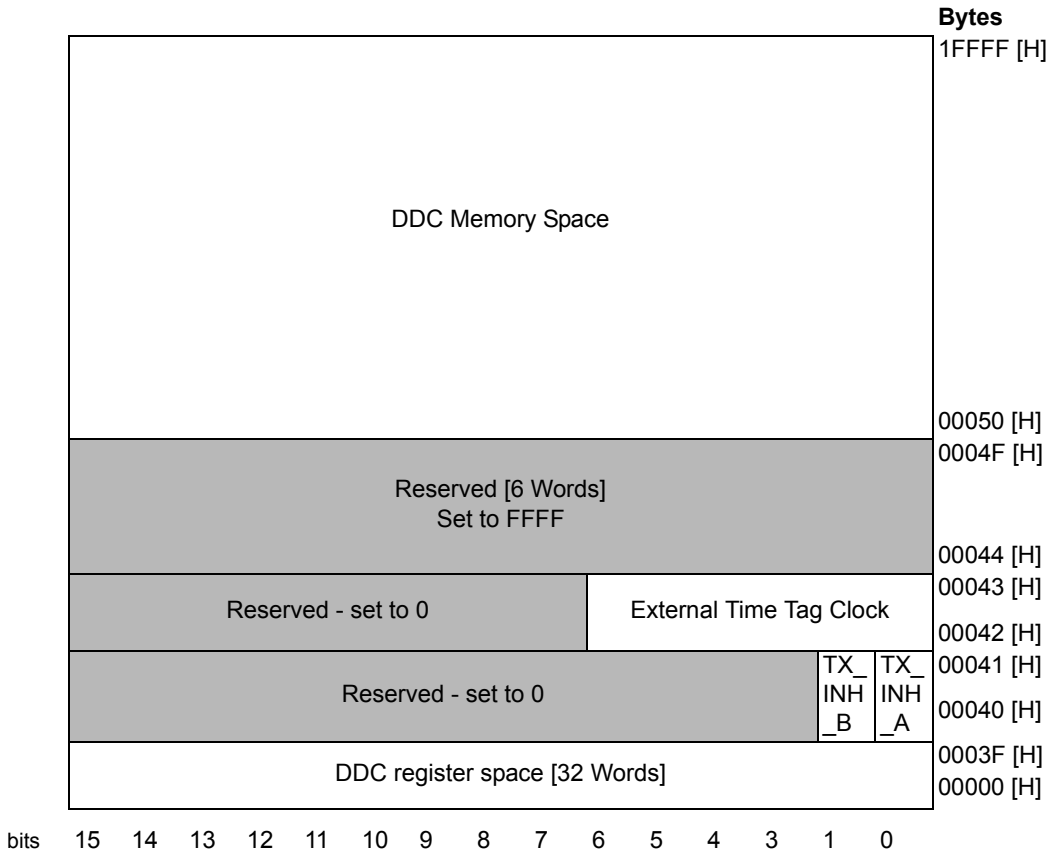
Appendix B *M4K1553MA* Module Specific Implementation of the DDC MicroAce Chip

The *M4K1553MA* module is an intelligent, MIL-STD-1553 interface module for the multimode, multiprotocol *EXC-4000* family of carrier boards. The *M4K1553MA* provides a user-friendly MIL-STD-1553 interface for remote terminals, monitors and bus controllers. It is based on the DDC MicroAce chip.

Appendix B outlines how the DDC MicroAce chip is implemented on the *M4K1553MA* module. For more details about the functionality of the chip see the *Enhanced Miniature Advanced Communications Engine (Enhanced Mini-ACE Series) User's Guides Volumes 1 (Architectural Reference) & 2 (Hardware Reference)*.

Appendix B-1 Memory Map

The *M4K1553MA* has a total of 64K words of internal memory.



B.1.1 DDC Register Space

Address: 00–3F (H)
Length 16 bits

See the *Enhanced Miniature Advanced Communications Engine (Enhanced Mini-ACE Series) User's Guides Volumes 1 (Architectural Reference)* for a detailed description of these registers.

See **Appendix B-2 M4K1553MA Configuration** on page B-3 for the configuration of the components of the *M4K1553MA* which will have an impact on the information on the registers detailed in the *User's Guide*.

B.1.2 Transmit Inhibit Register

Address: 40 (H)
Length 2 bits

The Transmit Inhibit register can be used to inhibit transmission on the 1553 Buses.

Bit	Description
2-15	Reserved – set to 0s
01	1 = Channel B inhibited/shutdown 0 = Normal operation of Channel B
00	1 = Channel A inhibited/shutdown 0 = Normal operation of Channel A

Transmit Inhibit Register

For a description of the pins TX_INH_A and TX_INH_B and their impact on the functionality of the DDC chip, see the *Enhanced Miniature Advanced Communications Engine (Enhanced Mini-ACE Series) User's Guides Volumes 1 (Architectural Reference)*.

B.1.3 External Time Tag Clock Register

Address: 42 (H)
Length 6 bits

The External Time Tag Clock register selects the external Time clock value used to increment the MicroAce's Time Tag register. In normal operation, the MicroAce uses an internal clock with several choices for the resolution of the Time tag register. See **Set_Timetag_Res_MA** on page 2-11 or the *Enhanced Miniature Advanced Communications Engine (Enhanced Mini-ACE Series) User's Guides Volumes 2 (Hardware Reference)*.

To set a resolution which is not offered or to synchronize between modules or boards, the chip can be programmed to use an external, user-defined clock. See the *Enhanced Miniature Advanced Communications Engine (Enhanced Mini-ACE Series) User's Guides Volumes 1 (Architectural Reference)*, **Configuration register #2**, **bits 07–09**.

The external clock is derived from the *EXC-4000* carrier board's HTTCLK signal (1 MHz). To select a different carrier board clock source, using the Time Tag Clock Select register, and for a description of the External signals, see the *User's Manual* for your carrier board.

A hardware or software reset will reset the external timer clock to the default value (64 μsec).

The External Time Tag Clock has a range of 4 µsec. (250 KHz) to 126 µsec. (8 KHz)

Bit	Description
06 – 15	Reserved - set to 0
00 – 05	Timer Clock Value

External Time Tag Clock Register

To calculate the Timer Clock Value [TCV]:

$$TCV = \frac{1,000,000}{2 \times F} - 1$$

F = Desired frequency (HZ) – [8 KHz-250 KHz]

Example: Desired programmable Timer Clock Frequency is 10 KHz (100 µsec.) resolution:

$$TCV = \frac{1,000,000}{2 \times 10,000} - 1 = (50 - 1) = 49(Dec) = 31(H)$$

Write 00031 (H) to the register.

B.1.4 DDC Memory Space

Address: **50 – 1FFFF (H)**
Length **16 bits**

The DDC MicroAce's internal RAM, beginning from address 0050 H, resides in this shared memory space. See the *Enhanced Miniature Advanced Communications Engine (Enhanced Mini-ACE Series) User's Guides Volumes 1 & 2 Architectural Reference* for details.

Appendix B-2 *M4K1553MA* Configuration

The *M4K1553MA* has been interfaced to the MicroAce chip in the following manner:

- 16BIT
- Buffered
- Nonzero Wait
- Non-Multiplexed address / data bus
- A 20 MHz external clock is provided as the system clock
- Subsystem Flag Input Signal (SSFLAG) to the DDC chip has been tied high[1]
- RT_AD_LAT input signal to the DDC chip has been tied high[2]
- Built-in-Test has been enabled.

1. *Enhanced Miniature Advanced Communications Engine (Enhanced Mini-ACE Series) User's Guides Volume 1: Architectural Reference*, Configuration Register #1, bit 08
2. *Enhanced Miniature Advanced Communications Engine (Enhanced Mini-ACE Series) User's Guides Volume 1: Architectural Reference*, Configuration Register #6, bit 05

Appendix C MIL-STD-1553 Formats

Appendix C provides diagrams of MIL-STD-1553 Word and Message formats.

C-1 Word Formats

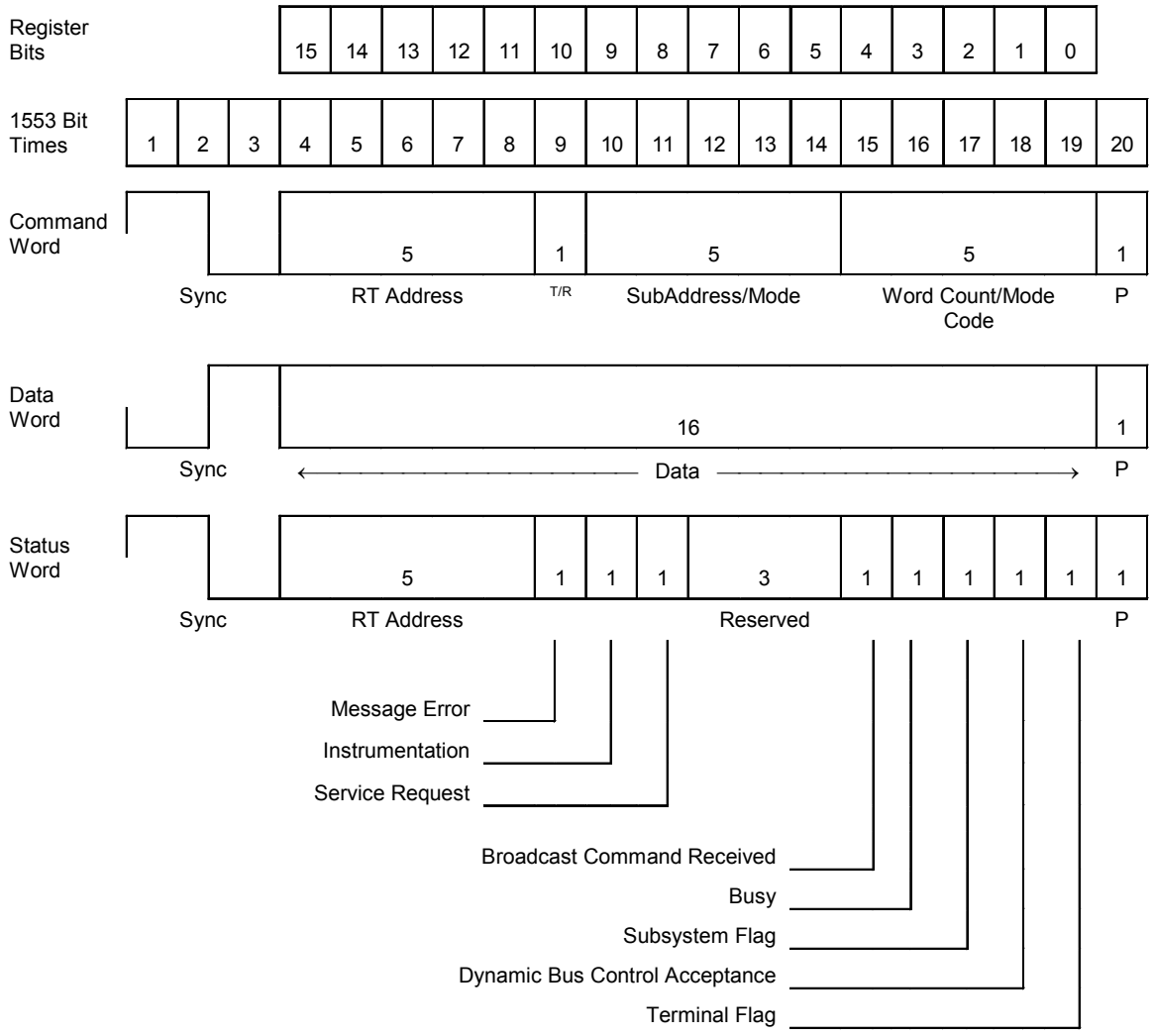


Figure C-1 MIL-STD-1553 Word Formats

Note: T/R = Transmit/Receive
P = Parity

C-2 Message Formats

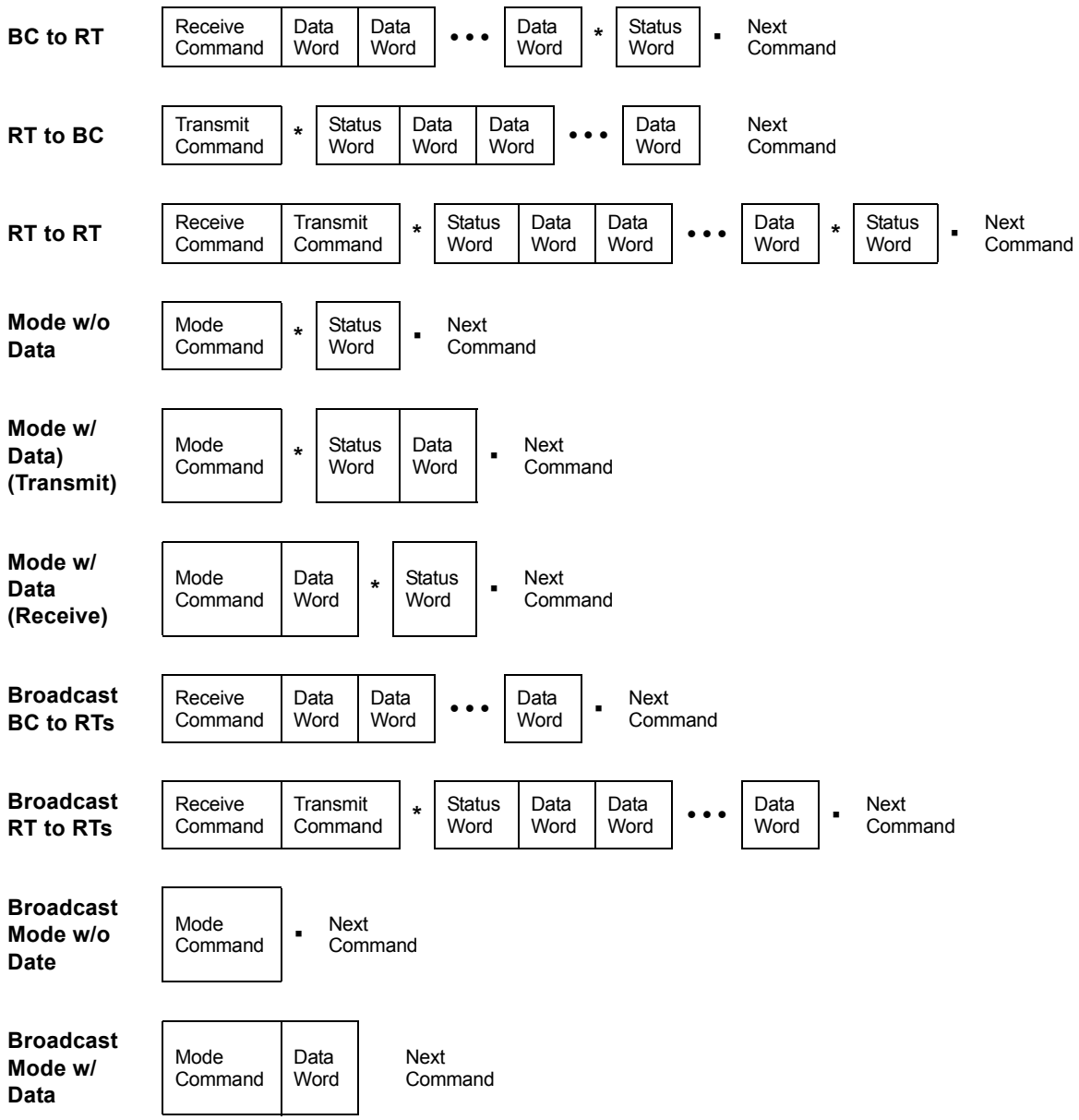


Figure C-2 MIL-STD-1553B Message Formats

Note: * = Response time
▪ = Intermessage Gaptime

Appendix D *M4K1553MA Software Tools* Installation Instructions

Appendix D provides installation instructions for adding *M4K1553MA Software Tools* to the *EXC-4000P104plus* carrier board. These Software Tools run under the following operating systems:

- Windows 9*x/ME*
- Windows NT4
- Windows 2000/XP

Warning: Whenever you handle an EXC-4000P104plus board, wear a suitably grounded electrostatic discharge wrist strap.

Windows 9*x /ME* Procedures

The *Excalibur Installation CD* contains the files for:

- *PCI Hardware Installation for Windows 9x/ME*
- *M4K1553MA Software Tools*

To install a *EXC-4000P104plus* board on a Windows 9*x* system:

1. Insert the board into an available PCI slot in your computer
2. Add *M4K1553MA Software Tools* to your Windows 9*x* system.

To verify that the board was installed correctly, run the test programs included on the *Excalibur Installation CD*.

To install the Excalibur EXC-4000P104plus board

1. Make sure the computer is turned off. Insert the board into one of the available slots. For more information, see the **Mechanical and Electrical Specifications** chapter in the *EXC-4000P104plus User's Manual.*
2. Turn on the computer and wait several moments while Windows 9*x/ME* boots up.
3. The message **Building driver information data** may or may not appear. After several seconds, the message **New Hardware Found** is displayed. Next, the **Update Device Driver Wizard** is displayed.

Note: If the **Update Device Driver Wizard** is not displayed, and you are upgrading from a previous version of the PCI Win 9*x*/ME Hardware Installation, follow these steps to invoke the **Update Device Driver Wizard.** Right-click **My Computer,** then click **Properties | Device Manager | Excalibur PCI Cards | Excalibur 4000PCI Card | Properties | Driver | Update Driver.**

4. Insert the *Excalibur Installation CD* in the drive and follow the on-screen instructions.
5. When the following message appears: **Windows found the following updated driver for this device: Excalibur EXC-4000 card**, click **Finish**.
6. Board installation is completed. If prompted to reboot the computer, do so now. Next, continue with board verification below.

To verify board installation:

1. Make certain the Excalibur *EXC-4000P104plus* board is in place in the computer.
2. Right-click **My Computer**. Select **Properties.** The **System Properties** dialog box appears.
3. In the **System Properties** dialog box, click the **Device Manager** tab.
4. Double-click **Excalibur PCI Card**. Verify that the Excalibur *EXC-4000P104plus* board is listed next to a gray diamond-shaped icon.

Board installation verification is successful.

Note: If you see an exclamation point (**!**) superimposed on the gray diamond, this indicates that the board is not properly installed. Check the following:

- There are not enough memory resources available in the system, free up more memory or IRQs.
- That the computer is Plug and Play compatible. Many computers claim to be "Plug and Play" but are not fully compatible with the Plug and Play specification.

To Add *M4K1553MA Software Tools* to Windows 9*x/ME* Systems

Note: If there is already a previous version of *M4K1553MA Software Tools* for the same module installed on the computer, the new version will overwrite it. We recommend that you first uninstall the previous version by selecting **Start | Settings | Control Panel | Add/Remove Programs.** If you want to save the earlier version, you can choose to install the new version to a different directory when the Install Wizard asks to which directory to install the software.

1. Insert the *Excalibur Installation CD* in the drive, click **Install Drivers, Applications, Utilities**.
2. Follow the on-screen instructions to select the software that matches your product.
3. When the **Excalibur Configuration Utility** (**`ExcConfig`**) screen appears, double-click the **Type** field. Select **4000PCI**.
4. Leave the **Auto** values and click **Save**.

Note: Remember the device number; it is the parameter for the Init_Module_MA function.

The *EXC-4000P104plus* board is now ready to run.

Running test programs

Excalibur provides test programs to verify that the board is operating properly. The source code is provided with the test programs as a guide to develop your own applications.

Go to **Start | Programs | *[Product Name]***, to run the test programs.

Windows NT4 Procedures

The *Excalibur Installation CD* contains the files for *M4K1553MA Software Tools*.

To install an *EXC-4000P104plus* board on a Windows NT4 system:

1. Insert the board into an available PCI slot in the computer.
2. Add *M4K1553MA Software Tools*
3. Reboot the computer
4. To verify that the board was installed correctly, run the test programs that are included on the *Excalibur Installation CD*

To install the Excalibur EXC-4000P104plus board

1. Make sure the computer is turned off.
2. Insert the board into one of the available slots.

For more information, see the **Mechanical and Electrical Specifications** chapter in the *EXC-4000P104plus User's Manual*.

To add *M4K1553MA Software Tools* to Windows NT4 systems

Note: If there is already a previous version of *Software Tools* for the same module installed on the computer, the new version will overwrite it. We recommend that you first uninstall the previous version by selecting **Start | Settings | Control Panel | Add/Remove Programs**. If you want to save the earlier version, you can choose to install the new version to a different directory when the Install Wizard asks to which directory to install the software.

1. Insert the *Excalibur Installation CD* in the drive, click **Install Drivers, Applications, Utilities**.
2. Follow the on-screen instructions to select the software that matches your product.
3. When the **Excalibur Configuration Utility** (**`ExcConfig`**) screen appears, double-click the Type field. Select **4000PCI**.
4. Leave the **Auto** values and click **Save**.

   Note: Remember the device number; it is the parameter for the Init_Module_MA function.
5. Reboot the computer at the end of the installation procedure.

The *EXC-4000P104plus* board is now ready to run.

Running test programs

Excalibur provides test programs to verify that the board is operating properly. The source code is provided with the test programs as a guide to develop your own applications.

Go to **Start | Programs | *[Product Name]***, to run the test programs.

Windows 2000/XP Procedures

The *Excalibur Installation CD* contains the files for:

- *PCI Hardware Installation for Windows 2000/XP*
- *M4K1553MA Software Tools*

To install an *EXC-4000P104plus* board on a Windows 2000/XP system:

1. Insert the board into an available PCI slot in your computer
2. Add *M4K1553MA Software Tools* to your Windows 2000/XP system.

To verify that the board was installed correctly, run the test programs included on the *Excalibur Installation CD*.

To install the Excalibur EXC-4000P104plus Board:

1. Make sure the computer is turned off. Insert the board into one of the available slots. For more information see the Installation section in the module's *User's Manual.*
2. Turn on the computer and wait several moments while Windows 2000/XP boots up.
3. The message **Building driver information data** may or may not appear. After several seconds, the message **New Hardware Found** is displayed. Next, the **Update Device Driver Wizard** is displayed.

Note: If the **Update Device Driver Wizard** is not displayed, and you are upgrading from a previous version of the PCI Windows Hardware Installation, follow these steps to invoke the **Update Device Driver Wizard.** Right-click **My Computer,** then click **Hardware | Device Manager | Excalibur PCI Cards | Excalibur 4000PCI card.** Double-click and select **Driver | Update Drivers.**

4. Insert the *Excalibur Installation CD* in the drive and follow the on-screen instructions.
5. When the following message appears: **Windows found the following updated driver for this device: Excalibur EXC-4000 card**, click **Finish.**
6. Board installation is completed. If prompted to reboot the computer, do so now. Next, continue with board verification below.

To verify board installation:

1. Make certain the Excalibur *EXC-4000P104plus* board is in place in the computer.
2. Right-click **My Computer | Properties**. The System Properties dialog box appears.
3. In the System Properties dialog box, click **Hardware | Device Manager.**
4. Double-click **Excalibur PCI Board**. Verify that the Excalibur PCI Board is listed next to a gray diamond-shaped icon.

Note: If an exclamation point **(!)** appears, superimposed on the gray diamond, this indicates that the board is not properly installed. Check that there are enough memory resources available in the system. Free up more memory or IRQs.

The board installation verification is successful.

To add *M4K1553MA Software Tools* under Windows 2000/XP

Note: If there is already a previous version of *Software Tools* for the same module installed on the computer, the new version will overwrite it. We recommend that you first uninstall the previous version by selecting **Start | Settings | Control Panel | Add/Remove Programs**. If you want to save the earlier version, you can choose to install the new version to a different directory when the Install Wizard asks to which directory to install the software.

1. Insert the *Excalibur Installation CD* in the drive, click **Install Drivers, Applications, Utilities.** Follow the on-screen instructions to select the software that matches your product.
2. When the **Excalibur Configuration Utility** (**`ExcConfig`**) screen appears, double-click the **Type** field. Select **4000PCI**.
3. Leave the **Auto** values and click **Save**.

Note: Remember the device number; it is the parameter for the Init_Module_MA function.

The *EXC-4000P104plus* board is now ready to run.

Running test programs

Excalibur provides test programs to verify that the board is operating properly. The source code is provided with the test programs as a guide to develop your own applications.

Go to **Start | Programs | *[Product Name*]**, to run the test programs.

Appendix E Multiple Board Support

M4K1553MA Software Tools supports the use of up to four *EXC-4000P104plus* boards simultaneously.

To use multiple boards:

1. Each board must be set for a unique ID. The ID of a board is set via the SW1 DIP switch. To set the ID of the board set the jumpers as follows:

Board ID	SW1 Switch contact			
	1	2	3	4
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1

Set DIP switch SW1 to a unique ID by setting the switch contacts *open* or *off* to represent logic '1'; and closed or on to represent logic '0'.

2. In the **`ExcConfig`** utility program, create a separate device number for each board. For each board enter the appropriate values in the Excalibur Configuration Screen, as described in the table below.

Name of field	Enter this value
Type	Select 4000PCI from the drop-down list
Unique Identifier	Set to the unique ID configured above in Step 1.
Memory	Set to automatic setting
Interrupt Setup	Set to automatic setting
I/O ports	Set to automatic setting

-

Appendix F Source Code References

Appendix F-1 *M4K1553MA* Software Tools Library

M4K1553MA Software Tools is set of "C" language functions designed to aid programmers of the *M4K1553MA* to write applications. **Appendix F-1** lists the files containing the functions. **Appendix F-2 Code Index** lists the functions by mode.

The files are divided into six categories:

Main files

Mode classes files

Module class files

"C" interface files

Support files

Interface to the kernel drivers

Main files

Headers defining the interface:

`MA_DataStructures.h`
`MA_Error.h`
`MA_Flags.h`
`MA_Proto.h`
`MA_Types.h`

Mode classes files

`MicroAceMode.h` Base class containing things relevant to all modes
`MicroAceMode.cpp`

BC mode:

`MicroAceBCMode.h` Class definition for BC mode
`MicroAceModuleBCModeWindowsSpecific.h` BC mode memory map
`MicroAceBCMode.cpp` Implementation of BC mode functions
`MicroAceBCModeWindowsSpecific.cpp` Implementation of BC mode functions requiring memory management

Monitor mode:

`MicroAceMonMode.h` Class definition for Monitor mode
`MicroAceMonMode.cpp` Implementation of monitor mode functions
`MicroAceMonModeWindowsSpecific.cpp` Implementation of Monitor mode functions requiring memory management

RT mode:

MicroAceRTMode.h Class definition for RT mode

MicroAceRTModeMemoryMap.h RT mode memory map

MicroAceRTMode.cpp Implementation of RT mode functions

MicroAceRTModeWindowsSpecific.cpp Implementation of RT mode functions requiring memory management

Module class files

Contains interface to one module, including implementations of the general functions and wrappers for the mode-specific functions

MicroAceModule.h

MicroAceModuleMemoryMap.h

MicroAceModule.cpp Implementation of general functions. wrapper functions for mode-specific functions

MicroAceModuleWindowsSpecific.cpp Implementation of general functions requiring memory management

"C" interface files

MicroAceModuleList.h Manages a list of modules

MicroAceModuleList.cpp

MA_Functions.cpp Contains a list of modules and a 'C' wrapper function for each software tools function, taking a handle specifying which module

Support files

1553MessageUtilities.h

1553MessageUtilities.cpp

ExcBasicValuePtr.h Very simple smart pointer

ExcBasicValuePtrWindowsSpecific.h Platform dependent simple smart pointer code

ExcBasicValuePtr.cpp

ExcBasicValuePtrWindowsSpecific.cpp

MA_Parity.h Function for calculating parity

MA_Parity.cpp

ExcTimer.h Class for using performance counter for timing

ExcTimerWindowsSpecific.cpp

MA_DiagAssert.h Debug-mode diagnostics

MA_DiagAssert.cpp

Interface to the kernel drivers

Deviceio.h Interaction with the kernel driver

error_devio.h Error codes for kernel drivers

exc4000.h Board level functions

excsysio.h Defined shared codes and structures between DLL and kernel drivers

Appendix F-2 Code Index

Appendix F-3 Error Messages

All functions in the *M4K1553MA Software Tools* are written as "C" functions, i.e., they return values of the enumerated type EXC_MA_DRIVER_STATUS. A negative value signifies an error.

Error Code	Value	Definition
EXC_MA_STATUS_OK	0	No errors
EXC_MA_ERROR_BAD_ MSG_ID	-1	Requested an undefined message
EXC_MA_ERROR_ILLEGAL_ INPUT	-2	Illegal value used as input
EXC_MA_ERROR_WRONG_ MODE	-3	Mode specific command called when in the wrong mode
EXC_MA_ERROR_BAD_ CHANNEL	-4	Tried to set channel to illegal value
EXC_MA_ERROR_NO_MEM_ FOR_SIM_MODE	-5	Init failed in Simulate mode due to Malloc call
EXC_MA_ERROR_NO_ SPACE_FOR_MESSAGE	-7	Not enough space in message stack for this message
EXC_MA_ERROR_TOO_ MANY_MESSAGES	-8	Exceeded maximum number of messages permitted
EXC_MA_ERROR_BAD_ MESSAGE_ID	-9	Attempted to place an undefined message into a message frame
EXC_MA_ERROR_NO_ SPACE_FOR_FRAME	-10	Not enough space in frame stack for this frame
EXC_MA_ERROR_BAD_ FRAME_ID	-11	Undefined frame id
EXC_MA_ERROR_TOO_ MANY_FRAMES	-13	Exceeded maximum number of frames permitted (20)
EXC_MA_ERROR_BAD_ FRM_MSGENTRY	-14	Invalid index of message in frame
EXC_MA_INVALID_ INTERRUPT	-15	Attempted to set an invalid interrupt
EXC_MA_STATUS_NO_ MORE_MESSAGES	-18	Attempt to read messages when no new msgs available
EXC_MA_RESET_TIMEOUT	-26	One of the self tests (protocol or RAM) did not complete within expected time
EXC_MA_ERROR_WRONG_ MODULE_TYPE	-27	The module at the specified location was not a module of the requested type
EXC_MA_ERROR_NO_ MODULE	-28	There is no module at the specified location
EXC_MA_ERROR_BAD_ HANDLE	-33	Invalid handle passed in
EXC_MA_ERROR_TOO_ MANY_BOARDS	-36	Too many boards initialized
EXC_MA_ERROR_BAD_ COMMAND_WORD	-60	Command word T/R bit does not match the requested command type

Error Code	Value	Definition
EXC_MA_STATUS_NOALTER	-62	This message is being transmitted
EXC_MA_NO_BC_MSG	-65	Attempt to read BC message that has not been received
EXC_MA_ERROR_ MINORFRAMEMSG	-84	Attempted an inappropriate action on a minor frame message
EXC_MA_ERROR_CANT_ ALTER_MODE_CODE	-150	Can not use Alter_Cmd on a mode code message
EXC_MA_ERROR_NO_ FRAME_DEFINED	-151	The frame you are trying to run does not exist. Probably means you are trying to run the BC without defining a frame
EXC_MA_ERROR_ ALREADY_RUNNING	-152	Perhaps you did not call Stop() after the last run? Perhaps you are in BC mode and the previous (non-continuous) Run_BC has not yet completed?
EXC_MA_INVALID_MODE	-153	Card is in an invalid mode (returned from Get_Mode)
EXC_MA_STATUS_ GETSTRING_TRUNCATED	-154	Error string was larger than the buffer passed in. The beginning of the string was copied.
EXC_MA_STOP_TIMEOUT	-156	Timed out waiting for module to respond to a stop request. Wait (checking EXC_MA_BOARD_HALTED in Get_Board_Status_MA), try again or reset.
eopenkernel	-1001	Cannot open kernel device; check **`ExcConfig`** settings
ekernelcantmap	-1002	Kernel driver cannot map memory
ereleventhandle	-1003	Error in kernel Release_Event_Handle
egetchintcount	-1005	Kernel error in Get_Channel_Interrupt_Count when attempting to retrieve interrupt count
egetintchannels	-1006	Kernel error in Get_Interrupt_Channels when attempting to retrieve interrupt count
ewriteiobyte	-1007	Error in kernel WriteIOByte
ereadiobyte	-1008	Error in kernel ReadIOByte
egeteventhand1	-1009	Error in kernel mGetEventHandle, first part
egeteventhand2	-1010	Error in kernel mGetEventHandle, second part
eopenscmant	-1011	Error in openSCManager in startkerneldriver
eopenservicet	-1012	Error in openservice in startkerneldriver
estartservice	-1013	Error in startservice in startkerneldriver
eopenscmanp	-1014	Error in openSCManager in stopkerneldriver
eopenservicep	-1015	Error in openservice in stopkerneldriver
econtrolservice	-1016	Error in controlservice in stopkerneldriver
eunmapmem	-1017	Error in kernel unmapmemory
egetirq	-1018	Error in Get_IRQ_Number

eallocresources	-1019	Error allocating resources. See **readme.pdf** for details on resource allocation problems.
egetramsize	-1020	Error in kernel GetRAMSize
ekernelwriteattrib	-1021	Error in kernel WriteAttributeMemory
ekernelreadattrib	-1022	Error in kernel ReadAttributeMemoryr
ekernelfrontdesk	-1023	Kernel frontdesk error
ekernelOscheck	-1024	Kernel Oscheck error
ekernelfrontdeskload	-1025	Kernel frontdeskload error
ekerneliswin2000compatible	-1026	Kernel is win2000 compatible error
ekernelbankramsize	-1027	Kernel banksize error
ekernelgetcardtype	-1028	Error in kernel GetCardType
emodum	-1029	Invalid module number specified
regnotset	-1030	Module not configured. Reboot after **ExcConfig** is run and board is in slot
ekernelbankphysaddr	-1031	Kernel GetBankPhysAddr error
ekernelclosedevice	-1032	Kernel CloseKernelDevice error
ekerneldevicenotopen	-1034	Kernel error: device not open
ekernelinitmodule	-1035	Kernel initialization error
ekernelbadparam	-1036	Kernel error: bad input parameter
ekernelbadpointer	-1037	Kernel error: invalid pointer to output buffer
ekerneltimeout	-1038	A Wait_For_Interrupt function returned with timeout
ekernelnotwin2000	-1039	Operating System is not Windows 2000
erequestnotification	-1040	Request_Interrupt_Notification error
ekernelnot4000card	-1041	Designated board is not a EXC-4000PCI
enotimersirig	-1042	Timers and IrigB not supported on this version of EXC-4000 board
eclocksource	-1059	If an invalid clock source was specified
eparmglobalreset	-1062	Illegal parameter used for `globalreset_flag` in StartTimer
etimernotrunning	-1063	Timer not running when ResetWatchdogTimer was called; did nothing
etimerrunning	-1064	Timer already running when StartTimer was called; did nothing
eparmreload	-1065	Illegal parameter used for `reload_flag` in StartTimer
eparminterrupt	-1066	Illegal parameter used for `interrupt_flag` in StartTimer
ebaddevhandle	-1067	Invalid handle specified. Use value returned by Init_Timers
edevtoomany	-1068	Init_Timers called for too many boards
einvalidOS	-1069	If an invalid operating system is used

Functions Index