# *M4K708*

## Software Tools

## Programmer's Reference


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

# Table of Contents

# 1 Introduction

Chapter 1 provides an overview for *M4K708 Software Tools* and the avionics communication hardware for the *M4K708 Weather Radar Display Databus* module on the EXC-4000 family of carrier boards.

## Getting Started

Before starting to write applications:

1. On the *Excalibur Installation CD*, included in the package you received with the hardware:
- Locate the appropriate *M4K708 Software Tools* for your hardware
- Download a copy of *M4K708 Software Tools: Programmer's Reference*
- Be sure to download a copy of your hardware's *User's Manual*.
2. Install the hardware. For hardware installation instructions, refer to the *User's Manual*.
3. Fill out the registration card and return it to your Excalibur representative.

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

### Installation

4000PCI Carrier Boards

For PCI and cPCI carrier boards, see **Appendix B: EXC-4000PCI Boards: Installation Instructions**for hardware and software installation instructions.

4000VME/VXI Carrier Boards

For VME or VXI carrier boards, see **Appendix C: EXC-4000VME Boards: Installation Instructions for PCI-MXI-2 Systems** for hardware and software installation instructions.

### Multiple Board Support

4000PCI Carrier Boards

The Excalibur Configuration Utility program (**`ExcConfig.exe`**) is a configuration setup program for PCI and cPCI boards. Use the utility program to specify which board in a machine is being used. *M4K708 Software Tools* supports the use of up to four PCI or cPCI boards simultaneously. Only if more than one board is used, is it necessary to run the utility program. If one board is used, the define value EXC_4000PCI can be used instead of the device number.

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

- **Appendix D: Multiple Board Support for EXC-4000 Carrier Boards** and
- In the **`readme.pdf`** file, on the *Excalibur Installation CD*, you received with your Excalibur board.

4000VME/VXI Carrier Boards

To configure multiple VME and VXI boards, see the section **EXC-4000VME and EXC-4000VXI Boards** on page D-2 in **Appendix D: Multiple Board Support for EXC-4000 Carrier Boards**.

### The Excalibur Systems Website

The software provided with Excalibur boards and modules is for Windows operating systems. The latest versions of the software and the *M4K708 Software Tools: Programmer's Reference* can be downloaded from our website. Check our website periodically for the latest updates: www.mil-1553.com.

In addition, information about our full range of products can be found at this site.

For further assistance, contact your Excalibur representative or write to excalibur@mil-1553.com.

## The *M4K708* Module

The *M4K708* is an interface module for the Excalibur EXC-4000 family of carrier boards. This module supports the ARINC 708/453 Airborne Weather Radar protocol.

### Overview of the Weather Radar Display Databus

The Weather Radar protocol consists of 1600-bit ARINC 708 words, which, when interpreted and plotted together, form a picture of weather patterns in the surrounding area. The various weather conditions are indicated by the colors displayed.

The picture is formed as follows: emanating from the center of a circle, a single radius line at a time is drawn, where each radius line is the data from a single ARINC 708 word.

Each ARINC 708 word is composed of 64 bits of control information, including the scan angle - the angle at which that line is to be drawn on the circle, and 512 'bins' - each one a 3-bit value specifying the color of a pixel to be drawn along that line.

Scan angle information is stored in the 12 bits between bit 52 (the 52$^{nd}$ bit received) and bit 63. Angle 0 degrees is "dead ahead" (up); angle 90 degrees is "right wing" (to the right). The angle to be used for drawing this line of data is determined by the combination of the 12 bits, which are defined as follows:

| Bit | Angle (in degrees) | Bit | Angle (in degrees) |
|---|---|---|---|
| 63 | 180 | 57 | 2.8125 |
| 62 | 90 | 56 | 1.40625 |
| 61 | 45 | 55 | 0.703125 |
| 60 | 22.5 | 54 | 0.3415625 |
| 59 | 11.25 | 53 | 0.17578125 |
| 58 | 5.625 | 52 | 0.087890625 |

**Table 1-1 Scan Angle**

The weather conditions represented by the 3-bit pixel data are:

| Pixel Value (3 bits) | Weather Condition |
|---|---|
| 0 | no precipitation |
| 1 | light precipitation |
| 2 | moderate precipitation |
| 3 | heavy precipitation |
| 4 | very heavy precipitation |
| 5 | reserved |
| 6 | medium turbulence |
| 7 | heavy turbulence |

**Table 1-2 Pixel Values**

Some radar displays do not require 512 pixels, they may use only 128 or 256 pixels. Such devices have the option of repeating the data more than once which would allow for data error detection. When less than 512 pixels are used, the actual pixels can occupy the first block of the 'bin' space or, alternatively, the actual pixels may be interleaved with the unused ones.

### M4K708 Module

The *M4K708* module contains two ARINC 708/453 channels, each software-selectable as transmit or receive.

#### Receive Mode

Each incoming ARINC 708 message is composed of 103 16-bit words comprising: two words of timetag, 100 words of ARINC 708 data, and a status word. The user sets the receive channel to receive data on the bus. When the user reads a word, the drivers will check if a complete message is available for reading and return it if it is. Each 16-bit word is checked to verify that the expected word type – time tag, data or status – was read and the entire ARINC 708 is checked for Manchester errors. Interrupts can be generated upon receipt of a specified number of ARINC 708 words, or upon error condition indicating receive data overrun.

#### Transmit Mode

Each outgoing ARINC 708 message is composed of ARINC 708 data words. The user can write up to 655 ARINC 708 words to a buffer. The drivers will inform the user if there is no space left on the module for more words, in which case the user should wait until the module empties the buffer by transmitting the data currently in the buffer. The user sets the transmit channel to send data on the bus, continuously, one-shot (only one ARINC 708 word at a time) or retransmit mode (the contents of the buffer are retransmitted over and over until stopped). Interrupts can be generated upon transmission of a specified number of ARINC 708 words, or upon error condition indicating that there is no data to transmit.

For more information about Receive and Transmit modes, see the *M4K708 Module User's Manual.*

## *M4K708 Software Tools* Functions

The *M4K708 Software Tools* is a set of 'C' language functions designed to aid users of Excalibur's *M4K708* modules to write test programs. These functions provide access to all of the *M4K708 Software Tools* functions in a structured and straightforward programming environment.

*M4K708 Software Tools* is available for Windows 9*x*/*ME,* NT, 2000 and XP. The Windows functions were written and tested using Borland C++ and Microsoft Visual C++.

### *M4K708 Software Tools* for the EXC-4000 Family of Carrier Boards

There are numerous modules available for use on the EXC-4000 carrier board. To date, Excalibur provides M4K708, M4K429/RT*x*, M4K1553P*x*, M4K1553/MCH, M4KDiscrete, M4KSerial and M4KH009 modules.

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

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

A general-purpose DLL accompanies the *M4K708* module for each of the EXC-4000 boards. Depending on the carrier board and compiler used, the names of the files are:

| Carrier board | Borland | Microsoft |
|---|---|---|
| 4000PCI | EXC4000.DLL | EXC4000MS.DLL |
| 4000VME/VXI | EXCV4000.DLL | EXCV4000MS.DLL |

The functions that operate at the 4000 carrier board level are contained in the **`EXC4000.c`** file for EXC-4000PCI carrier boards and the **`EXCv4000.c`** file for EXC-4000VME/VXI boards. The function, Get_4000Module_Type may be called for each board and module number, to check which, if any, modules are currently available at that location. (This function is also called automatically from Init_Module_708 to ascertain that the type of module is a *M4K708* module.) These DLLs are installed automatically in the Windows/System folder when the *Software Tools* are installed for modules.

The function Select_Time_Tag_Source_4000 sets the Time tag source for all the modules on the carrier board. The Time Tag source cannot be set for individual modules.

For other functions that operate at the carrier board level see the applicable carrier board's *Programmer's Reference*.

### Get_4000Module_Type for PCI boards

**Description**

Get_4000Module_Type for PCI boards checks which, if any, module is currently in the specified location. Up to four *M4K708* modules can be mounted on a 4000PCI carrier board.

**Syntax**

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

**Input Parameters**

| | |
|---|---|
| `device_num` | The define value EXC_4000PCI can be used instead of a device number. If more than one board is used, run **`ExcConfig`** to set the device number |
| `module_num` | A module position number:<br>A value 0 – 3 |

**Output Parameters**

| | |
|---|---|
| `modtype` | EXC4000_MODTYPE_708<br>If there is a 708 module |
| | EXC4000_MODTYPE_H009<br>If there is a H009 module |
| | EXC4000_MODTYPE_PX<br>If there is a P*x* module |
| | EXC4000_MODTYPE_MCH<br>If there is an MCH module |
| | EXC4000_MODTYPE_SERIAL<br>If there is a Serial module |
| | EXC4000_MODTYPE_RTX<br>If there is an RT*x* module |
| | EXC4000_MODTYPE_DIO<br>If there is a Discrete module |
| | EXC4000_MODTYPE_NONE<br>If there is no module present |

**Return Values**

| | |
|---|---|
| emodnum | If an invalid module number was specified |
| ekernelnot4000card | If the designated board is not a 4000PCI board |
| 0 | If successful |

### Get_4000Module_Type for VME/VXI boards

**Description**

Get_4000Module_Type for VME/VXI boards checks which, if any, module is in the currently specified location. Up to eight *M4K708* modules can be mounted on a VME/VXI carrier board.

**Syntax**

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

**Input Parameters**

| | |
|---|---|
| `device_num` | A device number (0–255) as set with the DIP switches.<br>See **Appendix C: EXC-4000VME Boards: Installation Instructions for PCI-MXI-2 Systems**. |
| `module_num` | A module position number:<br>A value 0 – 7 |

**Output Parameters**

| | |
|---|---|
| `modtype` | EXC4000_MODTYPE_708<br>If there is a 708 module |
| | EXC4000_MODTYPE_H009<br>If there is a H009 module |
| | EXC4000_MODTYPE_PX<br>If there is a P*x* module |
| | EXC4000_MODTYPE_MCH<br>If there is an MCH module |
| | EXC4000_MODTYPE_SERIAL<br>If there is a Serial module |
| | EXC4000_MODTYPE_RTX<br>If there is an RT*x* module |
| | EXC4000_MODTYPE_DIO<br>If there is a Discrete module |
| | EXC4000_MODTYPE_NONE<br>If there is no module present |

**Return Values**

| | |
|---|---|
| emodnum | If an invalid module number was specified |
| eopendefaultrm | If error in viOpenDefaultRM |
| eviopen | If error in viOpen |
| evimapaddress | If error in viMapAddress |
| 0 | If successful |

### Select_Time_Tag_Source_4000

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

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

**Input Parameters**

| | |
|---|---|
| `device_num` | **EXC-4000PCI/cPCI**<br>The define value EXC-4000PCI can be used instead of a device number. If more than one board is used, run ExcConfig to set the device number<br>**EXC-4000VME/VXI**<br>A device number (0–255) as set with the DIP switches.<br>See **Appendix C: EXC-4000VME Boards: Installation Instructions for PCI-MXI-2 Systems**. |
| `source` | INTERNAL_CLOCK<br>Uses the board's 4 μsec. timer<br>EXTERNAL_CLOCK<br>See "External Signals Connector" in the **Mechanical and Electrical Specifications** chapter in the applicable hardware *User's Manual* |

**Output Parameters** none

**Return Values**

| | |
|---|---|
| einval | If an invalid parameter was used as an input |
| eclocksource | If an invalid clock source was specified |
| 0 | If successful |

### Compiler Options

Programmers *must* use one of the following calling options, depending on the compiler used:

- The Borland DLL is compiled using **`_stdcall`** options
- The Microsoft DLL is compiled using **`_cdecl`** options

The driver functions in *M4K708 Software Tools* are supplied both in source form and linked as a DLL. When writing application programs, keep in mind that the module is a physical resource, and therefore you cannot run multiple copies of the program simultaneously.

Each function is presented with its formal definition, including data types of all input and output variables. A brief description of the purpose of the function is provided along with the legal values for inputs where applicable. All structures and flags used by *M4K708 Software Tools* functions are defined in **Appendix E Flags for Use with M4K708 Software Tools**.

Functions are written as 'C' functions, i.e., they return values. A negative value signifies an error. Full error messages may be printed using the Get_Error_String_708. function. (See **Appendix H: Error Messages**).

In Windows all user-defined programs must include the file **`proto_708.h`**. This file includes all the necessary header files and **`DLL`** function prototypes to operate *M4K708 Software Tools*.

### Conventions Used in the Programmer's Reference

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

Functions look like this.

Variables look like this

`Parameter` look like this.

**`File names`** look like this.

FLAGS look like this

**Note:** WORD = unsigned short int

# 2 *M4K708* Initialization Functions

Chapter 2 contains descriptions of the initialization functions necessary to write test programs for the *M4K708* 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 the inputs, where applicable.

The flags included in each function are defined in **Appendix E: Flags for Use with M4K708 Software Tools**.

The functions described in this chapter are:

Init_Module_708
Release_Module_708
GetHWRev_708
SetLoopback_708
GetModuleTimetag_708
ModuleTimetagReset_708
Get_Error_String_708

### Init_Module_708

**Description**

Init_Module_708 is the first function the user must call for each *M4K708* module on each device that is accessed in the application program.

Before exiting a program, call Release_Module_708 for each module initialized with Init_Module_708.

On **EXC-4000PCI** or **EXC-4000cPCI** carrier boards, Init_Module_708 enables the user to access up to four modules on a single board or any combination of up to 16 modules on four separate boards.

On **EXC-4000VME** or **EXC-4000VXI** carrier boards, Init_Module_708 enables the user to access up to eight modules on a single board or any combination of up to 32 modules on four separate boards.

The function may be called with the SIMULATE argument. If the SIMULATE argument is used, a portion of the memory equal to the size of the board's dual-port RAM is set aside. This area is then initialized with an id and version number for use in testing programs when no module is available.

Multiple modules may be simulated. It is possible to have real or SIMULATED modules in one application.

On **EXC-4000PCI** or **EXC-4000cPCI** carrier boards up to 17 real or SIMULATED modules may be initialized.

On **EXC-4000VME** or **EXC-4000VXI** carrier boards up to 33 or SIMULATED modules may be initialized.

More than one module may be SIMULATED simultaneously.

**Syntax**

Init_Module_708 (int `device_num`, int `module_num`)

**Input parameters**

| | | |
|---|---|---|
| | `device_num` | The device number is the index of the entry value set in `ExcConfig`: 0 - 4 |
| **or** | `SIMULATE` | If no module is present |
| | `module_num` | The module number of the *M4K708* module on the board specified by `device_num`: 0 - 4 |

**Note:**

If only one board is used, the define value EXC_4000PCI (default device number = 25) can be used instead of a device number. If more than one board is used the programmer must run the `ExcConfig` utility to set the device number.

**Output parameters**

none

**Return Values**

| | |
|---|---|
| eopenkernel | If there was an error opening a device |
| emodnum | Invalid module number specified |

**Init_Module_708 (cont.)**

| | |
|---|---|
| enomodule | If no EXC-4000 module present at specified location |
| ewrngmodule | If module specified on EXC-4000PCI board is not *M4K708* module |
| etimeoutreset | If timed out waiting for reset |
| eboardtoomany | If too many boards initialized |
| sim_no_mem | If no memory for simulation |
| enoid | If could not find a module at given device address |
| devhandle | If successful, the handle to the specified module on the board.<br>**EXC-4000CPI** and **EXC-4000cPCI**<br>A valid handle is a positive number ranging from 0–16.<br>**EXC-4000VME** and **EXC-4000VXI**<br>A valid handle is a positive number ranging from 0–32. |

## Release_Module_708

**Description**

Release_Module_708 releases any grabbed resources on a specific module.

Call the function for each module initialized with Init_Module_708, before exiting a program.

To continue accessing the module, call Init_Module_708.

**Syntax**

Release_Module_708 (int `devhandle`)

**Input Parameters**

| | |
|---|---|
| `devhandle` | The handle designated by Init_Module_708. |

**Output Parameters**

none

**Return Values**

| | |
|---|---|
| ebadhandle | If an invalid devhandle was specified; must be value returned by Init_Module_708 |
| 0 | If successful |

### GetHWRev_708

| | | |
|---|---|---|
| **Description** | GetHWRev_708 returns the hardware revision number. | |
| **Syntax** | GetHWRev_708 (int `devhandle`, int *`hwrev`) | |
| **Input Parameters** | `devhandle` | The handle designated by Init_Module_708. |
| **Output Parameters** | `hwrev` | Returns the current FPGA revision level of the module |
| **Return Values** | ebadhandle | If an invalid devhandle was specified; must be value returned by Init_Module_708 |
| | 0 | If successful |

### SetLoopback_708

| | | |
|---|---|---|
| **Description** | SetLoopback_708 sets up the module to loopback mode, where one channel on the module transmits to the other, internally, without a cable. | |
| **Syntax** | SetLoopback_708 (int `devhandle`, int `flag`) | |
| **Input parameters** | `devhandle` | The handle designated by Init_Module_708 |
| | `flag` | ENABLE<br>To set up internal loopback mode |
| | | DISABLE<br>To disable loopback mode |
| **Output parameters** | `none` | |
| **Return values** | ebadhandle | If an invalid devhandle was specified; must be value returned by Init_Module_708 |
| | einval | If an illegal value is used as an input |
| | 0 | If successful |

### GetModuleTimetag_708

| | | |
|---|---|---|
| **Description** | GetModuleTimetag_708 gets the current value of the running Time tag on the module. | |
| **Syntax** | GetModuleTimetag_708 (int_devhandle, unsigned int *timetag) | |
| **Input Parameters** | devhandle | The handle designated by Init_Module_708 |
| **Output Parameters** | timetag | 32-bit Time tag |
| **Return Values** | ebadhandle | If invalid handle specified; must be value returned by Init_Module_708. |
| | 0 | If successful |

### ModuleTimetagReset_708

| | | |
|---|---|---|
| **Description** | ModuleTimetagReset_708 sets the value of the running Time tag on the module to 0. | |
| **Syntax** | ModuleTimetagReset_708 (int_devhandle) | |
| **Input Parameters** | devhandle | The handle designated by Init_Module_708 |
| **Output Parameters** | none | |
| **Return Values** | ebadhandle | If invalid handle specified; must be value returned by Init_Module_708. |
| | 0 | If successful |

### Get_Error_String_708

**Description**

Get_Error_String_708 accepts the error of return values from other *M4K708 Software Tools* functions. This function returns the string containing a corresponding error message.

**Syntax**

Get_Error_String_708 (int `errcode`, int `errlen`, char* `errstring`)

**Example:**

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

Get_Error_String_708 (errorcode, ERRORLEN,
&Errorstr);
printf("error is: %s", ErrorStr);
```

**Input Parameters**

| | |
|---|---|
| `errcode` | The error code returned from a *Software Tools* call. |
| `errlen` | Maximum length of string to be returned - the message string that contains the corresponding error message |

**Output Parameters**

| | |
|---|---|
| `errorstring` | An array of 'errlen' characters, the message string that contains the corresponding error message. In case of bad input, this function returns a string denoting that. |

**Return Values**

| | |
|---|---|
| 0 | Always |

# 3 *M4K708* Configuration Functions

Chapter 3 contains descriptions of the configuration functions necessary to write test programs for the *M4K708* 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 the inputs, where applicable.

The flags included in each function are defined in **Appendix E: Flags for Use with M4K708 Software Tools**.

The functions described in this chapter are:



## Event Generation: Status, Interrupts and Output Triggers

Three events can occur:

- A bit is set in the Channel Status register - the status bits are read by GetStatus_708 and are cleared with ClearStatus_708. (See GetStatus_708 on page 4-2 and ClearStatus_708 on page 4-1)
- An interrupt is generated
- An output trigger is generated

### Channel Status Bit

The TXRX status bit is set for a receive channel when a specified number of ARINC 708 words is received by the channel. (See SetEventFrequency_708 on page 3-4)

The TXRX status bit is set for a transmit channel when a specified number of ARINC 708 words is received by the channel (See SetEventFrequency_708 on page 3-4)

The error status bit is set for a receive channel when receiving and the buffer is full – overrun condition.

An error status bit is set for a transmit channel when transmitting in CONTINUOUS or RETRANSMIT mode, and the buffer is empty.

### Interrupt/Trigger

An interrupt/trigger is generated when the conditions above are fulfilled, and the interrupt/trigger has been enabled for that condition, for the channel. (See SetInterrupt_708 on page 3-3 and SetTrigger_708 on page 3-4).

### SetupTransmitChannel_708

**Description** SetupTransmitChannel_708 resets the channel and sets it to transmit.

**Syntax** SetupTransmitChannel_708 (int `devhandle, int channel, int *chanhandle`)

| | | |
|---|---|---|
| **Input Parameters** | `devhandle` | The handle designated by Init_Module_708. |
| | `channel` | The channel number [0 or 1] |
| **Output Parameters** | `chanhandle` | The handle to the specified channel on the module. This handle is the first parameter in all channel specific functions. |
| **Return Values** | ebadhandle | If an invalid devhandle was specified; must be value returned by Init_Module_708 |
| | etimeoutreset | If timed out waiting for reset. |
| | einvalchan | If tried to set a channel to an illegal value |
| | 0 | If successful |

### SetupReceiveChannel_708

**Description** SetupReceiveChannel_708 resets the channel and sets it to receive.

**Syntax** SetupReceiveChannel_708 (int `devhandle, int channel, int *chanhandle`)

| | | |
|---|---|---|
| **Input Parameters** | `devhandle` | The handle designated by Init_Module_708. |
| | `channel` | The channel number [0 or 1] |
| **Output Parameters** | `chanhandle` | The handle to the specified channel on the module. This handle is the first parameter in all channel specific functions. |
| **Return Values** | ebadhandle | If an invalid devhandle was specified; must be value returned by Init_Module_708 |
| | etimeoutreset | If timed out waiting for reset. |
| | einvalchan | If tried to set a channel to an illegal value |
| | 0 | If successful |

### SetInterrupt_708

**Description** SetInterrupt_708 sets the condition or conditions under which the module is to generate interrupts for the *M4K708* module.

**Syntax** SetInterrupt_708 (int `chanhandle`, WORD `interrupt_type`)

**Input Parameters**

| | |
|---|---|
| `chanhandle` | The handle returned by SetupReceiveChannel_708 or SetupTransmitChannel_708 |
| `interrupt_type` | One or both ORed together:<br>TXRX_INTERRUPT<br>Interrupt after each time the specified number of ARINC 708 words is transmitted/received. The number is set by SetEventFrequency_708.<br>ERROR_INTERRUPT<br>Interrupt when channel is transmitting in CONTINUOUS or RETRANSMIT mode and buffer is empty or when channel is receiving and buffer is full - overrun condition<br>*or*<br>NO_INTERRUPT<br>To disable interrupt generation |

**Output Parameters** none

**Return Values**

| | |
|---|---|
| ebadchanhandle | If an invalid channel value is specified. Must be the handle returned by SetupReceiveChannel_708 or SetupTransmitChannel_708. |
| noirqset | If no interrupt allocated |
| einval | If an illegal value is used as an input |
| 0 | If successful |

### SetTransmitInterval_708

**Description** SetTransmitInterval_708 sets the interval between the beginning of one ARINC 708 word and the next ARINC 708 word in microseconds.

**Syntax** SetTransmitInterval_708 (int `chanhandle`, WORD `interval_micro`)

| | |
|---|---|
| `chanhandle` | The handle returned by SetupTransmitChannel_708 |
| `interval_micro` | 1600 – 65535: Interval value in microseconds<br>**Default** value is 5500 |

**Output Parameters** none

**Return Values**

| | |
|---|---|
| ebadchanhandle | If an invalid channel value is specified. Must be the handle returned by SetupTransmitChannel_708. |
| einval | If an illegal value is used as an input |
| 0 | If successful |

### SetTrigger_708

**Description** SetTrigger_708 sets the condition or conditions under which the module is to generate an output trigger.

**Syntax** SetTrigger_708 (int `chanhandle`, WORD `trigger_type`)

**Input Parameters**

| | |
|---|---|
| `chanhandle` | The handle returned by SetupTransmitChannel_708 or SetupReceiveChannel_708 |
| `trigger_type` | One or both ORed together:<br>TXRX_TRIGGER<br>Trigger after each time the specified number of ARINC 708 words is transmitted/received. The number is set by SetEventFrequency_708.<br>ERROR_TRIGGER<br>Trigger when channel is transmitting in CONTINUOUS or RETRANSMIT mode and buffer is empty or when channel is receiving and buffer is full - overrun condition<br>*or*<br>NO_TRIGGER<br>To disable output triggering |

**Output Parameters** none

**Return Values**

| | |
|---|---|
| ebadchanhandle | If an invalid channel value is specified. Must be the handle returned by SetupReceiveChannel_708 or SetupTransmitChannel_708. |
| einval | If an illegal value is used as an input |
| 0 | If successful |

### SetEventFrequency_708

**Description** SetEventFrequency_708 sets the desired number of ARINC 708 words to be transmitted/received either before setting the channel status bit, or generating an interrupt, or an output trigger.

**Syntax** SetEventFrequency_708 (int `chanhandle`, WORD `frequency`)

**Input Parameters**

| | |
|---|---|
| `chanhandle` | The handle returned by SetupReceiveChannel_708 or SetupTransmitChannel_708 |
| `frequency` | The frequency at which to generate a status bit or interrupt: **Default** value is 1<br>**Module rev B**: Maximum values<br>Transmit 655<br>Receive 636<br>**Module rev A**: values 1 – 80 |

**Output Parameters** none

**Return Values**

| | |
|---|---|
| ebadchanhandle | If an invalid channel value is specified. Must be the handle returned by SetupReceiveChannel_708 or SetupTransmitChannel_708. |
| einval | If an illegal value is used as an input |
| 0 | If successful |

# 4 *M4K708* Communication Functions

Chapter 4 contains descriptions of the receive and transmit communication functions necessary to write test programs for the *M4K708* 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 the inputs, where applicable.

The **communication** functions are:

| | |
|---|---|
| ClearStatus_708 | StartReceive_708 |
| GetStatus_708 | StartTransmit_708 |
| NumberWordsInBuffer_708 | Stop_708 |
| ReadWord_708 | WriteWord_708 |

The **ARINC 708 utility** functions are:

| | |
|---|---|
| GetData_708 | SetControlData_708 |
| SetData_708 | GetPixel_708 |
| GetControlData_708 | SetPixel_708 |

See **Appendix E: Flags for Use with M4K708 Software Tools** for typedef of control data structure.

### ClearStatus_708

| | | |
|---|---|---|
| **Description** | Call ClearStatus_708 to clear selected bits of the channel status/ interrupt status for the specified channel. | |
| **Syntax** | ClearStatus_708 (int `chanhandle`, WORD `clearflag`) | |
| **Input Parameters** | `chanhandle` | The handle returned by SetupTransmitChannel_708 or SetupReceiveChannel_708. |
| | `clearflag` | One *or* both ORed together:<br>TXRX_STATUS<br>Clears the bits indicating ARINC 708 words transmitted/received<br>ERROR_STATUS<br>Clears bits indicating error condition<br>*or*<br>ALL_STATUS<br>Clear all status bits |
| **Output Parameters** | none | |
| **Return Values** | ebadchanhandle | If an invalid channel value is specified. Must be the handle returned by SetupTransmitChannel_708 or SetupReceiveChannel_708. |
| | einval | If an illegal value is used as an input |
| | 0 | If successful |

### GetStatus_708

| | | |
|---|---|---|
| **Description** | Call GetStatus_708 to read the channel status for the specified channel. This value can be used to ascertain for which channel an interrupt or trigger was generated, and the type of event – TXRX or ERROR. | |
| **Syntax** | GetStatus_708 (int `chanhandle`, int `statusflag`) | |
| **Input Parameters** | `chanhandle` | The handle returned by SetupReceiveChannel_708 or SetupTransmitChannel_708 |
| **Output Parameters** | `statusflag` | One *or* both ORed together<br>TXRX_STATUS<br>The number of ARINC 708 words set in SetEventFrequency_708 was transmitted/received<br>ERROR_STATUS<br>Error occurred on channel<br>*or*<br>NO_STATUS<br>No channel status set |
| **Return Values** | ebadchanhandle | If an invalid channel value is specified. Must be the handle returned by SetupReceiveChannel_708 or SetupTransmitChannel_708. |
| | 0 | If successful |

### NumberWordsInBuffer_708

| | | |
|---|---|---|
| **Description** | NumberWordsInBuffer_708 returns the number of words currently in the buffer for the specified channel. | |
| **Syntax** | NumberWordsInBuffer_708 (int `chanhandle`, int *`numwords`) | |
| **Input Parameters** | `chanhandle` | The handle returned by SetupReceiveChannel_708. |
| **Output Parameters** | `numwords` | Number of words currently in the buffer returned for the specific channel.<br>Values: 0 – 65536 |
| **Return Values** | ebadhandle | If an invalid handle was specified; should be value returned by Init_Module_708 |
| | ebadchanhandle | If an invalid channel value is specified. Must be the handle returned by SetupReceiveChannel_708. |
| | 0 | If successful |

### ReadWord_708

**Description** ReadWord_708 reads one ARINC 708 word from the buffer for the specified receive channel.

**Syntax** ReadWord_708 (int `chanhandle`, WORD *`wordarray`, DWORD *`timetag`, WORD *`wordstatus`)

**Input Parameters**

| | |
|---|---|
| `chanhandle` | The handle returned by SetupReceiveChannel_708. |

**Output Parameters**

| | |
|---|---|
| `wordarray` | Pointer to 100 WORDs containing one ARINC 708 word |
| `timetag` | A 32-bit Timetag |
| `wordstatus` | INVALID_708WORD<br>Manchester error |
| | VALID_708WORD<br>Valid 708 word |

**Return Values**

| | |
|---|---|
| ebadchanhandle | If an invalid channel value is specified. Must be the handle returned by SetupReceiveChannel_708. |
| echantype | If channel type is not receive |
| ebadformat | If no proper ARINC 708 word found in receive buffer |
| enorcvword | If no ARINC 708 word was received |
| 0 | If successful |

### StartReceive_708

| | | |
|---|---|---|
| **Description** | Call StartReceive_708 to start receiving data in the buffer for the specified channel. | |
| **Syntax** | StartReceive_708 (int `chanhandle`) | |
| **Input Parameters** | `chanhandle` | The handle returned by SetupReceiveChannel_708 |
| **Output Parameters** | none | |
| **Return Values** | ebadchanhandle | If handle other than the one returned by SetupReceiveChannel_708 is used. |
| | echantype | If channel type is not receive |
| | 0 | If successful |

### StartTransmit_708

| | | |
|---|---|---|
| **Description** | Call StartTransmit_708 to start transmitting data in the buffer for the specified transmit channel. | |
| **Syntax** | StartTransmit_708 (int `chanhandle,` int `duration`) | |
| **Input Parameters** | `chanhandle` | The handle returned by SetupTransmitChannel_708 |
| | `duration` | DUR_ONESHOT<br>To send out one ARINC 708 word |
| | | DUR_CONTINUOUS<br>To continuously transmit until transmit buffer is empty - buffer can be written to continuously |
| | | DUR_RETRANSMIT<br>To transmit contents of a buffer again and again until stopped - data *cannot* be written to the buffer |
| **Output Parameters** | none | |
| **Return Values** | ebadchanhandle | If handle other than the one returned by SetupTransmitChannel_708 is used. |
| | echantype | If channel type is not transmit |
| | einval | If an invalid value was used as an input |
| | erunning | If channel already running |
| | 0 | If successful |

### Stop_708

**Description** Call Stop_708 to stop transmitting/receiving data in the buffer for the specified channel.

**Syntax** Stop_708 (int `chanhandle`)

**Input Parameters**

| | |
|---|---|
| `chanhandle` | The handle returned by SetupReceiveChannel_708 or SetupTransmitChannel_708 |

**Output Parameters** none

**Return Values**

| | |
|---|---|
| ebadchanhandle | If handle other than the one returned by SetupReceiveChannel_708 or SetupTransmitChannel_708 is used. |
| 0 | If successful |

### WriteWord_708

**Description** WriteWord_708 writes one ARINC 708 word to the buffer for the specified channel.

**Syntax** WriteWord_708 (int `chanhandle`, WORD *`wordarray`)

**Input Parameters**

| | |
|---|---|
| `chanhandle` | The handle returned by SetupTransmitChannel_708 |
| `wordarray` | Pointer to 100 WORDs containing one ARINC 708 word |

**Output Parameters** `none`

**Return Values**

| | |
|---|---|
| ebadchanhandle | If handle other than the one returned by SetupTransmitChannel_708 is used. |
| echantype | If function invalid for this channel type |
| eoneshot | Cannot write to buffer when channel running in ONESHOT mode |
| eretransmit | Cannot write to buffer when channel running in RETRANSMIT mode |
| enoxmtword | If no room to transmit ARINC 708 word |
| 0 | If successful |

## ARINC 708 WORD Utility Functions

See **Appendix E: Flags for Use with M4K708 Software Tools** for typedef of control data structure.

### GetData_708

| | | |
|---|---|---|
| **Description** | GetData_708 reads control data and all pixel information from ARINC 708 word into structures | |
| **Syntax** | GetData_708 (WORD *wordarray, t_controlData *controlData, WORD *pixelarray) | |
| **Input Parameters** | wordarray | Array containing one ARINC 708 word |
| **Output Parameters** | controlData | Structure containing control data |
| | pixelarray | Array of 512 WORDs, one for each pixel's data |
| **Return Values** | 0 | Always |

### SetData_708

| | | |
|---|---|---|
| **Description** | SetData_708 writes control data and all pixel information from structures into an ARINC 708 word. | |
| **Syntax** | SetData_708 (t_controlData *controlData, WORD *pixelarray, WORD *wordarray) | |
| **Input Parameters** | controlData | Structure containing control data |
| | pixelarray | Array of 512 WORDs, one for each pixel's data |
| **Output Parameters** | wordarray | Array containing one ARINC 708 word |
| **Return Values** | 0 | Always |

### GetPixel_708

| | | |
|---|---|---|
| **Description** | GetPixel_708 reads a specific pixel from the data portion of the ARINC 708 word. | |
| **Syntax** | GetPixel_708 (WORD *wordarray, int pixnum, WORD *pixel) | |
| **Input Parameters** | wordarray | Array of 100 WORDs containing one ARINC 708 word |
| | pixnum | Pixel number: a value 1 – 512 |
| **Output Parameters** | pixel | One WORD. Pixel data is in the lowest 3 bits |
| **Return Values** | einval | If an illegal value is used as an input |
| | 0 | If successful |

### SetPixel_708

| | | |
|---|---|---|
| **Description** | SetPixel_708 writes a specific pixel to the data portion of the ARINC 708 word. | |
| **Syntax** | SetPixel_708 (WORD *`wordarray`, int `pixnum`, WORD *`pixel`) | |
| **Input Parameters** | `wordarray` | Array of 100 WORDs containing one ARINC 708 word |
| | `pixnum` | Pixel number: a value 1 – 512 |
| | `pixel` | One WORD. Pixel data is in the lowest 3 bits |
| **Output Parameters** | `wordarray` | The updated array of 100 WORDs containing one ARINC 708 word |
| **Return Values** | einval | If an invalid value was used as an input |
| | 0 | If successful |

### GetControlData_708

| | | |
|---|---|---|
| **Description** | Call GetControlData_708 to read the control data portion from the ARINC 708 word array into a structure. | |
| **Syntax** | GetControlData_708 (WORD `*wordarray`, t_ControlData `*controlData`) | |
| **Input Parameters** | `wordarray` | Array of 100 words containing one ARINC 708 word |
| **Output Parameters** | `controlData` | Structure containing control data |
| **Return Values** | 0 | Always |

### SetControlData_708

| | | |
|---|---|---|
| **Description** | Call SetControlData_708 to add control data information into a ARINC 708 word. | |
| **Syntax** | SetControlData_708 (t_controlStruct `controlStruct`, WORD `*wordarray`) | |
| **Input Parameters** | `wordarray` | Array of 100 words containing one ARINC 708 word |
| | `controlStruct` | Structure containing various parts of the Control data |
| **Output Parameters** | `wordarray` | The updated array of 100 words containing one ARINC 708 word. |
| **Return Values** | 0 | Always |

# 5 Using Interrupt Functions

When writing a Windows program that processes interrupts, a separate thread is generally created to handle the interrupt processing. This thread calls Wait_for_Interrupt_708, in order to wait for the next interrupt. When the function returns, the interrupt is processed as needed. This method is demonstrated in the test program **`demo_708_int.c`** which is included with the *M4K708 Software Tools.*

**Note:** There is no need to reset the physical interrupt line in the interrupt thread; this is handled internally.

In cases of very high interrupt frequency, several interrupts may occur before the interrupt thread resumes execution. The Get_Interrupt_Count_708 function may be used to determine if multiple interrupts have occurred. Conversely, it is possible that the Wait_for_Interrupt_708 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_708 function and the call to Get_Interrupt_Count_708.) Once again, the Get_Interrupt_Count_708 function may be used to determine if the interrupt has already been processed.

The following functions are described in this chapter:

Get_Interrupt_Count_708

InitializeInterrupt_708

Wait_for_Interrupt_708

Wait_for Multiple_Interrupts_708

The flags included in each function are defined in **Appendix E: Flags for Use with M4K708 Software Tools**.

### Get_Interrupt_Count_708

**Description**

Get_Interrupt_Count_708 returns the total interrupt count for the specified module from the time the module was initialized with Init_Module_708.

**Syntax**

Get_Interrupt_Count_708 (int `devhandle`, unsigned long `*pdwInterruptCount`)

**Input Parameters**

| | |
|---|---|
| `devhandle` | The handle designated by Init_Module_708 |

**Output Parameters**

| | |
|---|---|
| `pdwInterruptCount` | Pointer to an unsigned long which receives the interrrupt count |

**Return Values**

| | |
|---|---|
| ebadhandle | If an invalid handle was specified; should be value returned by Init_Module_708 |
| egetintcount | If there was a kernel error |
| ekernelinitmodule | If error initializing kernel related data |
| ekernelbadparameter | If input parameter is invalid |
| ekernelbadpointer | If output parameter buffer is invalid |
| ekerneldevicenotopen | If specified device has not been opened |
| 0 | If successful |

### InitializeInterrupt_708

**Description**

InitializeInterrupt_708 may be called to initialize interrupt handling for the given module/card. In general, it is not necessary to call this function since the Wait_for_Interrupt_708 function initializes the interrupt handling automatically when it is first called. However, in certain situations, such as a program in which the Wait_for_Interrupt_708 function is not run in a separate thread but is executed sequentially in the main thread, one may wish to initialize the interrupt handling before calling Wait_for_Interrupt_708. Thus, if an interrupt occurs after the initialization but before the call to Wait_for_Interrupt_708, the latter call will return immediately, reporting the interrupt which occurred previously.

**Syntax**

InitializationInterrupt_708 (int `devhandle`)

**Input Parameters**

| | |
|---|---|
| `devhandle` | The handle designated by Init_Module_708 |

**Output Parameters**

none

**Return Values**

| | |
|---|---|
| ebadhandle | If an invalid handle is specified; should be value returned by Init_Module_708 |
| egetevenhandle1 | If there is an error in kernel function Get_Event_Handle, first part |
| egetevenhandle2 | If there is an error in kernel function Get_Event_Handle, second part |
| ekernelinitmodule | If error initializing kernel related data |
| ekernelbadparam | If input parameter is invalid |
| ekerneldevicenotopen | If specified device was not opened |
| 0 | If successful |

### Wait_for_Interrupt_708

**Description**

Wait_For_Interrupt_708 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_708 (int `devhandle,` unsigned int `timeout)`

**Example**

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

```
DWORD InterruptThread(int referenceParam)
{
  while (1)
  {
    int status;
    status = Wait_For_Interrupt_708(devhandle, INFINTE);
    if (status < 0)
    {
      // We don't check for ekerneltimeout since we passed
      // in a timeout value of INFINITE.
      // All other return values indicate error.
      // Process error…
       ExitThread(1);
    }
     // Process interrupt…
     // Check total number of interrupts
    Get_Interrupt_Count_708(devhandle, &numints);
  }
}
```

**Input Parameters**

| | |
|---|---|
| `devhandle` | The handle designated by Init_Module_708 |
| `timeout` | Timeout is specified in milliseconds, or INFINITE |

**Output Parameters**

none

**Return Values**

| | |
|---|---|
| ebadhandle | If an invalid handle is specified; should be value returned by Init_Module_708 |
| egetevenhandle1 | If there is an error in kernel function Get_Event_Handle, first part |
| egetevenhandle2 | If there is an error in kernel function Get_Event_Handle, second part |
| ekernelinitmodule | If error initializing kernel related data |
| ekernelbadparam | If input parameter is invalid |
| ekerneldevicenotopen | If specified device was not opened |
| | Successful if *either*: |
| ekerneltimeout | The wait timed out without receiving an interrupt *or* |
| 0 | If successful |

### Wait_for Multiple_Interrupts_708

**Description**

Wait_for_Multiple_Interrupts_708 waits for an interrupt on any of the specified modules. It suspends control of the calling thread while waiting, and returns control to the thread either upon receipt of the interrupt, or upon expiration of the time out. If `timeout` is set to INFINITE, then the call will return only upon receipt of the interrupt.

**Syntax**

Wait_for_Multiple_Interrupts_708 (int `nummodules`, int `*devhandle_array`, unsigned int `timeout`, unsigned long `*pdw_Interrupt_Bitfield`)

**Input Parameters**

| | |
|---|---|
| `nummodules` | Number of modules in the `devhandle_array` |
| `devhandle_array` | An array of module handles |
| `timeout` | Timeout is specified in milliseconds, or INFINITE |

**Output Parameters**

| | |
|---|---|
| `pdw_Interrupt_ Bitfield` | Pointer to an unsigned long which receives a bit field indicating which of the modules have interrupted (note that more than one module may have interrupted simultaneously). The modules are distributed in the bit field such that the lowest bit corresponds to the first module in the `devhandle_array`, and so on. |

**Return Values**

| | |
|---|---|
| egeteventhand1 | If there is an error in kernel function Get_Event_Handle, first part |
| egeteventhand2 | If there is an error in kernel function Get_Event_Handle, second part |
| ebadhandle | If an invalid handle specified in the `devhandle_array`; should be values returned by Init_Module_708. |
| ekernelinitmodule | If error initializing kernel related data |
| ekernelbadparam | If input parameter is invalid |
| ekerneldevicenotopen | If the specified device was not opened |
| ekernelbadpointer | If output parameter buffer is invalid |
| | Successful if *either*: |
| ekerneltimeout | The wait timed out without receiving an interrupt<br>*or* |
| 0 | If successful |

# Appendix A ARINC 708 Specifications

### Display Data Bus Format

| Bits | Data Display | Function details | On page |
|---|---|---|---|
| **01– 08** | Label | | |
| **09 – 10** | Control/Accept | **Table A-2 Control Accept Functions** | A-2 |
| **11** | Slave | 1 = Slave<br>0 = Master (Normal) | |
| **12 – 13** | Spare | | |
| **14** | Turbulence Alert | Automatic sensing of a Turbulence alert has occurred | |
| **15** | Weather Alert | Automatic sensing of a reflectivity Weather alert has occurred | |
| **16** | AntiClutter | Clutter elimination circuitry is in operation | |
| **17** | Sector Scan | Reduced Sector scan is in operation | |
| **18** | Stability Limits | Aircraft attitude and /or tilt control exceeds the system's design limits. | |
| **19** | Cooling Fault | | |
| **20** | Display Fault | | |
| **21** | Calibration (T-R) Fault | | |
| **22** | Altitude input Fault | | |
| **23** | Control Fault | | |
| **24** | Antenna Fault | | |
| **25** | Transmitter/ receiver Fault | | |
| **26** | Stabilization On | | |
| **27 – 29** | Mode | **Table A-3 Operating mode** | A-2 |
| **30 – 36** | Tilt | **Table A-4 Tilt data** | A-2 |
| **37 – 42** | Gain | **Table A-5 Gain data** | A-3 |
| **43 – 48** | Range | **Table A-6 Range Data** | A-3 |
| **49** | Spare | | |
| **50 – 51** | Data Accept | **Table A-7 Data accept** | A-3 |
| **52 – 63** | Scan Angle | **Table A-8 Scan angle** | A-3 |
| **64** | Spare | | |
| **65 – 67** | Bin 1 | | |
| • | • | | |
| • | • | **Table A-9 Weather Condition / Reflectivity Data** | A-4 |
| • | • | | |
| **1598 – 1600** | Bin 512 | | |

**Table A-1 Display Data Bus Format**

**Note:** The lowest order bit is referred to as bit 1.

### Control Accept Function

| Matrix Code | Bit 10 | Bit 9 | Function |
|---|---|---|---|
| **0** | 0 | 0 | Do not accept control |
| **1** | 0 | 1 | IND 1 accept control |
| **2** | 1 | 0 | IND 2 accept control |
| **3** | 1 | 1 | All INDs accept control |

**Table A-2 Control Accept Functions**

### Operating Mode

| Matrix Code | Bit 29 | Bit 28 | Bit 27 | Operating Mode |
|---|---|---|---|---|
| **0** | 0 | 0 | 0 | Standby |
| **1** | 0 | 0 | 1 | Weather [only] |
| **2** | 0 | 1 | 0 | Map |
| **3** | 0 | 1 | 1 | Contour |
| **4** | 1 | 0 | 0 | Test |
| **5** | 1 | 0 | 1 | Turbulence [only] |
| **6** | 1 | 1 | 0 | Weather & turbulence |
| **7** | 1 | 1 | 1 | Reserved [Calibration annunciation] |

**Table A-3 Operating mode**

**Note:** Weather (only) Reflectivity (Weather) only data should be transmitted on all azimuth addresses.

Turbulence (only) Turbulence only data should be transmitted on all azimuth addresses.

Weather & turbulence Turbulence data combined with reflectivity data is allowed as a means to transmit both weather only and weather plus turbulence words when Weather & turbulence mode is selected.

### Tilt Data

| Bit | Tilt in degrees |
|---|---|
| **36** | –16 |
| **35** | +8 |
| **34** | +4 |
| **33** | +2 |
| **32** | +1 |
| **31** | +0.5 |
| **30** | +0.25 |

**Table A-4 Tilt data**

**Note: TWO's** complement tilt

### Gain Data

| Bit | 42 | 41 | 40 | 39 | 38 | 37 | |
|---|---|---|---|---|---|---|---|
| | 1 | 1 | 1 | 1 | 1 | 1 | Cal |
| | 0 | 0 | 0 | 0 | 0 | 0 | Max |
| | 0 | 0 | 0 | 1 | 0 | 1 | -5 |
| | 0 | 0 | 1 | 0 | 1 | 1 | -11 |
| | 1 | 1 | 1 | 1 | 1 | 0 | -62 |

**Table A-5 Gain data**

### Range Data

| Bit | 48 | 47 | 46 | 45 | 44 | 43 | |
|---|---|---|---|---|---|---|---|
| | 0 | 0 | 0 | 0 | 0 | 1 | 5 NM |
| | 0 | 0 | 0 | 0 | 1 | 0 | 10 |
| | 0 | 0 | 0 | 1 | 0 | 0 | 20 |
| | 0 | 0 | 1 | 0 | 0 | 0 | 40 |
| | 0 | 1 | 0 | 0 | 0 | 0 | 80 |
| | 1 | 0 | 0 | 0 | 0 | 0 | 160 |
| | 1 | 1 | 1 | 1 | 1 | 1 | 315 |
| | 0 | 0 | 0 | 0 | 0 | 0 | 320 |

**Table A-6 Range Data**

### Data Accept

| Bit | 51 | 50 | Function |
|---|---|---|---|
| | 0 | 0 | Do not accept data |
| | 0 | 1 | Accept data 1 |
| | 1 | 0 | Accept data 2 |
| | 1 | 1 | Accept any data |

**Table A-7 Data accept**

### Scan Angle

| Bit | Angle (in degrees) | Bit | Angle (in degrees) |
|---|---|---|---|
| 63 | 180 | 57 | 2.8125 |
| 62 | 90 | 56 | 1.40625 |
| 61 | 45 | 55 | 0.703125 |
| 60 | 22.5 | 54 | 0.3415625 |
| 59 | 11.25 | 53 | 0.17578125 |
| 58 | 5.625 | 52 | 0.087890625 |

**Table A-8 Scan angle**

### Weather Conditions/Reflectivity Data

| Matrix code [Pixel Value 3 bits] | Weather Condition | Bin 1 Bit 67 | Bit 66 | Bit 65 |
|---|---|---|---|---|
| 0 | No precipitation [< Z2] | 0 | 0 | 0 |
| 1 | Light precipitation [Z2 - Z3] | 0 | 0 | 1 |
| 2 | Moderate precipitation [Z3 - Z4] | 0 | 1 | 0 |
| 3 | Heavy precipitation [Z4 to Z5] | 0 | 1 | 1 |
| 4 | Very heavy precipitation [> Z5] | 1 | 0 | 0 |
| 5 | Reserved[1] | 1 | 0 | 1 |
| 6 | Medium turbulence | 1 | 1 | 0 |
| 7 | Heavy turbulence | 1 | 1 | 1 |

**Table A-9 Weather Condition / Reflectivity Data**

1. Out of Calibration Indication

# Appendix B EXC-4000PCI Boards: Installation Instructions

Appendix B explains the procedure for installing *M4K708* for the:

- *M4K708* module on the EXC-4000PCI
- *M4K708* module on the EXC-c4000PCI

*M4K708 Software Tools* is for several different operating systems. Appendix B explains the procedure for installing it on the following systems.

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

**Warning:** *Whenever you handle an EXC-4000PCI 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*
- *M4K708 Software Tools*

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

1. Insert the board into an available PCI slot in your computer
2. Add *M4K708 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-4000PCI 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 hardware *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-4000PCI 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-4000PCI 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 *M4K708 Software Tools* to Windows 9*x/ME* 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_708 function.

**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.

To run the test programs in *M4K708 Software Tools,* the EXC-4000PCI must be linked up to a loopback cable. For details refer to the section *Mechanical and Electrical Specification* in the *M4K708 Module User's Manual.*

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

### Windows NT4 Procedures

The *Excalibur Installation CD* contains the files for:

- *M4K708 Software Tools*

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

1. Insert the board into an available PCI slot in the computer.
2. Add *M4K708 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-4000PCI board

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

For more information, see the section "Installation" in the hardware *[Product Name]: User's Manual.*

#### To add *M4K708 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_708 function.

5. Reboot the computer at the end of the installation procedure.

The EXC-4000PCI 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.

To run the test programs in *M4K708 Software Tools,* the EXC-4000PCI must be linked up to a loopback cable. For details refer to the section *Mechanical and Electrical Specification* in the *M4K708 Module User's Manual.*

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*
- *M4K708 Software Tools*

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

1. Insert the board into an available PCI slot in your computer
2. Add *M4K708 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-4000PCI 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 hardware *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-4000PCI 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 *M4K708 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_708 function.

The EXC-4000PCI 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.

To run the test programs in *M4K708 Software Tools,* the EXC-4000PCI must be linked up to a loopback cable. For details refer to the section *Mechanical and Electrical Specification* in the *M4K708 Module User's Manual.*

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

# Appendix C EXC-4000VME Boards: Installation Instructions for PCI-MXI-2 Systems

Appendix C explains the procedure for installing the *M4K708 Software Tools* for the Excalibur EXC-4000VME and EXC-4000VXI boards under a Windows operating system.

Installation of the EXC-4000VME board on a Windows system is accomplished by determining the board's logical address, setting the DIP switch on the board, and installing *M4K708 Software Tools*. To verify board installation, run the test-programs that are included on the *Excalibur Installation CD*.

All the files needed to run *M4K708* on Windows operating systems are on the *Excalibur Installation CD*.

**Note:** *M4K708 Software Tools* for VME boards were written for VISA standard.

**To install the EXC-4000VME board:**

1. Ensure the National Instruments PCI-MXI-2 system is installed correctly.
2. Before installing the Excalibur board, it is very important to determine the board's logical address. The board requires a 1MB area of memory. Choose a logical address that does not conflict with any other devices in your system. The board is set at default address 80H (128 Dec.).

*Warning:* If a logical address is already in use and it is also used for the EXC-4000VME, the board will not function properly.

3. Set the Board Logical Address DIP switch (SW1) according to the logical address chosen. See the section on DIP Switch Settings in the EXC-4000VME hardware *User's Manual*.
4. Turn on your VME and PC computers. Run the two programs that establish the connection to the VME (VXIINIT, the hardware initialization program, and Resman, the resources manager program).

**Note:** Later versions of the National Instruments PCI-MXI-2 may not require running VXIINIT.

# Appendix D Multiple Board Support for EXC-4000 Carrier Boards

### EXC-4000PCI and EXC-4000cPCI Carrier Boards

*M4K708 Software Tools* supports the use of up to four PCI and cPCI 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 Utility 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 |

### EXC-4000VME and EXC-4000VXI Boards

*M4K708 Software Tools* supports the simultaneous use of up to eight *M4K708* modules on a single VME or VXI board. The number of boards supported depends on the user's system.

A unique Logical Address must be set for each board. Use DIP switch SW1 to set the board's Logical Address.

#### Examples of Logical Address Switch (SW1) Settings:

| | | MSB | | | **Logical Address Switch (SW1** | | | | LSB |
|---|---|---|---|---|---|---|---|---|---|
| '1' | '1' | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
| A15 | A14 | A13 | A12 | A11 | A10 | A9 | A8 | A7 | A6 |

| **Logical Address** | | **Switch Settings** | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **Hex** | **Dec** | | | | | | | | |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 20 | 32 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 80 | 128 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 81 | 129 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| C0 | 192 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| FF | 255 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

**Example:** For a logical address of C0 (H) [=A16 Address F000 (H)], set position 1 and 2 to OFF or Open and all other switches to ON or Closed.

Switch ON or Closed = logic 0 at switch position
Switch OFF or Open = logic 1 at switch position

**Note:** 1. Numbers indicate switch positions.
2. Address lines A15 and A14 are always decoded as '1'.
3. Address lines A5–A0 are always decoded as '0'.

# Appendix E Flags for Use with *M4K708 Software Tools*

Flags are grouped according to the functions in which they are used. Some flags are used in more than one function, and they are duplicated in each section for clarity. Most flags are input parameters. Others are listed for the convenience of the programmer.

**Note:** Always use flags where provided, rather than the value associated with it, since values may change. For example, use SIMULATE with Init_Module rather than 0xFFFF.

The flags are grouped by function.

**Init_Module**

| | | |
|---|---|---|
| SIMULATE | 0xFFFF | Test drivers without a module present |
| EXC_4000PCI | | If only one board is used, the define value EXC_4000PCI can be used instead of a device number |

**Create/ParseControlData_708**

```
typedef struct
{
        WORD    label; //8 bits
        WORD    controlAccept; //2
        BOOL    slave;
        BOOL    turbulenceAlert;
        BOOL    weatherAlert;
        BOOL    antiClutter;
        BOOL    sectorScan;
        BOOL    stabilityLimits;
        BOOL    coolingFault
        BOOL    displayFault;
        BOOL    calibrationFault;
        BOOL    attitudeFault;
        BOOL    controlFault;
        BOOL    antennaFault;
        BOOL    txRcvFault;
        BOOL    stabilization;
        WORD    operatingMode; //3
        WORD    tilt; //7
        WORD    gain; //6
        WORD    range; //6
        WORD    dataAccept; //2
        WORD    scanAngle; //12
}t_controlStruct;
```

**StartTransmit_708**

| | | |
|---|---|---|
| DUR_ONESHOT | 0 | To send out one ARINC 708 word |
| DUR_CONTINUOUS | 1 | To continuously transmit until buffer empty |
| DUR_RETRANSMIT | 2 | To transmit contents of buffer again and again until stopped - data *cannot* be written to the buffer |

**GetStatus_708**

| | | |
|---|---|---|
| ERROR_STATUS | 1 | Error occurred on channel |
| TXRX_STATUS | 2 | The number of ARINC 708 words set in SetEventFrequency_708 was transmitted/received |
| NO_STATUS | 0 | No channel status bits set |

**ClearStatus_708**

| | | |
|---|---|---|
| ERROR_STATUS | 1 | Clear error status bit |
| TXRX_STATUS | 2 | Clear TXRX status bit |
| ALL_STATUS | 3 | Clear all status bits |

**SetInterrupt_708**

| | | |
|---|---|---|
| ERROR_INTERRUPT | 1 | Interrupt when channel is transmitting in CONTINUOUS or RETRANSMIT mode and buffer is empty or when channel is receiving and buffer is full - overrun condition |
| TXRX_INTERRUPT | 2 | Interrupt after each time the specified number of ARINC 708 words is transmitted/received. The number is set by SetEventFrequency_708. |
| NO_INTERRUPT | 0 | Disable Interrupts |

**SetTrigger_708**

| | | |
|---|---|---|
| ERROR_TRIGGER | 1 | Trigger when channel is transmitting in CONTINUOUS or RETRANSMIT mode and buffer is empty or when channel is receiving and buffer is full - overrun condition |
| TXRX_TRIGGER | 2 | Trigger after each time the specified number of ARINC 708 words is transmitted/received. The number is set by SetEventFrequency_708. |
| NO_TRIGGER | 0 | Disable output triggering |

**SetLoopback_708**

| | | |
|---|---|---|
| DISABLE | 0 | Disable loopback mode |
| ENABLE | 1 | Set up internal loopback mode |

**ReadWord_708**

| | | |
|---|---|---|
| INVALID_708WORD | 0 | Manchester error |
| VALID_708WORD | 1 | Valid ARINC 708 word |

**Module types for EXC-4000 carrier boards**

| | | |
|---|---|---|
| EXC4000_MODTYPE_SERIAL | 2 | A Serial module is present |
| EXC4000_MODTYPE_MCH | 3 | A MCH module is present |
| EXC4000_MODTYPE_RTX | 4 | An RTx module is present |
| EXC4000_MODTYPE_PX | 5 | A Px module is present |
| EXC4000_MODTYPE_MMSI | 6 | A MMSI module is present |
| EXC4000_MODTYPE_708 | 7 | A 708 module is present |
| EXC4000_MODTYPE_MA | 8 | A MicroAce module is present |
| EXC4000_MODTYPE_H009 | 9 | A H009 module is present |
| EXC4000_MODTYPE_DIO | 0xd | A Discrete module is present |
| EXC4000_MODTYPE_NONE | 0x1F | No module present |

# Appendix F *M4K708 Software Tools* Library

Appendix F includes a list of the files in the Excalibur *M4K708 Software Tools* needed to write user-defined applications. The files are divided into three categories:

**Source code and Header files** for the *M4K708 Software Tools* functions. Header files should be included in application programs as needed.

| File Extension | Description |
|---|---|
| ***.c** | source code |
| ***.h** | header file |

**DLL and associated *.lib files**

| File Extension | Description |
|---|---|
| ***.dll** | Borland compiler DLL |
| ***MS.dll** | Microsoft compiler DLL |
| ***.lib** | Index file used to create applications using Borland DLL functions |
| ***MS.dll** | Index file used to create applications using Microsoft DLL functions |

**Demo Programs** are examples of programs using *M4K708 Software Tools.* They can be used as a basis for user-defined programs. Demo programs include the following types of files.

| File Extension | Description |
|---|---|
| ***.c, *.h** | Demo source code |
| ***.ide** | Borland demo project files |
| ***.exe** | Borland demo executable files |
| ***.dsp**<br>***.dsw** | Microsoft demo project files |
| ***MS.exe** | Microsoft demo executable files |

| | File Name | Description |
|---|---|---|
| **Source Code Files** | **config_708** | Functions for configuring *M4K708* channels |
| | **comm_708** | Functions for communication over a *M4K708* module |
| | **deviceio_708** | Functions for interaction with kernel driver for Windows operating systems |
| | **error_708** | function for returning error messages |
| | **EXC4000** | For*M4K708* modules on PCI carrier boards - functions for extracting information associated with the carrier board. |
| | **EXCv4000** | For *M4K708* modules on VME/XVI carrier boards - functions for extracting information associated with the carrier board. |
| | **init_708** | Initialization and Release module functions |
| **Source Header files** | **config_708** | Header file for configuration functions |
| | **comm_708** | Header file for communication functions |
| | **deviceio_708** | Header file for interaction with kernel driver |
| | **error_devio** | Header file containing error codes for the Excalibur kernel drivers |
| | **error_708** | Header file containing error message codes |
| | **EXC4000** | For *M4K708* on PCI carrier boards - Header file containing prototypes for functions associated with the carrier board |
| | **EXCv4000** | For *M4K708* on VME/VXI carrier boards - Header file containing prototypes for functions associated with the carrier board. |
| | **Excsysio** | Header file which defines control codes in kernel drivers and defines strings for module names for use in Init_Module_708 calls to kernel drivers |
| | **Flags_708** | Header file containing flags for *M4K708*, for applications |
| | **Instance_708** | Header file for global module structure |
| | **proto_708** | Header file, prototypes of all functions. This will include all the header files needed for applications |
| | **708Incl** | Header file, include for recompiling the 708 **dll** code, not for applications |
| | **Reg_708** | Header file containing data structures |
| **Demo Programs** | **demo_708_int** | Interrupt demo |
| | **demo_708** | Demo illustrating functions of *M4K708 Software Tools* |
| | **demo_708_loopback** | Demo illustrating loopback functions |

| | File Name | Description | |
|---|---|---|---|
| **DLL and *.LIB files** | The table below shows the name of the DLL file and its corresponding LIB file for each module under the Borland and Microsoft compilers | | |
| | **Module** | **Borland** | **Microsoft** |
| | P*x* | PX | pxms |
| | MCH | MCH | mchms |
| | RT*x* | RTX | rtxms |
| | Discrete | M4Kdio | M4Kdioms |
| | Serial | excser | excserms |
| | H009 | M4KH009 | M4KH009ms |
| | 708 | exc708 | exc708ms |
| | MMSI | m4kmmsi | m4kmmsims |

# Appendix G Code Index

*M4K708 Software Tools* is a set of C language functions designed to aid users of the *M4K708* module to write test programs. Below is an alphabetical listing of all the functions and the name of the *Software Tools* file which contains its programming code.

| Driver Functions | Code File Name (*.c) |
|---|---|
| ClearStatus_708 | **comm_708** |
| Get_4000Module_Type for PCI boards | **EXC4000** |
| Get_4000Module_Type for VME/VXI boards | **EXCv4000** |
| Get_Error_String_708 | **error_708** |
| Get_Interrupt_Count_708 | **deviceio_708** |
| GetHWRev_708 | **init_708** |
| GetModuleTimetag_708 | **init_708** |
| GetPixel_708 | **comm_708** |
| GetStatus_708 | **comm_708** |
| Init_Module_708 | **init_708** |
| InitializeInterrupt_708 | **deviceio_708** |
| ModuleTimetagReset_708 | **init_708** |
| NumberWordsInBuffer_708 | **comm_708** |
| ReadWord_708 | **comm_708** |
| Release_Module_708 | **init_708** |
| Select_Time_Tag_Source_4000 | **EXC4000** |
| SetControlData_708 | **comm_708** |
| SetEventFrequency_708 | **config_708** |
| SetInterrupt_708 | **config_708** |
| SetLoopback_708 | **init_708** |
| SetPixel_708 | **comm_708** |
| SetTransmitInterval_708 | **config_708** |
| SetTrigger_708 | **config_708** |
| SetupReceiveChannel_708 | **config_708** |
| SetupTransmitChannel_708 | **config_708** |
| StartReceive_708 | **comm_708** |
| StartTransmit_708 | **comm_708** |
| Stop_708 | **comm_708** |
| Wait_for_Interrupt_708 | **deviceio_708** |
| Wait_for Multiple_Interrupts_708 | **deviceio_708** |
| WriteWord_708 | **comm_708** |

# Appendix H Error Messages

All functions for *M4K708 Software Tools* are written as C functions, i.e., they return values. A negative value signifies an error. Full error messages may be printed using the Get_Error_String_708 function. Below is a list of all *Software Tools* error messages, the negative value of each, and an explanation of the error

| Error | Value | Explanation |
|---|---|---|
| einval | -2 | Illegal value used as an input |
| sim_no_mem | -5 | No memory for simulation |
| etimeoutreset | -26 | Timed out waiting for reset |
| ewrngmodule | -27 | Module specified on EXC-4000 board in not a *M4K708* module |
| enomodule | -28 | No EXC-4000 module present at the specified location |
| ebadhandle | -33 | Invalid handle specified; should be value returned by Init_Module_708 |
| eboardtoomany | -36 | Too many modules initialized |
| noirqset | -53 | No interrupt allocated. |
| einvalchan | -200 | Tried to set channel to illegal value |
| ebadchanhandle | -203 | Invalid handle specified, should be value returned by channel setup routine |
| echantype | -204 | Function invalid for this channel type |
| enoxmtword | -205 | No room to transmit ARINC 708 word |
| enorcvword | -206 | No ARINC 708 word received |
| ebadformat | -207 | No proper ARINC 708 word found in receive buffer |
| erunning | -208 | Channel already running |
| eoneshot | -209 | Cannot write to buffer when channel running in ONESHOT mode |
| eretransmit | -210 | Cannot write to buffer when channel running in RETRANSMIT mode |

### For EXC-4000PCI boards only

| Error Code | Value | Explanation |
|---|---|---|
| eopenkernel | -1001 | Cannot open kernel device; check Excalibur Configuration Utility settings |
| ekernelcantmap | -1002 | Kernel driver cannot map memory |
| ereleventhandle | -1003 | Error in kernel Release_Event_Handle |
| egetintcount | -1004 | Error in kernel Get_Interrupt_Count |
| egetchintcount | -1005 | Error in kernel Get_Channel_Interrupt_Count |
| egetintchannels | -1006 | Error in kernel Get_Interrupt_Channels |
| ewriteiobyte | -1007 | Error in kernel writeiobyte |
| ereadiobyte | -1008 | Error in kernel readiobyte |
| egeteventhand1 | -1009 | Error in kernel Get_Event_Handle, first part |
| egeteventhand2 | -1010 | Error in kernel Get_Event_Handle, 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 write attribute |
| ekernelreadattrib | -1022 | Kernel read attribute error |
| ekernelfrontdesk | -1023 | Kernel frontdesk error |
| ekerneloscheck | -1024 | Kernel Oscheck error |
| ekernelfrontdeskload | -1025 | Kernel frontdeskload error |
| ekerneliswin2000compatible | -1026 | Kernel iswin2000compatible error |
| ekernelbankramsize | -1027 | Kernel banksize error |
| ekernelgetcardtype | -1028 | Kernel getcardtype error |
| emodnum | -1029 | Invalid module number specified |

| Error Code | Value | Explanation |
|---|---|---|
| regnotset | -1030 | Board not configured. Reboot after ExcConfig is run and board is in slot |
| ekernelbankphysaddr | -1031 | Error in GetBankPhysAddress |
| ekernelclosedevice | -1032 | Error in CloseKernelDrive |
| ekerneldevicenotopen | -1034 | Kernel error: device is not opened |
| ekernelinitmodule | -1035 | Kernel initialization error |
| ekernelbadparam | -1036 | Kernel error: bad input parameter |
| ekernelbadpointer | -1037 | Kernel error: invalid pointer to output buffer |
| ekerneltimeout | -1038 | Wait for Interrupt function returned with timeout |
| ekernelnotwin2000 | -1039 | Operating System is not Windows 2000 |
| erquestnotification | -1040 | Request_Interrupt_Notification error |
| ekernelnot4000Card | -1041 | Designated Board is not an EXC-4000 carrier board |
| 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 Start-Timer |
| etimernotrunning | -1063 | Timer not running when ResetWatchdogTimer was called; did nothing |
| etimerrunning | -1064 | Timer already running when StartTimer was called; did nothing |
| eparmreload | -1065 | Illegal parameter used for `reload_flag` in StartTimer |
| eparminterrupt | -1066 | Illegal parameter used for `interrupt_flag` in StartTImer |
| ebaddevhandle | -1067 | Invalid handle specified. Use value returned by Init_Timers |
| edevtoomany | -1068 | Init_Timers called for too many boards |
| einvalidOS | -1069 | If an invalid operating system is used |

### For EXC-4000VME/VXI boards only

| Error Code | Value | Explanation |
|---|---|---|
| eviclosedev | -1050 | Error in ViClose device |
| evicloserm | -1051 | Error in ViClose Default RM |
| eopendefaultrm | -1052 | Error in ViOpenDefault RM |
| eviopen | -1053 | Error in ViOpen |
| evimapaddress | -1054 | Error in viMapAddress |
| evicommand | -1055 | Error in VISA command |
| einstallhandler | -1056 | Error in VISA viInstallHandler |
| eenableevent | -1057 | Error in VISA viEnableEvent |
| eunistallhandler | -1058 | Error in VISA viUnistallHandler |
| eclocksource | -1059 | If an invalid clock source was specified |
| edevnum | -1060 | If a device number greater that 255 was used |
| einstr | -1061 | If a device was not successfully initialized by Init_Module_708 |

# Functions Index

January 2006, Rev A-2