

ARTISAN®
TECHNOLOGY GROUP

Your **definitive** source for quality pre-owned equipment.

Artisan Technology Group
(217) 352-9330 | sales@artisantg.com | artisantg.com

Full-service, independent repair center
with experienced engineers and technicians on staff.

We buy your excess, underutilized, and idle equipment
along with credit for buybacks and trade-ins.

Custom engineering
so your equipment works exactly as you specify.

- Critical and expedited services
- In stock / Ready-to-ship
- Leasing / Rentals / Demos
- ITAR-certified secure asset solutions

Expert team | Trust guarantee | 100% satisfaction

All trademarks, brand names, and brands appearing herein are the property of their respective owners.

Visit our website - *Click HERE*

CONDOR ENGINEERING

MS1553 / MS1773

User's Manual



Copyrights

Software Copyright © 1993-2000 Condor Engineering, Inc.

This software product is copyrighted and all rights are reserved. The distribution and sale of this product are intended for the use of the original purchaser only per the terms of the License Agreement.

This User's Manual is copyrighted and all rights are reserved.

This document may not, in whole or part, be; copied; photocopied; reproduced; translated; reduced or transferred to any electronic medium or machine-readable form without prior consent in writing from Condor Engineering, Inc.

Microsoft is a registered trademark of Microsoft Corporation
Windows is a registered trademark of Microsoft Corporation

MS1553/MS1773 User's Manual (1600100-0023)

Condor Engineering, Inc.
Santa Barbara, CA 93101
(805) 965-8000
(805) 963-9630 (fax)
support@condoreng.com

Document Revision: August 29, 2000
Document Version: 4.11
Software Revision: 4.20

Contents

Part 1 MS1553 / MS1773 Installation

Chapter 1 Overview

About this Manual 2

Chapter 2 MS1553 Hardware Interface Description

Introduction 3
ISA Bus Memory Map 3
Byte Accesses 6
Switch Register 6
Interrupts 7

Chapter 3 Using CP with the Hardware

Direct Hardware RT Initialization Example 9

Chapter 4 MS1553PC / MS1773PC Installation

Installation Instructions 13
MS1553PC/1773PC Jumper Settings 15
- Changing Interrupt Line Jumper on the MS1773PC-1300 17
- Board Base Address 18
- Interrupt line select (IRQ3-7,9-12,14-15) 18
- High-Priority Interrupt Enable 19
- Bus Coupling 20

MS1553PC/1773PC DIP Switch Settings 20
- Switch SW1 and SW2 – Channel Bus Coupling 21
- Switch SW3 – Board Address/Configuration Information 21
- Switch SW4 – Interrupt Line Select (IRQ) 22

Chapter 5 MS1553PC104 Installation

Hardware Overview 25
- Memory Map for MS1553PC104 26
- Switch Register 26
- Timer 28

Discrete Input/Output Signals....29
Installation Instructions....31
MS1553PC104 Jumper Settings....33
Board Base Address....33
Interrupt Line Select (IRQ3-7, 9-12,14-15)....34
High-Priority Interrupt Enable....35
Bus Coupling....36
Input/Output Connector Pin Definition....37
Terminal Address Bits....38

Chapter 6 MS1553VMX Installation

Introduction....39
Hardware Overview....39
Switch Register....39
Interrupts....40
Memory Map....41
Timers....42
MS1553VMX VXI Compatibility....43
MS1553VMX Installation....45
MS1553VMX Jumper Locations....46
Performing Diagnostics....46
MS1553VMX Jumper Settings....47
Jumpers E3 and E5 Bus Coupling....47
Jumper Block E1 – A24, Board Address....48
Jumper Block E2 – VXI Registers Base Address....48
Input/Output Connector Pin Definition....49
Interface Software....50

Chapter 7 MS1553VME Installation

Introduction....51
Hardware Overview....51
Switch Register....52
Interrupts....52
Memory Map (Word Offset Format)....53
MS1553VME Installation....53
Performing Diagnostics....54
MS1553VME DIP Switch Settings....54
Switch SW1 – Channel A Bus Coupling....55
Switch SW3 – Channel B Bus Coupling....55
Switch SW4 – Board Address....56
Switch SW2 – Configuration Information....56
Interface Software....57

Chapter 8 MS1553MC Installation

System Requirements 59
MS1553MC Hardware Installation 59
MS1553MC DIP Switch Settings 60
MS1553MC Software Installation 61
Verifying Installation 62

Chapter 9 MS1553 Switch Register

Introduction 63
Switch Registers 63
    MS1553PC-3/4 Switch Register 63
    MS1553PC96 Switch Register 64
    MS1553PC104 Switch Register 64
    MS1553VMX Switch Register 65
    MS1553VME Switch Register 65
    Switch Register Notes 65

Part 2 C Driver

Chapter 10 C Driver

Introduction 67
Multiple Board Support 69
Compatibility with Previous Versions 69
Re-Building the API 69
Compilation Notes 70

Chapter 11 Bus Controller Architecture

Background 73
A Quick Glance at the Control Word for the BCRTM 75
Temporary Frame Operation versus Permanent Frame Execution 75
Building Frames 77
The Returned Status Structure 78
    Interrupt Status Word Definition for BC Mode 79
Delay Values 80
Building Permanent Frames 80

Chapter 12 Remote Terminal Architecture

Introduction 83
RT Descriptor Space 85
Remote Terminal Subaddress Descriptor 86
The Status Structure 86

Chapter 13 Bus Monitor Architecture

Introduction 89

Chapter 14 Routine Descriptions

Introduction 91
Command Summary 91
ms1553m_bc_1noskip 94
ms1553m_bc_auto_retry 95
ms1553m_bc_build_error 96
ms1553m_bc_build_frame 97
ms1553m_bc_chg_data 98
ms1553m_bc_clear_counts 99
ms1553m_bc_delay 100
ms1553m_bc_end_of_frame 101
ms1553m_bc_exec_frame 102
ms1553m_bc_free_frame 103
ms1553m_bc_get_adr 104
ms1553m_bc_get_cblk_ptr 105
ms1553m_bc_get_dbuf_ptr 106
ms1553m_bc_halt_frame 107
ms1553m_bc_mtoggle 108
ms1553m_bc_poll 109
ms1553m_bc_restart 111
ms1553m_bc_rtrt 112
ms1553m_bc_rtrtmon 114
ms1553m_bc_rtrtmon_same 116
ms1553m_bc_set_eol 118
ms1553m_bc_set_pcr 119
ms1553m_bc_skip 120
ms1553m_bc_tr_data 121
ms1553m_bc_tr_grp_data 123
ms1553m_bc_tr_mode_code 125
ms1553m_bc_tr_mode_same 127
ms1553m_bc_tr_same 129
ms1553m_bcrt_inquire 131
ms1553m_bcrtm_active 132
ms1553m_bcrtm_exit 133
ms1553m_init (BC) 134
ms1553m_init (RT) 138
ms1553m_init (BM) 142
ms1553m_m_close 145
ms1553m_m_cur_cbptr 146
ms1553m_m_flen 147

ms1553m_m_fread 148
ms1553m_m_fsearch 149
ms1553m_m_fseek 150
ms1553m_m_fwrite 151
ms1553m_m_get_msg 152
ms1553m_m_next_cbptr 153
ms1553m_m_open 154
ms1553m_m_record 155
ms1553m_m_select_rt 156
ms1553m_m_start 157
ms1553m_m_start_cbptr 158
ms1553m_m_stop 159
ms1553m_rt_busy 160
ms1553m_rt_chg_adr 161
ms1553m_rt_clear_counts 162
ms1553m_rt_dbusc 163
ms1553m_rt_free_link 164
ms1553m_rt_get_sa_dbuf_ptr 165
ms1553m_rt_get_sa_mstat_ptr 166
ms1553m_rt_instr 167
ms1553m_rt_reset_timeout 168
ms1553m_rt_set_timeout 169
ms1553m_rt_srq 170
ms1553m_rt_ssfail 171
ms1553m_rt_t_update 172
ms1553m_rt_tflag 173
ms1553m_rt_tr_data 174
ms1553m_rt_tr_link 176
ms1553m_rt_tr_mcdata 178
ms1553m_rt_tr_short_data 179
ms1553m_rtm_clear_tag_timer 180

Chapter 15 Generic 1553 Routine Descriptions

Command Summary 181
Single Board Command Summary 181
ms1553m_bit_test 182
ms1553m_break_hit 183
ms1553m_break_irupt 184
ms1553m_clear_break_flag 185
ms1553m_config_sw 186
ms1553m_get_version 187
ms1553m_int_handler 188
ms1553m_register_address 189

ms1553m_register_read 190
ms1553m_register_write 191
ms1553m_reset 192
ms1553m_set_interrupt 193
memwcpy 196

Part 3 Visual Basic Interface

Chapter 16 Visual Basic Interface

Introduction 197
Defining MST1553.DLL Functions and Constants 197
Calling ms1553m_init() from Visual Basic 198
MS1553 Functions Not Supported in Visual Basic 198
Working with Unsigned Integers in Visual Basic 199
Using MST1553 Callbacks in Visual Basic 200
    An Example Callback Function: 201
Using the ms1553m_bcrtm_exit Function 202
Visual Basic Sample Application 202
VB1553.FRM Source Code 203

Appendix A Linking the Driver and Sample Programs

Introduction 209
Linking the Driver with a DOS Application 209
Linking the Driver with a WINDOWS Application 210
Sample "C" Programs 210
    BC.C 211
    RT.C 217
    MON.C 222

Appendix B VMECFG Source File

Source Code 229

Appendix C Cabling and Connectors

Introduction 239
    MIL-STD-1553 Modules 239
    MIL-STD-1773 Modules 239

Appendix D Power Requirements

Requirements 241

CHAPTER 1

Overview

This manual provides hardware installation and software development information for the following Condor interfaces:

- MS1553PC MIL-STD-1553 to ISA bus interface
- MS1773PC MIL-STD-1773 to ISA bus interface
- MS1553PC104 MIL-STD-1553 to PC/104 bus interface
- MS1553VME MIL-STD-1553 to VME or VXI bus interface
- MS1553VMX MIL-STD-1553 to VME or VXI bus interface
- MS1553MC MIL-STD-1553 to Micro Channel bus interface
- MS1553GRD MIL-STD-1553 to GRiD ISA bus interface

The Industry Standard Architecture (ISA) bus is found on all IBM-compatible computers and the Micro Channel Architecture (MCA) is used on IBM's Personal System/2 (PS/2) and RS/6000 line of computers.

This manual also describes all of the functions that are provided with the Condor 1553 utility library. It can be used as a template for software development for programmers who wish to write their own driver. Regardless of the driver you are implementing, this manual will help you in defining software functions and their structures for BC, RT, and monitor modes of operation, including test functions and interrupt service routines. A series of sample programs is also available which can assist you in your development.

Within the software sections of this manual, reference is made to the "MS1553" board. This is intended to be a generic reference to any of the boards listed above. When there is a difference in the boards, which affects the software descriptions in Chapter 10, the function description will note the difference.

About this Manual

This manual is divided into three sections and four Appendices:

- **Part 1, MS1553/MS1773 Installation** – provides hardware and software development information as well as installation instructions.
- **Part 2, C Driver** – describes the "C" language driver supplied by Condor Engineering, Inc. for its MS1553 product line.
- **Part 3, Visual Basic** – describes the interface to the Microsoft Visual Basic development environment using the Condor Engineering-supplied Microsoft® Windows™ 32-bit DLL.
- **Appendices** – Four appendices provide information on linking the driver to your application as well as providing sample programs, cabling and connectors required for the MS1553 and MS1773 modules. A final section on power requirements is included.

CHAPTER 2

MS1553 Hardware Interface Description

Introduction

This chapter provides the hardware interfacing information required by a programmer to directly communicate with the IBM PC-compatible hardware.

If you are using the Condor-supplied Microsoft C or Borland C/C driver libraries in a DOS environment, or the Condor-supplied Microsoft WINDOWS™ DLLs in a WINDOWS 3.x, 95, or NT environment need only read this section for a basic understanding of the hardware.

If you are developing your own software driver in another high-level language, in assembly language, or in a different operating environment, this document presents a logical and descriptive approach to the software development process.

One other resource is necessary: The United Technologies Microelectronics Center (UTMC) data sheet for the BCRTM controller chip, which is used on the card. This document is included with all board shipments. Additional copies are available through Condor Engineering, Inc.

ISA Bus Memory Map

The hardware is designed to be totally memory mapped within the PC's memory space in 32KB or 64KB increments. Condor board models with 8KW RAM can only map into a 32KB range (and only need 32KB).

The valid segment address ranges are listed in the following table.

Part Numbers	MS1553PC-1 MS1553PC-3 MS1773PC-3 MS1553GRD-1	MS1553PC-2 MS1553GRD-2	MS1553PC-4 MS1773PC-4	MS1553MC-1	MS1553MC-2
Valid Address Ranges (8KW)	8000H - 87FFH 8800H - 8FFFH 9000H - 97FFH 9800H - 9FFFH A000H - A7FFH A800H - AFFFH B000H - B7FFH B800H - BFFFH C000H - C7FFH C800H - CFFFH D000H - D7FFH D800H - DFFFH E000H - E7FFH E800H - EFFFH	8000H - 87FFH 8800H - 8FFFH 9000H - 97FFH 9800H - 9FFFH A000H - A7FFH A800H - AFFFH B000H - B7FFH B800H - BFFFH C000H - C7FFH C800H - CFFFH D000H - D7FFH D800H - DFFFH E000H - E7FFH E800H - EFFFH		C000H - C7FFH C800H - CFFFH D000H - D7FFH D800H - DFFFH	C000H - C7FFH C800H - CFFFH D000H - D7FFH D800H - DFFFH
Valid Address Ranges (32KW)		8000H - 8FFFH 9000H - 9FFFH A000H - AFFFH B000H - BFFFH C000H - CFFFH D000H - DFFFH E000H - EFFFH F000H - FFFFH	8000H - 8FFFH 9000H - 9FFFH A000H - AFFFH B000H - BFFFH C000H - CFFFH D000H - DFFFH E000H - EFFFH F000H - FFFFH		C000H - CFFFH D000H - DFFFH

The base address is set by DIP switches or jumpers on the board for PC models and by the configuration software (ADF file) on the Micro Channel architecture model boards.

For all Condor model boards, the dual-ported memory exists as the lower 8KW portion of this window, while an upper portion is reserved for use for the BCRTM, switch register, and dual-ported memory above 8KW (available on 32KW RAM models only).

Because of this mapping, there are actually 31,483 (8,096 plus 23,487) words of memory available for use on the 32KW model boards. The following table shows the memory map in both byte-offset and word-offset formats.

Addresses in Standard Intel Byte-Offset Format

Base Address (BA)	8000H to F800H
Memory	(BA) through (BA+3FFFH)
BCRTM Registers	(BA+4000H) through (BA+4023H)
Reserved Area	(BA+4024H) through (BA+47FFH)
Switch Register	(BA+4800H)
Reserved Area	(BA+4801H) to (BA+487FH)
Memory, above 16KB	(BA+4880H) to (BA+FFFFH)

Addresses in CP Word-Offset Format

Base Address (BA)	8000H to F800H
Memory	(BA) through (BA+1FFFH)
BCRTM Registers	(BA+2000H) through (BA+2011H)
Reserved Area	(BA+2012H) through (BA+23FFH)
Switch Register	(BA+2400H)
Reserved Area	(BA+2401H) to (BA+243FH)
Memory, above 8KW	(BA+2440H) to (BA+7FFFH)

If you have an 8KW board installed, no other board may be installed within the 32 Kbytes address space mapped by the board. However, that does not prevent a memory manager, such as QEMM, 386MAX, EMM386, etc., from allocating an Upper Memory Block (UMB) in the otherwise unused address space from (BA+5000h) through (BA+7FFFh) (standard Intel byte-offset addresses). This method can be used to gain 12 Kbytes of otherwise unusable upper memory. For additional information, see the documentation for your memory manager.

Byte Accesses

If the PC were allowed to write only bytes into the dual-ported memory, a synchronization problem would exist where the PC could have updated only one byte of a 16-bit word while the BCRTM could read the entire 16-bit word with only part of it being correct.

Eight bit boards are designed with a front-end which only reads and writes words on the 1553 side while allowing bytes to the PC. Thus, when the low byte is written the entire word is written to memory. You must be very careful to *write only words* to the board. If bytes are written, an interrupt could occur which could cause the interrupt routine to corrupt the register if it writes to the board.

When a word is written to the board, interrupts can't occur between each byte. Remember, when using string, move instructions to use only words as well. The read mode for the board works similarly. A word is read and saved in two registers when the low byte is read. When the high byte is read the contents of one of the bus registers, which contains the high byte is supplied. Therefore you must also *only read words*.

For 16-bit model boards, the same rule applies: *only read and write words* to the board. If you do not use a Condor-supplied driver library, you should be aware that some compilers optimize for single byte accesses. Either switch off this optimization or write your code so that words always are read or written to the board.

Switch Register

The software may read positions 5 through 8 of the configuration switch. These switches determine the RAM size and the size of the board's address range (window). See the installation chapters.

They appear as a four bit number in the LSBs of the word read with switch five being the MSB and switch eight the LSB. The address of the switch register is the base board address plus 4800H in the standard Intel byte-offset format. A closed switch is a zero and an open switch is a one. The MS1553MC board uses a "POS" register rather than DIP switches. The configuration setup menu may be used to change it. For MS1553PC-3 and MS1553PC-4 models, the switch register is in factory programmed logic on the board, and is not programmable by the user.

For MS1553PC-1 and MS1553PC-2 models, bit 15 (MSB) and bit 14 are used for improvement of RT interrupt servicing.

Interrupts

The interrupt (IRQ) is selected by setting only one DIP switch closed for the available interrupts. All 8-bit model boards have IRQ2-7 available. All 16-Bit model boards have IRQ3-7, 9-12, 14, and 15 available. The MS1553PC-3, MS1553PC-4, MS1773PC-3, and MS1773PC-4 boards use jumpers rather than switches. The MS1553MC board uses a POS register rather than DIP switches. You can change the POS register using the configuration setup menu.

If the high-priority interrupt enable switch is closed, the selected interrupt is the 'OR' of the high priority interrupt and the standard interrupt from the chip. If the switch is open, only the standard interrupt may be enabled. Setting the bits in the Interrupt Reset Register of the BCRTM chip clears the interrupt.

When using the Condor-supplied software library, ensure that the high-priority interrupt is enabled. This is expected by the software, and gives you the option of disabling or enabling the high-priority interrupts in the software driver.

A special note to IRQ 2/9 users: If you are using IRQ 2 or 9, remember that on an AT-class host, the host mother-board interrupt 2 is physically wired to IRQ 9 on the interrupt controller. You should specify IRQ 9 to any software that controls the add-on board. This is true even if the add-on board is an 8-bit board, and claims to use IRQ 2, since the host, not the add-on board, sets the exact IRQ number (2 or 9).

CHAPTER 3

Using CP with the Hardware

CP is a DOS-based program for use with MS1553 interface cards. It is not supported under Windows 95 or NT. If you are going to write your own software for the board, you may need to test some algorithms or register settings in an interactive environment. CP provides the method for this test since it can read and write directly to memory locations in the PC. You must remember that the only difference when using the segment/offset formats for CP is that the offset is in words rather than bytes. This was done to eliminate confusion since all of CP and the 1553 protocol are word oriented. What this means to you is that the byte offset must be divided by two. Sample CP programs are included on the disk sent with the hardware.

Direct Hardware RT Initialization Example

The following routine is an example of a CP routine that initializes remote terminal 1. The CP routine RT.CP is called with the segment address for the board to initialize. The program legalizes subaddress one of RT 1 for both receive and transmit. Both the input and output buffers are mapped to the same location so that any data that comes in for that subaddress is immediately available for reading. This is written as a CP subroutine that requires the segment portion of the address for the board to be passed as a parameter:

```
call  rt D000
o<    rt D000
      ; set up the addresses
      mov 0 r60                 ; this is the board address
      mov %1 r61

      ; setup the main control registers
      ; offsets are in "CP word-offset format"
      mov 0 >r60:2002           ; set the descriptor space address
      mov 800 >r60:2006         ; set the log list pointer
      mov 0 >r60:2007           ; clear the hi-priority interrupt bits
      mov 0 >r60:2009           ; clear the standard interrupt bits
```

```
mov 1 >r60:200a          ; set the rt address to one, odd parity

; setup the rt descriptors for receive
mov 600 >r60:4           ; set two to be illegal
mov 0   >r60:0           ; legalize subaddress 1
mov 200 >r60:1           ; set the message status pointer
mov 400 >r60:2           ; set the data list pointer

; and for transmit
mov 600 >r60:84          ; set two to be illegal
mov 0   >r60:80          ; legalize subaddress 1
mov 600 >r60:81          ; set the message status pointer
mov 400 >r60:82          ; set the data list pointer

; enable the rt
mov 181 >r60:2000
```

If we were to write this in C (for DOS) and call it RT.C, it would appear as follows:

```
main( argv, argc )
char **argv;
unsigned short argc;
{
    unsigned short *base;
    unsigned long address;

    /*   if there aren't the right number of parameters, show what we want  */
    if ( argc != 2 )
    {    printf( "rt segment\n" );
         exit( 1 );
    }

    /*   convert the address */
    sscanf( argv[1], "%lx", &adr );
    base = (unsigned short *) (address << 16);

    /*   set the descriptor space register to zero */
    base[0x2002] = 0;

    /*   set the log list pointer */
    base[0x2006] = 0x800;

    /*   clear the hi-priority and standard interrupt bits */
    base[0x2007] = base[0x2009] = 0;

    /*   set the rt address to one, with odd parity */
    base[0x200a] = 1;
```

```
        /*  setup the rt descriptors for receive, subaddress two will be illegal, one
            is legal */

        base[4] = 0x600;
        base[0] = 0;
        base[1] = 0x200;        /* message status pointer */
        base[2] = 0x400;        /* data list pointer */

        /*  setup the rt descriptors for transmit, subaddress two will be illegal, one
            is legal */
        base[0x84] = 0x600;
        base[0x80] = 0;
        base[0x81] = 0x200;     /* message status pointer */
        base[0x82] = 0x400;     /* data list pointer */

        /*  enable the rt */
        base[0x2000] = 0x181;
    }
```

The absolute addresses are defined in the C driver, but this example is intended to show you the lowest level possible for interfacing to the board.

This method must be modified somewhat when running under a protected-mode operating system such as Windows (V3.x, 95 or NT). Under such an operating system physical addresses do not map directly into software addresses. The MS1553 API supplies functions that can be used to create a pointer that DOES map the MS1553 board (See the API function "ms1553m_register_address" for further information).

CHAPTER 4

MS1553PC / MS1773PC Installation

This section explains how to install the MS1553PC, MS1773PC & MS1553GRD PC boards. All DIP switch, jumper, and configuration file settings are defined.

If possible, use a static free workstation and be sure to follow good ESD practices. Before beginning the installation process, inspect the packaging material to insure that the board hasn't been damaged during shipment. If it has, contact the shipper right away.

Installation Instructions

Carefully remove the installation media and documentation and set them aside. Then follow the steps in this section:

1. Select the board address. The default address for these boards is D000H. Make a note of the address you select so that you can make it known to the software.

   The address is jumper selectable in the upper memory block (UMB) of the IBM PC. This is the address space between 640K and 1024K. This space is also used for other devices so you must be sure to select an area that is not being used by something else.

2. A program called MSD.EXE (Microsoft Diagnostics) is provided with most current Microsoft Windows and DOS systems and helps you to select the address. To run this program, type:

   msd

Note: This assumes the program is already on your PATH. It usually is since it is typically located in the DOS and/or Windows folder of your system.

   Select the "Memory" box and look for a free address range that is large enough for the address space of the board. It is best if your extended memory manager (e.g. QEMM, EMM386, etc.) is not running during this step.

Avoid selecting an address that directly conflicts with any other installed devices. The selected memory address space must not directly overlap any other devices. In addition, any indirect conflict must be avoided. This occurs when a 16-bit memory-mapped device (such as the MS1553PC-3 or -4) is placed in the same 128K region (i.e. segment address ranges: A000-BFFF or C000-DFFF or E000-FFFF) as an 8-bit memory-mapped device. This is caused by an inherent architectural limitation of the ISA bus.

When selecting a base memory address, be aware of the potential for conflict with expanded memory managers (e.g. EMM386, QEMM386) which also use the UMB. You can avoid a problem by configuring the memory manager to exclude the address space you select from its list of available mapping regions. For example, when starting EMM386, use the command line switch: /X=D000-D7FF for an MS1553PC-3 at the default address of D000H. (You may instead use the command line switch: /X=D000-D4FF, which permits EMM386 to use the unused area of the 8KW board from D500-D7FF). See your memory manager documentation for a complete discussion of these switches. Memory managers are typically loaded at system startup from CONFIG.SYS.

3. Insert the installation media and install the software. For Win32 platforms, the installation program should run automatically following insertion of the CD. If it does not, execute Setup.exe from the \Setup\Disk1 folder. For Win16 and DOS platforms, copy the entire contents of the folder \MS1553-SW to your hard drive using xcopy or the file manager.

4. Remove the MS1553PC or MS1773PC board from the anti-static bubble pack and lay it on top of the bag. This prevents static discharge from damaging the board when you change the jumpers.

5. To set the DIP switches for the MS1553GRD module remove the four screws from the front of the module. This is the end with the two BJ-77 connectors protruding from the case. Be careful as the BJ-77 connectors are wired to the boards internally. Carefully slide the boards from the module. They unfold to expose the DIP switches. As the boards unfold, lay them on top of the bag.

   Refer to the “MS1553PC/1773PC DIP Switch Settings” or “MS1553PC/1773PC Jumper Settings” section of this manual to set the DIP switches or jumpers on the board to the desired settings for your operating environment.

   For MS1553PC and MS1773PC models – remove the cover from the PC in which the board is to be placed. Remove the blank mounting plate from one of the unused slots in your computer. Install the board and bolt the guide plate to the PC with the screw that was removed in the earlier step.

   For MS1553GRD models –carefully fold the boards back into their face-to-face position and slide them back into the module, again being

careful not to break the wires connected to the BJ-77's. Now replace the four screws to complete the assembly. Install the MS1553GRD module in your GRiD computer by sliding it into the module slot in the back of the computer until it locks in position. At this time it is installed and ready for use.

6. Connect your 1553 bus to the connectors provided on the back of the interface card. The top connector is bus A and the other is bus B. If this is the last connection on the bus use a "Tee" and connect a 70-ohm terminator to provide the proper termination for the bus. Otherwise, use a "Tee" with a cable on each end of the "Tee".
7. For MIL-STD-1773 boards, from top to bottom are channel A transmitter, channel A receiver, channel B transmitter, channel B receiver. Connect four 850 nm fiber-optic cables with low-profile SMA connectors (or ST connectors if you ordered the ST option).
8. Power-up the computer.

To test the configuration:

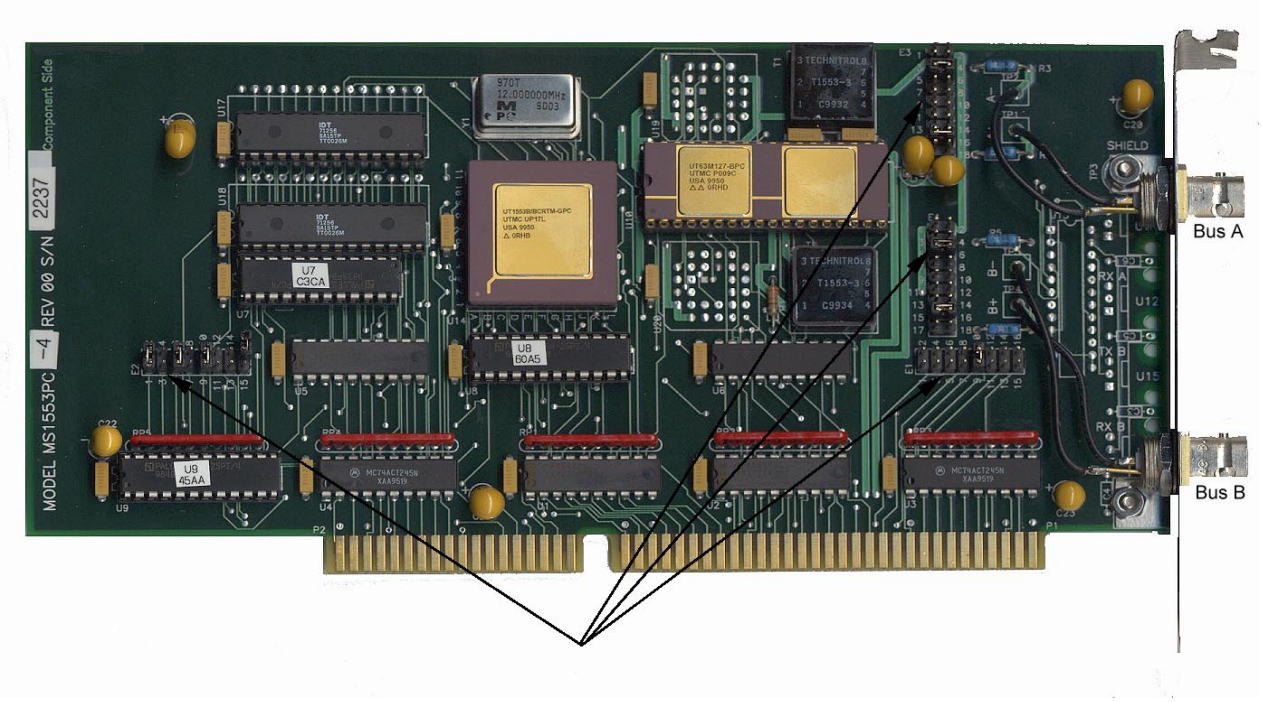
1. Execute the Test Config program for your host operating system. On a Win32 platform this program is named tstcfg32.exe, located in the folder \Examples\C\Win32 and accessed from the Test Config shortcut. For Win3.x and DOS it is named tstcfg16.exe or tstcfg.exe under the \Examples\C\Win16 and \Examples\C\DOS folders, respectively.
2. Test Config will prompt for the information concerning your board's configuration. The system then tests the configuration. If there is a problem, diagnostic information is printed on the screen. The most likely problems are mapping the board to the same memory space as another card or the same interrupt as another card.
3. Once these steps are successfully complete the board is ready for use.

MS1553PC/1773PC Jumper Settings

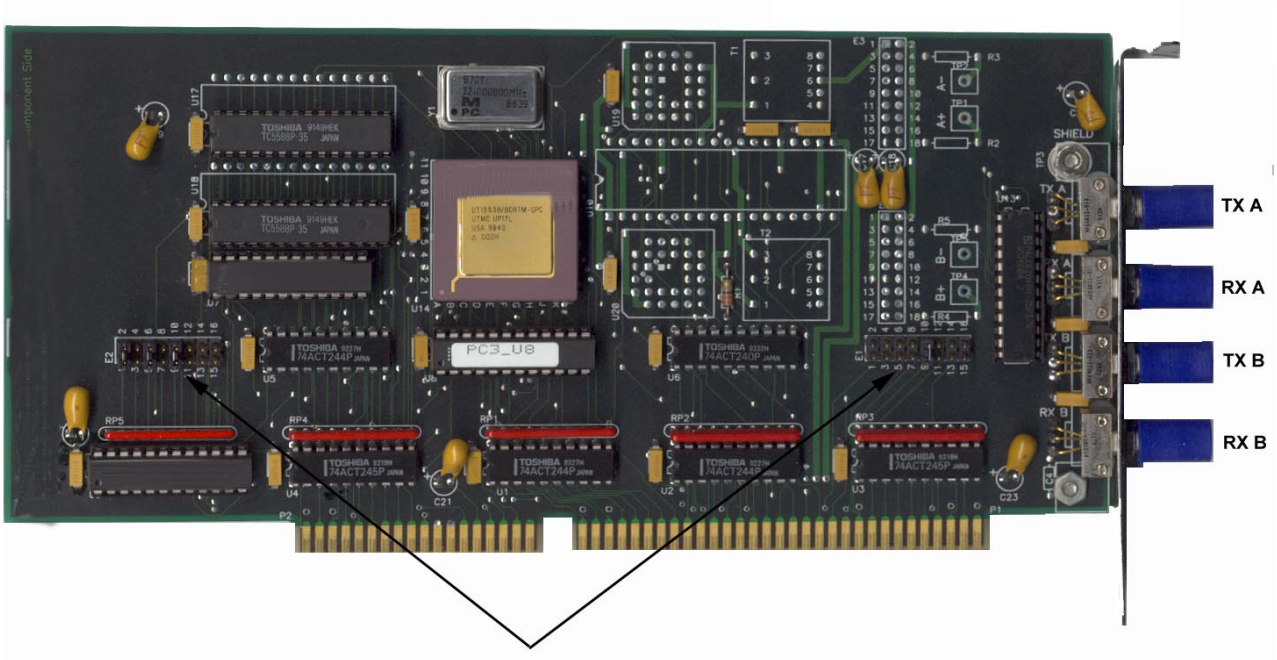
This section provides jumper settings for the MS1553PC and MS1773PC boards.

The MS1553PC-3/-4 interface board contains four jumper blocks used to determine certain special information required by the hardware. The MS1773PC-3/-4 interface board contains only two jumper blocks. The setting description is identical to the MS1553PC-3/-4 settings.

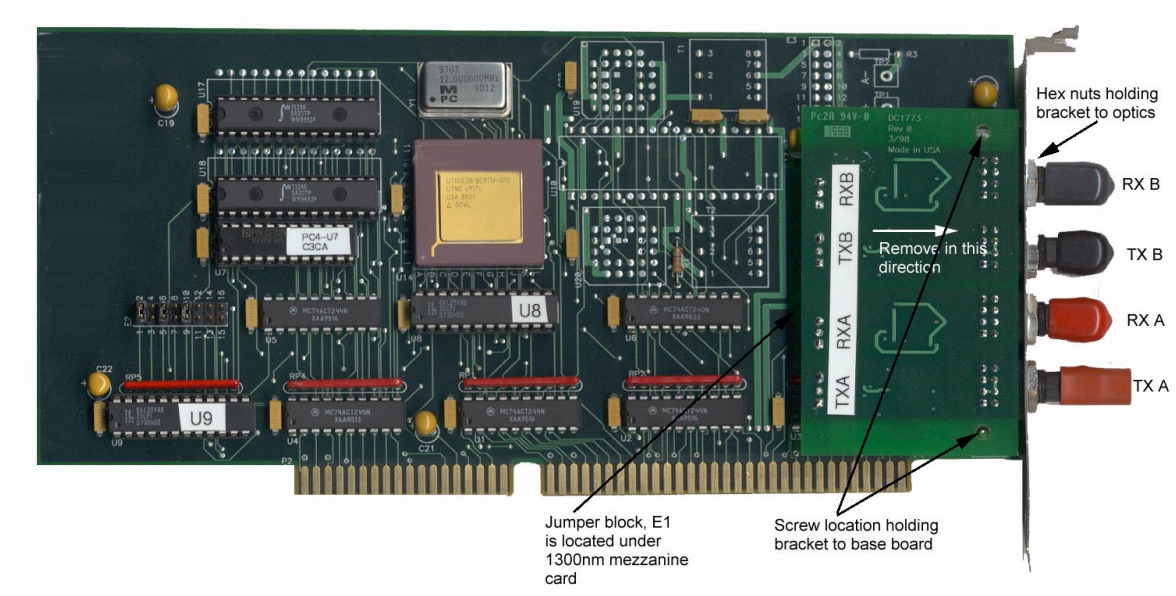
Please remember to record the settings so that you don't have to remove the cover from your PC at a later date.



MS1553PC Jumper and Connector Locations



MS1773PC Jumper and Connector Locations



MS1773PC-1300 Jumper and Connector Locations

Changing Interrupt Line Jumper on the MS1773PC-1300

If you need to change the interrupt line jumper setting, remove the 1300nm mezzanine card as follows:

Caution: This product can be damaged by static, please use appropriate ESD safeguards when performing the following procedure.

1. Remove the two screws (see figure below for screw locations) holding the bracket to the base.
2. Slide the mezzanine card & bracket assembly slightly forward as show on drawing so the mounting tabs clear the base board.
3. Carefully remove the mezzanine card and bracket assembly to gain access to the jumpers.
4. Change the interrupt jumper setting.
5. Re-assemble in reverse order, paying close attention not to damage the board to board wiring.

The hex nuts fastening the optics to the bracket require an opening in the computer's rear panel of ½ inch. If your computer has narrower openings, you may have to remove the hex nuts and reinstall the mezzanine card after installing the card. Before trying this procedure, please check that all the hex nuts have been installed with their sides parallel to the sides of the opening as shown below.

Board Base Address

Jumper Positions one through eight of E2 determine the board address as follows:

Address	Jumper	Jumper	Jumper	Jumper
8000H	E2-1 to E2-2	E2-3 to E2-4	E2-5 to E2-6	E2-7 to E2-8
8800H	E2-3 to E2-4	E2-5 to E2-6	E2-7 to E2-8	
9000H	E2-1 to E2-2	E2-5 to E2-6	E2-7 to E2-8	
9800H	E2-5 to E2-6	E2-7 to E2-8		
A000H	E2-1 to E2-2	E2-3 to E2-4	E2-7 to E2-8	
A800H	E2-3 to E2-4	E2-7 to E2-8		
B000H	E2-1 to E2-2	E2-7 to E2-8		
B800H	E2-7 to E2-8			
C000H	E2-1 to E2-2	E2-3 to E2-4	E2-5 to E2-6	
C800H	E2-3 to E2-4	E2-5 to E2-6		
D000H	E2-1 to E2-2	E2-5 to E2-6		
D800H	E2-5 to E2-6			
E000H	E2-1 to E2-2	E2-3 to E2-4		
E800H	E2-3 to E2-4			
F000H	E2-1 to E2-2			
F800H	All Jumpers Open			

Memory contention is the most common reason that a PC adapter board fails after installation. Run a utility, such as MSD.EXE to determine your current memory usage before installing the board. Most "upper memory" usage has been standardized for common peripherals, but D000H through EFFFH is reserved for specialized adapter boards. The board allows maximum flexibility in configuring memory usage.

Remember that models MS1553PC-4 and MS1773PC-4 must be mapped to an even 64KB-segment address to work properly. We ship all boards from the factory with the base address set at D000H.

Interrupt line select (IRQ3-7,9-12,14-15)

Jumper blocks E1 and E2 are used to select the interrupt line used for the PC. Only one jumper must be in at any time. Interrupt conflict is one of the most likely reasons that PC adapter boards don't work when first installed. Check your PC manual for available interrupts. Often, you will find that IRQ 5 is not being used, but the Condor board allows you to choose any interrupt that is available on your PC.

IRQ#	Jumper Placement
IRQ9	E1-15 to E1-16
IRQ7	E1-13 to E1-14
IRQ6	E1-11 to E1-12
IRQ5	E1-9 to E1-10
IRQ4	E1-7 to E1-8
IRQ3	E1-5 to E1-6
IRQ10	E1-3 to E1-4
IRQ11	E1-1 to E1-2
IRQ12	E2-15 to E2-16
IRQ15	E2-13 to E2-14
IRQ14	E2-11 to E2-12

High-Priority Interrupt Enable

The high-priority interrupt enable jumper is used for enabling the high priority interrupts for the board. If you do not wish to service these interrupts from the BCRTM chip (and you do not wish the driver software to service them), disable them by removing the jumper.

These interrupts are the following:

- Dynamic bus control acceptance.
- Illogical command error.
- Subsystem fail.
- BIT fail.
- End of BIT.
- Data overrun.

The normal position for this jumper should be "on", which enables the interrupt. If the jumper is *installed*, these interrupts can be individually enabled, read, and disabled by the driver software.

When using the Condor-supplied software library, ensure that the high-priority interrupt is enabled. This is expected by the software, and gives you the option of disabling or enabling the high-priority interrupts in the software driver.

High-Priority Interrupt	Jumper
Enable	E2-9 to E2-10
Disabled	no jumper

Bus Coupling

Note: This section is not applicable to MS1773PC-3 and MS1773PC-4 boards, which use fiber optic cabling.

The bus outputs can be coupled in three different ways:

- Direct coupled (short stub) - Isolation Transformer 1:1
- Direct coupled (short stub) - Isolation Transformer 1.4:1
- Transformer coupled (long stub)

Use direct coupling when the distance from the isolation transformer to the data bus does not exceed one foot. Use transformer coupling when the distance exceeds one foot but is less than 25 feet. There are isolation transformers on the board for both direct and transformer coupling.

There are 53.6-ohm isolation resistors on board for direct coupling. Transformer coupling requires external transformer and isolation resistors at the junction to the main 1553 bus to match the stub impedance to the bus.

Jumper block E3 is for channel A bus coupling; jumper block E4 is for channel B bus coupling. Each selection requires two jumpers per channel.

Coupling	Jumper (channel A)	Jumper (Channel B)
Direct (1:1)	E3-5 to E3-6 E3-15 to E3-16	E4-5 to E4-6 E4-15 to E4-16
Direct (1.4:1)	E3-7 to E3-8 E3-13 to E3-14	E4-7 to E4-8 E4-13 to E4-14
Long Stub	E3-9 to E3-10 E3-11 to E3-12	E4-9 to E4-10 E4-11 to E4-12

MS1553PC/1773PC DIP Switch Settings

This section provides DIP switch settings for the MS1553PC-1, MS1553PC-2, MS1553GRD-1, & MS1553GRD-2 boards.

The MS1553PC interface board contains four DIP switches used to determine information required by the hardware. Remember to record the settings so that you do not have to remove the cover from your PC at a later date. The switch setting of "off" is synonymous with "open"; "on" is synonymous with "closed".

Switch SW1 and SW2 – Channel Bus Coupling

The bus outputs can be coupled in three different manners:

- Direct coupled (short stub) - Isolation Transformer 1:1
- Direct coupled (short stub) - Isolation Transformer 1.4:1
- Transformer coupled (long stub)

Use direct coupling when the distance from the isolation transformer to the data bus does not exceed one foot. Use transformer coupling when the distance exceeds one-foot but is less than 25 feet. There are isolation transformers on the board for both direct and transformer coupling.

There are 53.6 ohm isolation resistors on board for direct coupling. Transformer coupling requires an external transformer and isolation resistors at the junction to the main 1553 bus to match the stub impedance to the bus. Switch SW1 is for channel A bus coupling. Switch SW2 is for channel B bus coupling. Each selection requires two switches be "on" while the rest are "off".

SW1 Switch Numbers								
Coupling	1	2	3	4	5	6	7	8
Direct (1:1)	off	on	off	off	off	off	on	off
Direct (1.4:1)	off	off	on	off	off	on	off	off
Long Stub	off	off	off	on	on	off	off	off

SW2 Switch Numbers								
Coupling	1	2	3	4	5	6	7	8
Direct (1:1)	off	on	off	off	off	off	on	off
Direct (1.4:1)	off	off	on	off	off	on	off	off
Long Stub	off	off	off	on	on	off	off	off

Switch SW3 – Board Address/Configuration Information

Switch positions one through four of switch SW3 determine the board address as follows:

SW 3 Switch Numbers				
	1	2	3	4
8000H	on	on	on	on
8800H	on	on	on	off
9000H	on	on	off	on
9800H	on	on	off	off
A000H	on	off	on	on
A800H	on	off	on	off
B000H	on	off	off	on
B800H	on	off	off	off

SW 3 Switch Numbers				
	1	2	3	4
C000H	off	on	on	on
C800H	off	on	on	off
D000H	off	on	off	on
D800H	off	on	off	off
E000H	off	off	on	on
E800H	off	off	on	off
F000H	off	off	off	on
F800H	off	off	off	off

Switches five through eight of SW3 contain configuration information that is read back by the software to verify the options in use.

Option	Switch	On/Off	Model
Reserved	8	off	all
8KW RAM	7	on	all
32KW RAM	7	off	PC-2/GRD-2
32KB PC window	6	on	all
64KB PC window	6	off	PC-2/GRD-2
Reserved	5	on	all

Switch six indicates that you are using either a 32KB or 64KB window into the board from the PC. If it is a 64KB window, it must be on an even boundary (i.e. A000, B000 D000).

Switch seven informs the driver of the amount of static RAM that is on the board. The standard is 8KW. If you have the "-2" option, which is 32KW, this switch is off. The actual amount of memory available is also dependent on the window selected. If you have selected a 32KB window, you have 16KW of usable RAM; the remainder is unused.

Switch SW4 – Interrupt Line Select (IRQ)

This switch selects the interrupt line to be used for the PC. Only one switch must be on at any time. The settings are as follows:

IRQ Number	SW4
2	(this is IRQ9 on an AT)
3	6
4	5
5	4
6	3
7	2

The last switch is used for enabling the high priority interrupts for the board. If you do not wish to service these interrupts from the BCRTM chip (and you do not want the driver software to service them), disable the high priority interrupts by turning switch 8 off. These interrupts are:

- Dynamic bus control acceptance.
- Illogical command error.
- Subsystem fail.
- BIT fail.
- End of BIT.
- Data overrun.

The normal position for this switch should be "on", which enables the interrupt. If the switch is *on*, then these interrupts can be individually enabled, read, and disabled by the driver software.

CHAPTER 5

MS1553PC104 Installation

This section tells provides a hardware overview and explains how to install the MS1553PC104 PC board. All jumper and configuration file settings are defined.

If possible, use a static free workstation and be sure to follow good ESD practices. Before beginning the installation process, inspect the packaging material to insure that the board hasn't been damaged during shipment. If it has, contact the shipper right away.

Hardware Overview

This section provides the hardware overview of the MS1553PC104 PC board. It includes memory map and timer and register descriptions.

The MS1553PC104 is a 16-bit PC/104 compatible board that is memory mapped to any 32-Kbyte, 64-Kbyte, or 128-Kbyte ISA upper memory block address. The memory window size is jumper programmable. The memory base address is also jumper programmable. The API software does not support the 128-Kbyte-memory configuration at this time.

The BCRTM sees memory as 16-bit words only. Because the memory is dual-ported between the BCRTM and the host ISA bus, all ISA bus accesses to the card must be 16-bit accesses. Addressing the board in byte mode may cause memory or register contents to be corrupted and a synchronization problem would exist.

In addition to MIL-STD-1553 functions, the MS1535PC104 board includes a general-purpose timer and six low current discrete signals. The timer is useful for high resolution, long interval timing and time tagging of MIL-STD-1553 and discrete events. Each discrete can be individually programmed as input, output, or bi-directional. A flexible interrupt on state change feature is also included.

Memory Map for MS1553PC104

	Offset from Base Address in Standard Intel Byte-Offset Format	Offset from Base Address in CP Word-Offset Format	Notes
Memory	0 to 3FFFH	0 to 1FFFH	
BCRTM Registers	4000H to 4023H	2000H to 2011H	
Reserved Area	4024H to 47FFH	2012H to 23FFH	
Switch Register	4800H	2400H	read only
Interrupt Control Register	4802H	2401H	
Interrupt Status Register	4804H	2402H	read only
Timer Reset Register	4806H	2403H	write only
Timer Latch Register	4808H	2404H	write only
Latched Timer LSB Register	4808H	2404H	read only
Latched Timer MSB Register	480AH	2405H	read only
Compare LSB Register	480CH	2406H	
Compare MSB Register	480EH	2407H	
Discrete I/O Register	4810H	2408H	
Discrete Output Enable Reg.	4812H	2409H	
Discrete Event Trigger Compare	4814H	240AH	
Discrete Event Trigger Mask	4816H	240BH	
Discrete Event Time Tag LSB	4818H	240CH	read only
Discrete Event Time Tag MSB	481AH	240DH	read only
Reserved Area	481CH	240EH	
Reserved Area	481EH	240FH	
Reserved Area	4820H to 487FH	2410H to 243FH	
Memory, above 16KB	4880H to FFFFH	2440H to 7FFFH	32/64K only
Memory, above 64KB	10000H to 1FFFFH	8000H to FFFFH	64K only

Switch Register

For MS1553PC104 models, the switch register is in factory-programmed logic on the board; it isn't user-programmable. Bit 8 is set when activity is detected on bus A. Bit 9 is set when activity is detected on bus B. Reading this register clears these two bits.

Switch Register (32Kb)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	1	B	A	0	0	1	1	1	0	0	1

Switch Register (64Kb)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	1	B	A	0	1	0	1	1	1	1	1

Switch Register (128Kb)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	1	B	A	0	1	1	1	1	1	1	1

Interrupt Control Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	DE	TO	TC	X	X	X	X	X	X	X	X	X	X	X

DE Logic one in bit 13 enables the discrete event interrupt.

TO Logic one in bit 12 enables the timer overflow interrupt.

TC logic one in bit 11 enables the timer compare interrupt.

Interrupt Status Register (read only)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	X	X	X	X	X	X	X	DE	TO	TC	X	X

TC A logic one in bit 2 indicates that a timer compare interrupt is active.

TO A logic one in bit 3 indicates that a timer overflow interrupt is active.

DE A logic one in bit 4 indicates that a discrete event interrupt is active. A read of this register clears bits 2 through 4.

Timer

A 32-bit general-purpose timer is included on the board. The timer has a resolution of one microsecond, yielding a period of approximately 71.58 minutes before overflow. The free-running timer is cleared to a count of zero after reset or by executing a write to the Timer Reset register. The state of the counter is read by executing a write to the Timer Latch Register (this transfers the counter state to a holding register) and then reading the Timer Least Significant Word and Timer Most Significant Word (in any order). The contents of the Timer MSW and LSW registers remain unchanged until another latch timer command is executed.

An interrupt-driven timer compare function is included. With the timer compare interrupt enabled, an interrupt is generated when the timer value is equal to the value loaded into the timer compare register. Bit 2 in the Interrupt Status register is the flag that indicates that a timer compare interrupt has occurred. Bit 11 of the Interrupt register enables the timer compare interrupt.

An interrupt can be generated when the timer count overflows. Bit 3 in the Interrupt Status register is the flag that indicates that a timer overflow interrupt has occurred. Bit 12 of the Interrupt register enables the timer overflow interrupt.

Timer Reset (write only)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
No data - a write to this location resets the 32-bit, one microsecond timer															

Timer Latch (write only)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
No data - a write to this location causes the timer value to be latched. The latched data can then be read from the timer latch.															

Timer Least Significant Word (read only)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Latched timer value least significant 16-bits (read only)															

Timer Most Significant Word (read only)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Latched timer value most significant 16-bits (read only)															

Timer Compare Register Least Significant Word

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Timer compare value least significant 16-bits. An interrupt can be generated when the timer is equal to the timer compare value.															

Timer Compare Register Most Significant Word

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Timer compare value most significant 16-bits. An interrupt can be generated when the timer is equal to the timer compare value.															

Discrete Input/Output Signals

The MS1553PC104 has six general purpose, 5-volt digital discrete input/output signals. Each bit can be programmed as an output using the Output Enable register.

An interrupt can be generated as a function of one or more of the discretes. The Trigger Mask register selects which bits is used to generate the discrete event interrupt. The Trigger Compare register selects the state pattern used for the compare. The discrete event interrupt, when enabled, is generated when the discrete lines transition from a non-matching state to a matching state. The discrete lines are sampled at a 12MHz rate to detect the state change. Bit 4 in the Interrupt Status register is the flag that indicates that a discrete event interrupt has occurred. Bit 13 of the Interrupt register enables the discrete event interrupt.

When a discrete event interrupt occurs, the timer value is loaded into a special holding register. The contents of this holding register are read using the Discrete Event Time Tag registers.

The discrete lines have light pull-up resistors (approximately 100K ohms). These pull-ups allow switches and drivers that don't source current to be read directly. Each discrete can sink and source 8 mA. Board damage may occur if discrete lines are driven to a voltage less than zero volts or greater than 5.0 volts.

Discrete Data Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A write to this location sets discrete outputs, a read returns the state of the discrete lines Only the six least significant bits are used on the MS1553PC104.															

Discrete Output Enable Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A logic one enables the corresponding discrete output, and a logic zero disables the corresponding discrete output driver. This register is zero after reset. Only the six least significant bits are used on the MS1553PC104.															

Discrete Event Trigger Compare Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
This value contains the bit pattern used to compare against the discretes when generating the discrete event interrupt. Only the six least significant bits are used.															

Discrete Event Trigger Mask Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A logic one enables the corresponding discrete bit to be compared against the corresponding bit in the discrete event trigger compare register. Only the six least significant bits are used.															

Discrete Event Time Tag Least Significant Word (read only)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
This register contains the least significant 16-bits of the time tag for the last discrete event which was detected															

Discrete Event Time Tag Most Significant Word (read only)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
This register contains the most significant 16-bits of the time tag for the last discrete event which was detected															

Installation Instructions

Carefully remove the installation media and documentation and set them aside. Then follow the steps in this section:

1. Select the board address. The default address for these boards is D000H. Make a note of the address you select so that you can make it known to the software.

   The address is jumper selectable in the upper memory block (UMB) of the IBM PC. This is the address space between 640K and 1024K. This space is also used for other devices so you must be sure to select an area that is not being used by something else.

2. A program called MSD.EXE (Microsoft Diagnostics) is provided with Microsoft Windows 3.x and DOS 6.x systems and helps you to select the address. To run this program, type:

   msd

Note: This assumes the program is already on your PATH. It usually is since it is typically located in the DOS and/or Windows folder of your system.

   Select the "Memory" box and look for a free address range that is large enough for the address space of the board. It is best if your extended memory manager (e.g. QEMM, EMM386, etc.) is not running during this step.

   Avoid selecting an address that directly conflicts with any other installed devices. The selected memory address space must not directly overlap any other devices. In addition, any indirect conflict must be avoided. This occurs when a 16-bit memory-mapped device (such as the MS1553PC104-3 or -4) is placed in the same 128K region (i.e. segment address ranges: A000-BFFF or C000-DFFF or E000-FFFF) as an 8-bit memory-mapped device. This is caused by an inherent architectural limitation of the ISA bus.

   When selecting a base memory address, be aware of the potential for conflict with expanded memory managers (e.g. EMM386, QEMM386) which also use the UMB. You can avoid a problem by configuring the memory manager to exclude the address space you select from its list of available mapping regions. For example, when starting EMM386, use the command line switch: /X=D000-D7FF for a 32KW board at the default address of D000H. (You may instead use the command line switch: /X=D000-D4FF, which permits EMM386 to use the unused area of the 8KW board from D500-D7FF). See your memory manager documentation for a complete discussion of these switches. Memory managers are typically loaded at system startup from CONFIG.SYS.

3. Insert the installation media and install the software. For Win32 platforms, the installation program should run automatically following insertion of the CD. If it does not, execute Setup.exe from the

\Setup\Disk1 folder. For Win16 and DOS platforms, copy the entire contents of the folder \MS1553-SW to your hard drive using xcopy or the file manager.

4. Remove the MS1553PC104 board from the anti-static bubble pack and lay it on top of the bag. This prevents static discharge from damaging the board when you change the jumpers.

   Refer to the "" section to set the jumpers on the board to the desired settings for your operating environment.

5. Install this board in a 16-bit PC/104 stack. At this time it is installed and ready for use.

6. Connect the 1553 bus to the connector on the interface card. Note that if this is the last connection on the bus, a 70-ohm terminator should be connected to provide the proper termination for the bus.

7. Power-up the computer.

To test the configuration:

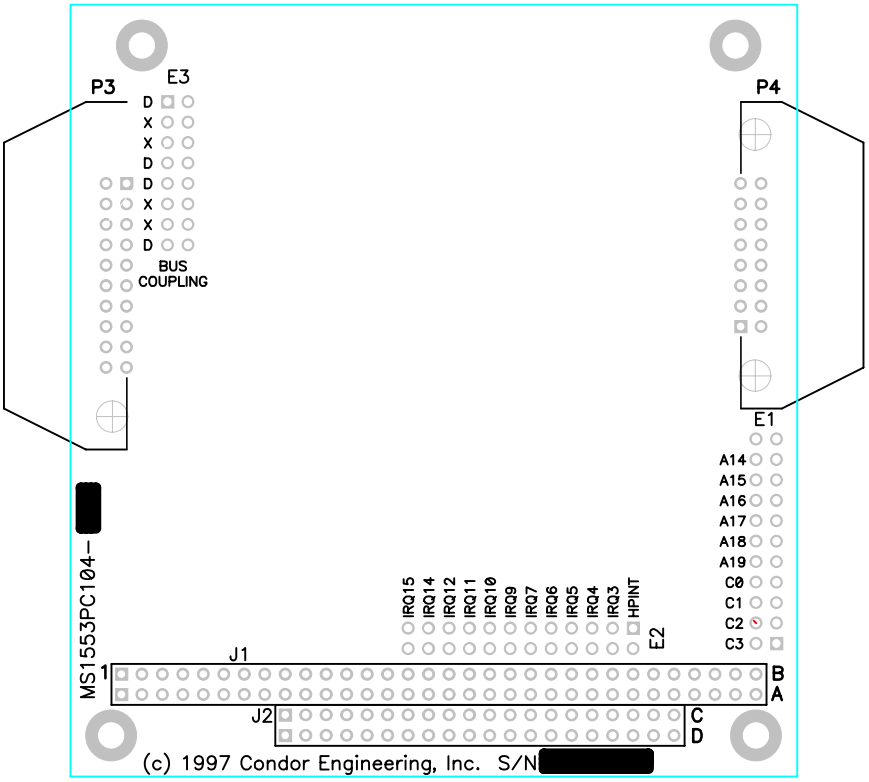
1. Execute the Test Config program for your host operating system. On a Win32 platform this program is named tstcfg32.exe, located in the folder \Examples\C\Win32 and accessed from the Test Config shortcut. For Win3.x and DOS it is named tstcfg16.exe or tstcfg.exe under the \Examples\C\Win16 and \Examples\C\DOS folders, respectively.

2. Test Config will prompt for the information concerning your board's configuration. The system then tests the configuration. If there is a problem, diagnostic information is printed on the screen. The most likely problems are mapping the board to the same memory space as another card or the same interrupt as another card.

3. Once these steps are successfully complete the board is ready for use.

MS1553PC104 Jumper Settings

This section provides jumper settings for the MS1553PC104.



The MS1553PC104 interface board contains four jumper blocks used to determine certain special information required by the hardware.

Board Base Address

Jumper Block E1 determines the board address as follows. If a jumper is on or installed, the corresponding address bit is '0'. Remove the jumper to set the corresponding address to a '1'.

Jumper	Board Label	Function
E1-21 to E1-22	C3 (note 1)	Configuration bit 3 (note 3)
E1-19 to E1-20	C2 (note 2)	Configuration bit 2 (note 3)
E1-17 to E1-18	C1	Configuration bit 1 (note 3)
E1-15 to E1-16	C0	Configuration bit 0 (note 3)
E1-13 to E1-14	A19	Base address bit 19
E1-11 to E1-12	A18	Base address bit 18
E1-9 to E1-10	A17	Base address bit 17
E1-7 to E1-8	A16	Base address bit 16

Jumper	Board Label	Function
E1-5 to E1-6	A15	Base address bit 15
E1-3 to E1-4	A14	Base address bit 14
E1-1 to E1-2	*blank*	reserved - leave open

Notes:

1. The C3 jumper controls the operation of the MEMCS16* signal. When the C3 jumper is not installed, MEMCS16* is driven per specification, strictly as a function of the LA address lines. With the jumper not installed, the MS1553PC104 board cannot reside in a 128 Kbyte memory region shared with 8 bit devices. When the C3 jumper is installed, the board also decodes the upper SA address lines when driving the MEMCS16* signal, thus allowing the board to co-exist with 8 bit devices in the same 128 Kbyte memory region. However, the MEMCS16* timing is no longer "per spec", though many ISA boards operate this way. It is recommended that the C3 jumper be installed.

2. This jumper is not currently used, leave OPEN.

3. MS1553PC104 Configuration bit settings are shown in the following table:

Jumpers C1 and C0	Available Board Memory and required bus address space
In, In	32Kb - 8Kwords
Out, In	64Kb - 32Kwords
In, Out	32Kb - 16Kwords
Out, Out	128Kb - 64Kwords

Memory contention is the most common reason that a PC adapter board fails after installation. Run a utility, such as MSD.EXE to determine your current memory usage before installing the board. Most upper memory block usage has been standardized for common peripherals, but D000H through EFFFH is reserved for specialized adapter boards. The board allows maximum flexibility in configuring memory usage.

We ship all boards from the factory at D000H in the 32Kb configuration.

Interrupt Line Select (IRQ3-7, 9-12,14-15)

Jumper block E1 is used to select the interrupt line to be used. Interrupt conflict is one of the most likely reasons that PC adapter boards do not work when first installed. Check your PC manual for available interrupts. Usually, IRQ 5 is not being used, but the Condor board allows you to choose any interrupt that is available on your PC.

Jumper	Board Label	Function
E2-23 to E2-24	IRQ15	Select interrupt IRQ15 (note 1)
E2-21 to E2-22	IRQ14	Select interrupt IRQ14 (note 1)
E2-19 to E2-20	IRQ12	Select interrupt IRQ12 (note 1)
E2-17 to E2-18	IRQ11	Select interrupt IRQ11 (note 1)
E2-15 to E2-16	IRQ10	Select interrupt IRQ10 (note 1)
E2-13 to E2-14	IRQ9	Select interrupt IRQ9 (note 1)
E2-11 to E2-12	IRQ7	Select interrupt IRQ7 (note 1)
E2-9 to E2-10	IRQ6	Select interrupt IRQ6 (note 1)
E2-7 to E2-8	IRQ5	Select interrupt IRQ5 (note 1)
E2-5 to E2-6	IRQ4	Select interrupt IRQ4 (note 1)
E2-3 to E2-4	IRQ3	Select interrupt IRQ3 (note 1)
E2-1 to E2-2	HPINT	Enable BCRTM high priority interrupts

Note 1: Only one IRQ jumper should be installed.

High-Priority Interrupt Enable

The high-priority interrupt enable jumper is used for enabling the high priority interrupts for the board. If you do not wish to service these interrupts from the BCRTM chip (and you do not wish the driver software to service them), disable them by removing the jumper.

These interrupts are:

- Dynamic bus control acceptance.
- Illogical command error.
- Subsystem fail.
- BIT fail.
- End of BIT.
- Data overrun.

The normal position for this jumper should be "on", which enables the interrupt. If the jumper is *installed*, then these interrupts can be individually enabled, read, and disabled by the driver software.

When using the Condor-supplied software library, ensure that the high-priority interrupt is enabled. This is expected by the software, and gives you the option of disabling or enabling the high-priority interrupts in the software driver.

Bus Coupling

The bus outputs can be coupled in two different ways:

- Direct coupled (short stub)
- Transformer coupled (long stub)

Use direct coupling when the distance from the isolation transformer to the data bus does not exceed one foot. Use transformer coupling when the distance exceeds one foot but is less than 25 feet. There are isolation transformers on the board for both direct and transformer coupling.

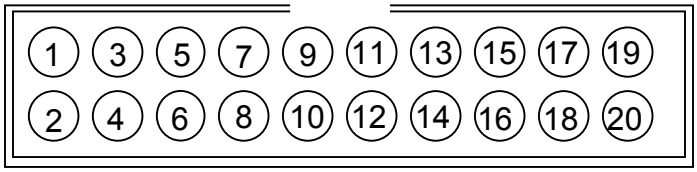
There are 53.6 ohm isolation resistors on board for direct coupling. Transformer coupling requires an external transformer and isolation resistors at the junction to the main 1553 bus to match the stub impedance to the bus.

Jumper block E3 is for A bus coupling.

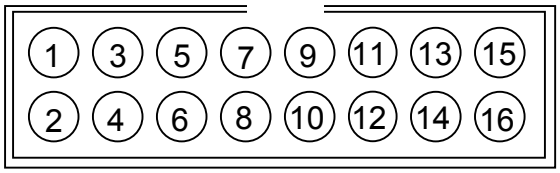
Jumper	Board Label	Bus Coupling
E3-8 to E3-9	D	direct coupling (short stub)
E3-7 to E3-10	X	transformer coupling (long stub)
E3-6 to E3-11	X	transformer coupling (long stub)
E3-5 to E3-12	D	direct coupling (short stub)
E3-4 to E3-13	D	direct coupling (short stub)
E3-3 to E3-14	X	transformer coupling (long stub)
E3-2 to E3-15	X	transformer coupling (long stub)
E3-1 to E3-16	D	direct coupling (short stub)

Input/Output Connector Pin Definition

This section provides I/O connector pin definitions for the MS1553PC104. Pin 1 is identified by a square pad on the solder side of the board. The embossed triangle-shaped notch on the plastic connector housing should be ignored.



Front View - P3 Connector



Front View - P4 Connector

Pin	Function	Pin	Function
P3-1		P4-1	Discrete I/O Bit 0
P3-2		P4-2	GND
P3-3		P4-3	Discrete I/O Bit 1
P3-4		P4-4	GND
P3-5	Channel A +	P4-5	Discrete I/O Bit 2
P3-6	Channel A -	P4-6	GND
P3-7	Channel A shield	P4-7	Discrete I/O Bit 3
P3-8		P4-8	GND
P3-9	Channel B +	P4-9	Discrete I/O Bit 4
P3-10	Channel B -	P4-10	GND
P3-11	Channel B shield	P4-11	Discrete I/O Bit 5
P3-12		P4-12	GND
P3-13	Terminal Address Parity	P4-13	
P3-14	Terminal Address Bit 4	P4-14	GND
P3-15	Terminal Address Bit 3	P4-15	
P3-16	Terminal Address Bit 2	P4-16	GND
P3-17	Terminal Address Bit 1		
P3-18	Terminal Address Bit 0		
P3-19	GND		
P3-20	GND		

Terminal Address Bits

The terminal address bits are programmed to a logic zero by connecting to GND. They are programmed to a logic one by leaving them unconnected.

Condor Engineering-supplied API software doesn't directly support using the hardware Terminal Address bits, since the software first clears the register containing these bits during self-test and initialization, then re-programs the register with the value supplied by the user in the ms1553m_init() function call.

If this function is to be used by user software, than you must supply software to read and save the value during boot up of the host computer, before any ms1553m_*() functions are called. Then, when calling ms1553m_init() the saved value is passed to the API which then initializes the RT address to the specified value.

CHAPTER 6

MS1553VMX Installation

Introduction

This chapter provides hardware installation and software development information for the MS1553VMX board. This product is a second-generation, enhanced version of the MS1553VME board described in the chapter, “MS1553VME Installation”.

Hardware Overview

The MS1553VMX board is configured as an A24/D16 slave board that is memory mapped to any 32KW in A24 address space. The address select jumper block (E1) on the board selects A23 through A16 of the address map. See the memory map on the following page.

The BCRTM sees memory as 16-bit words only. If you try to address the board in either byte or long word (32-bit) modes, you cause a bus error. If the VME bus were allowed to write only bytes into the dual-ported memory, a synchronization problem would exist where the VME bus could update only one byte of a word while the BCRTM could read the entire word with only part of it being correct.

Switch Register

The switch register (SW2) is "read" only and indicates the following configuration information to the software:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	1	B	A	1	1	1	1	1	1	1	0

The switch register is in factory programmed logic on the board, and isn't user-programmable. Bit 8 is set when activity is detected on bus A. Bit 9 is set when activity is detected on bus B. Reading this register clears these two bits.

Interrupts

The MS1553VMX board functions as a VME D16 Interrupter module. Based on the configuration of the BCRTM chip, the MS1553VMX board can generate an interrupt on the VME bus. The interrupts on the board are selected by writing the status/ID to the 16-bit interrupt register located at BA+MSA+2401H. This register may only be read during the interrupt acknowledge cycle for the board.

Interrupt Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	HP	X	TO	TC	INT			ID							

ID Interrupt ID: Bits 0 through 7 represent the 8-bit interrupt vector, with bit 0 representing the LSB. The conditions that generate an interrupt are controlled by software controlled bits in the BCRTM chip registers. See the UTMC BCRTM data sheet for more information.

INT Interrupt Number: Bits 8 through 10 of the register represent the interrupt number in binary format. Valid values are 001 (IRQ1) through 111 (IRQ7).

TC Timer Compare: A logic "1" in bit 11 enables the timer compare interrupt.

HP High Priority Interrupt Enable: A logic "1" in bit 14 enables high priority interrupts from the BCRTM.

Bits 13 and 15 are not used. It is recommended that you write "0's" to these bits so that your driver and interrupt handler remain compatible with possible future upgrades.

The board is an Interrupt On Register Acknowledge (RORA) module. This means that resetting a bit in the High-Priority Interrupt Status/Reset Register of the BCRTM chip clears the interrupt.

Interrupt Status Register (read only)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	X	X	X	X	X	X	X	X	TO	TC	HP	ST

ST A logic one in bit 0 indicates that a standard priority interrupt from the BCRTM is active.

HP A logic one in bit 1 indicates that a high priority interrupt from the BCRTM is active.

TC A logic one in bit 2 indicates that a timer compare interrupt is active.

TO A logic one in bit 3 indicates that a timer overflow interrupt is active. A read of this register clears bits 2 and 3.

Memory Map

MS1553VMX memory locations in standard VME address space (A24)	Offset from Base Address in Standard Intel Byte-Offset Format	Offset from Base Address in CP Word-Offset Format	Notes
Memory	0 to 3FFFH	0 to 1FFFH	
BCRTM Registers	4000H to 4023H	2000H to 2011H	
Reserved Area	4024H to 47FFH	2012H to 23FFH	
Switch Register	4800H	2400H	read only
Interrupt Control Register	4802H	2401H	
Interrupt Status Register	4804H	2402H	read only
Timer Reset Register	4806H	2403H	write only
Timer Latch Register	4808H	2404H	write only
Latched Timer LSB Register	4808H	2404H	
Latched Timer MSB Register	480AH	2405H	read only
Compare LSB Register	480CH	2406H	
Compare MSB Register	480EH	2407H	
Reserved Area	4810H to 487FH	2408H to 243FH	
Memory, above 16KB	4880H to FFFFH	2440H to 7FFFH	

MS1553VMEL memory locations in short VME address space (A16)	Offset from Base Address in Standard Intel Byte-Offset Format	Offset from Base Address in CP Word-Offset Format
VXI ID/Logical Address Register	0	0
VXI Device Type	2	1
VXI Status/Control Register	4	2
VXI Offset Register	6	3
VXI unused registers	8H - 3FH	4 - 1FH

Timers

A 32-bit general-purpose timer is included on the board. This timer has a resolution of one microsecond, yielding a period of approximately 71.58 minutes before overflow. The free-running timer is cleared to a count of zero after reset or by executing a write to the Timer Reset register. The state of the counter is read by executing a write to the Timer Latch Register (this transfers the counter state to a holding register). Then read the Timer Least Significant Word and Timer Most Significant Word (in any order). The contents of the Timer MSW and LSW registers remain unchanged until another latch timer command is executed.

An interrupt-driven timer compare function is included. When the timer compare interrupt enabled, an interrupt is generated when the timer value is equal to the value loaded into the timer compare register. Bit 2 in the Interrupt Status register is the flag that indicates that a timer compare interrupt has occurred. Bit 11 of the Interrupt register enables the timer compare interrupt.

An interrupt can be generated when the timer count overflows. Bit 3 in the Interrupt Status register is the flag that indicates that a timer overflow interrupt has occurred. Setting bit 12 of the Interrupt register enables the timer overflow interrupt.

Timer Reset (write only)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
No data - a write to this location resets the 32-bit, one microsecond timer															

Timer Latch (write only)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
No data - a write to this location causes the timer value to be latched. The latched data can then be read from the timer latch.															

Timer Least Significant Word (read only)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Latched timer value least significant 16-bits (read only)															

Timer Most Significant Word (read only)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Latched timer value most significant 16-bits (read only)															

Timer Compare Register Least Significant Word

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Timer compare value least significant 16-bits. An interrupt can be generated when the timer is equal to the timer compare value.															

Timer Compare Register Most Significant Word

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Timer compare value most significant 16-bits. An interrupt can be generated when the timer is equal to the timer compare value.															

MS1553VMX VXI Compatibility

The MS1553VMX includes the required VXI register-based device registers. The base address for these registers is programmed using jumper block E2. You enable the VXI registers by installing at least one jumper on jumper block E2. The base address for the VXI registers is 0xC000 plus an offset defined by E2 (see the section, "MS1553VMX Jumper Settings" for details). The base address of the operational registers is selected using the VXI offset register; jumper block E1 is not be used.

If no jumpers are installed on jumper block E2, the board does not respond to short VME address space accesses. The base address for the operational registers is selected by jumper block E1.

VXI ID/Logical Address Register (read only)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Device Class (register-based)		Address Space (A16/ A24)		Manufacturer ID (3732 or E94H for Condor Engineering)											
1	1	0	0	1	1	1	0	1	0	0	1	0	1	0	0

VXI Device Type Register (read only)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Required Memory (64K bytes)				Model Code (256 for MS1553PC104)											
0	1	1	1	0	0	0	1	0	0	0	0	0	0	0	0

VXI Status/Control Register (read)

15	14	13 - 4	3	2	1	0
A24/A32 Active	MODID*	Device Dependent	Ready	Passed	Device Dependent	
(Zero at reset)	0	0x000	1 (This board does not automatically execute self test at startup)	1 (This board does not automatically execute self test at startup)	0	0

VXI Status/Control Register (write)

15	14 - 2	1	0
A24/A32 Active	Device Dependent	Sysfail Inhibit	Reset
1=active, 0=inactive	*unused*	*unused (board does not drive SYSFAIL*)*	*unused*

MS1553VMX Installation

VXI Offset Register

15 - 8	7 - 0
Address bits 23 - 16	Address bits 15 - 8
These define the base address for the A24 operational registers	*unused (board requires 64K bytes of address space)*

MS1553VMX Installation

This section tells how to install the MS1553VMX board. All jumper, and configuration file settings are defined.

Please take a moment to familiarize yourself with the installation procedure to insure the proper operation of your 1553 interface card.

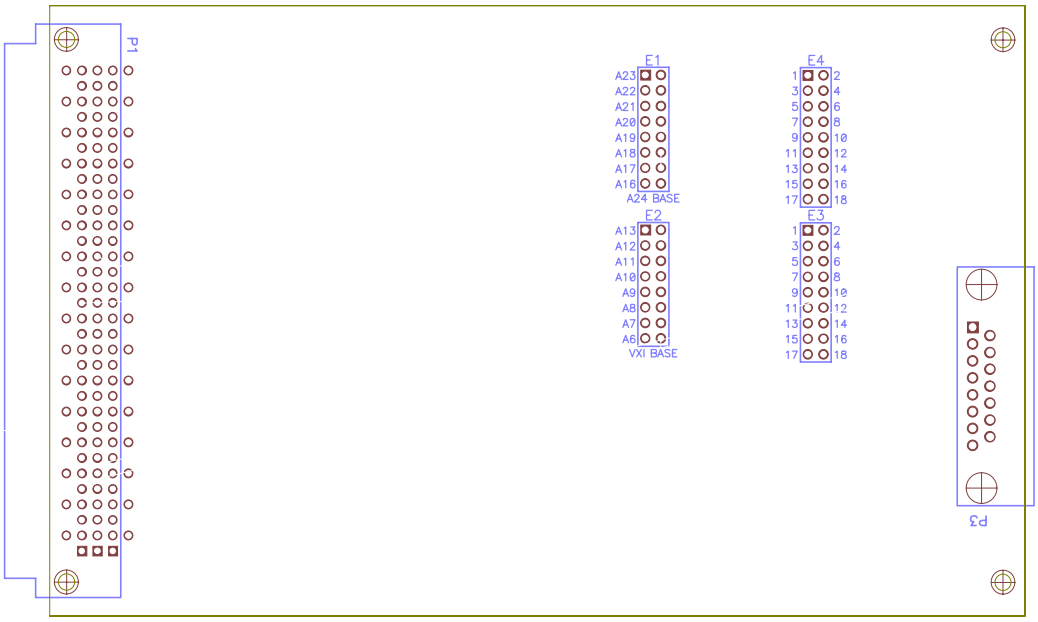
1. Inspect the packing material to ensure the board has not been damaged in shipment. If it has, please notify the shipping company immediately.
2. Carefully remove the documentation and the distribution media.
3. Remove the MS1553VMX board from the anti-static bubble-pack bag and lay the board on top of the bag. This prevents static discharge from damaging the board while the switches are being set.
4. Refer to the section, “MS1553VMX Jumper Settings”, to set the jumpers on the board to the desired settings for your operating environment.
5. Remove the blank mounting plate from one of the unused slots in your computer. Install the MS1553VMX board and bolt the faceplate to the VME card frame.
6. Connect your 1553 bus to the connectors provided on the front panel of the interface card.

   If this is the last connection on the bus, a 70-ohm terminator must be connected to provide the proper termination for the bus.
7. Power-up the computer. Now, you are ready to test the configuration.

MS1553VMX Installation

MS1553VMX Jumper Locations

This section provides jumper settings for the MS1553VMX.



Performing Diagnostics

A source file is supplied via the distribution media named "tstcfg" to perform diagnostics and report configuration information for the board.

If you choose to use this program, you must either upload the file to your VME computer and build an executable, or modify the program for your operating environment. If your environment is Microware's real-time operating system OS9, you can obtain an OS9-formatted Condor diskette containing a file called tstcfg.

To run tstcfg:

1. Execute tstcfg.exe.
2. Press **Enter**.

   You are asked for the information concerning your configuration. The configuration is then tested.

   If there is a problem, diagnostic information is printed on the screen. The most likely problem you will encounter is mapping the board to the same memory space as another card or the same interrupt as another card.
3. Once this step is complete, the hardware installation is complete.

MS1553VMX Jumper Settings

The MS1553VMX interface board contains four jumper blocks used to determine certain special information required for the hardware to function properly in the environment that it is to be used. Remember to record the settings so that you will not have to remove the cover from your VME chassis again.

Jumpers E3 and E5 Bus Coupling

The two typical bus-coupling configurations are as follows:

Jumper	Direct coupled (short stub)	Transformer coupled (long stub)
E3-17 to E3-18		
E3-15 to E3-16	Installed	
E3-13 to E3-14		
E3-11 to E3-12		Installed
E3-9 to E3-10		Installed
E3-7 to E3-8		
E3-5 to E3-6	Installed	
E3-3 to E3-4		
E3-1 to E3-2	Installed	Installed
E4-17 to E4-18		
E4-15 to E4-16	Installed	
E4-13 to E4-14		
E4-11 to E4-12		Installed
E4-9 to E4-10		Installed
E4-7 to E4-8		
E4-5 to E4-6	Installed	
E4-3 to E4-4		
E4-1 to E4-2	Installed	Installed

Jumper Block E1 – A24, Board Address

Jumper	Board Label	Function
E1-15 to E1-16	A16	Base address bit 16
E1-13 to E1-14	A17	Base address bit 17
E1-11 to E1-12	A18	Base address bit 18
E1-9 to E1-10	A19	Base address bit 19
E1-7 to E1-8	A20	Base address bit 20
E1-5 to E1-6	A21	Base address bit 21
E1-3 to E1-4	A22	Base address bit 22
E1-1 to E1-2	A23	Base address bit 23

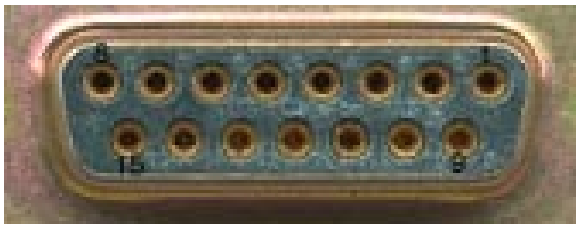
Jumper Block E2 – VXI Registers Base Address

Jumper	Board Label	Function
E2-15 to E2-16	A6	VXI Base address bit 6
E2-13 to E2-14	A7	VXI Base address bit 7
E2-11 to E2-12	A8	VXI Base address bit 8
E2-9 to E2-10	A9	VXI Base address bit 9
E2-7 to E2-8	A10	VXI Base address bit 10
E2-5 to E2-6	A11	VXI Base address bit 11
E2-3 to E2-4	A12	VXI Base address bit 12
E2-1 to E2-2	A13	VXI Base address bit 13

Note: This jumper block configures the base address for the VXI registers in short VME address space. The two most significant address bits (bits 14 and 15) are always a logic one. An installed jumper programs the address bit to a zero and an open jumper programs the bit to a one. Leaving all of the VXI base address jumpers open disables the VXI registers.

Input/Output Connector Pin Definition

The standard configuration of the MS1553VMX provides its I/O on two BNC type Twinax connectors that are clearly marked Bus A and Bus B on the end panel. Some configurations provide the I/O on a 15-pin subminiature D connector. These connections are defined as follows:



Pin	Function
1	Channel B +
2	Channel B -
3	GND
4	Channel A +
5	Channel A -
6	Channel A shield
7	GND
8	GND
9	Channel B shield
10	Terminal Address Bit 0
11	Terminal Address Bit 1
12	Terminal Address Bit 2
13	Terminal Address Bit 3
14	Terminal Address Bit 4
15	Terminal Address Parity

The terminal address bits are programmed to a logic zero by connecting to GND and are programmed to a logic one by leaving them unconnected.

Interface Software

The basic architecture of the MS1553VMX is virtually identical to the architecture of the MS1553PC. For this reason, the source code for the MS1553PC utility libraries is included with each MS1553VMX. These routines are documented in the C Driver section of this manual and can be modified to be compatible with the operating system and compiler you are using. These routines have been built with standard IBM PC-based development tools: Microsoft and Borland C compilers, so you may need to make some modifications.

Machine- and Operating-System-specific changes are normally necessary in the following areas:

- **Syntax** – The supplied source code conforms to the Microsoft and Borland "C/C++" compiler's syntax requirements for a "C" program. The only "C++" extension being used is the use of the "C++" style comments, which use a "//" to begin a comment which extends for the remainder of the line.
- **16-bit vs. 32-bit compilers** – The supplied source code can be compiled for either 16- or 32-bit environments. Most likely the 32-bit selection is more useful, since in that mode the "C" extensions for the PC platform addressing are disabled.
- **Mapping the board into host processor space** – You need to supply a function that takes a physical address of the board and returns a "C" pointer that is valid for accessing the board.
- **Byte vs. Word Board Accesses** – The MS1553 series of products must not be accessed as bytes.
- **Interrupt Function** – The supplied source code depends on hardware interrupts. The supplied function needs to be called by the Operating System when the board generates an interrupt.

CHAPTER 7

MS1553VME Installation

Introduction

This chapter provides hardware installation and software development information for Condor's MS1553VME board.

Hardware Overview

The MS1553VME board is configured as an A24/D16 slave board that is memory mapped to any 16KW (or 32KW) block boundary from the base address of the A24/D16 memory map through the base address plus 0x008000H (or 0x010000H). The address select DIP switch (SW4) on the board selects A23 through A15 of the address map. The configuration switch (SW2) determines if the board is mapped into 16KW or 32KW. The first 8KW of the MS1553VME's address space accesses the first 8KW of RAM. The BCRTM registers, the Switch register, the interrupt register, and a small reserved area follow. Finally, the memory above 8KW (on MS1553VME-2 and MS1553VME-4 versions only) is mapped. This scheme makes 31,583 words useful of the 32KW RAM. See the memory map on the following page.

The BCRTM sees memory as 16-bit words only. If you try to address the board in either byte or long word (32-bit) modes, you cause a bus error. If the VME bus were allowed to write only bytes into the dual-ported memory, a synchronization problem would exist where the VME bus could update only one byte of a word while the BCRTM could read the entire word with only part of it being correct.

Switch Register

The switch register (SW2) is "read" only and indicates the following configuration information to the software:

bit 0	Reserved
bit 1	8 KW RAM (0) or 32 KW RAM (1)
bit 2	32 KB Window (0) or 64 KB Window (1)
bit 3	Reserved
bit 4-15	Undetermined

The four switches are located in the four LSB's of the base address plus 2400H (BA+MSA+2400H). See the MS1553 DIP Switch Settings section of this document for details concerning the meaning of these switches. A closed switch is a zero and an open switch is a one.

Interrupts

The MS1553VME board functions as a VME D16 Interrupter module. Based on the configuration of the BCRTM chip, the MS1553VME board can generate an interrupt on the VME bus. The interrupts on the board are selected by writing the status/ID to the 16-bit interrupt register located at BA+MSA+2401H. This register may only be read during the interrupt acknowledge cycle for the board. Bit 15 is "read only" and a logic "1" means that the interrupt register has been written to. It will always be "1" when read during the interrupt acknowledge cycle. Bit 14 is the High Priority Interrupt Enable. Writing a "1" in this bit position enables high priority interrupts; writing a "0" disables them. Bits 11 through 13 are not used and it is recommended that you write "0's" to these bits so that your driver and interrupt handler will remain compatible with possible future upgrades. Bits 8 through 10 of the register represent the interrupt number in binary format. Valid values are 001 (IRQ1) through 111 (IRQ7). Bits 0 through 7 represent the 8-bit interrupt vector, with bit 0 representing the LSB. The conditions that generate an interrupt are controlled by software controlled bits in the BCRTM chip registers. See the UTMC BCRTM data sheet for more information.

The interrupt register contains the following data:

bits 0 - 7	Interrupt ID (0 to 255)
bits 8 - 10	Interrupt number (IRQ1 - 7)
bits 11 - 13	Reserved
bit 14	High Priority interrupt enable
bit 15	Interrupt Vector Set flag

The board is an Interrupt On Register Acknowledge (RORA) module. This means that resetting a bit in the High-Priority Interrupt Status/Reset Register of the BCRTM chip clears the interrupt.

Memory Map (Word Offset Format)

Base Address (BA)	Based on memory map of VME bus for A24/D16 data transfers
Memory Select Address (MSA)	Address selected by SW4 switches in address select table
Memory	(BA + MSA) through (BA+MSA+1FFFH)
BCRTM Registers	(BA+MSA+2000H) through (BA+MSA+2011H)
Reserved	(BA+MSA+2012H) through (BA+MSA+23FFH)
Switch Register	(BA+MSA+2400H)
Interrupt Register	(BA+MSA+2401H)
Reserved	(BA+MSA+2402H) to (BA+MSA+243FH)
Memory, above 8KW	(BA+MSA+2440H) to (BA+MSA+7FFFH)

Note: "+" means "OR".

MS1553VME Installation

This section tells how to install the MS1553VME board. All DIP switch, jumper, and configuration file settings are defined.

Please take a moment to familiarize yourself with the installation procedure to insure the proper operation of your 1553 interface card.

1. Inspect the packing material to ensure the board has not been damaged in shipment. If it has, please notify the shipping company immediately.
2. Carefully, remove the documentation and the distribution media.
3. Remove the MS1553VME board from the anti-static bubble-pack bag and lay the board on top of the bag. This prevents static discharge from damaging the board while the switches are being set.
4. Refer to the section, “MS1553VME DIP Switch Settings”, to set the DIP switches on the board to the desired settings for your operating environment.

5. Remove the blank mounting plate from one of the unused slots in your computer. Install the MS1553VME board and bolt the faceplate to the VME card frame.
6. Connect your 1553 bus to the connectors provided on the front of the interface card.

   The connectors are marked bus A and B. If this is the last connection on the bus, use a "Tee" and connect a 70-ohm terminator to provide the proper termination for the bus. Otherwise, you should use a "Tee" with a cable on each end of the "Tee".
7. Power-up the computer. Now, you are ready to test the configuration.

Performing Diagnostics

A source file is supplied via the distribution media named "tstcfg" to perform diagnostics and report configuration information for the board.

If you choose to use this program, you must either upload the file to your VME computer and build an executable, or modify the program for your operating environment. If your environment is Microware's real-time operating system OS9, you can obtain an OS9-formatted Condor diskette containing a file called tstcfg.

To run tstcfg:

1. Execute tstcfg.exe.
2. Press **Enter**.

   You are asked for the information concerning your configuration. The configuration is then tested.

   If there is a problem, diagnostic information is printed on the screen. The most likely problem you will encounter is mapping the board to the same memory space as another card or the same interrupt as another card.
3. Once this step is complete, remove the software.

   The hardware installation is complete.

MS1553VME DIP Switch Settings

The MS1553VME interface board contains four DIP switches used to determine certain special information required for the hardware to function properly in the environment that it is to be used. Remember to record the settings so that you don’t have to remove the cover from your VME chassis some time later. For all of the following, the switch setting "off" is synonymous with "open"; "on" is synonymous with "closed".

Switch SW1 – Channel A Bus Coupling

The bus outputs can be coupled in four different ways:

- Direct to the bus transceiver chip.
- Direct bus connection via the transformer.
- Long stub transformer connection via series resistors.
- Long stub transformer connection with no series resistors.

The typical connections are items 2 and 4. Item 4 requires an external transformer at the junction to the main 1553 bus. This transformer is required to match the stub impedance to the bus. Item 1 should not be connected to a standard 1553 bus, but can be connected to other units if they are using the same mode of direct connection. Each selection requires two switches be "on" while the rest are "off". Switch 1 (SW1) is the 10-position switch paralleling several resistors along the left side of the board (when looking at the component surface).

SW 1 Switch numbers								
	1	2	3	4	5	6	7	8
Item 1	on	off	off	off	off	off	off	on
Item 2	off	on	off	off	off	off	on	off
Item 3	off	off	on	off	off	on	off	off
Item 4	off	off	off	on	on	off	off	off

Switch positions 9 and 10 connect the transformer shields to the twinax cable shields for the A and B channels, respectively. If the switch is turned on, it connects the shield.

Switch SW3 – Channel B Bus Coupling

The bus outputs can be coupled in four different ways:

- Direct to the bus transceiver chip.
- Direct bus connection via the transformer.
- Long stub transformer connection via series resistors.
- Long stub transformer connection with no series resistors.

The typical connections are items 2 and 4. Item 4 requires an external transformer at the junction to the main 1553 bus. This transformer is required to match the stub impedance to the bus. Item 1 shouldn’t be connected to a standard 1553 bus, but can be connected to other units if they are using the same mode of direct connection. Each selection requires two switches be "on" while the rest are "off".

	SW 2 Switch numbers							
	1	2	3	4	5	6	7	8
Item 1	on	off	off	off	off	off	off	on
Item 2	off	on	off	off	off	off	on	off
Item 3	off	off	on	off	off	on	off	off
Item 4	off	off	off	on	on	off	off	off

Switch SW4 – Board Address

SW4 is a ten-position DIP switch located close to the bottom right-hand corner of the board when the board is turned with the top plane facing up. Use the words "MS1553VME C1990 Condor Engineering Inc." to reference the bottom of the board.

Switch positions one through nine control the board address. The switches are set to match (BA + MSA) as described in the MS1553 Address Mapping Table. The switch positions in the table below correspond to the address bits A23 through A15. A closed (ON) switch position represents a logic 0; an open switch represents a logic 1. Switch position 10 is not used.

Switch Position	1	2	3	4	5	6	7	8	9	10
Address Bit	A23	A22	A21	A20	A19	A18	A17	A16	A15	0

For example, if the beginning of the A24/D16 memory map starts at 7F0000H and you want the board to respond to addresses starting at 7F8000H, set switch positions 2 through 9 to an open (OFF) setting. If you want to respond to addresses starting at 808000H, only switch positions 1 and 9 should be open.

Switch SW2 – Configuration Information

SW2 switches one through four contain configuration information that is read back by the software to verify what options are in use. Not all of the options exist for every board. The last column of the table will indicate the revision of the board required to support the option.

Option	Switch	On/Off	Model	Revision
reserved	4	off	all	all
8KW RAM	3	On	all	all
32KW RAM	3	off	VME-2,-4	all
32KB Window	2	on	all	all
64KB Window	2	off	VME-2,-4	
reserved	1	on	all	all

Switches 1 and 4 are reserved for future board options. Switch 3 informs the driver of the amount of static RAM that is on the board. The MS1553VME-1 and MS1553VME-3 models support only 8KW. The MS1553VME-2 or MS1553VME-4 models have 32KW on-board RAM and may use either 8KW or 32KW. Switch 2 determines the window size available in which to map the board. Models MS1553VME-1 and MS1553VME-3 require a 32KB window. Models MS1553VME-2 and MS1553VME-4 normally use a 64KB window. If only 32KB of memory space is available, the switch can be set "on" but is not able to access the top 16KW of RAM.

For models MS1553VME-1 and MS1553VME-3 switches 1-3 are in the "on" position. For models MS1553VME-2 and MS1553VME-4 switch 1 is "on" and switches 2 and 3 typically are "off".

Interface Software

The basic architecture of the MS1553VME is virtually identical to the architecture of the MS1553PC. For this reason, the source code for the MS1553PC utility libraries is included with each MS1553VME. These routines are documented in the C Driver section of this manual and can be modified to be compatible with the operating system and compiler you are using. These routines have been built with standard IBM PC-based development tools – Microsoft and Borland C compilers, so some modifications may need to be made by the end user.

Machine- and Operating System-specific changes are normally necessary in the following areas:

- **Syntax** – The supplied source code conforms to the Microsoft and Borland “C/C++” compiler’s syntax requirements for a “C” program. The only “C++” extension being used is the use of the “C++” style comments, which use a “//” to begin a comment which extends for the remainder of the line.
- **16-bit vs. 32-bit compilers** – The supplied source code can be compiled for either 16- or 32-bit environments. Most likely the 32-bit selection is more useful, since in that mode the “C” extensions for the PC platform addressing are disabled.

- **Mapping the board into host processor space** – You need to supply a function that takes a physical address of the board and returns a "C" pointer that is valid for accessing the board.
- **Byte vs. Word Board Accesses** – The MS1553 series of products must not be accessed as bytes.
- **Interrupt Function** – The supplied source code depends on hardware interrupts. The supplied function needs to be called by the Operating System when the board generates an interrupt.

CHAPTER 8

MS1553MC Installation

This section tells how to install the MS1553MC board. All DIP switch, jumper, and configuration file settings are defined.

If possible, use a static free workstation. Before beginning the installation process, inspect the packaging material to insure that the board hasn't been damaged during shipment. If it has, contact the shipper right away.

System Requirements

You need the following equipment to install the Condor MS1553MC board.

- IBM MCA computer or compatible.
- Condor MS1553MC-1 or MS1553MC-2 board.
- Two pigtail cables terminated with a 9-pin D connector on one end and a Trumpeter twinax BNC connector on the other end. These cables are supplied with the board.
- Condor-supplied software (MS1553) on one 5 1/4" high density diskette or one 3 1/2" high density diskette. The MS1553 software includes a Microsoft C library, a Borland C/C++ library, a Microsoft WINDOWS™-compatible DLL, CP, configuration and test software, and an ADF file.
- Working copy of your MCA Reference Diskette.

MS1553MC Hardware Installation

Inspect the packing material to insure the board has not been damaged in shipment. If it has, please notify the shipping company immediately. Then follow the steps in this section:

1. Remove the MS1553MC board from the anti-static bubble-pack bag and lay the board on top of the bag. This will prevent static discharge from damaging the board while the switches are being set.
2. Set the DIP switches. See DIP Switch Settings below.
3. Remove the cover from the MCA computer.
4. Remove the blank mounting plate from one of the unused adapter slots in your computer.
5. Install the MS1553MC-1 or MS1553MC-2 board into the unused slot. The board should slide in from the side.
6. Bolt the mounting plate on the board to the computer with the screw that was removed from the blank mounting plate.
7. Replace the cover to the MCA computer.
8. Connect the pigtail cables to the back of the system unit.

   The cables are 12 inches long MIL-STD-1553B stubs. If longer or shorter stub lengths are required, consult the factory. The right connector is for Channel A; the left connector for Channel B.

Note: If this is the last connection on the MIL-STD-1553B bus, then a tee connector should be used with a 70-ohm terminator on one end to provide the proper termination for the bus. Tee connectors, 70-ohm terminators and complete terminated MIL-STD-1553B cables are also available from Condor Engineering, Inc.

MS1553MC DIP Switch Settings

The bus outputs can be coupled in three different manners:

- Direct coupled (short stub) - Isolation Transformer 1:1
- Direct coupled (short stub) - Isolation Transformer 1.4:1
- Transformer coupled (long stub)

Use direct coupling when the distance from the isolation transformer to the data bus does not exceed one foot. Use transformer coupling when the distance exceeds one foot but is less than 25 feet. There are isolation transformers on the board for both direct and transformer coupling. There are 53.6 ohm isolation resistors on board for direct coupling. Transformer coupling requires external transformer and isolation resistors at the junction to the main 1553 bus to match the stub impedance to the bus. Switch SW1 is for channel A bus coupling. Switch SW2 is for channel B bus coupling. Each selection requires two switches be "on" while the rest are "off".

SW1 Switch Numbers								
Coupling	1	2	3	4	5	6	7	8
Direct (1:1)	off	on	off	off	off	off	on	off
Direct (1.4:1)	off	off	on	off	off	on	off	off
Long Stub	off	off	off	on	on	off	off	off

SW2 Switch Numbers								
Coupling	1	2	3	4	5	6	7	8
Direct (1:1)	off	on	off	off	off	off	on	off
Direct (1.4:1)	off	off	on	off	off	on	off	off
Long Stub	off	off	off	on	on	off	off	off

MS1553MC Software Installation

To install the software, follow these instructions:

1. Insert your working copy of the Reference Diskette in drive A.
2. Power on the MCA computer.
3. The IBM banner or equivalent appears and prompts you to press Enter.

   The Main Menu appears.
4. Choose 5. Copy (merge) an option diskette.

   The option diskette is the Condor supplied 3½-inch diskette with @626C.ADF in the root folder. The MCA menu provides step-by-step instructions for merging the Condor ADF file to the reference diskette.
5. Choose 3. Set configuration.
6. Choose 2. Change configuration on the Set Configuration menu to change the defaults.
7. Move the cursor to the choice lists to select the memory address range, interrupt request level and BCRTM high priority interrupt.

   The choice list for Memory Address Range has six selections:

   - [32 KB: 0C0000h-0C7FFFh] (default)
   - [32 KB: 0C8000h-0CFFFFh]
   - [32 KB: 0D0000h-0D7FFFh]
   - [32 KB: 0D8000h-0DFFFFh]
   - [64 KB: 0C0000h-0CFFFFh]
   - [64 KB: 0D0000h-0DFFFFh]

   For the MS1553MC-1, the last two choices aren't valid. If there is a memory address conflict with another board in the system, an asterisk

appears next to the choice. You must choose another memory address range in this case.

The board may be set to use any interrupt in the Micro Channel system. The default is [IRQ5], which is usually available.

The high priority interrupts are the following:

- Dynamic bus control acceptance
- Illogical command error
- Subsystem Fail
- BIT fail
- Data overrun

8. The high priority interrupts are enabled in most applications so set the default to [enabled]. If enabled, the application software can individually enable, read, and clear the high priority interrupts.

   The BCRTM End of Message Interrupt may also be enabled.

9. Press F10 to save your changes.
10. Press Esc to return to the Set Configuration menu.
11. Press Esc followed by Enter to reset the computer.

    You need to remove the reference diskette to boot from the hard drive.

Verifying Installation

To verify installation:

1. Change to the config folder as follows: a:\config
2. Execute the test configuration program by typing: testcfg
3. Press Enter.

   This program performs a memory test, a memory controller test, check for the BCRTM chip, and perform an inquire to test the interrupt.

Note: The test may fail if you have an interrupt level conflict or memory map conflict. The program "MSD.EXE" or a similar program may be used to test the available memory ranges in your system.

   Refer to the MS1553 CP and Menu System Users Manual and to Part 2, C Driver, for additional test programs that test actual MIL-STD-1553B bus transfers.

4. If the board is removed from the computer or replaced into a different adapter slot, then the next time you power on the computer it will display error "165". Replace the board or set the configuration again as explained in the MCA user documentation.

CHAPTER 9

MS1553 Switch Register

Introduction

Each member of the Condor Engineering MS1553 product line contains a software-readable configuration register called the "switch register". This register is used to identify the board type and features to the Application Program Interface (API), which uses the information to setup internal data structures and options.

This section documents the various versions of this register and the meaning of each of the bits, so that this information can be made available to a user's program. The API functions ms1553m_register_read() and ms1553m_config_sw() can be used to read and return information from this register.

Switch Registers

MS1553PC-3/4 Switch Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	ID	1	1	P	1	1	1	1	1	1	1	S	W	R	C

Where:

ID 0 Identifies PC3/PC4 board

S 0 for PC3/4; 1 for Sabtech interface

W 0 for 32 Kb window; 1 for 64 Kb window

R 0 for 8 Kw RAM; 1 for 32 Kw RAM, but if WR = 11 then 64 Kb window and 32 Kw RAM. If WR = 01 then 32 Kb window and

16 Kw RAM. All other combinations give 32 Kb window and 8 Kw RAM

C 0 for BCRT; 1 for BCRTM chip installed*

P 1 for PC3/4; 0 for PC96. PC-3 reads 0xbff1, PC-4 reads 0xbff

MS1553PC96 Switch Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	B	A	1	1	1	1	0	0	W	R	C

Where:

W 0 for 32 Kb window; 1 for 64 Kb window

R 0 for 8 Kw RAM; 1 for 32 Kw RAM, but if WR = 11 then 64 Kb window and 32 Kw RAM. If WR = 01 then 32 Kb window and 16 Kw RAM* all other combinations give 32 Kb window and 8 Kw RAM

C 0 for BCRT; 1 for BCRTM chip installed

Bit 11 1 for PC3/4; 0 for PC96

B 0 for no activity on B Bus; 1 for activity seen

A 0 for no activity on A Bus; 1 for activity seen

MS1553PC104 Switch Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	1	B	A	0	W	W	1	1	R	R	0

Where:

B 0 for no activity on B Bus; 1 for activity seen

A 0 for no activity on A Bus; 1 for activity seen

WW RAM Window Size: 00 for 8 Kw; 01 for 16 Kw; 10 for 32 Kw; and 11 for 64 Kw

RR RAM Memory Size: 00 for 8 Kw; 01 for 16 Kw; 11 for 32 Kw; and 11 for 64 Kw

RR 11 then 64 Kb window and 32 Kw RAM*

RR 01 then 32 Kb window and 16 Kw RAM*

MS1553VMX Switch Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	1	B	A	1	1	1	1	1	1	1	0

Where:

B 0 for no activity on B Bus; 1 for activity seen

A 0 for no activity on A Bus; 1 for activity seen

MS1553VME Switch Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
U	U	U	U	U	U	U	U	U	U	U	U	Res	W	R	Res

Where:

U Undefined

Res Reserved

W 0 for 32 Kb window; 1 for 64 Kb window

R 0 for 8 Kw RAM; 1 for 32 Kw RAM

Switch Register Notes

The -96 board can be distinguished by a zero in bit 11.

The -104 board can be distinguished by a zero in bit 7

The -VME and -VMX boards can't be distinguished from a -3/4 board except by reading and writing the interrupt register.

CHAPTER 10

C Driver

Introduction

This section of the document describes the driver supplied by Condor Engineering, Inc. for its MS1553/1773 product line. This driver has evolved into an extremely powerful tool kit of over 70 routines. Many of the enhancements are due to feedback from customers, who have shared their requirements with us. We have made every attempt to evaluate these requirements and develop a tool kit that allows you to perform complex functions quickly and efficiently.

This driver is supplied in four forms:

- Microsoft "C/C++"-MS/DOS compatible link library (Large Model)
- Borland "C/C++"-MS/DOS compatible link library (Large Model)
- Microsoft Windows 3.X-compatible Dynamic Link Library (DLL).
- Microsoft Windows 95/98/NT/2000 drivers

The Windows DLL is *language independent*, and can be called from *any* Windows program.

The driver routines are documented in two chapters: Chapter 14 describes the BC, RT, and Monitor routines. Chapter 15 describes the generic tool kit routines.

Within each chapter the routines are listed in alphabetical order. In some cases, a routine is listed multiple times, such as ms1553m_init(), since different parameters are passed for the different modes. Each routine is named according to a naming convention that indicates the modes of operation which apply to the routine:

- ms1553m_init() Initialization for BC, RT, or Monitor mode
- ms1553m_bc_... BC mode routines
- ms1553m_rt_... RT mode routines
- ms1553m_m_... Monitor mode routines

- ms1553m_rtm_... Routines used in either RT or Monitor mode
- ms1553m_bcrtm_... Routines used in either BC, RT or Monitor mode
- ms1553m_... Generic tool kit routines

The MS1553/1773 series of boards operate in one of three modes:

- Bus Controller Mode (BC)
- Single Remote Terminal Mode (RT)
- Bus Monitor Mode (M)

If a given routine is *not* a generic routine, the board *must have been initialized* into one of the three operating modes *prior* to the use of the routine. This initialization is accomplished by calling the function "ms1553m_init()", passing it the parameters which describe the desired mode of operation.

If the routine *is* a generic routine, then you must read the "*Description*" section for that routine to determine what initialization, if any, is required before calling the routine.

Failure to properly initialize the driver library by calling "ms1553m_init()" when required will result in undefined behavior and possible memory corruption under MS-DOS®, and a General Protection Fault when using the Windows™-compatible DLL under Windows.

Once you have called "ms1553m_init()" you must remember to call "ms1553m_bcrtm_exit()" before your application returns to the operating system. Under MS-DOS, this can be accomplished with a normal call, or by installing it as an "atexit()" routine. Under Windows, the DLL attempts to call "ms1553m_bcrtm_exit()" automatically just before it is unloaded, but if a fault occurs (such as a General Protection Fault) in either the driver DLL or the application code, Windows does *not* unload the DLL, and the driver cannot cleanup the interrupts.

Calling "ms1553m_bcrtm_exit()" multiple times, or calling it before calling "ms1553m_init()" is harmless.

Multiple Board Support

The MS1553 API has been modified to support the use of multiple MS1553xxx boards in a single system.

This was done be adding a new argument to each of the functions which identifies the board to be used. The "board" argument identifies one of four possible boards, numbered 0 through 3.

The function names which support multiple boards were changed from ms1553_*() to ms1553m_*(). Only a few functions that are not board-specific were not renamed. These functions are described in Chapter 15, “Generic 1553 Routine Descriptions”.

Compatibility with Previous Versions

As described in the previous paragraph, release Version 4.00 of the MS1553 API features the ability to support multiple MS1553 boards in a single chassis. This change necessitated a change in the calling convention for most of the API functions.

To convert an old function name to the new function name, replace the "ms1553_" in the function name with "ms1553m_". Then add a "board" number argument to the beginning of the old calling sequence to get the new API function calling sequence.

Re-Building the API

The standard distribution contains the source code for the API. You may rebuild the API from these source files, using the compiler of your choice.

The as-distributed source code and libraries were build using:

- MS-DOS Library (ms1553bc.lib) – Borland C/C++ Version 5.02
- MS-DOS Library (ms1553ms.lib) – Microsoft C/C++ Version 1.52
- Windows 3.xx Library and DLL – Borland C/C++ Version 5.02
- Windows 95/98/NT/2000 Borland Library and DLL – Borland C/C++ Version 5.02
- Windows 95/98/NT/2000 Microsoft Library and DLL – Microsoft C/C++ Version 4.00

Many versions the Microsoft and Borland compilers, both older and newer, are compatible with these libraries. If your compiler/linker is not compatible with the as-built libraries, you can simply rebuild the library you need with your compiler.

- To build the MS-DOS version of the library, you need to define the label "_DOS".
- To build the Windows 3.1 version of the library, define the label "_Windows".
- To build the 32-bit Windows version of the library, define the labels "_Windows" and "__WIN32__".

If you are not using the PC-AT platform, or you are using non-Microsoft operating systems, you will probably need to modify the source code somewhat. Look in the file "setupirq.c" for the platform-specific functions that will have to be modified.

Contact Condor Engineering if you need assistance in converting the API to a different platform.

Compilation Notes

All Borland libraries were built under Borland C++ 5.02. The 16-bit Microsoft DOS library was built with Microsoft C++ 1.52. The 32-bit Microsoft library file was built using Microsoft C++ 4.0.

The example programs should be built as console applications. Include in the project the example source file, and the correct version of MST1553.LIB for your chosen compiler. If you need to link with the MST1553.DLL, and neither .lib file is compatible with your linker, your package might provide a utility (sometimes called implib.exe) that will create a compatible .lib file directly from the mst1553.dll file.

Beware that selecting optimization can have adverse effects on your code. For example, if you have a section of code:

```
while (!(x_err[0].completed) )
  ;
```

that waits for the "completed" flag to be set by the ms1553 interrupt routine (indicating that the associated message has been transacted on the 1553 bus), the compiler may "optimize" the code by removing the test from the loop, thus defeating the purpose of the code.

It is always a good idea to develop a program with the optimizer turned OFF, then once the program is running, re-compile with the optimizer ON and re-test (Borland and Microsoft both recommend this procedure).

Some other recommendations:

When setting up a project, set the byte alignment to 1 byte alignment. Note that this is not normally the default!

If you are performing direct accesses to the board, using the function ms1553_register_address() to obtain a pointer to the board's dual-port RAM or registers, be sure that your compiler generates only WORD accesses and not byte accesses. Byte accesses will result in corrupted data being transferred. The MS1553 internal function "memwcpy()" is available for use, see the MS1553.H file for the calling sequence.

If you are using a physical pointer (in DOS) or a pointer you have created using DPMI (in 16-bit Windows) to access the board directly, remember that this method will not work in a 32-bit environment. You will need to use the pointer obtained from ms1553_register_address(). If this routine is called with an argument of zero, the resulting pointer can be used to map the entire board memory space.

CHAPTER 11

Bus Controller Architecture

This chapter provides a basic understanding of the Bus Controller Hardware Architecture. The BCRTM chip is an extremely flexible device that has many capabilities that can be used to build and execute lists of messages, or *frames*. This chapter points out these basic functions so that you can understand the power that is available locally on the card.

Background

First, you must understand that the BCRTM is (only) capable of executing a frame locally on the card from the dual-ported static RAM that resides there. This means the time-critical functions, including frame timing functions, do not involve stealing cycles from the host processor. The BCRTM is capable of executing time delays controlled by a 12 MHz crystal oscillator with a granularity of 16 microseconds. This results in fewer host cycles and accurate timing. Also, when studying the driver, remember that all of the BC commands end up compiled into the memory on the card and *not* in host memory. The following figure shows the format of a single command compiled into memory:

COMMAND BLOCK

HEAD POINTER (Previous Command)
CONTROL WORD
COMMAND WORD 1
COMMAND WORD 2 (RT-RT ONLY)
DATA LIST POINTER
STATUS WORD 1
STATUS WORD 2 (RT-RT ONLY)
TAIL POINTER (Next Command)
DRIVER DATA AREA (8 Words)

DATA BUFFER

DATA WORD 0
DATA WORD 1
• • •
DATA WORD 31
TAIL POINTER

Each command consists of a command block and an associated data buffer. The command block has a pointer to the data buffer, as well as a pointer to the next command block. The BCRTM chip automatically follows these pointers when executing the list of commands, or *frame*.

When the driver initializes the MS1553/1773 into BC mode, the memory on the card is initialized to contain a user-specified number of two-word mode-code-with-data-word buffers. Following that area is a user-specified number of thirty-three word data buffers (defined above). Next, is the interrupt log-list. This circular list contains all of the information about the type and order of the interrupts that have occurred. The remaining room in memory is filled with command blocks as shown above.

The command block and data buffer areas, as they are called, can be viewed as a pool. When a frame is built, the required number of command blocks and data buffers are removed from the pool and become a part of the frame, leaving the remaining pool for other frames or temporary buffers for one-time type commands. If you execute the command to free a frame, all of the command blocks and data buffers are then re-linked back into this pool to be reused.

Each command for the BC has an accompanying delay term that is the amount of time that must transpire from the beginning of the current command to the beginning of the next command. This is very important for frames, since it is used to achieve the proper timing for the minor frames. Later in this manual is a more detailed discussion of this issue, but for now it is necessary to know here that the delays exist and are handled locally on the card to maintain proper accuracy.

The next important function to understand is the SKIP capability. This BCRTM function allows commands to be skipped, while still executing the delay terms to keep timing constant in the frame. This means that while a frame is executing you may cause messages to appear and disappear without halting execution. There are several driver commands that make use of this function allowing you to hide BC messages and make them appear only when necessary.

Finally, there is the role of the host in the driver. Each command generates an interrupt to the host, which is logged in the interrupt log-list on the card. The host then takes the interrupt and "reads" through the log-list to "see" what has happened. If the command was a transmit, data are moved from the board to the application layer, at the user-specified location, along with status information. The interrupt routine also keeps track of the message and error counts for you in the status structure. If high-priority interrupts are enabled, such as message error or subsystem fail, these are handled by this routine. These high-priority interrupt handlers can also be user-specified. If User Defined Interrupt routines are present the interrupt routine then calls the subroutine that has been linked in at runtime for handling that particular interrupt.

The host isn’t involved in the major time-critical functions. It is involved only in the intercommunication between the final data and the application.

A Quick Glance at the Control Word for the BCRTM

Only the MS1553 driver uses the Control Word in each Command Block. Certain commands allow you to make use of some of the special features found here, but you must *not* set any of these bits directly.

The Message Error bit and Interrupt and Continue bit are strictly under driver control. The Skip bit, End of List bit and Polling bit are under user-level control, depending on the build-frame routine loop and polling options you select. The Auto Retry bit, RT-RT Transfer (with or without Monitor), and the Time Delay are also under user control by way of driver routine options.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ERR	SKIP	INTR	POLL	AUTO	EOL	RTRT	MON	DELAY (7...0)							

ERR	15	Error
SKIP	14	Skip
INTR	13	Interrupt and Continue
POLL	12	Poll
AUTO	11	Auto Retry
EOL	10	End of List
RTRT	09	RT-RT Transfer
MON	08	Monitor RT-RT Transfer
DELAY	07..00	Delay

Temporary Frame Operation versus Permanent Frame Execution

In Bus Controller mode, the MS1553/1773 driver supports two modes of operation: Temporary Frame (or immediate) Mode, and Permanent Frame Mode. Before using the BC routines, you should have a clear understanding of the differences between the two modes.

Temporary Frame Mode is best used when the application requires 1553 messages, which depend on conditions that are detected by the host processor.

Permanent Frame Mode is best used when the application requires a fixed, repetitive list of messages, which are executed at a fixed rate. This is the classical 1553 “bus list” mode, which often consists of both “major” and “minor” frames.

In either mode, the application calls the various driver routines that compile the specified commands into the memory on the board. This process continues until the required list of messages has been created.

You may stack as many (or as few) commands together as fit into MS1553 memory.

The next operation depends on the driver mode. If the driver is in Temporary Frame Mode, the application directs the driver to execute the current bus list. The driver executes each of the messages in the list, and updates the status structures defined for each message as it is executed. This process continues until every message in the list has been completed. The driver then deletes the list of messages just executed (reclaiming the command blocks and data buffers) and returns control to your application.

If, however, the driver is Permanent Frame Mode, when the application requests execution of the created bus list, there are two options: execute the frame and return to the application immediately, or execute the frame and wait until it is complete before returning to the application.

If the driver is directed to wait until the frame is complete, the execution is very similar to Temporary Frame Mode, except that the list of messages (the *frame*) is *not* deleted by the driver, and is therefore available for future use. The application could, for example, create two permanent frames, and alternate between them without needing to re-create them before each use.

The process is slightly different when executing a permanent frame and specifying *no wait*. The driver starts the frame executing, then returns to the application. As the messages are executed, the driver interrupt routine updates the status structures and copies the data to/from the application's data buffers. If the frame ends with a "terminate", it executes once and quits. If the frame contains a loop, it continually loops until instructed to halt by the ms1553m_bc_halt_frame() routine.

In temporary mode, the application can examine the status structures, knowing the frame has been executed once and the structures have been updated. If this is a permanent frame, the application *cannot* examine the status structures. The application code must first wait for the status areas to be updated, and then clear the appropriate areas after testing them. For example, if you wish to know when a particular command has executed, you must have cleared the structure member called "completed" first, so when the interrupt routine sets this to a TRUE, the application knows that the data and status are updated. If the frame contains a loop, then the application must wait for the next time the command is executed, examine and then clear the status, etc.

You do *not* have to create permanent frames. You may execute a transmit or receive command which is compiled on the board and executed in a temporary frame. This can be done at any time (as long as another frame is not currently executing), whether or not there are permanent frames that exist on the board. You are allowed to create as many permanent frames as you have memory for, prior to execution.

To simply show the difference between Temporary Mode and Permanent Mode, the Permanent Mode is "bracketed" with the following:

```
/*   indicate we are going to compile a frame */
     ms1553m_bc_build_frame( TRUE );
/*   save the start address to loop back to and to know where to find the frame
     for later execution
*/
start = ms1553m_bc_get_adr();
frame commands.....
/*   indicate we are at the end of this frame.  The TRUE means we want to
     continually loop and "start" is the address to loop back to.
*/
ms1553m_bc_end_of_frame( TRUE, start );
/*   finally when all frames are built, stop the compilation process
*/
ms1553m_bc_build_frame( FALSE );
```

In temporary mode, you would just execute the "frame commands" portion.

Building Frames

Generally, in BC mode, the BC must execute major frames that are sample timing loops built by repeating minor frames. The process generates a major frame time that is the lowest frequency in which all other sample frequencies are sub-multiples.

For example, if you wish to sample one set of data at 20 HZ and another at 10 HZ, the major frame would be 10 HZ and the 20 HZ minor frame would repeat twice during that period. As the difference in frequency increases, the number of times minor frames must repeat increases. This creates a situation where the multiple repetitions of a command really want to reference the same data areas so that only one update is required to update all commands.

The commands listed with the last part of the name being "_same" are in the driver to allow you to handle exactly this situation. They allow the second through the Nth occurrence of a command to be referenced to the same data areas. To update the area use the ms1553mbc_chg_data() command. This command updates the data while the frame is executing and ensures the integrity of the data in the buffer so there are no data phasing errors.

If you wish to "hide" messages in your frames that appear and disappear based upon some conditions in your system, there are three commands that allow you to perform this function. Keep in mind the delay terms are always executed, even if the command is hidden.

This must happen to maintain a constant loop time. These functions are:

- ms1553m_bc_skip()
- ms1553m_bc_1noskip()
- ms1553m_bc_mtoggle()

With these functions you can achieve most special requirements for "hidden" message manipulation.

The Returned Status Structure

Each routine that accepts a status pointer has this area updated by the interrupt routine when a command is executed. The following is the list of variables in the structure:

```
typedef struct status_info        /* Used in BC and RT modes only        */
{
   BT_U16BOOL completed;      /* Set TRUE when message complete   */
   BT_U16BOOL error;             /* Set TRUE if an error was detected    */
   BT_U16BIT  runtime_error;  /* Code for error(s) which occured        */
   BT_U16BIT  status1;           /* BC mode: 1553 Status Word           */
   BT_U16BIT  status2;          /* BC mode, RT->RT msg, XMT Status Word
*/
   BT_U16BIT  msg_status;   /* if BC, mirrors interrupt status word  */
                        /* if RT, packed "MWB, ME, WC, Time Tag" */
                        /*   from the Message Status Word List.  */
                        /* Bit 15(MSB): Subsystem Fail asserted  */
                        /* Bit 14: Message was broadcasted       */
                        /* Bit 13: Message Error                 */
                        /* Bits 12-8: Word Count                 */
                        /* Bits 7-0: Time Tag (64us LSB)         */
   BT_U32BIT  msg_count;     /* Number of messages without errors     */
   BT_U32BIT  error_count;    /* Number of messages with errors       */
   BT_U16BIT  delta_time;      /* RT mode, time since last message (us),*/
                              /* wraps around after 255*64 microseconds*/
} STAT_INFO;
```

The "completed" flag is set TRUE by the interrupt routine when the particular command associated with this pointer has been executed. This indicates all data and status have been updated. When "auto retry" is enabled, the "completed" flag *does not* indicate that all retries have been attempted since each retry generates a separate interrupt.

If any errors occurred, the "error" flag is set TRUE. The "runtime_error" word contains an error code to indicate which run time error(s) have occurred. This error code is the "OR" of the possible error code bits, so there may be more than one error at a time. For BC mode these are:

- BC_NO_ERROR
- BC_MESSAGE_ERROR
- BC_NO_RESPONSE

If the value is equal to BC_NO_ERROR, no BC run time errors have occurred.

The "status1" word is the RT status returned for this particular command. If it is a RT-to-RT transfer, "status1" contains the receiving RT status word and "status2" contains the transmitting RT status word.

The "msg_status" variable mirrors the interrupt status word when in the BC mode.

Interrupt Status Word Definition for BC Mode

All bits in the Interrupt Status Word are active high and have the following functions:

Bit Number	Description
BIT 14	No Response Time-out (Message Error condition). Further defines the Message Error condition to indicate that a Response Time-out condition has occurred.
BIT 3	Polling Comparison Match. Indicates polling comparison interrupt.
BIT 2	Retry Fail. Indicates all the programmed retries have failed.
BIT 1	Message Error. Indicates a Message Error has occurred.
BIT 0	Interrupt and Continue. This bit corresponds to the interrupt and continue function described in the control word of the Command Block.

The interrupt routine accumulates a count of the number of messages without errors in "msg_count" and the number of messages with errors in "error_count".

The "delta_time" variable is only valid in RT mode.

Delay Values

The delay values that are specified for all commands, other than ms1553m_bc_delay(), are sixteen bit values in microseconds. The granularity of the delay on the board is 16 microseconds, therefore, the driver rounds to the nearest 16-microsecond interval to attempt to generate the correct delay. In the control word for the command block, only eight bits are available for the delay for any particular command block. If a larger number is required, the driver generates the correct number of "empty" command blocks in which the delay values are non-zero, but the skip bit is set indicating that there is no command to execute, just a delay. This is important to note if you ever display the contents of the memory on the card, since at first you might wonder where a number of empty packets were generated. For large delays, the ms1553m_bc_delay() routine is passed a long word, thereby allowing generation of as large a delay as there is memory on the board.

Note: There is a danger of swamping the host processor with interrupts if special care isn't taken with respect to the delay term. For example, if you were transmitting one packet of five words and used a delay of zero, an interrupt would occur approximately every 154 microseconds. This swamps a typical PC/AT class machine. You therefore must use a delay term to ensure enough time for the host machine to be able to service the interrupts. If the host becomes swamped with interrupts, it appears to hang, since it is constantly in the interrupt service routine.

Building Permanent Frames

More than one permanent frame can exist concurrently (limited only by the memory available on the MS1553PC-x board). Which frame, if any, is executing at a given time is determined by the application. A permanent frame or permanent message list remains valid until it is released.

The following description explains how a permanent message list can be successfully built, executed (perhaps multiple times), and finally released.

1. The application must call ms1553m_init() to initialize the MS1553 routines, and setup the Bus Controller mode.
2. The application must call ms1553m_bc_build_frame(TRUE) to set the global driver variable "building_frame" to true. This informs the MS1553 routines that a permanent frame build is in progress, and clears the global "frame_error" error flag. This is the error code returned by ms1553m_bc_build_error().
3. The application then calls ms1553m_bc_get_adr() to obtain the address of the first command block. This address serves as the primary way of identifying a message list in all subsequent message building routines, and must be specified in the call to ms1553m_bc_exec_frame() to execute the message list.

4. The application makes calls to any MS1553 routine. During this phase the message list is built up, that is, no start enables are issued by any MS1553 routine. See step 9 below.
5. After the message list is built the application makes a call to ms1553m_bc_end_of_frame(T/F, addr). This function allows the application to specify whether the message list just built should, when executed, execute once and terminate, or it should loop indefinitely. If it should loop indefinitely, then "addr" contains the address of the message that is to be looped back to when the message list has been executed. "addr" is obtained from ms1553m_bc_get_adr(), and does *not* have to be the first message in the message list.
6. This function does *not* start the execution of the frame (See step 9 below to start execution.)
7. The procedure ms1553m_bc_build_error() is used to inform the application of any errors encountered during the message/frame building phase. This routine does *not* clear the global error flag that is returned by this routine.
8. The application can now loop back to step (3) above, and call ms1553m_bc_get_adr(), and begin building another frame. When the proper number of frames have been created, the application should call ms1553m_bc_build_frame(FALSE) to indicate that the frame(s) are complete. Note that ms1553m_bc_build_frame(FALSE) clears the global error flag which is returned by ms1553m_bc_build_error().
9. The procedure ms1553m_bc_set_pcr(poll_code) is only used when the message buffer will contain any ms1553m_bc_poll calls. Prior to the ms1553m_bc_exec_frame() call the application calls ms1553m_bc_set_pcr() to set up the polling compare register with the correct mask value.
10. The list of messages (permanent frame) is now ready to be executed. The procedure ms1553m_bc_exec_frame() is used by the application to start this process. It is passed the address of the first command block of the frame which was obtained in step 3 above, and returns to the application after starting execution of the frame.
11. In the event that the message list executing is looping indefinitely, the procedure ms1553m_bc_halt_frame() can be called to terminate the execution. It is passed the address of the first command block of the frame which was obtained in step 3 above.
12. Finally, after a message list is no longer needed, the application may call ms1553m_bc_free_frame() to delete the frame. This releases the command blocks used by the list and makes them available to the free list pool. At this point, the application may go to step 2 and begin building another frame for execution, or it may call ms1553m_bcrtm_exit() to terminate processing.
13. If several large message lists are created, the free pool of available command blocks may diminish to the point that no more blocks are

available. In this case, you must either re-design the lists to be shorter, or release a message list using ms1553m_bc_free_frame(), to free up dual-port memory on the MS1553PC-x board.

14. The procedure ms1553m_bc_skip() allows the application to set or clear the skip bit in the specified command block. Setting the skip bit will make the 1553 skip this command block until it is cleared.
15. The procedure ms1553m_bc_1noskip() allows the application to clear the skip bit in the specified command block. The interrupt routine after processing the log list entry for the command blocks execution resets the skip bit in the command block. Thus the command block only executes once.
16. The procedure ms1553m_bc_chg_data() transfers data to the data area of the appropriate command block (RT Receive), which is part of a looping message list.
17. The procedure ms1553m_bc_tr_same() compiles a transmit or receive message which uses the same data area as another previously compiled message. The data pointer is replaced with the getadr tag.
18. The procedure ms1553m_bc_tr_mode_same() compiles a transmit or receive mode code which uses the same data area as another previously compiled message. The data pointer is replaced with the getadr tag.
19. The procedure ms1553m_bc_rtrtmon_same() will compile a RT-RT monitor message which uses the same data area as another previously compiled message. The data pointer is replaced with the getadr tag.

For an example of building and using permanent frames see the example programs supplied on disk.

CHAPTER 12

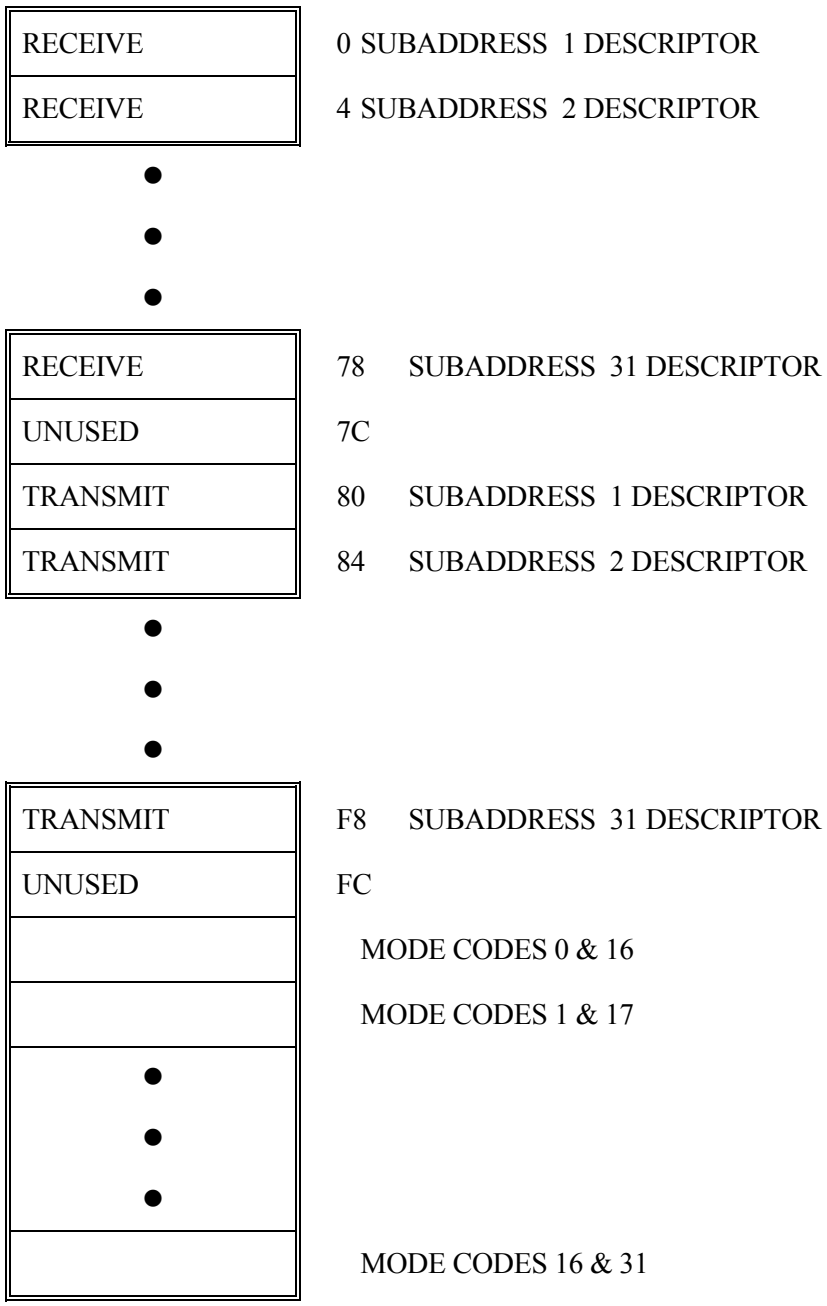
Remote Terminal Architecture

Introduction

This chapter describes the Remote Terminal Hardware Architecture.

Before beginning the discussion of the driver for Remote Terminal mode, the chapter provides a brief discussion of the hardware features that are available on the board. This is not intended to be a detailed discussion of the BCRTM; you should refer to the BCRTM data sheet for that information.

The BCRTM expects to find a section of RAM set up with three contiguous arrays of structures, one for receives, one for transmits, and one for mode codes. Each element of the array controls various parameters for each subaddress. The BCRTM data sheet calls this area the descriptor space. It is here that the BCRTM finds the information that governs the location and queue size for each subaddress.


RECEIVE
0 SUBADDRESS 1 DESCRIPTOR
RECEIVE
4 SUBADDRESS 2 DESCRIPTOR
RECEIVE
78 SUBADDRESS 31 DESCRIPTOR
UNUSED
7C
TRANSMIT
80 SUBADDRESS 1 DESCRIPTOR
TRANSMIT
84 SUBADDRESS 2 DESCRIPTOR
TRANSMIT
F8 SUBADDRESS 31 DESCRIPTOR
UNUSED
FC
MODE CODES 0 & 16
MODE CODES 1 & 17
MODE CODES 16 & 31

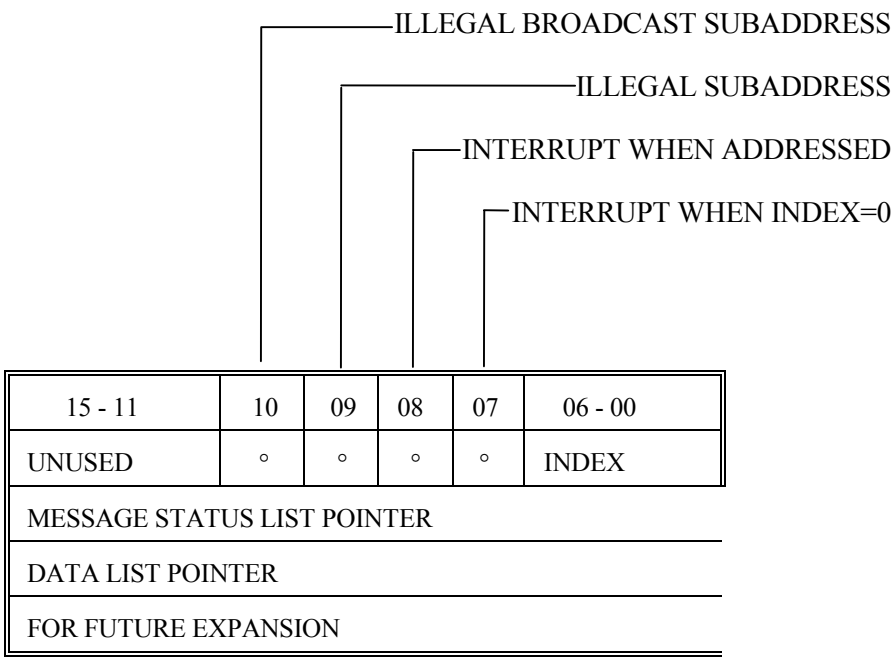
RT Descriptor Space

Each descriptor consists of four words:

1. A Control Word
2. Message Status List Pointer
3. A Data List Pointer
4. A reserved fourth word.

The pointers indicate where the data and status is to be stacked linearly in a queue, if the index value is non-zero. If the index value is zero, the BCRTM assumes only a one-message deep buffer, and overwrites the area with each new message. The control word indicates whether the subaddress is legal for operation for standard and broadcast messages. The interrupt modes are set here also.

You are not responsible for setting up these pointers, the ms1553m_init() routine will perform the function for you based upon the information you supply. You should *never* modify the settings of this table or the driver may not function correctly. The interrupt routine modifies these variables at times to implement a circular queue system.



15 - 11	10	09	08	07	06 - 00
UNUSED	◦	◦	◦	◦	INDEX
MESSAGE STATUS LIST POINTER					
DATA LIST POINTER					
FOR FUTURE EXPANSION					

Remote Terminal Subaddress Descriptor

The BCRTM handles the time-critical functions and notifies the host of the presence of new data or the fact that data has been transmitted. Also, with the pointer-oriented nature of the chip, memory can be partitioned easily and may be non-contiguous. But because of the flexibility, initializing the device requires a significant amount of information.

The Status Structure

For the RT driver routines, the status structure elements are not all used. The structure is as follows:

```
typedef struct status_info
{
    BOOLEAN completed;
    BOOLEAN error;
    unsigned short runtime_error;
    unsigned short status1;
    unsigned short status2;
    unsigned short msg_status;
    unsigned long  msg_count;
    unsigned long  error_count;
    unsigned short delta_time;
} STAT_INFO;
```

The "completed" flag indicates the requested command has been completed and the data has either been transmitted or received. The error field indicates that an error has occurred. For RT mode, it indicates a run time error only. You must test for which run time error:

- RT_DESCRIPTOR_OVERFLOW
- RT_INVALID_DESCRIPTOR
- RT_NO_RESPONSE

The "descriptor overflow" error occurs when the data you are attempting to place in the queue is overflowing the queue size declared in ms1553m_init(). The "invalid descriptor" error occurs when the subaddress you are attempting to access has not been legalized in the ms1553m_init() routine. The "no response" error occurs when you have the host timeout function enabled, ms1553m_set_timeout(), and the application is waiting for data, but it has not arrived in the allotted time period.

The "status1" and "status2" fields are not used. These are for BC and Monitor modes. The "msg_status" is an important field. It contains the status information concerning the message that was just completed. It also includes the word count needed when a receive command was executed. The bit field for this word is:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
N/A	MWB	MER	WORD COUNT					TIME TAG							

N/A	15	Not Used
MWB	14	Message Was Broadcasted
ME	13	Message Error
WORD COUNT	12...8	Word Count
TIME TAG	7...0	Time Tag

Notes: Remember a word count of zero is really 32, just as in the MIL-STD-1553 protocol.

The time tag is in 64 microsecond resolution and is a free-running eight bit counter. The routine ms1553m_rtm_clear_tag_timer() clears it. Remember, however, that time periods larger than 16.32 milliseconds causes the counter to "roll over", so absolute accuracy is lost. The field "delta_time" in the structure is maintained by the driver and is the difference between the current and previous time tag.

The "msg_count" field contains the count of messages received or transmitted without errors, and the "error_count" contains the number of messages received or transmitted with errors.

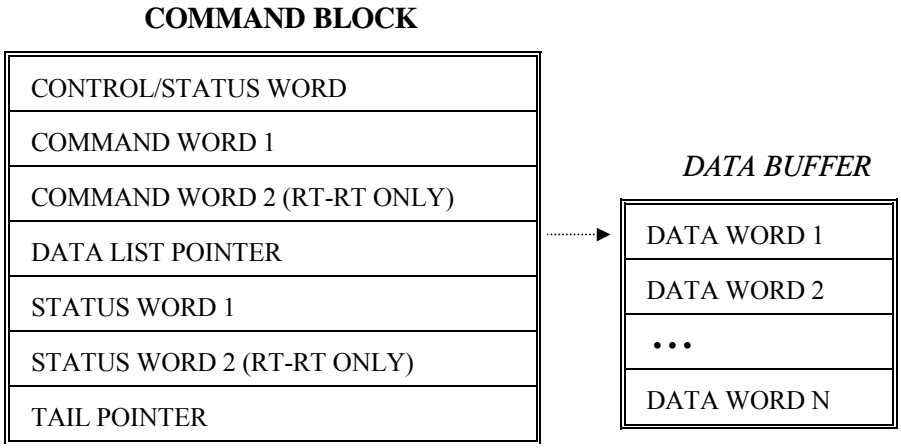
CHAPTER 13

Bus Monitor Architecture

Introduction

This chapter discusses Bus Monitor Architecture.

The architecture of the bus monitor is much like that of the bus controller, except it operates in reverse. The bus monitor mode requires that memory be set up as a singularly linked list of command blocks that will be filled with command word, status word, and data as it appears upon the bus. The bus controller is a doubly linked list of the commands to send on the bus and the monitor is a singularly linked list of command blocks that are filled with the commands as they pass by on the bus. You can see the similarity in the diagram below:



To provide the quickest method of accessing all data for storing it to disk, the 32-word data buffer is stored contiguous to the command block. This means that there is a seven-word command block followed by a thirty-two-word data buffer. Because the monitor function must operate very quickly, you must understand more of the low-level details of the hardware. The toolkit we provide directly addresses the board, so data are moved the fewest amount of times. Also, interrupts are available, but not used to save precious time. The goal is to achieve 100% bandwidth recording to disk using a minimum of an 8 MHz PC/AT with a 28 MS disk drive and a 1:1

interleave controller. This has been accomplished with the toolkit provided.

The following diagram shows the bits in the first word of the structure, the control/status word. You will have access to the word, and must understand the meaning of the bits within it.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CBA	ME	INTR	BUS	RESERVED				TIME TAG							

CBA	15	Command Block Active
ME	14	Message Error
INTR	13	Interrupt When Addressed
BUS	12	Bus A/B
RESERVED	11...8	Reserved
TIME TAG	7...0	Time Tag

The "command block active" bit is set to a one by the hardware when all of the data has been successfully written. The application software must test this bit to ensure valid data. It also must be cleared by the application software when the data has been processed.

The "message error" bit indicates if a message error occurred while receiving this message.

The "interrupt when addressed" *must* always stay zero. Setting this may cause unpredictable results!

Bus A/B indicates from which bus the message was received.

Finally, the "time tag" is the eight-bit number from the free-running counter internal to the BCRTM. This tag is in 64-microsecond resolution.

With this level of hardware understanding, you should be ready to begin the discussion of the driver routines available.

CHAPTER 14

Routine Descriptions

Introduction

This chapter provides descriptions of Bus Controller, Remote Terminal, and Bus Monitor routines.

The routines are listed in alphabetical order, by routine name. Routines like ms1553m_init(), which are called with different arguments depending on the current mode of operation (e.g., BC, RT or Monitor) are listed multiple times, describing the necessary arguments for the specific mode.

Command Summary

ms1553m_bc_1noskip(board, get_adr_ptr)
ms1553m_bc_auto_retry(board, t_f)
ms1553m_bc_build_error(board)
ms1553m_bc_build_frame(board, turn_on)
ms1553m_bc_chg_data(board, get_adr_ptr, data_ptr)
ms1553m_bc_clear_counts(board, get_adr_ptr, all)
ms1553m_bc_delay(board, usec)
ms1553m_bc_end_of_frame(board, loop, loop_msg_addr)
ms1553m_bc_exec_frame(board, get_adr_ptr, wait)
ms1553m_bc_free_frame(board, get_adr_ptr)
ms1553m_bc_get_adr(board)
ms1553m_bc_get_cblk_ptr(board, get_adr_ptr)
ms1553m_bc_get_dbuf_ptr(board, get_adr_ptr)
ms1553m_bc_halt_frame(board, get_adr_ptr)
ms1553m_bc_mtoggle(board, get_adr_ptr)
ms1553m_bc_poll(board, &rt_list[0], cnt, mask, stat_info, send, delay)
ms1553m_bc_restart(board)
ms1553m_bc_rtrt(board, src_rt, src_sa, dest_rt, dest_sa, cnt, stat_info,send,delay)

ms1553m_bc_rtrtmon(board,src_rt, src_sa, dest_rt, dest_sa, cnt, data_ptr,stat_info,send,delay)

ms1553m_bc_rtrtmon_same(board, s_rt, s_sa, d_rt, d_sa, cnt, get_adr_ptr, stat_info,send,delay)

ms1553m_bc_set_eol(board, BCmsgoffset)

ms1553m_bc_set_pcr(board, poll_mask)

ms1553m_bc_skip(board, get_adr_ptr, sflag)

ms1553m_bc_tr_data(board,rt_adr,tr,s_adr, cnt, data_ptr, stat_info, send, delay)

ms1553m_bc_tr_grp(board, rt_adr, tr, &sa_lst[0], sa_cnt, data_ptr, &stat_info[0], send, delay)

ms1553m_bc_tr_mode_code(board, rt_adr, tr, mode, data_ptr, stat_info, send, delay)

ms1553m_bc_tr_mode_same(board, rt_adr, tr, mode, get_adr_ptr, stat_info, send, delay)

ms1553m_bc_tr_same(board, rt_adr, tr, s_adr, cnt, get_adr_ptr, stat_info, send, delay)

ms1553m_bcrt_inquire(board)

ms1553m_bcrtm_active(board)

ms1553m_bcrtm_exit(board)

ms1553m_init(board, BC_MODE, i_init, NULL, irq, board_address)

ms1553m_init(board, RT_MODE, i_init, m_init, irq, board_address)

ms1553m_init(board, M_MODE, i_init, NULL, irq, board_address)

ms1553m_m_close(board)

ms1553m_m_cur_cbptr(board)

ms1553m_m_flen(board, last_block)

ms1553m_m_fread(board, dest, count)

ms1553m_m_fsearch(board, match, mask, cb_num)

ms1553m_m_fseek(board, cb_num)

ms1553m_m_fwrite(board, src, count)

ms1553m_m_get_msg(board, mon_command)

ms1553m_m_next_cbptr(board, cbptr)

ms1553m_m_open(board, path)

ms1553m_m_record(board, sec_to_record)

ms1553m_m_select_rt(board, rt_mask)

ms1553m_m_start(board)

ms1553m_m_start_cbptr(board)

ms1553m_m_stop(board)

ms1553m_rt_busy(board, t_f)

ms1553m_rt_chg_adr(board, new_rt_adr)

ms1553m_rt_clear_counts(board, tr, subaddr)

ms1553m_rt_dbusc(board, t_f)

ms1553m_rt_free_links(board, tr, &sa_lst[0], sa_cnt)

ms1553m_rt_get_sa_dbuf_ptr(board, tr, subaddr)

ms1553m_rt_get_sa_mstat_ptr(board, tr, subaddr)
ms1553m_rt_instr(board, t_f)
ms1553m_rt_reset_timeout(board)
ms1553m_rt_set_timeout(board, t_f, milliseconds)
ms1553m_rt_srq(board, t_f)
ms1553m_rt_ssfail(board, t_f)
ms1553m_rt_t_update(board, &sa_lst[0], sa_cnt, data_ptr)
ms1553m_rt_tflag(board, t_f)
ms1553m_rt_tr_data(board,tr, s_adr, cnt, data_ptr, stat_info, wait)
ms1553m_rt_tr_link(board, tr, &sa_lst[0], sa_cnt, data_ptr, &stat_info[0])
ms1553m_rt_tr_mcdata(board, tr, mode, data_word)
ms1553m_rt_tr_short_data(board, trbit, s_adr, cnt, data_ptr, stat_info)
ms1553m_rtm_clear_tag_timer(board)

ms1553m_bc_1noskip

Description

The routine is used when a message is to appear only once and disappear again. For example, if a series of "hidden" messages for various cockpit displays are in a frame, the ms1553m_bc_chg_data() command could be used to update data for a display and then ms1553m_bc_1noskip() could be called to update that display only once and disappear again.

The driver interrupt routine handles the changing of the “skip” bits after the command has executed once.

Synopsis

ms1553m_bc_1noskip(board, get_adr_ptr);

UINT board;
The board number to be accessed, 0 - 3.

unsigned short get_adr_ptr;
This is the address returned by ms1553m_get_adr() for the command block which is to be skipped.

Include Files

ms1553.h

Return Value

None.

ms1553m_bc_auto_retry

Description

This routine enables or disables the retry bit in the control word of the command block during BC command execution. Each command compiled for the BC contains separate control of the auto-retry function. Globally, the function can be enabled or disabled by a bit in the initialize field during the ms1553m_init() command. By default ms1553m_init() enables the retry mode.

Calling this routine affects all commands compiled after it is executed. This routine can be called on a command-by-command basis, if desired.

Synopsis

ms1553m_bc_auto_retry(board, t_f);

UINT board;
The board number to be accessed, 0 - 3.

unsigned long t_f;
t_f = TRUE/FALSE. True to enable the each command block auto-retry mode.

Include Files

ms1553.h

Return Value

None.

ms1553m_bc_build_error

Description

During the frame building process, the driver "ORs" together the returned error code for each command that is executed and save the result in a variable within the driver.

This routine allows you access to that variable at any time to see if any errors have occurred. If it is zero, no errors have occurred.

This routine doesn't clear the internal driver variable which accumulates all of the error codes, you must call "ms1553m_bc_build_frame(TRUE)" to clear the accumulated error codes.

Synopsis

status = ms1553m_bc_build_error(board);

UINT board;
The board number to be accessed, 0 - 3.

unsigned short status;
The accumulated error codes occurring during the frame building process.

Include Files

ms1553.h

Return Value

COMMAND_BLOCK_POOL_EMPTY

DATA_BLOCK_POOL_EMPTY

BAD_COUNT

DELAY_TOO_LARGE

MODE_CODE_BLOCK_POOL_EMPTY

INFINITE_LOOP_AND_WAIT

FRAME_BUILD_IN_PROGRESS

FRAME_BUILD_NOT_IN_PROGRESS

NO_COMMANDS_IN_FRAME

ms1553m_bc_build_frame

Description

Call this routine to cause the driver to enter the permanent frame building mode of operation if a TRUE is passed, or enter temporary mode if FALSE. If the driver is already in permanent mode and a TRUE is passed, the FRAME_BUILD_IN_PROGRESS error code is returned.

Synopsis

ms1553m_bc_build_frame(board, turn_on);

UINT board;
The board number to be accessed, 0 - 3.

BOOLEAN turn_on;
When set to TRUE this variable informs the driver that all following ms1553 routines will comprise a new permanent frame build. If false, the current permanent frame being built is complete and the driver returns to temporary mode.

Include Files

ms1553.h

Return Value

FRAME_BUILD_IN_PROGRESS

ms1553m_bc_chg_data

Description

This routine is used to update data for an RT receive command which is compiled in a frame on the board. As discussed in the section on frames, this is how to update data that must be sent to the remote terminals. Since the data must exist in the dual-ported RAM on the board for the BCRTM to send it on the bus, this provides the pathway to update it.

For this routine to work properly, you must have more than one command in a frame where one of them does *not* refer to this data buffer. It may be a skipped command, an ms1553m_bc_delay() command, a mode code, etc. The time between the end of the previous BC command and the next BC command should be greater than 4.1 milliseconds, although this time can decrease with the use of a faster host processor. This time is required because the driver must ensure that no command is transmitting the buffer at the current time so that moving the data won't cause phasing problems.

Synopsis

ms1553m_bc_chg_data(board, get_adr_ptr, data_ptr);

UINT board;
The board number to be accessed, 0 - 3.

unsigned short get_adr_ptr;
This is the address returned by ms1553m_bc_get_adr() for the message whose data area on the card is to be updated.

unsigned short *data_ptr;
Pointer to the new data for the command.

Note: The caller's data buffer should not cross a segment boundary.

Include Files

ms1553.h

Return Value

None.

ms1553m_bc_clear_counts

Description

This routine clears the error and message counters for either the specified command or the entire frame, on a per-board basis.

The interrupt routine maintains counts for each command indicating the number of successful messages which have been sent and the number of messages with errors. The counts are found in the status structure and are updated by the interrupt routine.

Synopsis

ms1553m_bc_clear_counts(board, get_adr_ptr, all);

UINT board;
The board number to be accessed, 0 - 3.

unsigned short get_adr_ptr;
Address returned from ms1553m_bc_get_adr for a command block.

unsigned all;
all = TRUE/FALSE. True to clear entire frame.

Include Files

ms1553.h

Return Value

None.

ms1553m_bc_delay

Description

When a delay is required in a frame that is longer than the time that can be specified for an individual message, you can insert the required delay in the frame with this command. This delay is accomplished by inserting the proper number of skipped messages provided there is sufficient memory.

Synopsis

ms1553m_bc_delay(board, usec);

UINT board;
The board number to be accessed, 0 - 3.

unsigned long usec;
Delay in microseconds.

Include Files

ms1553.h

Return Value

COMMAND_BLOCK_POOL_EMPTY

ms1553m_bc_end_of_frame

Description

This command is used to indicate the end of a frame. You are given the choice of looping back to a point in the frame continually, if "loop" is true. If "loop" is false, the frame executes once and stops. The address to loop back to doesn't have to be the first address in the frame. If there is some initialization data that must be sent to your system, which only needs to be sent once, it can be placed at the beginning of the frame and the loop point can be following those commands.

Following this command, you can execute another ms1553m_bc_get_adr() and begin compiling another frame. When all frames are then compiled, you will call ms1553m_bc_build_frame(FALSE) to indicate the frames are complete.

Synopsis

ms1553m_bc_end_of_frame(board, loop, loop_msg_addr);

UINT board;
The board number to be accessed, 0 - 3.

BOOLEAN loop;
This variable when set to TRUE informs the BCRTM driver that the built frame should loop indefinitely when executed.

unsigned short loop_msg-addr;
This is the address returned from ms1553m_bc_get_adr() for the command block which is to be executed after the last one in the frame, if "looping" is selected. It does not have to be the first block in the frame.

Include Files

ms1553.h

Return Value

FRAME_BUILD_NOT_IN_PROGRESS

NO_COMMANDS_IN_FRAME

ms1553m_bc_exec_frame

Description

This command instructs the BCRTM to execute the frame whose starting address is passed. If the "wait" parameter is true, the frame is checked to ensure it is not an infinite loop. If it is not, the frame executes once, after which this routine returns to the calling program. If it is an infinite loop, it does *not* execute the frame, but returns an error code.

Normally, this routine is called to begin execution of an infinitely looping frame, the "wait" parameter is then false. Once an infinitely looping frame has been started, the ms1553m_bc_halt_frame() command can be used to halt its execution.

Synopsis

ms1553m_bc_exec_frame(board, get_adr_ptr, wait);

UINT board;
The board number to be accessed, 0 - 3.

unsigned short get_adr_ptr;
The address returned from ms1553m_bc_get_adr() for the start of a frame.

BOOLEAN wait;
If wait is TRUE then remain until all the messages in buffer have been transmitted before returning, this should only be used if the "loop" parameter for the ms1553m_bc_end_of_frame() command was false, which generates a one-pass frame.

Include Files

ms1553.h

Return Value

INFINITE_LOOP_AND_WAIT

ms1553m_bc_free_frame

Description

This routine is called to delete a specified frame, so as to free all of the command and data blocks associated with the frame, and return them to the memory pool for reuse.

Synopsis

ms1553m_bc_free_frame(board, get_adr_ptr);

UINT board;
The board number to be accessed, 0 - 3.

unsigned short get_adr_ptr;
This is the address returned by ms1553m_bc_get_adr() for the first message in the frame that is to be deleted.

Include Files

ms1553.h

Return Value

None.

ms1553m_bc_get_adr

Description

This routine is used to return the address of the next command block that is about to be compiled. This address is used for later frame commands such as ms1553m_bc_end_of_frame(), ms1553m_bc_..._same(), and ms1553m_exec_frame().

This routine is called following the call to ms1553m_bc_build_frame() to save the start address of the frame that is about to be built. This address will be required later for use in the call to ms1553m_bc_exec_frame(), which begins execution of the frame.

Synopsis

adr = unsigned short ms1553m_bc_get_adr(board);

UINT board;
The board number to be accessed, 0 - 3.

unsigned short adr;
The address returned is the 16-bit word offset from the base address of the board to the beginning of the next command block to be compiled.

Include Files

ms1553.h

Return Value

The address of the next command block to be compiled.

ms1553m_bc_get_cblk_ptr

Description

This routine returns the absolute pointer to the command block whose address is passed. This allows direct access to the BCRTM command block (which includes some extra data used by the driver). Do *not* modify any of the added variables, other than the message and error count variables. Otherwise, many of the driver routines may no longer function correctly.

You must take care to address all structure members as declared, never recast as bytes, or permit the compiler to optimize your code to access bytes.

When using the driver Windows DLL, this routine provides a method of obtaining a protected-mode pointer to a command block on the board. If the normal real-mode "Segment:Offset" pointer was used, the application would fail with a "General Protection Fault".

Synopsis

cblk_ptr = struct extended_command_block *ms1553m_bc_get_cblk_ptr(board, get_adr_ptr);

UINT board;
The board number to be accessed, 0 - 3.

unsigned short get_adr_ptr;
The address returned by ms1553m_bc_get_adr for the command block whose pointer is desired.

unsigned short *cblk_ptr;
A pointer to the specified command block.

Include Files

ms1553.h

Return Value

An absolute pointer to the extended command block.

ms1553m_bc_get_dbuf_ptr

Description

This routine is called to obtain an absolute pointer to a data buffer associated with a specified command block. This function returns a pointer to an unsigned short array. You may then directly address the data on the board.

You *must* always address data on the board as unsigned shorts and *never* as bytes.

When using the driver Windows DLL, this routine provides a method of obtaining a protected-mode pointer to a data buffer on the board. If the normal real-mode "Segment:Offset" pointer was used, the application would fail with a "General Protection Fault".

Synopsis

buf_ptr = unsigned short *ms1553m_bc_get_dbuf_ptr(board, get_adr_ptr);

UINT board;
The board number to be accessed, 0 - 3.

unsigned short get_adr_ptr;
Address returned by ms1553m_bc_get_adr for the command block whose data buffer address is desired.

unsigned short *buf_ptr;
A pointer to the data buffer associated with the specified command block.

Include Files

ms1553.h

Return Value

An absolute pointer to the data buffer associated with the command block.

ms1553m_bc_halt_frame

Description

This routine is called to terminate the execution of a frame that is currently executing. The call must supply the start address of the frame to be terminated.

The routine ensures that the execution halts prior to beginning the next pass through the frame. It always stops at the beginning of the frame, it doesn't just halt the BCRTM at the current location. This was done to ensure an orderly halt with complete frames being executed.

Synopsis

ms1553m_bc_halt_frame(board, get_adr_ptr);

UINT board;
The board number to be accessed, 0 - 3.

unsigned short get_adr_ptr;
This is the start address, returned by ms1553m_bc_get_adr(), for a frame currently executing which is to be halted.

Include Files

ms1553.h

Return Value

None.

ms1553m_bc_mtoggle

Description

This routine is used for those circumstances when you have two messages for a device, and either the first or the second is to be executed, but not both. The function is effectively a "momentary toggle" in which the first message disappears for one cycle and the second message appears for one cycle. Then the first message is continually sent, and the second continually skipped or "hidden". Use care when using this function! The specified way the commands are generated must be followed *exactly*!

To use the function, generate the two messages contiguously to one another. The first message is the "normally closed" function of the toggle, or the message that is normally executed. For this message, you *must not* use a non-zero delay term! The second message is then generated; it may have a non-zero delay. After the second message is generated, which is the "normally open" or "hidden" message, you must set the skip flag *true* for it. When you wish to execute the function, you pass the address pointer for the first message to this routine.

Here is an example:

```
nc = ms1553m_bc_get_adr(board)

ms1553m_bc_tr_data( board, 1, RBIT, 2, 32, d1ptr, s1ptr, FALSE, 0 );
/*    delay MUST be 0 */
no = ms1553m_bc_get_adr(board);
ms1553m_bc_tr_data( board, 1, RBIT, 2, 32, d2ptr, s2ptr, FALSE, 3000 );
ms1553m_bc_skip( board, no, TRUE );
/* when executing the function */
ms1553m_bc_mtoggle( board, nc );
```

Synopsis

```
ms1553m_bc_mtoggle( board, get_adr_ptr );
```

UINT board;
The board number to be accessed, 0 - 3.

unsigned short get_adr_ptr;
This is the word offset to the message in the frame that was returned from ms1553m_bc_get_adr.

Include Files

ms1553.h

Return Value

None.

ms1553m_bc_poll

Description

Poll each RT found in the user-supplied "rt_list" for its status word, "AND" the status word with the "mask" and then test the result for a non-zero result. If the result is non-zero, set the "status1" in "stat_info" to the returned status word and set the "completed" flag in "stat_info" to indicate the completion.

If "wait" is true for temporary mode, the routine returns when an RT has been found that meets the criteria. If false, the application must test the "completed" flag. When used in a permanent frame the "wait" flag is ignored.

If this is a permanent frame the mask is ignored, since there can be multiple polling frames in memory and the mask is stored in a chip register rather than memory. For permanent frames you must call ms1553m_bc_pcr(mask) to set the polling code register in the chip, then execute the permanent frame.

Synopsis

ms1553m_bc_poll(board, &rt_list[0], cnt, mask, stat_info, send, delay);

UINT board;
The board number to be accessed, 0 - 3.

struct poll_list rt_list[cnt];
An array of "cnt" elements containing remote terminal addresses which should be polled.

```
struct poll_list
{
    unsigned short rt_adr;
};
```

unsigned short cnt;
The number of entries in the list.

unsigned short mask;
A word to be "ANDed" with each polled remote terminal's status word.

struct status_info *stat_info;
A data structure which will contain the results of the poll.

BOOLEAN send;
If true, send the current temporary buffer and wait until a non-zero result from the "AND" of the mask and the status is found. Otherwise, return and the application must test the "completed" flag in the "stat_info" structure for completion when the buffer is executed.

unsigned short delay;
The delay time between each poll taken in microseconds.

Include Files

ms1553.h

Return Value

COMMAND_BLOCK_POOL_EMPTY

ms1553m_bc_restart

Description

This routine is normally called by a user interrupt function which has specified the "UI_END_OF_COMMAND_BLOCK_LIST" interrupt notification with a call to the ms1553m_set_interrupt() function. The user function is called when a BC message is processed, which has the "END_OF_LIST" bit set, and must call this function to restart the Bus Controller.

Failure to call this function will result in the Bus Controller remaining in a stopped state.

To set the "END_OF_LIST" bit in a BC message, call the ms1553m_bc_set_eol() function, passing the address offset of the desired BC message. This offset can be obtained via the function ms1553m_bc_get_adr()..

Synopsis

ms1553m_bc_restart(board);

UINT board;
The board number to be accessed, 0 - 3.

Include Files

ms1553.h

Return Value

0

ms1553m_bc_rtrt

Description

This routine sets up a command block for a remote terminal to remote terminal transmission. The data flow is from the source RT to the destination RT, and the BC does not monitor the data. Thus, there is no data returned to the BC, all data are sent between the remote terminals. If you wish to capture the data, you must use the ms1553m_bc_rtrtmon() routine.

Synopsis

ms1553m_bc_rtrt(board, src_rt, src_sa, dest_rt, dest_sa, cnt, stat_info,send,delay);

UINT board;
The board number to be accessed, 0 - 3.

unsigned short src_rt;
The remote terminal sending the message.

unsigned short src_sa;
The subaddress of the remote terminal sending the message.

unsigned short dest_rt;
The remote terminal receiving the message.

unsigned short dest_sa;
The subaddress of the remote terminal receiving the message.

unsigned short cnt;
The number of words in the message.

struct status_info *stat_info;
A pointer to the status structure that is to be updated by the interrupt routine to indicate the status of this command. The status words in the structure are updated where "status1" is the destination RT and "status2" is the source.

BOOLEAN send;
This switch is only used for the temporary mode of the driver. If you are compiling a frame it is ignored. The switch indicates whether to send the commands(s) in the current temporary buffer(s), wait for completion, and reclaim the storage. If it is FALSE, commands will continue to be stacked.

unsigned short delay;
The delay time between each message transmission in microseconds with a range of 0 to 65535, from the beginning of this message to the beginning of the next one.

Include Files

ms1553.h

Return Value

COMMAND_BLOCK_POOL_EMPTY

ms1553m_bc_rtrtmon

Description

This routine sets up a command block for a remote terminal to remote terminal transmission. The data flow is from the source RT to the destination RT with an additional destination in the BC. The BC monitors the transmission, and the data are placed in the array pointed to by "data_ptr" by the interrupt routine.

Synopsis

ms1553m_bc_rtrtmon(board,src_rt, src_sa, dest_rt, dest_sa, cnt, data_ptr,stat_info,send,delay);

UINT board;
The board number to be accessed, 0 - 3.

unsigned short src_rt;
The remote terminal sending the message.

unsigned short src_sa;
The subaddress of the remote terminal sending the message.

unsigned short dest_rt;
The remote terminal receiving the message.

unsigned short dest_sa;
The subaddress of the remote terminal receiving the message.

unsigned short cnt;
The number of words in the message.

unsigned short *data_ptr;
The monitored data will be transferred to this location.

Note: The caller's data buffer should not cross a segment boundary.

struct status_info *stat_info;
A pointer to the status structure that is to be updated by the interrupt routine to indicate the status of this command. The status words in the structure are updated where "status1" is the destination RT and "status2" is the source.

BOOLEAN send;
This switch is only used for the temporary mode of the driver. If you are compiling a frame it is ignored. The switch indicates whether to send the commands(s) in the current temporary buffer(s), wait for completion, and reclaim the storage. If it is FALSE, commands will continue to be stacked.

unsigned short delay;
The delay time between each message transmission in microseconds with a range of 0 to 65535, from the beginning of this message to the beginning of the next one.

Include Files

ms1553.h

Return Value

COMMAND_BLOCK_POOL_EMPTY

ms1553m_bc_rtrtmon_same

Description

This procedure sets up a command block for a remote terminal to remote terminal transmission. The data flow is from the source RT to the destination RT with an additional destination in the BC. The BC monitors the transmission, and the data are placed in the array pointed to by the "data_ptr" associated with the command whose "get_adr_ptr" specified.

This version of the command uses the same, previously defined data buffer on the board, thereby saving valuable data buffer space, when you are using the second through the Nth repetition of the same command for a frame. This is important to note when you are compiling frames in which the RT to RT commands with monitoring must be repeated many times. By using the other form you will use a data buffer on the board for each command. This routine folds all of the required data buffers into the same data area.

You may specify different remote terminals and subaddresses for each command, setting them to the same data buffer area, if you wish.

Synopsis

ms1553m_bc_rtrtmon_same(board, s_rt, s_sa, d_rt, d_sa, cnt, get_adr_ptr, stat_info,send,delay);

UINT board;
The board number to be accessed, 0 - 3.

unsigned short s_rt;
The remote terminal sending the message.

unsigned short s_sa;
The subaddress of the remote terminal sending the message.

unsigned short d_rt;
The remote terminal receiving the message.

unsigned short d_sa;
The subaddress of the remote terminal receiving the message.

unsigned short cnt;
The number of words in the message.

unsigned short *get_adr_ptr;
The address returned from a ms1553m_bc_get_adr() function executed prior to executing a ms1553m_bc_tr_data() function that sets up the data area on the board which is to receive or transmit common data.

struct status_info *stat_info;
A pointer to the status structure that is to be updated by the interrupt routine to indicate the status of this command. The status words in the structure are updated where "status1" is the destination RT and "status2" is the source.

BOOLEAN send;
This switch is only used for the temporary mode of the driver. If you are compiling a permanent frame it is ignored.
If TRUE, this switch indicates that the driver is to execute the commands(s) in the current temporary buffer(s), wait for completion, and reclaim the storage.
If FALSE, commands will continue to be stacked.

unsigned short delay;
The delay time between each message transmission in microseconds with a range of 0 to 65535, from the beginning of this message to the beginning of the next one.

Include Files

ms1553.h

Return Value

COMMAND_BLOCK_POOL_EMPTY

ms1553m_bc_set_eol

Description

This routine sets the "END_OF_LIST" bit in the specified BC message. The offset to the BC message is obtained via a call to the ms1553m_bc_get_adr() function.

When the BC encounters a BC message with the "END_OF_LIST" bit set, it causes an interrupt and suspends processing of messages. The interrupt routine *must* call ms1553m_bc_restart() to cause the bus controller to resume message processing.

You must register an interrupt function which calls the ms1553m_bc_restart() function. This function must be registered with the ms1553m_set_interrupt() function, and must request notification of the "UI_END_OF_COMMAND_BLOCK_LIST" interrupt cause.

Synopsis

ms1553m_bc_set_eol(board, BCmsgoffset);

UINT board;
The board number to be accessed, 0 - 3.

unsigned short BCmsgoffset;
The offset to the desired BC message.

Include Files

ms1553.h

Return Value

0

ms1553m_bc_set_pcr

Description

This routine is called prior to a ms1553m_bc_exec_frame call in order to set up the polling compare register with the correct mask.

This call is only necessary if the frame uses a call to ms1553m_bc_poll.

Synopsis

ms1553m_bc_set_pcr(board, poll_mask);

U16BIT board;
The board number to be accessed, 0 - 3.

unsigned short poll_mask;
This is the mask to be inserted into the polling compare register.

Include Files

ms1553.h

Return Value

None.

ms1553m_bc_skip

Description

If "sflag" parameter is true, the skip flag for the specified command block is set. This forces the BCRTM to execute the delay for the command, but not execute the actual command. This means the message is skipped or it is now "hidden".

The process is reversed when sflag is false.

Synopsis

ms1553m_bc_skip(board, get_adr_ptr, sflag);

UINT board;
The board number to be accessed, 0 - 3.

unsigned short get_adr_ptr;
This is the address returned by ms1553m_bc_get_adr for the command block that will have the skip flag set or cleared.

unsigned short sflag;
If TRUE, set the flag to skip the command. If FALSE, clear the skip flag and the message executes.

Include Files

ms1553.h

Return Value

None.

ms1553m_bc_tr_data

Description

Compile a command block on the card to instruct a RT/subaddress to either receive or to transmit data. The T/R bit is with respect to the RT. Thus, for the RT to receive data from the BC, you are instructing the RT to receive. This command may be placed in a frame, which means it isn't executed until the frame is executed. Likewise, if you are compiling it in temporary mode, it does not execute until the current temporary buffer is executed. This means either this command or a later one must have the "send" flag TRUE.

Synopsis

ms1553m_bc_tr_data(board,rt_adr,tr,s_adr, cnt, data_ptr, stat_info, send, delay);

UINT board;
The board number to be accessed, 0 - 3.

unsigned short rt_adr;
The remote terminal transmitting/receiving this message.

unsigned short tr;
Transmit/Receive bit (TBIT or RBIT).

unsigned short s_adr;
The subaddress within the remote terminal to transmit/receive the message.

unsigned short cnt;
The number of data words in the message.

unsigned short *data_ptr;
If the command is an RT receive, this will point to the calling routine's data buffer that is to be placed in the data buffer on the board associated with the command when it is executed. If the command is an RT transmit, this points to the final destination for the data received from the RT. The interrupt routine will update the data at this location, once it has been totally received by the hardware.

Note: The caller's data buffer should not cross a segment boundary.

struct status_info *stat_info;
A pointer to the status structure that is to be updated by the interrupt routine to indicate the status of this command.

BOOLEAN send;

This switch is only used for the temporary mode of the driver. If you are compiling a frame it is ignored.
If TRUE, the driver will transmit all of the commands(s) in the current temporary buffer(s), wait for completion, and reclaim the storage. If it is FALSE, commands will continue to be stacked.

unsigned short delay;

The delay time between each message transmission in microseconds with a range of 0 to 65535, from the beginning of this message to the beginning of the next one.

Include Files

ms1553.h

Return Value

The following error codes are possible:

- COMMAND_BLOCK_POOL_EMPTY
- DATA_BLOCK_POOL_EMPTY

ms1553m_bc_tr_grp_data

Description

This routine is used when a large data buffer is to be received or transmitted using multiple subaddresses. Once the definition for the subaddresses is defined, in the subaddress/count array "sa_lst", the driver routine then packs and unpacks the data in the specified order and amounts from a contiguous block of memory located at the "data_ptr". When used in temporary mode, the driver overhead is reduced, versus using multiple ms1553m_bc_tr_data() commands, thereby creating better response time.

In permanent mode, this routine is more of a convenience than anything else, in that it computes the source or destination offset positions in the data and status for you from the "sa_lst" counts.

Synosis

ms1553m_bc_tr_grp(board, rt_adr, tr, &sa_lst[0], sa_cnt, data_ptr, &stat_info[0], send, delay);

UINT board;
The board number to be accessed, 0 - 3.

unsigned short rt_adr;
The remote terminal transmitting/receiving this message.

unsigned short tr;
Transmit/Receive bit (TBIT or RBIT).

struct sa_list sa_lst[sa_cnt];
An array of "sa_cnt" elements of "sa_list" structures. Each entry in the structure contains the subaddress and the number of data words in the message for each subaddress.

```
struct sa_list
{
    unsigned short s_adr;
    unsigned short count;
};
```

unsigned short sa_cnt;
The number of subaddress/count elements in "sa_lst".

unsigned short *data_ptr;
A pointer to a user supplied block of memory containing the data to be transmitted/received. Each subaddress message cannot contain more than 32 words of data. All data words to be transmitted or received are placed contiguously in memory starting at the location pointed to by data_ptr and in the order specified by the "sa_lst" structure.

Note: The caller's data buffer should not cross a segment boundary.

struct status_info stat_info[sa_cnt];
An array of "sa_cnt" elements of status_info structures which will contain the results of the message transmission. Each element corresponds to the equivalent subaddress element of "sa_lst".

BOOLEAN send;
This switch is only used for the temporary mode of the driver. If you are compiling a frame it is ignored. The switch indicates whether to send the commands(s) in the current temporary buffer(s), wait for completion, and reclaim the storage. If it is FALSE, commands will continue to be stacked.

unsigned short delay;
The delay time between each message transmission in microseconds with a range of 0 to 65535, from the beginning of this message to the beginning of the next one.

Include Files

ms1553.h

Return Value

COMMAND_BLOCK_POOL_EMPTY

DATA_BLOCK_POOL_EMPTY

ms1553m_bc_tr_mode_code

Description

This routine transmits a mode code and uses a unique data word buffer if it is a mode-code-with-data-word. If the mode code is between 16 and 31 (a mode-code-with-data-word), then the associated data word pointed to by "data_ptr" will be transferred also.

Synopsis

ms1553m_bc_tr_mode_code(board, rt_adr, tr, mode, data_ptr, stat_info, send, delay);

UINT board;
The board number to be accessed, 0 - 3.

unsigned short rt_adr;
The remote terminal transmitting/receiving this mode code.

unsigned short tr;
Transmit/Receive bit (TBIT or RBIT).

unsigned short mode;
The mode code to send.

unsigned short *data_ptr;
If the command is an RT receive, this will point to the calling routine's data buffer that is to be placed in the data buffer on the board associated with the command when it is executed. If the command is an RT transmit, this points to the final destination for the data received from the RT. The interrupt routine will update the data at this location, once it has been received by the hardware For mode codes, this is only one word and only valid for mode codes with data words.

Note: The caller's data buffer should not cross a segment boundary.

struct status_info *stat_info;
A pointer to the status structure that is to be updated by the interrupt routine to indicate the status of this command.

BOOLEAN send;
This switch is only used for the temporary mode of the driver. If you are compiling a frame it is ignored.
If TRUE, the driver will transmit the commands(s) in the current temporary buffer(s), wait for them to compile, and then reclaim the storage.
If FALSE, commands will continue to be stacked in the temporary frame.

unsigned short delay;

The delay time between each message transmission in microseconds with a range of 0 to 65535, from the beginning of this message to the beginning of the next one.

Include Files

ms1553.h

Return Value

COMMAND_BLOCK_POOL_EMPTY

ILLEGAL_MODE_CODE

DATA_BLOCK_POOL_EMPTY

ms1553m_bc_tr_mode_same

Description

This routine, as the others that end in "_same", is used for the second through the Nth occurrence of a mode code command in a frame where the data area is to be shared. It uses the same data word on the board as well as the final application destination. Obviously, this is for mode-code-with-data-word type commands.

Synopsis

ms1553m_bc_tr_mode_same(board, rt_adr, tr, mode, get_adr_ptr, stat_info, send, delay);

UINT board;
The board number to be accessed, 0 - 3.

unsigned short rt_adr;
The remote terminal transmitting/receiving this mode code.

unsigned short tr;
Transmit/Receive bit (TBIT or RBIT).

unsigned short mode;
The mode code to send.

unsigned short *get_adr_ptr;
The address returned from a ms1553m_bc_get_adr() function executed prior to executing a ms1553m_bc_tr_mode_code() function that sets up the data area on the board which is to receive or transmit common data mode code data words.

struct status_info *stat_info;
A pointer to the status structure that is to be updated by the interrupt routine to indicate the status of this command.

BOOLEAN send;
This switch is only used for the temporary mode of the driver. If you are compiling a frame it is ignored.
If TRUE, the driver will transmit all of commands(s) in the current temporary buffer(s), wait for them to compile, and then reclaim the storage.
If FALSE, commands will continue to be stacked in the temporary frame.

unsigned short delay;

The delay time between each message transmission in microseconds with a range of 0 to 65535, from the beginning of this message to the beginning of the next one.

Include Files

ms1553.h

Return Value

COMMAND_BLOCK_POOL_EMPTY

ILLEGAL_MODE_CODE

DATA_BLOCK_POOL_EMPTY

ms1553m_bc_tr_same

Description

This routine can be used for the second through the Nth repetition of an ms1553m_bc_tr_data() command within a frame.

To understand why this is useful, we must understand the way in which a typical frame is built. To achieve a particular frequency for any particular command, it must be repeated one or more times, with appropriate delays. If you were to use multiple ms1553m_bc_tr_data() commands, they would consume a separate data area for each command. If you wish to update the data that is being sent to an RT, you would have to update the data for each command, wasting host CPU time, generating phasing errors on your data in the frame, and wasting memory on the board.

By saving the address from ms1553m_bc_get_adr() for the first command, the ms1553m_bc_tr_same() routine can compile a command which will use the same data area on the board, as well as in the application layer. This applies to all of the "_same" routines in the driver library.

The other parameters passed allow you to specify different parameters for the remainder of the command, if desired. The end result is common data areas. If you wish to update data for this common area, on the fly, you will call the ms1553m_bc_chg_data() command, and the data area on the board will be updated for ALL commands referencing it.

Synopsis

ms1553m_bc_tr_same(board, rt_adr, tr, s_adr, cnt, get_adr_ptr, stat_info, send, delay);

UINT board;
The board number to be accessed, 0 - 3.

unsigned short rt_adr;
The remote terminal transmitting/receiving this message.

unsigned short tr;
Transmit/Receive bit (TBIT or RBIT).

unsigned short s_adr;
The subaddress within the remote terminal to transmit/receive the message.

unsigned short cnt;
The number of data words in the message.

unsigned short get_adr_ptr;
The address value returned from the ms1553m_bc_get_adr() function executed prior to executing a ms1553m_bc_tr_data() function that sets up the data area on the board which is to receive or transmit common data.

struct status_info *stat_info;
A pointer to the status structure that is to be updated by the interrupt routine to indicate the status of this command.

BOOLEAN send;
This switch is only used for the temporary mode of the driver. If you are compiling a frame it is ignored. If TRUE, the driver executes all of the commands(s) in the current temporary buffer(s), wait for them to compile, and reclaim the storage. If FALSE, commands continue to be stacked in the temporary frame.

unsigned short delay;
The delay time between each message transmission in microseconds with a range of 0 to 65535, from the beginning of this message to the beginning of the next one.

Include Files

ms1553.h

Return Value

COMMAND_BLOCK_POOL_EMPTY

ms1553m_bcrt_inquire

Description

This is an obsolete function that was intended to help determine if the hardware is really plugged into the chassis, in case your application program is run on a machine in which there is no MS1553-xxx board.

This routine attempts to verify the presence of the hardware by performing a memory test, a BIT function, and some proprietary checks upon the board. The results are then passed back to the calling program. As our naming convention implies, this only works after the board has been initialized as a BC or RT (e.g., ms1553m_init() has been successfully called). You must re-initialize the board after the test, since the tests are destructive.

All of the tests which this routine executes are also executed by ms1553m_init() with the singular exception of the BCRTM BIT test. This is the test executed by the function ms1553m_bit_test(). For this reason this function is considered obsolete.

Synopsis

unsigned short ms1553m_bcrt_inquire(board);

UINT board;
The board number to be accessed, 0 - 3.

Include Files

ms1553.h

Return Value

The following bits may be set in the word returned from the inquire. You must therefore test for each one, since they all may be there.

- INQ_BITFAIL indicates the chip did not pass the BIT function, which would be a good indicator that the board is not present.
- INQ_BITTIMEOUT indicates the chip didn't report the BIT status through the interrupt system, which normally indicates a wrong interrupt number.
- INQ_HI_PRI_SW_OFF is a warning bit for your program. This flag is letting you know that if the program requires high priority interrupts, they have not been enabled on the board or the revision hardware in the chassis does not support them. (This is true for Revision 1.00 only.) If you do not make use of high priority interrupts, you can ignore the bit.

ms1553m_bcrtm_active

Description

This function tests if the BCRTM chip is currently in the active state. This means it has been initialized. For the BC, it also means it is executing a frame, For an RT, it also means it is actively waiting for commands from the BC, or if Monitor, is in the state of monitoring the bus with the RT selection mask selected.

Synopsis

unsigned short ms1553m_bcrtm_active(board);

UINT board;
The board number to be accessed, 0 - 3.

Include Files

ms1553.h

Return Value

TRUE if active

FALSE if not.

ms1553m_bcrtm_exit

Description

This routine *must* always be called just prior to exiting from your program to the operating system, provided the driver has been initialized in one of the three states. The interrupt system has been enabled and must be disabled properly in order to "clean up" the driver data structures for a clean exit.

If this routine is not called and the application exits back to the operating system, the next interrupt from the board will probably crash the host computer.

Multiple calls to this routine, and calling this routine before calling ms1553m_init() are both harmless.

This routine must be called for each board which has been initialized with a call to ms1553m_init().

Synopsis

ms1553m_bcrtm_exit(board);

UINT board;
The board number to be accessed, 0 - 3.

Include Files

ms1553.h

Return Value

None.

ms1553m_init (BC)

Description

To initialize the board properly as a BC, four words are required in the initialization structure "i_init" shown below. These are found in the "ms1553.h" file in the "typedef" struct called "initial_info" under the structure "bc_only":

```
struct bc_only
{
    unsigned short control;
    unsigned short max_data_buffers;
    unsigned short hi_pri_irupts;
    unsigned short max_mc_buffers;
} bc;
```

The first word, "control", contains certain user-specified bits that will be placed in control register zero of the BCRTM chip. Keep in mind these bits must be "ORed" together for form the final word. The bits are defined in the file "ms1553.h" and are listed below:

RETRY_ON_BUSY	/* retry if the RT is busy */
RETRY_ON_MSG_ERROR	/* retry if a message error occurs */
RETRY_ON_TIMEOUT	/* retry if the RT does not respond */
RETRY_ON_BC_MSG_ERROR	/* retry if there is a BC error */
RETRY_COUNT_ON_0	/* the LSB of a count of 1 to 4 retries prior to generating an error */
RETRY_COUNT_ON_1	/* the MSB of the retry count, the count is of the form (n-1) */
CHANNEL_SELECT_A	/* channel A is the primary channel */
CHANNEL_SELECT_B	/* channel B is the primary channel */
RETRY_ON_ALTERNATE_BUS	/* all retries are on the alternate bus from the primary one */
BC_SELECT	/* the BC is selected with this bit */
RT_RESP_32US	/* set the maximum RT response to 32 usec prior to a timeout error */
RT_RESP_16US	/* set the RT response to 16 usec */

No other bits must be set in this word. The driver initializes all other bits.

The next word contains the maximum number of unique 32 word data buffers you require on the board. To determine this, add up the number of transmit and receive commands in your frames. Each one uses a unique data buffer. If you use the ms1553m_bc_tr_same() or ms1553m_bc_rtrtmon_same() commands for the second through the Nth instance of commands, you need only one buffer for the first command and

then all the routines which end in "_same" refer to that buffer. See the detail on those two commands for a description of their use.

The third word contains the high-priority interrupts you wish to be enabled. The user-specified bits are the following:

MESSAGE_ERROR_ENABLE
SUBSYSTEM_FAIL_ENABLE
DYNAMIC_BUS_CONTROL_ENABLE
ILLOGICAL_COMMAND_ENABLE
DATA_OVERRUN_ENABLE
END_OF_BIT_ENABLE
BIT_WORD_FAIL_ENABLE

You may set any or all of these bits to zero. If you are making use of the user-defined high-priority interrupt routines, you must enable the appropriate bits by "ORing" the combination of these symbols to form the resultant word. The user-defined interrupt routines are discussed later.

Finally, the fourth word is similar to the second in that it is the number of one-word buffers required for unique mode-code-with-data-word instances. At least one mode code buffer is required. As with data buffers, if the ms1553m_tr_mode_same() commands are used, you need only one word for the first mode code.

You may specify enough information to allow the initialization routine to tailor the on-card memory to your specific requirements. The board address and interrupt lines are then setup for the driver and the card is activated. Once this routine is executed, you *must* execute an ms1553m_bcrtm_exit() prior to returning the operating system or the driver cannot "clean up" the interrupt handling, and your system may hang.

Synopsis

ms1553m_init(board, BC_MODE, i_init, NULL, irq, board_address , interrupt);

BT_UINT board;
The board number to be accessed, 0 - 3. In 32-bit Windows, this is the device number to open, and the board_address parameter is ignored.

BT_UINT mode_of_operation;
The symbol BC_MODE in ms1553.h

struct initial_info *i_init;
A pointer to a structure containing the initialization information for the control registers used for the BC mode of operation.

NULL
This pointer is only required for RT mode and can therefore be zero or NULL.

BT_UINT irq;
The interrupt number of the board. This must agree with the DIP switch setting. Note that if the host computer is an AT-class machine, use should specify interrupt “9”, never “2”, if you are using the IRQ 2/9 setting on the board. This applies even to an 8-bit board, which has a DIP Switch setting for IRQ “2”. Only if this is an XT-class machine (generally, only 8088-based machines fall into this class) should you specify IRQ “2” in the call to this function.

BT_U16BIT *board_address
The starting location for the 1553 on board memory which corresponds to the DIP switch setting of the board. In 32-bits, the lowest 2 bits of this parameter are used to determine the device number to open. If zero, than the "board" specifies the device number to open.

BT_UINT interrupt_mode_enable
You should specify “INTERRUPT” to enable hardware interrupts (recommended) or “POLLED” to specify polled (non-interrupt) operation. See the generic function ms1553m_int_handler() for more information.

Include Files

ms1553.h

Return Value

INIT_MEMORY_FULL	Specified number of data/command blocks exceed the available memory.
INIT_MEMORY_SIZE_ERROR	Memory specified by jumpers does not respond or does not exist.
INIT_MEMORY_ALLOC_ERROR	MALLOC of memory failed.
INIT_ILLEGAL_IRUPT_NUMBER	IRQ is < 2 or > 15.
INIT_NOT_RESPONDING	Board/chip is not responding.
INIT_CANT_MAP_ADDR	Cannot map the physical board address into protected address space.
INIT_ILLEGAL_MODE	Mode not RT_MODE, BC_MODE, or M_MODE.
INIT_NOT_BCRTM	Chip does not support monitor mode.
INIT_BAD_PARAMETER	Board number > 3 or NULL pointer passed in

The memory full error indicates the queue sizes requested require more than available on the board. The memory size error occurs when the configuration DIP switch on the board indicates either 16KW or 32KW of memory are available, but when the ms1553m_init() routine tests the area, the memory does not respond. The illegal interrupt number means you have selected an IRQ that isn’t of the legal set.

If the mode passed is not of the set BC_MODE, RT_MODE, or M_MODE, the illegal mode error is generated. Finally, the memory allocation error occurs when the driver attempts to allocate memory on the host for implementing the FIFO's used to ensure the messages and status get routed to the correct destinations, and cannot get the memory from the operating system.

ms1553m_init (RT)

Description

As indicated earlier, initializing the RT involves a significant amount of information. This is passed in the two structures. We will first discuss the initial_info struct for the RT:

```
struct rt_only
{
    BT_U16BIT rt_address;
    BT_U16BIT control;
    BT_U32BITreceive_illegal_broadcast;
    BT_U32BITreceive_sub_addr;
    BT_U32BITtransmit_sub_addr;
    U16BIT rnum_messages[30];
    U16BIT tnum_messages[30];
    U16BIT rmsg_length[30];
    U16BIT tmsg_length[30];
    U16BIT hi_pri_irupts;
} rt;
```

The first parameter, "rt_address", is the desired RT address for the board from 0 through 30. The next word, "control", indicates which bus or busses to use, A and/or B. These bits are:

CHANNEL_A_ENABLE

CHANNEL_B_ENABLE

You must "OR" these together to form the resultant word.

Next, are three 32-bit words that are basically bit fields. A "C" bit field definition is not used since it is not a portable construct to different machines. The LSB of the word corresponds to subaddress one. A one in a bit position makes the subaddress illegal, a zero legalizes it. The first word is for broadcast commands, the second for RT receive commands, and the third for RT transmit commands.

Next are two arrays, "rnum_messages" and "tnum_messages". Each element of the array contains a number that is the maximum number of messages to expect in the queue for receive and transmit. The first array element is for subaddress one. The next two arrays, "rmsg_length" and "tmsg_length" contain the size of each message in the queue. Normally, this is 32, but sometimes cases arise where the number is different. The ms1553m_init() routine then computes the proper queue sizes and indices for the hardware and setups all of the pointers properly.

Finally, the "hi_pri_irupt" parameter allows enabling the Built-in-Test (BIT) and other high priority interrupts, if you are using User Defined Interrupt routines, which are covered later in this manual. These defines are found in "ms1553.h" and are:

STANDARD_ENABLE_INIT
MESSAGE_ERROR_ENABLE
END_OF_COM_BLOCK_ENABLE
BIT_WORD_FAIL_ENABLE
END_OF_BIT_ENABLE
SUBSYSTEM_FAIL_ENABLE
DYNAMIC_BUS_CONTROL_ENABLE
ILLOGICAL_COMMAND_ENABLE
DATA_OVERRUN_ENABLE

The next structure is the "mode_code" structure. The chip handles mode codes automatically and the driver allows you the option of generating interrupts when they occur. You must beware that if the high-priority interrupts are not enabled on the board and you have installed "User Defined Interrupts" for mode codes, your routines will not execute! Since the servicing is totally up to your routines, you are given maximum control over the generation of the interrupts as well as the legalization. Following is the discussion of the various parameters for this structure:

```
typedef struct mode code
{
        unsigned short int_on_reception;
        unsigned short illegalize_broadcast_no_word;
        unsigned short illegalize_mode_code_no_word;
        unsigned short illegalize_broadcast_with_word;
        unsigned short illegalize_transmit_with_word;
        unsigned short illegalize_receive_with_word;
        unsigned short int_when_mode_code_transacted;
} MODE_CODE;
```

With all but the last parameter, each must be viewed as a bit field of size 16-bits. There is a possible 32 mode codes, but they are split into 16 without data words and 16 with data words. Not all are defined at this time, but space is reserved for future expansion. For all of the parameters the LSB is the right-most bit, bit zero.

The "int_on_reception" word governs mode codes 0 through 15 or the mode codes without data words. A one in any bit position enables the interrupt for that mode code.

The "illegalize_broadcast_no_word" defines which of the mode code without data words are illegal for broadcast. This means that a one in any bit position makes that mode code ILLEGAL.

Similarly, the "illegalize_mode_code_no_word" defines which of the mode code without data words are illegal for reception. A one in any bit position makes that mode code ILLEGAL.

The next three parameters:

"illegalize_broadcast_with_word"
"illegalize_transmit_with_word"
"illegalize_receive_with_word"

are handled the same, but apply to the mode codes 16 through 31. You will note that since a data word is involved, some of the mode codes can be receive and some are transmit so you MUST legalize the correct type for that mode code.

The "int_when_mode_code_transacted" word governs the generation of interrupts for the mode code with data word codes 16 through 31. A one in any bit position will enable the interrupt for that mode code.

Synopsis

ms1553m_init(board, RT_MODE, i_init, m_init, irq, board_address, interrupt);

BT_UINT board;
The board number to be accessed, 0 - 3.

BT_UINT mode_of_operation
This parameter indicates which mode into which the board is to be initialized. Use the symbol RT_MODE that is defined in ms1553.h.

struct initial_info *i_init;
A structure containing all the initialization information needed for the RT mode of operation. This will be shown below in the description section.

struct mode_code *m_init;
A structure defining the RT mode code initialization information, which will also be discussed below.

BT_UINT irq;
The interrupt number of the board. This must agree with the DIP switch setting. Note that if the host computer is an AT-class machine, use should specify interrupt “9”, never “2”, if you are using the IRQ 2/9 setting on the board. This applies even to an 8-bit board, which has a DIP Switch setting for IRQ “2”. Only if this is an XT-class machine (generally, only 8088-based machines fall into this class) should you specify IRQ “2” in the call to this function

BT_U16BIT *board_address
The starting location for the 1553 memory. This is determined by the DIP switch settings on the board.

BT_UINT interrupt_mode_enable
You should specify “INTERRUPT” to enable hardware interrupts (recommended) or “POLLED” to specify polled (non-interrupt) operation. See the generic function ms1553m_int_handler() for more information.

Include Files

ms1553.h

Return Value

INIT_MEMORY_FULL

INIT_MEMORY_SIZE_ERROR

INIT_ILLEGAL_IRUPT_NUMBER

INIT_ILLEGAL_MODE

INIT_MEMORY_ALLOC_ERROR

The memory full error indicates the queue sizes requested require more memory than is available on the board. The memory size error occurs when the configuration DIP switch on the board indicates either 16KW or 32KW of memory are available, but when the ms1553m_init() routine tests the area, the memory does not respond. The illegal interrupt number means you have selected an IRQ that isn’t of the legal set. If the mode passed is not of the set BC_MODE, RT_MODE, or M_MODE, the illegal mode error is generated.

Finally, the memory allocation error occurs when the driver attempts to allocate memory on the host for implementing the FIFO's used to ensure the messages and status get routed to the correct destinations, and cannot get the memory from the operating system.

ms1553m_init (BM)

Description

Monitor initialization of the hardware is handled differently from the BC and RT forms. The section attempts to clarify these differences to provide a basic understanding, so that you can make good use of the tool kit provided. It doesn't delve into hardware detail, since this is not required.

First, for BC and RT modes, interrupts are an important part of the interface due to the interactive nature, command and response, type of application to hardware requirements. In monitoring you are interested in data collection and recording as primary functions, in other words we are just a "listener". When recording to disk, for some amount of time, we wish to have the ability to record at 100% bandwidth, provided the host hardware can support it. The recorder interface software, therefore wants to be designed for minimal overhead. To accomplish this, interrupts from the board are disabled and all accesses to the data and command information are direct to the hardware, skipping an extra movement to the application layer, as with the RT and BC modes. This provides for a very stream-lined interface, but also requires a lower-level interface on your part that means a deeper understanding of the hardware for monitor mode.

The memory is formatted into a circularly linked list of monitor command blocks, as discussed earlier. When started, the BCRTM fills each block, sets the "active bit" for the block, and proceeds to the next block. With the exception of a very small area used for the interrupt loglist, which is reserved for later expansion, the entire memory is this list of command blocks.

Since most everything is a fixed algorithm, there is very little information required to get things started. The monitor structure is shown below and consists of two words called "control" and "mon_control".

```
struct monitor_only
{
    unsigned short control;
    unsigned short mon_control;
} monitor;
```

The "control" word is the same as that used for the RT mode. You are to set up which channels are to be used by specifying either or both of the following symbols:

CHANNEL_A_ENABLE

CHANNEL_B_ENABLE

In the "mon_control" word only two bits are used: "SELECT_TERMINALS" and "MONITOR_ALL". If you select the "monitor all" function, everything is recorded at all times requiring the application to filter the data. If you use the "select terminals", the BCRTM will only record data for the remote terminals you select when calling the ms1553m_m_select_rt() routine. Generally, this is the preferred method, since it offers the greatest flexibility.

The application layer must perform filtering at the subaddress or data level.

Synopsis

ms1553m_init(board, M_MODE, i_init, NULL, irq, board_address, interrupt);

BT_UINTboard;
The board number to be accessed, 0 - 3.

BT_UINTmode_of_operation;
This parameter must be set to the M_MODE define found in the "ms1553.h" file.

struct initial_info *i_init;
A structure containing the initialization information for the monitor mode of operation. This is found in the "ms1553.h" file union called "initial_info".

NULL;
Not used in monitor mode so it can be NULL.

BT_UINTirq;
The interrupt number of the board. This must agree with the DIP switch setting. Note that if the host computer is an AT-class machine, use should specify interrupt “9”, never “2”, if you are using the IRQ 2/9 setting on the board. This applies even to an 8-bit board, which has a DIP Switch setting for IRQ “2”. Only if this is an XT-class machine (generally, only 8088-based machines fall into this class) should you specify IRQ “2” in the call to this function

BT_U16BIT *board_address
The starting location for the 1553 memory. This is determined by the DIP switch or jumper settings on the board.

BT_UINT interrupt_mode_enable
You should specify “INTERRUPT” to enable hardware interrupts or “POLLED” (recommended) to specify polled (non-interrupt) operation. See the generic function ms1553m_int_handler() for more information

Include Files

ms1553.h

Return Value

INIT_MEMORY_SIZE_ERROR

INIT_ILLEGAL_IRUPT_NUMBER

INIT_ILLEGAL_MODE

The memory size error occurs when the configuration DIP switch on the board indicates either 16KW or 32KW of memory is available, but when the ms1553m_init() routine tests the area, the memory does not respond. The illegal interrupt number means you have selected an IRQ that is not of the legal set. If the mode passed is not of the set BC_MODE, RT_MODE, or M_MODE, the illegal mode error is generated.

Note:

See the function “ms1553m_m_get_msg” for a simplified method for reading Bus Monitor messages from the board.

ms1553m_m_close

Description

Closes the currently open file. If no file is open, return an error.

Synopsis

ms1553m_m_close(board);

UINT board;
The board number to be accessed, 0 - 3.

Include Files

ms1553.h

Return Value

M_NOT_IN_MONITOR_MODE

M_FILE_ERROR

M_FILE_NOT_OPEN

The "file error" return indicates an error has occurred from the file system for the operating system in which the driver is operating. The "not in monitor mode" error means the driver and hardware has not been successfully initialized into monitor mode.

ms1553m_m_cur_cbptr

Description

This routine is used when the application is interactively monitoring the data being saved in memory on the board. It returns a pointer to the current monitor command block being written in memory by the BCRTM. If no bus activity is currently being written to memory, it is the next block that is written by the chip. From this point, the application can access the data directly. There are two more routines to be used in this mode, ms1553m_m_start_cbptr() and ms1553m_m_next_cbptr().

Synopsis

```
struct m_command_block *ms1553m_m_cur_cbptr( board );

UINT board;
```

The board number to be accessed, 0 - 3.

Include Files

ms1553.h

Return Value

A pointer to the current monitor command block being written, or about to be written.

ms1553m_m_flen

Description

Computes the count of the last valid command block in the currently open file and return it in the "last_block" variable passed. The current file position is left unchanged.

Synopsis

ms1553m_m_flen(board, last_block);

UINT board;
The board number to be accessed, 0 - 3.

unsigned long *last_block;
A pointer to where the count of the last valid command block in the currently open disk file shall be placed.

Include Files

ms1553.h

Return Value

M_NOT_IN_MONITOR_MODE

M_FILE_NOT_OPEN

ms1553m_m_fread

Description

Reads the specified number of command blocks from the current file position to the destination memory location specified. If fewer command blocks are available on the disk, moves what is available and returns an error. The "active" flag for the control/status word is cleared for the command block following the last one read, in this case. The file position will be left following the last command block read.

Synopsis

ms1553m_m_fread(board, dest, count);

U16BIT board;
The board number to be accessed, 0 - 3.

struct m_command_block *dest;
A pointer to one or more contiguous command block structures where the data are to be placed.

unsigned short count;
A count of the number command blocks to read to "dest".

Include Files

ms1553.h

Return Value

M_NOT_IN_MONITOR_MODE

M_FILE_NOT_OPEN

M_FILE_ERROR

M_FILE_EOF

M_FILE_FEWER_CBLKS_READ

ms1553m_m_fsearch

Description

This routine searchs the disk from the current file position for a match to the control block specified. To properly use this function, you must fully understand the match process algorithm. The match data are copied to a temporary command block and each word is "ANDed" with the corresponding word from the "mask" command block. Next , each word from the command block on the disk is "ANDed" with the "mask" and compared to the corresponding word in the temporary command block. If the entire command block matches, a zero is returned to indicate a match was found, and the file pointer remains at this command block. The command block number is returned in 'cb_num', which can then be used for relative seeks using the 'ms1553m_m_fseek() command. If any word does not match, the file position is incremented to the next block and the search continues. If the end-of-file is reached, an error is returned and the process stops.

Synopsis

ms1553m_m_fsearch(board, match, mask, cb_num);

UINT board;
The board number to be accessed, 0 - 3.

struct m_command_block *match;
A pointer to the command block which contains the data to be matched.

struct m_command_block *mask;
A pointer to the command block which contains the mask which must first be applied to the data being searched.

unsigned long *cb_num;
A pointer to where the returned command block number will be placed.

Include Files

ms1553.h

This technique allows the most flexible matching patterns possible, but remember to mask out words such as the tail pointer for the command block and other data that never match or are not meaningful.

Return Value

0 if a match is found

M_NOT_IN_MONITOR_MODE

M_FILE_NOT_OPEN

M_FILE_EOF

ms1553m_m_fseek

Description

This routine performs a file seek for the currently open file. The parameter specified is in command block units. This ensures positioning in complete data units and provides a clean path for future expansion. If you desire to set up an append function for the record routine, pass the symbol M_EOF.

Synopsis

ms1553m_m_fseek(board, cb_num);

UINT board;
The board number to be accessed, 0 - 3.

unsigned long cb_num;
The command block to seek on the disk. This is *not* a byte position. This is specified in command block units since that represents a complete unit of data and will be the most portable construct for future expansion. To position for append use the symbol M_EOF.

Include Files

ms1553.h

Return Value

M_NOT_IN_MONITOR_MODE

M_FILE_NOT_OPEN

ms1553m_m_fwrite

Description

Writes the specified number of command blocks from the source location in memory at the current file position. If room for the entire number of command blocks is unavailable on the disk, returns a file error. The file pointer is positioned following the last block written.

Synopsis

ms1553m_m_fwrite(board, src, count);

UINT board;
The board number to be accessed, 0 - 3.

struct m_command_block *src;
A pointer to one or more contiguous command block structures where the data are to be written from in memory to disk.

unsigned short count;
A count of the number command blocks to written to disk.

Include Files

ms1553.h

Return Value

M_NOT_IN_MONITOR_MODE

M_FILE_NOT_OPEN

M_FILE_ERROR

ms1553m_m_get_msg

Description

This routine fetches the next available message from the Bus Monitor message list. It provides a compact, simple method for obtaining a formatted version of the BM message buffer, that is also compatible with languages such as Visual Basic.

It also performs the house keeping functions, such as testing to see if the buffer is valid, releasing the buffer, clearing the RT-RT command words and chaining to the next buffer.

Its use is recommended whenever the application needs to read and process BM buffers in real time.

Synopsis

ms1553m_m_get_msg(board, ciptr);

UINT board;
The board number to be accessed, 0 - 3.

MON_COMMAND_DATA FAR *ciptr;
This is a pointer to a user-supplied instance of the structure ”MON_COMMAND_DATA”, which is defined in the file ms1553.h

Include Files

ms1553.h

Return Value

0 indicates that the supplied structure now contains a new BM message

M_NOT_IN_MONITOR_MODE

M_NO_DATA_AVAILABLE

Notes

This function offers a greatly simplified method for reading Bus Monitor messages from the board. The MON_COMMAND_DATA structure that is returned by this function has much of the message information decoded, and the function handles the entire house keeping needed by the board.

See the example program MON.C for an example Bus Monitor that uses this function, or the Visual Basic example program.

ms1553m_m_next_cbptr

Description

This routine is used when the application is interactively monitoring the data being saved in memory on the board. It returns a pointer to the next monitor command block in memory, relative to the command block pointer passed. This routine would be used when you wish to step from one command block to another in memory as your application processes the data. There are two more routines to be used in this mode, ms1553m_m_start_cbptr() and ms1553m_m_cur_cbptr().

Synopsis

```
struct m_command_block *ms1553m_m_next_cbptr( board, cbptr );

UINT board;
```

The board number to be accessed, 0 - 3.

```
struct m_command_block *cbptr;
```

A pointer to the command block from which you desire to step to the next command block in the link-list.

Include Files

ms1553.h

Return Value

A pointer to the next monitor command block in the linked-list, relative to the pointer passed.

ms1553m_m_open

Description

Opens a file for low level I/O in binary mode for both read and write. This needs to be done only if record functions are being used. Otherwise, interactive monitoring doesn't require this function. The open options are not available to you so that the driver can ensure optimization for speed.

Synopsis

ms1553m_m_open(board, path);

UINT board;
The board number to be accessed, 0 - 3.

char *path;
A pointer to a user supplied file name.

Include Files

ms1553.h

Return Value

M_FILE_ALREADY_OPEN

M_FILE_ERROR

M_NOT_IN_MONITOR_MODE

The "file error" return indicates an error has occurred from the file system for the operating system in which the driver is operating. The "not in monitor mode" error means the driver and hardware has not been successfully initialized into monitor mode.

ms1553m_m_record

Description

This routine records all of the specified RT data traffic to the currently open file for the specified number of seconds or until a control-break is hit for early termination. If you have initialized the driver with the "SELECT_TERMINALS" option, you must use the ms1553m_m_select_rt() routine first to select which remote terminals are to be monitored. If the record function is to begin in a file position within a file which already contains data, you must first call ms1553m_m_fseek() with the appropriate position indicator.

Synopsis

ms1553m_m_record(board, sec_to_record, update_function, call_interval);

BT_UINT board;
The board number to be accessed, 0 - 3.

BT_U32BITsec_to_record;
Data will be recorded for the specified number of seconds, or until a control-break is hit.

void (*update_function) (unsigned int)
The pointer to a function that will be called during the recording process evey “call_interval” seconds. It is passed the number of seconds that have elapsed since the last time it was called.

BT_UINT call_interval
The number of seconds between calls to the update_function.

Include Files

ms1553.h

Return Value

M_NOT_IN_MONITOR_MODE

M_FILE_ERROR

M_FILE_NOT_OPEN

ms1553m_m_select_rt

Description

This routine is used if you have used the "SELECT_TERMINALS" option for the "mon_control" word required when the ms1553m_init() was executed. The "rt_mask" parameter gives you the option of choosing which remote terminals are to be monitored by the hardware. This provides the ability to filter the bus traffic through the hardware. The resultant data saved by the monitor system in the command blocks are only the remote terminals in question. This reduces the filtering required by the application.

Synopsis

ms1553m_m_select_rt(board, rt_mask);

UINT board;
The board number to be accessed, 0 - 3.

unsigned long rt_mask;
The bit positions containing a "1" in this 32-bit word indicate the RT(s) to monitor where the LSB represents RT zero. The default is none selected (all zeros).

Include Files

ms1553.h

Return Value

M_NOT_IN_MONITOR_MODE

ms1553m_m_start

Description

Starts the BCRTM monitoring the bus with the currently set parameters. The command blocks are filled, and the "active bit" is set. The process continues until stopped by the application. This routine always begins at the first command block in memory.

Synopsis

ms1553m_m_start(board);

UINT board;
The board number to be accessed, 0 - 3.

Include Files

ms1553.h

Return Value

M_NOT_IN_MONITOR_MODE

ms1553m_m_start_cbptr

Description

This routine is used when the application is interactively monitoring the data being saved in memory on the board. It returns a pointer to the first monitor command block in memory. From this point, the application can access the data directly. There are two more routines to be used in this mode, ms1553m_m_cur_cbptr() and ms1553m_m_next_cbptr().

Synopsis

struct m_command_block *ms1553m_m_start_cbptr(board);

U16BIT board;
The board number to be accessed, 0 - 3.

Include Files

ms1553.h

Return Value

A pointer to the first monitor command block.

ms1553m_m_stop

Description

Stops the BCRTM from monitoring the bus. Activity from this point forward is no longer placed in the command blocks in memory.

Synopsis

ms1553m_m_stop(board);

UINT board;
The board number to be accessed, 0 - 3.

Include Files

ms1553.h

Return Value

M_NOT_IN_MONITOR_MODE

ms1553m_rt_busy

Description

This function sets the “busy” bit in the RT status word returned from the RT to the state requested by the true/false flag passed. You may set the flag at anytime, but remember that for most MIL-STD-1553 systems it is not an approved method of "holding off" a response from an RT.

For short periods, or if your system is designed to a different specification, this gives you access to the RT Busy status bit.

Synopsis

ms1553m_rt_busy(board, t_f);

UINT board;
The board number to be accessed, 0 - 3.

unsigned short t_f;
True passed for this variable will set the status word bit to a one, a false will set the status word bit to a zero.

Include Files

ms1553.h

Return Value

None.

ms1553m_rt_chg_adr

Description

Sets the RT address to the value passed. This overrides the selection made in the "initial_info" union for ms1553m_init().

Synopsis

ms1553m_rt_chg_adr(board, new_rt_adr);

UINT board;
The board number to be accessed, 0 - 3.

unsigned short new_rt_adr;
The new RT address for the board. This must be between 0 and 30.

Include Files

ms1553.h

Return Value

None.

ms1553m_rt_clear_counts

Description

This routine can be called at anytime to clear the accumulated counts for both "msg_count" and "err_count" found in the status structure. It also clears the accumulators for both, found in the internal data structures for the interrupt routine.

Synopsis

ms1553m_rt_clear_counts(board, tr, subaddr);

UINT board;
The board number to be accessed, 0 - 3.

unsigned short tr;
Transmit/Receive bit (TBIT, RBIT).

unsigned short subaddr;
Subaddress for which the "msg_count" and "err_count" are to be cleared.

Include Files

ms1553.h

Return Value

None.

ms1553m_rt_dbusc

Description

This function sets the dynamic bus control bit, in the status word returned from the RT, to the state requested by the true/false flag passed.

Synopsis

ms1553m_rt_dbusc(board, t_f);

UINT board;
The board number to be accessed, 0 - 3.

unsigned short t_f;
True passed for this variable will set the status word bit to a one, a false will set the status word bit to a zero.

Include Files

ms1553.h

Return Value

None.

ms1553m_rt_free_link

Description

This functions performs the opposite function of the ms1553m_rt_tr_link() command. All linked subaddresses are placed back in there normal state. The interrupt routine no longer updates the application layer. If you initialized the queue size for any of the subaddresses to be larger than one message deep, the queue is set to one permanently after a link has been used. So you must be aware of this effect. Linked subaddresses should be initialized with queue sizes of one.

Synopsis

ms1553m_rt_free_links(board, tr, &sa_lst[0], sa_cnt);

UINT board;
The board number to be accessed, 0 - 3.

unsigned short tr;
Transmit/Receive bit (TBIT, RBIT).

struct sa_list sa_lst[sa_cnt];
An array of "sa_cnt" elements of "sa_list" structures. Each entry in the structure contains the subaddress and the number of data words in the message for each subaddress.

unsigned short sa_cnt;
The number of subaddress/count elements.

Include Files

ms1553.h

Return Value

None.

ms1553m_rt_get_sa_dbuf_ptr

Description

This routine is provided as a low-level toolkit-type routine. It returns an absolute pointer to the data buffer area for the selected subaddress and T/R selection. This provides the application direct access to the data buffer.

You must take care to address all structure members as declared, never recast as bytes, or permit the compiler to optimize your code to access bytes.

When using the driver Windows DLL, this routine provides a method of obtaining a protected-mode pointer to a data buffer on the board. If the normal real-mode "Segment:Offset" pointer is used, the application fails with a "General Protection Fault".

Synopsis

unsigned short *ms1553m_rt_get_sa_dbuf_ptr(board, tr, subaddr);

UINT board;
The board number to be accessed, 0 - 3.

unsigned short tr;
Transmit/Receive bit (TBIT, RBIT).

unsigned short subaddr;
Subaddress for which we wish the message status pointer.

Include Files

ms1553.h

Return Value

An absolute pointer to the requested data buffer area.

ms1553m_rt_get_sa_mstat_ptr

Description

This routine is provided as a low-level toolkit-type routine. It returns an absolute pointer to the message status word area for the selected subaddress and T/R selection. T/R selection. This provides the application direct access to the message status area.

You must take care to address all structure members as declared, never recast as bytes, or permit the compiler to optimize your code to access bytes.

When using the driver Windows DLL, this routine provides a method of obtaining a protected-mode pointer to a structure on the board. If the normal real-mode "Segment:Offset" pointer is used, the application fails with a "General Protection Fault".

Synopsis

unsigned short *ms1553m_rt_get_sa_mstat_ptr(board, tr, subaddr);

UINT board;
The board number to be accessed, 0 - 3.

unsigned short tr;
Transmit/Receive bit (TBIT, RBIT).

unsigned short subaddr;
Subaddress for which we wish the message status pointer.

Include Files

ms1553.h

Return Value

An absolute pointer to the requested message status area.

ms1553m_rt_instr

Description

This function sets the instrumentation bit, in the 1553 status word returned from the RT, to the state requested by the true/false flag passed.

Synopsis

ms1553m_rt_instr(board, t_f);

UINT board;
The board number to be accessed, 0 - 3.

unsigned short t_f;
True passed for this variable will set the status word bit to a one, a false will set the status word bit to a zero.

Include Files

ms1553.h

Return Value

None.

ms1553m_rt_reset_timeout

Description

Resets the host time out counter. See “ms1553m_rt_set_timeout”for information concerning the time out function in general. This reset is required if you wish to make use of the time out for the "link" function calls. Since the "link" is called only once to establish the link, this routine is required for the application program to have the ability to reset the current count when data are received or transmitted. The other routines, such as ms1553m_rt_tr_data(), resets the timeout counter automatically when the data are received.

Synopsis

ms1553m_rt_reset_timeout(board);

UINT board;
The board number to be accessed, 0 - 3.

Include Files

ms1553.h

Return Value

None.

ms1553m_rt_set_timeout

Description

This function generates an error if no data are received or transmitted within a specified amount of time. For example, if the 1553 bus is disconnected, the host RT software would normally just "hang" waiting for the data. If this function is enabled, the host real-time clock is used to generate a time out delay as specified. Then, if the time out occurs, the open RT requests will be satisfied with a time out error. The error is returned as a run-time error in the status structure.

Synopsis

ms1553m_rt_set_timeout(board, t_f, milliseconds);

UINT board;
The board number to be accessed, 0 - 3.

unsigned short t_f;
True passed for this variable will enable the time out function, a false will disable the function.

unsigned long milliseconds;
The number of milliseconds to set the time out value if "t_f" is true. The granularity of this function is dependent upon the real time clock function available in the host.

Include Files

ms1553.h

Return Value

None.

ms1553m_rt_srq

Description

This function sets the service request bit, in the 1553 status word returned from the RT, to the state requested by the true/false flag passed.

Synopsis

ms1553m_rt_srq(board, t_f);

UINT board;
The board number to be accessed, 0 - 3.

unsigned short t_f;
True passed for this variable will set the status word bit to a one, a false will set the status word bit to a zero.

Include Files

ms1553.h

Return Value

None.

ms1553m_rt_ssfail

Description

This function sets the subsystem fail bit, in the 1553 status word returned from the RT, to the state requested by the true/false flag passed.

Synopsis

ms1553m_rt_ssfail(board, t_f);

UINT board;
The board number to be accessed, 0 - 3.

unsigned short t_f;
True passed for this variable will set the status word bit to a one, a false will set the status word bit to a zero.

Include Files

ms1553.h

Return Value

None.

ms1553m_rt_t_update

Description

This command is designed to be used *only* for subaddresses which have been "linked" with the ms1553m_rt_tr_link() command. The transmit form of the link has the problem where the data area for transmit on the board may be data that is too old. This might occur when you have a case where data are received from the BC and you wish to immediately have them ready to be transmitted back for verification. If the normal link is used, the previous data would go out from the board and upon completion, the new data moved by the interrupt routine to the board for the next sample. This delay isn't acceptable in some circumstances. Therefore, the data on the board must be updated or "freshened" immediately.

This routine ensures that the data are moved to the board, but there is *no* phasing error. Once the host starts moving the data, it completes, ensuring that the data transmitted is a complete set. Either the old data or the new data are transmitted, but not a combination of both.

Synopsis

ms1553m_rt_t_update(board, &sa_lst[0], sa_cnt, data_ptr);

UINT board;
The board number to be accessed, 0 - 3.

struct sa_list sa_lst[sa_cnt];
An array of "sa_cnt" elements of "sa_list" structures. Each entry in the structure contains the subaddress and the number of data words in the message for each subaddress.

Description unsigned short sa_cnt;
The number of subaddress/count elements.

unsigned short *data_ptr;
A pointer to a user supplied block of memory containing the data to be transmitted. Each subaddress message cannot contain more than 32 words of data. All data words to be transmitted or received are placed contiguously in memory starting at the location pointed to by data_ptr and in the order specified by the "sa_lst" structure.

Note: The caller's data buffer should not cross a segment boundary.

Include Files

ms1553.h

Return Value

RT_INVALID_DESCRIPTOR

ms1553m_rt_tflag

Description

This function sets the terminal flag bit, in the 1553 status word returned from the RT, to the state requested by the true/false flag passed.

Synopsis

ms1553m_rt_tflag(board, t_f);

UINT board;
The board number to be accessed, 0 - 3.

unsigned short t_f;
True passed for this variable will set the status word bit to a one, a false will set the status word bit to a zero.

Include Files

ms1553.h

Return Value

None.

ms1553m_rt_tr_data

Description

This routine is used to transmit or receive data to or from a subaddress. For a transmit, the data are moved from the application layer pointed to by "data_ptr" to the queue on the board. If other data are stacked to be transmitted, they are placed at the end of the queue so the data are sent in the proper order. For a receive, the data area for the subaddress is checked to see if new data are in the queue. If it is, it is moved to the application layer at "data_ptr". Otherwise, the request is queued, along with the destination pointer for the data. When the data are received, they are moved to the "data_ptr" and the status is updated. Both commands allow you to issue a "wait" so the routine will return only after the request has been completed.

You may wish to enable the host timeout function for this command, so see the ms1553m_rt_set_timeout() command.

Synopsis

ms1553m_rt_tr_data(board,tr, s_adr, cnt, data_ptr, stat_info, wait);

UINT board;
The board number to be accessed, 0 - 3.

unsigned short tr;
Transmit/Receive bit (TBIT, RBIT).

unsigned short s_adr;
The subaddress which will be transmitting/receiving the message.

unsigned short cnt;
The number of words in the message that is to be transmitted. For receive this is ignored, since the BC is controlling the word count.

unsigned short *data_ptr;
A pointer to the application's data area that either contains the data to be transmitted or is the destination for the data being received.

Note: The caller's data buffer should not cross a segment boundary.

struct status_info *stat_info;
A pointer to the status structure that is to be updated by the interrupt routine to indicate the status of this command.

BOOLEAN wait;

If true, do not return until the command has been completed and the data has either been transmitted or received. If false, the application must wait for the "completed" flag in the status structure to be set by the interrupt routine.

Include Files

ms1553.h

Return Value

RT_INVALID_DESCRIPTOR

RT_DESCRIPTOR_OVERFLOW

ms1553m_rt_tr_link

Description

This command is designed for special requirements where you are transmitting or receiving data split among multiple subaddresses. When this occurs, there are some special "tricks" which are used in the driver to cut overhead and make the application code easier as well. The routine assumes a contiguous data buffer in which the data are split as indicated by the array of structures "sa_lst", which contains the subaddress and word count for each. The driver then sets up a linked-list of pointers to the data areas linking the order of the data and subaddresses as described in the structure. Once this is done, the command has established a "link" between the hardware buffers on the board and the "data_ptr" area in the application. The driver jams the queue size to one for a link, so you must beware of this if you remove the link later. This "link" is one in which the interrupts for all but the last subaddress in the list are disabled to cut overhead.

When the last subaddress is accessed and the command is completed, the interrupt routine moves the data from the board to the "data_ptr" for a receive or from the "data_ptr" to the board for a transmit. This synchronizes the movement of the data to the BC. The application program need only clear the completed flag in one of the elements of the array of status structures "stat_info" and loop waiting for the completed flag to set, indicating the completion of the command. The data may then be moved from "data_ptr" while the hardware is transmitting or receiving the next set.

To remove the link, and return to normal operation, see the ms1553m_rt_free_links() command. Another command of use here is the ms1553m_rt_t_update(). This is used if you wish to "freshen" the data in the transmit buffers on the board. See the description of ms1553m_rt_t_update().

Note: Remember this command should only be executed once for each set of subaddresses for receive and transmit. Once the link is established you only need to work with the completed flags.

Synopsis

```
ms1553m_rt_tr_link( board, tr, &sa_lst[0], sa_cnt, data_ptr, &stat_info[0] );
```

UINT board;
The board number to be accessed, 0 - 3.

unsigned short tr;
Transmit/Receive bit (TBIT, RBIT).

struct sa_list sa_lst[sa_cnt];
An array of "sa_cnt" elements of "sa_list" structures. Each entry in the structure contains the subaddress and the number of data words in the message for each subaddress.

```
struct sa_list
{
    unsigned short s_adr;
    unsigned short count;
};
```

unsigned short sa_cnt;
The number of subaddress/count elements.

unsigned short *data_ptr;
A pointer to a user supplied block of memory containing the data to be transmitted/received. Each subaddress message cannot contain more than 32 words of data. All data words to be transmitted or received are placed contiguously in memory starting at the location pointed to by data_ptr and in the order specified by the "sa_lst" structure.

Note: The caller's data buffer should not cross a segment boundary.

struct status_info stat_info[sa_cnt];
An array of "sa_cnt" elements of status_info structures which will contain the results of the message transmission. Each element corresponds to the equivalent subaddress element of "sa_lst".

Include Files

ms1553.h

Return Value

RT_INVALID_DESCRIPTOR

ms1553m_rt_tr_mcdata

Description

If the "tr" bit is set to TBIT, this routine writes the specified data word into the mode code data word area on the board. When the BC executes a "mode code with data word" command, this word is transmitted to the BC. If the "tr" bit is set to RBIT, this routine reads the specified data word from the mode code data word area on the board and copies it to the location pointed to by "data_word".

Synopsis

ms1553m_rt_tr_mcdata(board, tr, mode, data_word);

UINT board;
The board number to be accessed, 0 - 3.

unsigned short tr;
Transmit/Receive bit (TBIT, RBIT).

unsigned short mode;
The mode-code-with-data-word whose data word is to be updated.

unsigned short *data_word;
A pointer to the data word that is to be set for the mode code if it is a transmit or a pointer to the destination where the received mode code data word is to be placed.

Include Files

ms1553.h

Return Value

None.

ms1553m_rt_tr_short_data

Description

This routine is used to transmit or receive short data words (3 words or less), to or from a subaddress. For a transmit, the data are moved from the application layer pointed to by "data_ptr" to the queue on the board. In receive mode the data are moved from the queue on the board along with the status to the application layer at "data_ptr".

Synopsis

ms1553m_rt_tr_short_data(board, trbit, s_adr, cnt, data_ptr, stat_info);

UINT board;
The board number to be accessed, 0 - 3.

unsigned short trbit;
Transmit/Receive bit (TBIT, RBIT).

unsigned short s_adr;
The subaddress which will be transmitting/receiving the message.

unsigned short cnt;
The number of words in the message that is to be transmitted. For receive this is ignored, since the BC is controlling the word count.

unsigned short *data_ptr;
A pointer to the application's data area that either contains the data to be transmitted or is the destination for the data being received.

Notes:
1. The caller's data buffer should not cross a segment boundary.
2. *data_ptr should not cross segment boundaries.

struct status_info *stat_info;
A pointer to the status structure that is to be updated by the interrupt routine to indicate the status of this command.

Include Files

ms1553.h

Return Value

RT_INVALID_DESCRIPTOR

RT_DESCRIPTOR_OVERFLOW

ms1553m_rtm_clear_tag_timer

Description

Clears the eight-bit time tag counter internal to the BCRTM to zero. All time tags are then relative to zero at this reset. The time tag counter begins counting immediately after this call completes.

Synopsis

ms1553m_rtm_clear_tag_timer(board);

UINT board;
The board number to be accessed, 0 - 3.

Include Files

ms1553.h

Return Value

None.

CHAPTER 15

Generic 1553 Routine Descriptions

This chapter provides descriptions of generic 1553 driver routines and user-defined interrupt routines. Most of the generic driver routines may only be called *after* ms1553m_init() has been called.

Failure to observe this rule results in memory corruption when using the DOS version of the driver, and a General Protection Fault when using the Windows .DLL version of the MS1553 driver.

Command Summary

ms1553m_bit_test(board)
ms1553m_break_hit(void)
ms1553m_break_irupt(void)
ms1553m_clear_break_flag(void)
ms1553m_config_sw(board)
ms1553m_get_version(void)
ms1553m_register_address(board, register_name)
ms1553m_register_read(board, register_name)
ms1553m_register_write(board, register_name,value
ms1553m_reset(board)
ms1553m_set_interrupt(board, name_of_routine, ui_code)
memwcpy(volatile U16BIT * dest, volatile U16BIT * source, int llength)

Single Board Command Summary

Previous versions of the MS1553 API only supported a single installed board. Current versions allow the use of up to 4 MS1553xxx boards in a PC at once. The API was changed to expect a first argument of the board number, while the previous version provided no such support.

ms1553m_bit_test

Description

This routine forces the BCRTM to run the built-in-test function. To do this the driver assumes it can take over the BIT "User Defined Interrupts". If your program uses these, you must restore them after the test. On return, a zero indicates a pass condition. If the high priority interrupts are not enabled on the board, the corresponding warning is returned. This is only a warning and does not mean the test failed. It means that the BIT "User Defined Interrupts" do not function because they are not enabled. Refer to the Hardware Interface Description and Installation chapters for information regarding how to enable the interrupt for your hardware.

Note: This function may *not* be called before ms1553m_init has been called. After this function has been executed, a re-initialization is required (e.g., ms1553m_init() must be called, as well as any other setup functions).

Synopsis

unsigned short ms1553m_bit_test(board);

UINT board;
The board number to be accessed, 0 - 3.

Include Files

ms1553.h

Return Value

Remember that the returned word is the "OR" of the bits indicated below:

0 indicates the test passed.

BIT_FAIL	BCRTM chip failed its internal diagnostics
BIT_TIMEOUT	End of BIT interrupt did not occur/interrupt not set
BIT_HI_PRI_SW_OFF	High priority interrupt switch is off (warning only)
BIT_CHA_FAIL	BCRTM chip failed channel A diagnostics
BIT_CHB_FAIL	BCRTM chip failed channel B diagnostics

ms1553m_break_hit

Description

The application program can call this routine to test the state of the driver's break flag. Usually, the break interrupt handler can call the ms1553m_break_irupt() routine and use this routine within the application to see if a break was hit after a "wait" condition was pending. If it was, the application's termination handling routines can be executed and the break flag must then be cleared with the ms1553m_clear_break_flag().

If this routine is called before ms1553m_init() is called, it returns a FALSE.

There is only one break flag for the entire driver; for that reason this function does *not* take the "board" argument.

Synopsis

unsigned short ms1553m_break_hit(void);

Include Files

ms1553.h

Return Value

A TRUE if the break flag is set; otherwise a FALSE.

ms1553m_break_irupt

Description

When writing the application software for the system, it is the responsibility of that application software to detect the control-break and control-c functions. When these occur, the software should execute the ms1553m_break_irupt() function call to inform the driver to abort all current "wait" conditions.

Once the driver has returned to the calling program, this break flag should be tested by the application using ms1553m_break_hit() following any pending driver calls with the "wait" function. The application can then process the exception. This break flag for the driver remains set until the application software executes the ms1553m_clear_break_flag() function to clear it.

This routine may be called before ms1553m_init() is called, but then it has no effect.

There is only one break flag for the entire driver; for that reason this function does *not* take the "board" argument.

Synopsis

```
ms1553m_break_irupt( void );
```

Include Files

ms1553.h

Return Value

None.

ms1553m_clear_break_flag

Description

When writing the application software for the system, it is the responsibility of that application software to detect the control-break and control-c functions. When these occur, the software should execute the ms1553m_break_irupt() function call to inform the driver to abort all current "wait" conditions.

Once the driver has returned to the calling program, this break flag should be tested by the application using ms1553m_break_hit() following any pending driver calls with the "wait" function. The application can then process the exception. This break flag for the driver remains set until the application software executes the ms1553m_clear_break_flag() function to clear it.

This routine may be called before calling ms1553m_init().

There is only one break flag for the entire driver; for that reason this function does *not* take the "board" argument.

Synopsis

```
ms1553m_clear_break_flag( void );
```

Include Files

ms1553.h

Return Value

None.

ms1553m_config_sw

Description

The function allows you to view the RAM size available on the MS1553 hardware, and other potentially useful characteristics of the board.

Note: You must *not* call this function before calling ms1553m_init().

Synopsis

unsigned short ms1553m_config_sw(board);

UINT board;
The board number to be accessed, 0 - 3.

Include Files

ms1553.h

Return Value

For each of the following, remember that these are bit fields within the returned word. Therefore, to test for each field you must first "AND" the returned word with CONFIG_*_MASK and then compare the result to the choices below it. If the window size selected on the board is 16KW (32KB) and the memory is 32KW (64KB), you receive a size of 16KW and the upper portion of the memory is non-accessible.

CONFIG_MEMSIZE_MASK	Memory size mask
CONFIG_MEMSIZE_8K	
CONFIG_MEMSIZE_16K	
CONFIG_MEMSIZE_32K	
CONFIG_CHIP_MASK	Chip type mask
CONFIG_BCRTM	Chip installed is a BCRTM
CONFIG_BCRT	Chip installed is a BCRT
CONFIG_SABTECH_MASK	Sabtech mask
CONFIG_SABTECH_AVAIL	Sabtech I/F avail
CONFIG_PC_ONLY	Sabtech not avail
CONFIG_PC96_SUPPORT_MASK	PC96 configuration mask
PC96_CONFIGURATION_BIT	MS1553PC96 board
PC96_BUS_A_ACTIVITY_BIT	MS1553PC96, activity detected on bus A
PC96_BUS_B_ACTIVITY_BIT	MS1553PC96, activity detected on bus B

ms1553m_get_version

Description

The function returns the version ID of the current MS1553 library software. This function is especially useful in a Windows .DLL environment, where it might be possible use either a newer or an older version of the library than was compiled into the application. This is possible since the DLL and the application are separate files.

There is only one break flag for the entire driver; for that reason this function does *not* take the "board" argument.

This routine can be called at any time.

Synopsis

unsigned short ms1553m_get_version(void);

Include Files

ms1553.h

Return Value

The value of “VERSION_ID_NUMBER”, which is defined in “ms1553.h” is returned by this function.

ms1553m_int_handler

Description

The function is only used when operating the API in the POLLED mode (see ms1553m_init for BC, RT and BM).

In POLLED mode there is no interrupt process to copy the 1553 data between the board and your buffers. Calling this function periodically allows the API to detect that the data has been sent or received, and update the STATUS_INFO structures and move the data.

Synopsis

void ms1553m_int_handler(BT_UINT board);

Include Files

ms1553.h

Return Value

None.

Notes

Normally the MS1553 API software is used in an interrupt-driven environment, but there are cases when a polled interface would be of value.

One such example might be a machine which did not have a hardware interrupt available, or when porting the software to an environment which did not support interrupts, or where such support was too complex to implement in the time available for the conversion.

To use the MS1553 API in a polled environment pass the “POLLED” constant to the ms1553m_init function.

Then when calling any function which takes a "WAIT" parameter, do NOT specify "WAIT". Rather specify "NOWAIT" to return from the function right away.

Then code a polling loop in your function which calls the API function "int8259_handler(UINT board)". In the loop which calls the int8259_handler() function, also test the "completed" word in the "status_info" for non-zero.

When the "completed" word is set than the message has been completed. All of the structures have been updated and all of the data words have been transferred.

ms1553m_register_address

Description

This routine is used to obtain the address of a specified MS1553PC register. If you are using the real-mode DOS driver, the address of any quantity on the MS1553PC board can be computed from the "Segment:Offset" address of the physical board, and this routine is not strictly necessary. However, in a protected mode, like Windows, when using the .DLL interface, the real-mode address must be converted to a protected-mode address. Any attempt to use a real-mode address in protected mode causes in a General Protection Fault.

This routine can be used to obtain the protected-mode address that is needed for directly accessing the MS1553. This address may be used in cases where the programmer wants to access the MS1553 board directly, bypassing the driver. In addition to the register names, which are defined, any word-offset can be passed to this routine, such as zero, which returns a pointer to the beginning of the board.

When using the address returned by this routine, take care to access the board as 16-bit words only, never as bytes!

This routine must not be called before ms1553m_init() has been called, or after ms1553m_exit() has been called. Also, the address returned by this routine may not be valid after ms1553m_exit() has been called.

Synopsis

unsigned short _far *ms1553m_register_address(board, register_name);

UINT board;
The board number to be accessed, 0 - 3.

unsigned short register_name;
This is the word offset of the register/data word whose address is to be returned. See "ms1553m_register_read()" for a list of register names which are defined in "ms1553.h".

Include Files

ms1553.h

Return Value

Address of the specified register is returned.

ms1553m_register_read

Description

The specified (16-bit word) register is read, and the value is returned to the caller. This function is useful in a Windows DLL environment, since the MS1553 board must be addressed using a protected-mode address, and any attempt to use the real-mode "Segment:Offset" address causes in a General Protection Fault.

Note: You must *not* call this function before calling ms1553m_init().

Synopsis

unsigned short ms1553m_register_read(board, register_name);

UINT board;
The board number to be accessed, 0 - 3.

unsigned short register_name;
This is the word offset of the register/data word to be read. The following register names are defined in "ms1553.h":

CONTROL_REGISTER	Register 0
STATUS_REGISTER	Register 1
BC_CURRENT_CMD_BLK_REG	Register 2
RT_DSCPTR_SPACE_ADR_REG	Register 2
M_CURRENT_CMD_BLK_REG	Register 2
POLLING_COMPARE_REGISTER	Register 3
BIT_WORD_REGISTER	Register 4
CURRENT_COMMAND_REGISTER	Register 5
INTERRUPT_LOG_LIST_PTR	Register 6
BCRT_ENABLE_HIGH_PRI_INT_REG	Register 7
BCRT_STATUS_HIGH_PRI_INT_REG	Register 8
STANDARD_INT_ENABLE_REG	Register 9
RT_ADDRESS_REGISTER	Register 10
BUILT_IN_TEST_REGISTER	Register 11
RESET_START_REGISTER	Register 12
REMOTE_TERMINAL_TIMER_RESET	Register 13
M_CONTROL_REGISTER	Register 14
SWITCH_REGISTER	Raw board config switch register

Include Files

ms1553.h

Return Value

Value of specified register is returned.

ms1553m_register_write

Description

The specified (16-bit word) register is written. This function is useful in a Windows DLL environment, since the MS1553 board must be addressed using a protected-mode address, and any attempt to use the real-mode "Segment:Offset" address causes in a General Protection Fault.

Note: You must *not* call this function before calling ms1553m_init().

Synopsis

void ms1553m_register_write(board, register_name,value);

UINT board;
The board number to be accessed, 0 - 3.

unsigned short register_name;
This is the word offset of the register/data word to be written. The following register names are defined in "ms1553.h":

unsigned short value;
The value to be written.

CONTROL_REGISTER	Register 0
STATUS_REGISTER	Register 1
BC_CURRENT_CMD_BLK_REG	Register 2
RT_DSCPTR_SPACE_ADR_REG	Register 2
M_CURRENT_CMD_BLK_REG	Register 2
POLLING_COMPARE_REGISTER	Register 3
BIT_WORD_REGISTER	Register 4
CURRENT_COMMAND_REGISTER	Register 5
INTERRUPT_LOG_LIST_PTR	Register 6
BCRT_ENABLE_HIGH_PRI_INT_REG	Register 7
BCRT_STATUS_HIGH_PRI_INT_REG	Register 8
STANDARD_INT_ENABLE_REG	Register 9
RT_ADDRESS_REGISTER	Register 10
BUILT_IN_TEST_REGISTER	Register 11
RESET_START_REGISTER	Register 12
REMOTE_TERMINAL_TIMER_RESET	Register 13
M_CONTROL_REGISTER	Register 14
SWITCH_REGISTER	Raw board config switch register

Include Files

ms1553.h

Return Value

None.

ms1553m_reset

Description

Issues a reset to the BCRTM that resets the internal encoder/decoders.

Note: You must *not* call this function before calling ms1553m_init().

Synopsis

ms1553m_reset(board);

UINT board;
The board number to be accessed, 0 - 3.

Include Files

ms1553.h

Return Value

None.

ms1553m_set_interrupt

Description

This facility is provided to handle special high priority interrupt situations which arise such as message errors and dynamic bus control, as well as mode codes which are received when in RT mode. The following is a list of the user-defined interrupts available:

```
UI_HP_MESSAGE_ERROR      /* bc or rt mode (Reg 8 bit 1) */
UI_END_OF_CMD_BLOCK_LST  /* bc mode       (Reg 8 bit 2) */
UI_BIT_WORD_FAIL         /* bc or rt mode (Reg 8 bit 3) */
UI_END_OF_BIT            /* bc or rt mode (Reg 8 bit 4) */
UI_SUBSYSTEM_FAIL        /* bc or rt mode (Reg 8 bit 5) */
UI_DYNAMIC_BUS_MODE_CODE /* rt mode       (Reg 8 bit 6) */
UI_ILLOGICAL_COMMAND     /* bc or rt mode (Reg 8 bit 7) */
UI_DMA_DATA_OVERRUN      /* bc or rt mode (Reg 8 bit 8) */

UI_LL_MESSAGE_ERROR      /* bc or rt(Log List Status bit 1)    */
UI_RETRY_FAIL            /* bc only (Log List Status bit 2)    */
UI_ILLEGAL_COMMAND       /* rt only (Log List Status bit 4)    */
UI_POLL_COMPARISON_FAIL  /* bc only (Log List Status bit 3)    */
UI_ILLEGAL_BROADCAST     /* rt only (Log List Status bit 5)    */
UI_RESPONSE_WITH_MSG_ERR /* rt only (Log List Status bit 13)   */
UI_END_OF_BC_CMD_BLOCK   /* bc only (any message complete)     */

UI_MC_WITHOUT_DATA_WORD  /* rt only (Log List Status bit 6 +   */
                         /*  RT Mode Code descriptor W0 Bit 15)*/
UI_MC_WITH_DATA_WORD     /* rt only (Log List Status bit 7 +   */
                         /*  RT Mode Code descriptor W0 Bit 8) */
UI_RT_RECEIVE_MSG        /* rt only, Receive ANY non MC msg,   */
                         /* BEFORE data transfer.              */
UI_RT_TRANSMIT_MSG       /* rt only, Transmit ANY non MC msg   */
                         /* BEFORE data transfer.              */
UI_RT_RECEIVE_MSG_DONE   /* rt only, Receive any non MC msg    */
                         /* AFTER API has SUCCESSFULLY         */
                         /* transfered data to user's buffer   */
UI_RT_TRANSMIT_MSG_DONE  /* rt only, Transmit any non MC msg   */
                         /* AFTER API has SUCCESSFULLY         */
                         /* transfered data from user's bufr   */
UI_HW_INTERRUPT_START    /* start of hardware interrupt fn     */
UI_HW_INTERRUPT_END      /* end of hardware interrupt fn       */
```

Some of these interrupts are not really useful to most applications, but are provided for completeness. Each interrupt routine should be defined to be a standard function call, not an interrupt procedure. This is due to the fact that the driver interrupt routine checks the table for a non-NULL address. If one is available, the driver interrupt routine calls it as a function.

The parameters passed to each function are defined in the file ms1553.h. Refer there for calling sequences and parameter definitions.

If you wish to "uninstall" a routine, you may execute the ms1553m_set_interrupt() routine with a NULL for the routine address. Extreme care must be taken when using these functions. You must fully understand the consequences of anything that is performed by the application upon the operating system or the hardware. Any DOS or BIOS system calls in any application interrupt routine including indirect calls are illegal and normally cause severe unpredictability in the 1553 driver.

Windows NT, or 32-bit Windows 95 programs, don't have this problem since the user interrupt function is called from a high-priority user-mode thread, *not* an interrupt function.

Disable stack checking when using this function since your function is called using the interrupt stack, not the stack supplied by your program. Additionally, when targeting 16-bit DOS or Windows you should specify DS != SS when compiling your program.

Note: Do *not* call this function before calling ms1553m_init() since ms1553m_init() clears all of the interrupt routine addresses.

For DOS programs, the user function is called according to the "cdecl" calling convention. For 16-bit Windows programs, the calling convention is "WINAPI" (far Pascal), and for 32-bit Windows programs the calling convention is "_stdcall".

See the "USERINTERRUPTFUNCTION" definition, and the prototype for the ms1553m_set_interrupt() function in the ms1553.h file for more information about the calling convention and arguments passed to the user interrupt function specified by the "name_of_routine" argument specified below.

Synopsis

ms1553m_set_interrupt(board, name_of_routine, ui_code);

UINT board;
The board number to be accessed, 0 - 3.

void name_of_routine();
The name of the application routine that is to be installed in the user interrupt table. A NULL passed here will de-install the vector.

unsigned short ui_code;
The user interrupt code found in ms1553.h where the vector should be installed.

Include Files

ms1553.h

Return Value

UI_INVALID_IRUPT_NUMBER.

memwcpy

Description

Copies data to and from the board using WORD accesses only.

You must use only WORD accessed to the MS1553xxx family of boards. While all API accesses are known to be WORD accesses only, the typical "C" library function "memcpy()" and its equivalent functions are typically BYTE moves. These accesses will fail.

This function is provided to eliminate this problem.

Caution Operation with overlapping arrays is not defined.

You *may* call this function before calling ms1553m_init(), or after calling ms1553m_bcrtm_exit(), since it is a generic memory copy function, not specifically related to the MS1553-xxx API.

Synopsis

memwcpy(volatile U16BIT * dest, volatile U16BIT * source, int llength);

U16BIT * dest;
Pointer to the destination address.

U16BIT * source;
Pointer to the source address.

int length;
Number of WORDS to be copied.

Include Files

ms1553.h

Return Value

None.

CHAPTER 16

Visual Basic Interface

Introduction

This chapter discusses the interface between the Condor Engineering-supplied DLL (MST1553.DLL) and the Microsoft Visual Basic programming environment.

Microsoft Visual Basic Versions 4.0 and 5.0 support the use of external DLL's such as the MST1553.DLL which is supplied by Condor Engineering for use with the MS1553 series of boards. Complete documentation of the function calls provided is included in Part 2 of this document beginning with Chapter 10 and extending through Chapter 15. Please refer to these chapters for detailed information about the supplied Application Programmers Interface (API).

A text file, MST1553.TXT, is provided to aid the Version 4.0 or 5.0 Visual Basic user in using the MST1553 DLL. The text file contains Declare statements, Const Statements, and User Defined data types.

While Version 4.0 of Visual Basic is supported, Visual Basic Version 5.0 is recommended since it supports the *AddressOf* operator, which is required for implementing callbacks.

Defining MST1553.DLL Functions and Constants

Before you can call a function from the MST1553.DLL "C" function library, you must define it to the Visual Basic programming environment. You can manually copy and paste definitions from the supplied MST1553.TXT file to your project or you can use the Microsoft API Text Viewer Utility included with Visual Basic. For more information on the API Text Viewer consult the Microsoft Visual Basic documentation.

Although you can import the entire MST1553.TXT file into your Visual Basic project, Microsoft recommends that you include only the

declarations, constants, etc. that are actually needed by your program to reduce memory usage.

Calling ms1553m_init() from Visual Basic

The function ms1553m_init() is the first function that must be called to setup the interface between the user's program and the MS1553 board. Successfully calling this function is a big first step in writing a Visual Basic program that uses the MS1553 board.

The ms1553m_init() function's third argument (initial_info) is a pointer to a union. Since Visual Basic doesn't have an equivalent to the C union, this function is "aliased" to three functions:

Visual Basic Functions	Operation	User Defined Type
ms1553m_init_bc()	Initializes to BC mode	bc_only
ms1553m_init_rt()	Initializes to RT mode	rt_only
ms1553m_init_mon()	Initializes to Monitor Mode	monitor_only

Notes: For the board address parameter, pass a long variable equal to the board number. For example:

lngBoard = &H01 . You can also pass the number (0 -> 3) directly in the function.

The fourth parameter in the Visual Basic initialization functions is the mode_code user-defined data type. This should be passed only when calling ms1553m_init_rt. For ms1553m_init_bc and ms1553m_init_mon pass a 0 in place of a mode_code data type since mode_code data are not used in these functions.

MS1553 Functions Not Supported in Visual Basic

The following MS1553 API functions are not currently supported in Visual Basic. Contact Condor Engineering for the latest information.

- ms1553m_m_start_cbptr()
- ms1553m_m_cur_cbptr()
- ms1553m_m_next_cbptr()
- ms1553m_rt_get_sa_mstat_ptr()
- ms1553m_rt_get_sa_dbuf_ptr()
- ms1553m_bc_get_dbuf_ptr()
- ms1553m_bc_get_cblk_ptr()
- ms1553m_register_address()

Working with Unsigned Integers in Visual Basic

Visual Basic does not support unsigned integers. Since the MST1553 DLL uses unsigned integers problems can arise when attempting to set values in the upper half of the range.

Example: When MST1553.DLL uses an argument of C type *unsigned short*, the type is *integer* in Visual Basic. Both are 16-bit values, but the Visual Basic variable has a range of -32768 to 32767, and the C argument has a range of 0 to 65535. A Visual Basic error is generated if an *integer* type is set to value greater than 32767.

Solutions: There are two solutions to this problem. The easiest is to set the value of variables directly in Hex. To set an integer variable to 65535 use myVariable = &HFFFF. (note the &H syntax) . The second solution is to convert the desired unsigned value to the signed equivalent. This can be accomplished in a small utility function :

```
Function  u_conv ( unsigned as Long) as Integer
   Dim signed as Integer
   If unsigned > 32767 then
        signed = unsigned - 65536
        Else
        signed = unsigned
   End If
        u_conv = signed
End Function
```

When using returned values from the MST1553 DLL the opposite conversion can be made. In many cases the returned values form MST1553 DLL functions need to be compared to predefined values.

For more information on using Visual Basic with unsigned integers, consult Microsoft Article: “Q112673 How to Pass & Return Unsigned Integers to DLLs from VB” .

Using MST1553 Callbacks in Visual Basic

For callbacks to work properly, the callback function declarations must be set up correctly. If not, Visual Basic usually crashes. For this reason it is a good idea to save your work often when working with callbacks.

Callback functions allow the MST1553 DLL and underlying 1553 hardware to call functions asynchronously in your program when predefined events occur. The MST1553 DLL must first know the address of your function so it can be called when the proper event(s) occur. Visual Basic 5 introduced the *AddressOf* operator. This operator allows the address of a Visual Basic Function or Sub to be passed as an argument.

Bus Controller Interrupts:	Callback Parameters
UI_HP_MESSAGE_ERROR	(board, dum1, dum2);
UI_END_OF_CMD_BLOCK_LST	(board, dum1, dum2);
UI_BIT_WORD_FAIL	(board, dum1, dum2);
UI_END_OF_BIT	(board, dum1, dum2);
UI_SUBSYSTEM_FAIL	(board, dum1, dum2);
UI_DYNAMIC_BUS	(board, dum1, dum2);
UI_ILLOGICAL_COMMAND	(board, dum1, dum2);
UI_DMA_DATA_OVERRUN	(board, dum1, dum2);
UI_LL_MESSAGE_ERROR	(board, CmdBlkPtr, dum1);
UI_RETRY_FAIL	(board, CmdBlkPtr, dum1);
UI_POLL_COMPARISON_FAIL	(board, CmdBlkPtr, dum1);
UI_END_OF_BC_CMD_BLOCK	(board, CmdBlkPtr, dum1);
UI_HW_INTERRUPT_START	(board, dum1, dum2);
UI_HW_INTERRUPT_END	(board, dum1, dum2);

Remote Terminal Interrupts:	Callback Parameters
UI_MESSAGE_ERROR	(board, dum1, dum2)
UI_BIT_WORD_FAIL	(board, dum1, dum2)
UI_END_OF_BIT	(board, dum1, dum2)
UI_SUBSYSTEM_FAIL	(board, dum1, dum2)
UI_DYNAMIC_BUS	(board, dum1, dum2)
UI_ILLOGICAL_COMMAND	(board, dum1, dum2)
UI_DMA_DATA_OVERRUN	(board, dum1, dum2)
UI_LL_MESSAGE_ERROR	(board, SA_MC_Ptr, dum2)
UI_ILLEGAL_COMMAND	(board, SA_MC_Ptr, dum2)
UI_ILLEGAL_BROADCAST	(board, SA_MC_Ptr, dum2)
UI_RESPONSE_WITH_MSG_ERR	(board, SA_MC_Ptr, dum2)

Remote Terminal Interrupts:	Callback Parameters
UI_MC_WITHOUT_DATA_WORD	(board, mode_code, dum2)
UI_MC_WITH_DATA_WORD	(board, mode_code, data_wd)
UI_RT_RECEIVE_MSG	(board, subaddress, word_count)
UI_RT_TRANSMIT_MSG	(board, subaddress, word_count)
UI_HW_INTERRUPT_START	(board, dum1, dum2)
UI_HW_INTERRUPT_END	(board, dum1, dum2)

The following is a description of the callback parameters:

- board – the board number that caused the interrupt.
- CmdBlkPtr – WORD offset of the BC command block which caused the interrupt, relative to the start of the board.
- SA_MC_Ptr – WORD offset of the Subaddress/modecode descriptor which caused the interrupt, relative to the start of the board.
- mode_code – the mode code number.
- mc_data_wd – the mode code data word that was received.
- subaddress – the subaddress number which caused the interrupt.
- word_count – the number of words in the message which caused the interrupt (1-32).
- dum1. dum2 – dummy place holder, reversed for future versions.

Callback functions are void functions so the Visual Basic callback procedures must be Subs.

An Example Callback Function:

```
Public Sub int_handler(ByVal a As Long, ByVal b As Integer, ByVal d As Integer)
   Form1.Label1 = "This was called from Handler"
End Sub
```

The first parameter is declared "ByVal As Long" and the second and third parameters are declared "ByVal as Integer". All callback functions send these three parameters so all callback Subs must be declared as above. Remember to place callback Subs in a Visual Basic Standard Module.

To "install" the callback with the MST1553 DLL use the *ms1553m_set_interrupt* function as follows:

```
ms1553m_set_interrupt 0, AddressOf int_handler, UI_END_OF_BC_CMD_BLOCK
```

The *AddressOf* operator preceding the callback sub name. Visual Basic 4.0 doesn't support the *AddressOf* operator and therefore *cannot* process callbacks.

Using the ms1553m_bcrtm_exit Function

When running programs from the design environment in Visual Basic be sure to call the *ms1553m_bcrtm_exit* function before the program terminates. If you fail to do so, Visual Basic crashes. You can call *ms1553m_bcrtm_exit()* in a Form_Unload procedure to make sure it is called prior to exit. The *ms1553m_bcrtm_exit()* function can be called multiple times without harm.

Visual Basic Sample Application

A sample Visual Basic 5.0 project is provided and consists of the following files:

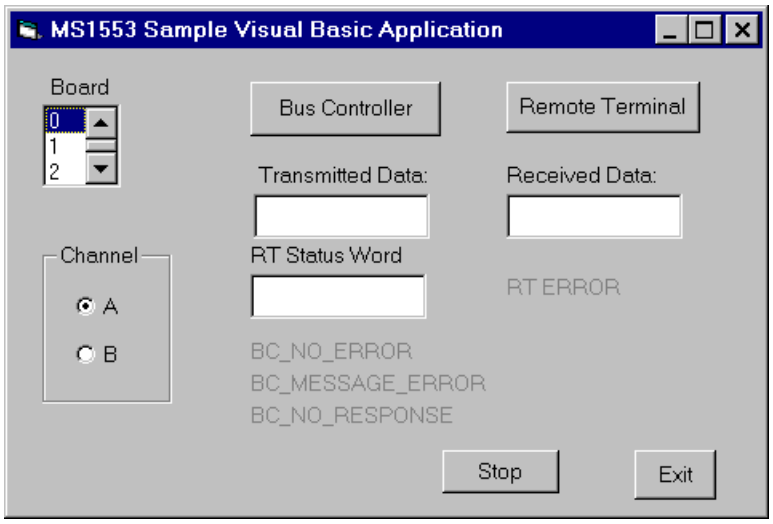
VB1553.FRM	Visual Basic Form
VB1553.BAS	Visual Basic Standard Module
VB1553.VBP	Visual Basic Project File
VB1553.EXE	Executable created by Visual Basic
Bmform.FRM	Visual Basic Form
frmSplash.FRM	Visual Basis Form

The application can act either as a 1553 Bus Controller, Remote Terminal or Bus Monitor. When in BC mode, one data word is transmitted to the selected RT address and sub-address. The data word is continuously transmitted with the data starting at zero and incrementing to 255. A 50mS delay occurs between each transmission. The returned RT status word, as well as BC_NO_ERROR, BC_MESSAGE_ERROR and BC_NO_RESPONSE errors are displayed. The data transmitted appears in the *Transmitted Data* textbox.

When in RT mode the selected RT address and sub-address is monitored. When data are received it is displayed in the *Received Data* textbox.

In Bus Monitor mode the selected RT addresses (0-31) are monitored. The display screen has a single region that is used to display the most recent message of each type (BC-RT, RT-BC, etc.).

If you want to run the VB example, but you do not have Visual Basic installed on your machine, you can run the setup.exe program in the examples\vb\setup directory. This will setup the necessary Visual Basic components needed to run the executable version of the example.



VB1553.FRM Source Code

Note that the screen shot above and the source code listings below are the previous version, before the incorporation of the monitor mode, and the selectable RT address and subaddress. Please see the distribution disks for the latest version of the VB example program.

```
' /*=====================================================*
' * FILE:                    V B 1 5 5 3 . F R M
' *=====================================================*
' *
' * COPYRIGHT (C) 1994 - 1998 BY
' *          CONDOR ENGINEERING, INC., SANTA BARBARA, CALIFORNIA
' *          ALL RIGHTS RESERVED.
' *
' *THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND
' *COPIED ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND
' *WITH THE INCLUSION OF THE ABOVE COPYRIGHT NOTICE.  THIS
' *SOFTWARE OR ANY OTHER COPIES THEREOF MAY NOT BE PROVIDED OR
' *OTHERWISE MADE AVAILABLE TO ANY OTHER PERSON.  NO TITLE TO
' *AND OWNERSHIP OF THE SOFTWARE IS HEREBY TRANSFERRED.
' *
' *THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT
' *NOTICE AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY CONDOR
' *ENGINEERING.
' *
' *=====================================================*/

' This program is written for Visual Basic 5.0.  It demonstrates
' several functions of the MST1553 DLL.  The MST1553.DLL file must
' be in the path for this application to run.

Option Explicit

Private Function bc_initialize()

    Dim ret As Integer
    Dim info As bc_only
    Dim irq As Integer
```

```
    info.max_data_buffers = 1
    info.max_mc_buffers = 1
    bc_initialize = 0   'return value if no errors
    gBoard = lstBrd.List(lstBrd.ListIndex)
    'get Channel
    If optA.Value = True Then
        info.control = CHANNEL_SELECT_A
    Else
        info.control = CHANNEL_SELECT_B
    End If
    ret = ms1553m_init_bc(0, BC_MODE, info, 0, 2, gBoard)
    If ret <> 0 Then
        err_handler "ms1553m_init_bc", ret
        bc_initialize = -1
    End If

End Function


Private Sub cmdExit_Click()
    ms1553m_break_irupt
    ms1553m_bcrtm_exit gBoard
    End
End Sub


Private Sub cmdRT_Click()

    Dim count1 As Long
    Dim ret As Integer
    Dim data_ptr(1) As Integer
    Dim status As status_info

    'set up board in RT mode
    If rt_initialize <> 0 Then Exit Sub
    gQuit = 1
    frmMain.MousePointer = vbHourglass

    Do While gQuit = 1
        ret = ms1553m_rt_tr_data(0, RBIT, 1, 1, data_ptr(0),
status, False)
        If ret <> 0 Then
            err_handler "ms1553m_rt_tr_data", ret
            Exit Sub
        End If
        Do While status.completed = False And gQuit = 1
            DoEvents  'allow user to hit "Stop"
        Loop
        DoEvents  'allow user to hit "Stop"
        'set completed flag so board can set it back to true
        status.completed = False
        'update data and error displays
        txtRx = data_ptr(0)
        If status.ms_error <> 0 Then
            lblMessErr.Enabled = True
        Else
            lblMessErr.Enabled = False
        End If
    Loop  'continue until "Stop" is clicked
    frmMain.MousePointer = vbNormal

End Sub


Private Sub cmdCancel_click()
    frmMain.MousePointer = vbNormal
    ms1553m_break_irupt
    gQuit = 0  'break transmit or receive loops
```

```
End Sub


Private Sub Form_Load()

    Dim j As Integer
    'load board lsit box and set default to 0
    optA.Value = True
    For j = 0 To 3
        lstBrd.AddItem j
    Next j
    lstBrd.ListIndex = 0

End Sub


Private Sub Form_Unload(Cancel As Integer)
    'call exit function before terminating
    ms1553m_bcrtm_exit 0
End Sub


Private Sub cmdBC_click()

    Dim ret As Integer
    Dim status As status_info
    Dim data_ptr(1) As Integer      'transmit data

    If bc_initialize <> 0 Then      'put board in BC mode
        Exit Sub
    End If
    gQuit = 1
    data_ptr(0) = 0                 '0 is first data sent
    frmMain.MousePointer = vbHourglass
    Do While gQuit = 1
        DoEvents  'allow user to click on Stop
        'send data
        ret = ms1553m_bc_tr_data(0, 1, RBIT, 1, 1, data_ptr(0),
status, True, 1000)
        If ret <> 0 Then
            err_handler "ms1553m_bc_tr_data", ret
        End If

        If data_ptr(0) = 255 Then data_ptr(0) = 0  'loop on 255
        data_ptr(0) = data_ptr(0) + 1
        Sleep 100  'Windows API call to wait 100mS
        'update transmit data
        txtXmit = data_ptr(0)
        'update RT status
        txtStatus = status.status1
        'update error displays
        If status.runtime_error = BC_NO_ERROR Then
            lblNoError.Enabled = True
        Else
            lblNoError.Enabled = False
        End If

        If status.runtime_error And BC_MESSAGE_ERROR Then
            lblMessageError.Enabled = True
        Else
            lblMessageError.Enabled = False
        End If

        If status.runtime_error And BC_NO_RESPONSE Then
            lblNoResponse.Enabled = True
        Else
            lblNoResponse.Enabled = False
        End If
```

```
    Loop        'continue until "Stop" is clicked
    frmMain.MousePointer = vbNormal

End Sub


Public Sub err_handler(func As String, err1 As Integer)

    Dim strErr As String
    'display message box with error
    Select Case err1
        Case INIT_CANT_MAP_ADDR
            strErr = "INIT_CANT_MAP_ADDR"
        Case INIT_ILLEGAL_MODE
            strErr = "INIT_ILLEGAL_MODE"
        Case INIT_BAD_PARAMETER
            strErr = "INIT_BAD_PARAMETER"
        Case INIT_MEMORY_ALLOC_ERROR
            strErr = "INIT_MEMORY_ALLOC_ERROR"
        Case INIT_NOT_RESPONDING
            strErr = "INIT_NOT_RESPONDING"
        Case INIT_ILLEGAL_IRUPT_NUMBER
            strErr = "INIT_ILLEGAL_IRUPT_NUMBER"
        Case INIT_MEMORY_SIZE_ERROR
            strErr = "INIT_MEMORY_SIZE_ERROR"
        Case Else   'show hex value for other errors
            strErr = "Error Code: " & Hex(err1) & "(Hex)"
    End Select
    MsgBox func & vbCrLf & strErr, vbExclamation, "1553 VB Demo"

End Sub


Private Function rt_initialize()

    Dim ret As Integer
    Dim info As rt_only
    Dim mcode As mode_code

    rt_initialize = 0
    info.rt_address = 1 'RT address 1
    info.receive_illegal_broadcast = &HFFFFFFFF
    info.receive_sub_addr = &HFFFFFFFE   'subaddresses 1 legal
    info.transmit_sub_addr = &HFFFFFFFE  'subaddresses 1 legal
    info.rmsg_length(0) = 32             'default message size
    info.rnum_messages(0) = 1            'only 1 message

    'get board from list box
    gBoard = lstBrd.List(lstBrd.ListIndex)
    'get channel
    If optA.Value = True Then
        info.control = CHANNEL_A_ENABLE
    Else
        info.control = CHANNEL_B_ENABLE
    End If
    ret = ms1553m_init_rt(0, RT_MODE, info, mcode, 2, gBoard)
    If ret <> 0 Then
        err_handler "ms1553m_init_rt", ret
        rt_initialize = -1
    End If

End Function
```

VB1553.BAS File:

```
' /*=============================================================*
' * FILE:                    V B 1 5 5 3 . B A S
```

```
' *=========================================================*
' *
' * COPYRIGHT (C) 1994 - 1998 BY
' *          CONDOR ENGINEERING, INC., SANTA BARBARA, CALIFORNIA
' *          ALL RIGHTS RESERVED.
' *
' *THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND
' *COPIED ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND
' *WITH THE INCLUSION OF THE ABOVE COPYRIGHT NOTICE.  THIS
' *SOFTWARE OR ANY OTHER COPIES THEREOF MAY NOT BE PROVIDED OR
' *OTHERWISE MADE AVAILABLE TO ANY OTHER PERSON.  NO TITLE TO
' *AND OWNERSHIP OF THE SOFTWARE IS HEREBY TRANSFERRED.
' *
' *THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT
' *NOTICE AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY CONDOR
' *ENGINEERING.
' *
' *=========================================================*/

'General Public Variables
Public gBoard As Long
Public gQuit As Integer

'MST1553 User Defined Types
Type mode_code
     int_on_reception As Integer
     illegalize_broadcast_no_word As Integer
     illegalize_mode_code_no_word As Integer
     illegalize_broadcast_with_word As Integer
     illegalize_transmit_with_word As Integer
     illegalize_receive_with_word As Integer
     int_when_mode_code_transacted As Integer
End Type
Type rt_only
     receive_illegal_broadcast As Long
     receive_sub_addr As Long
     transmit_sub_addr As Long
     rt_address As Integer
     control As Integer
     rnum_messages(30) As Integer
     tnum_messages(30) As Integer
     rmsg_length(30) As Integer
     tmsg_length(30) As Integer
     hi_pri_irupts As Integer
End Type
Type status_info
    completed As Boolean
    ms_error As Boolean
    runtime_error As Integer
    status1 As Integer
    status2 As Integer
    msg_status As Integer
    msg_count As Long
    error_count As Long
    delta_time As Integer
End Type
Type bc_only
     control As Integer
     max_data_buffers As Integer
     hi_pri_irupts As Integer
     max_mc_buffers As Integer
End Type

Type sa_list
     s_adr As Integer
     count As Integer
End Type
```

```
'MST1553 Constants
Global Const RT_MODE = 0
Global Const CHANNEL_A_ENABLE = &H80
Global Const CHANNEL_B_ENABLE = &H100
Global Const RBIT = 0
Global Const CHANNEL_SELECT_A = &H80
Global Const CHANNEL_SELECT_B = 0
Global Const BC_MODE = 1
Global Const INIT_CANT_MAP_ADDR = &H80
Global Const BC_NO_ERROR = 0
Global Const BC_NO_RESPONSE = 2
Global Const BC_MESSAGE_ERROR = 1
Global Const INIT_ILLEGAL_MODE = &H8
Global Const INIT_BAD_PARAMETER = &H100
Global Const INIT_MEMORY_ALLOC_ERROR = &H10
Global Const INIT_NOT_RESPONDING = &H40
Global Const INIT_ILLEGAL_IRUPT_NUMBER = &H4
Global Const INIT_MEMORY_SIZE_ERROR = &H2

'MST1553 Function Declarations
Declare Function ms1553m_rt_tr_data Lib "mst1553.DLL" (ByVal
board&, ByVal trbit%, ByVal s_adr%, ByVal count%, dptr%, status_ptr
As status_info, ByVal wait As Boolean) As Integer
Declare Function ms1553m_init_bc Lib "mst1553.DLL" Alias
"ms1553m_init" (ByVal board&, ByVal mode_of_operation%, ii_ptr As
bc_only, mc_ptr&, ByVal irq%, ByVal board_address&) As Integer
Declare Function ms1553m_init_rt Lib "mst1553.DLL" Alias
"ms1553m_init" (ByVal board&, ByVal mode_of_operation%, ii_ptr As
rt_only, mc_ptr As mode_code, ByVal irq%, ByVal board_address&) As
Integer
Declare Function ms1553m_bcrtm_exit Lib "mst1553.DLL" (ByVal
board&) As Integer
Declare Function ms1553m_break_irupt Lib "mst1553.DLL" () As
Integer
Declare Function ms1553m_reset Lib "mst1553.DLL" (ByVal board&) As
Integer
Declare Function ms1553m_bc_tr_data Lib "mst1553.DLL" (ByVal
board&, ByVal rt_adr%, ByVal trbit%, ByVal s_adr%, ByVal count%,
msg_ptr%, status_ptr As status_info, ByVal send As Boolean, ByVal
usecs%) As Integer

'Windows API Function Declarations
Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)
```

APPENDIX A

Linking the Driver and Sample Programs

Introduction

This appendix discusses linking the driver with the application and provides two sample “C” programs. Additional information and sample programs are available on the distribution diskette.

You should always set your compiler options to 1 byte structure alignment, LARGE model (16-bit applications) and disable stack checking.

Linking the Driver with a DOS Application

When you link the driver with your MS-DOS application, you must link it as a library. Libraries for Borland “C/C++” (MS1553TC.LIB) and Microsoft “C/C++” (MS1553MC.LIB) are both provided, in Large Model only. All pointers that are passed to driver routines must be “far” pointers. Compile your programs with stack checking disabled if you are using any interrupt functions.

When linking with a “C++” routine, you should note that the driver library is a “C” library, and therefore, the function names should *not* be mangled. This also means that any function names passed to the “ms1553m_set_interrupt()” routine must also be declared “ extern “C” ". You may need to modify the file “ms1553.h” to declare the driver routines as extern “C”.

The ms1553.h file assumes LARGE model; it does not explicitly declare functions and pointers as FAR, but they all must be FAR for the library to function correctly.

Linking the Driver with a WINDOWS Application

The driver for a Microsoft WINDOWS Version 3.1 application is supplied as DLL. To link your application with the driver, you should link with the library file "MS1553.LIB". This sets up your application to use the DLL, which is named "MS1553.DLL. When your application executes, MS1553.DLL must be loadable by Windows. This means that it must be in either the default folder, the Windows folder, or in a folder referenced in your PATH environment variable.

Windows DLL's are *language independent*, and can therefore be called from any language that supports Windows DLL's. Any functions passed to the "ms1553m_set_interrupt()" routine must be declared as "WINAPI" functions, just as if they were exported DLL functions.

The 32-bit libraries are linked with the multi-threaded libraries. If you are using interrupt functions (ms1553m_set_interrupt() function) your user functions are called from a thread created by the driver to service interrupts. For this reason you should link your application with a multi-threaded library.

Function calling conventions are fully defined in the file ms1553.h.

Check the "readme" file on the distribution diskette for the latest status on the libraries and DLLs.

Sample "C" Programs

Two C programs are included as examples of how to make use of the driver routines. The set of programs must be run together, with the RT being started first then the BC. You are required to supply the segment and interrupt request number to each program when they are started. The files were compiled with Borland C Version 5.02. The versions on the distribution diskette may have been updated from the versions listed here; see the actual code for the most up-to-date listings. The versions on the distribution diskette can also be compiled and linked with the Microsoft tools.

The distribution diskette also contains an example Windows program, which uses the DLL version of the driver.

BC.C

```
/*=============================================================================
=*
 * FILE:                        B C . C

*=============================================================================
*
 *
 * COPYRIGHT (C) 1994 - 2000 BY
 *          CONDOR ENGINEERING, INC., SANTA BARBARA, CALIFORNIA
 *          ALL RIGHTS RESERVED.
 *
 *          THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND
 *          COPIED ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND
WITH
 *          THE INCLUSION OF THE ABOVE COPYRIGHT NOTICE.  THIS SOFTWARE OR
ANY
 *          OTHER COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE
 *          AVAILABLE TO ANY OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF THE
 *          SOFTWARE IS HEREBY TRANSFERRED.
 *
 *          THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT
 *          NOTICE AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY CONDOR
 *          ENGINEERING.
 *

*=============================================================================
*
 *
 * FUNCTION:  Demonstrate BC functionality, using either the Microsoft
 *            MS-DOS library "MS1553MC.LIB", the Borland MS-DOS library
 *            "MS1553BC.LIB", or the MS1553 DLL for MS-WINDOWS,
"MS1553.DLL".
 *
 * ENTRY POINTS:
 *
 * DESCRIPTION:
 *      This routine will first send 4 packets to RT 1 at subaddresses
 *      20, 21, 22, 23.  The ms1553m_bc_tr_grp_data() is used with the SEND
 *      option set to TRUE.  This causes it to process all three messages
 *      before returning to the caller.  Next this routine performs 4 RT-BC
 *      messages from RT 1 at subaddresses 20, 21, 22, 23 again using the
 *      group message routine ms1553m_bc_tr_grp_data().
 *
 *      This process is repeated first on Bus A, then on Bus B.  The data
 *      received from the RT is tested to insure that it is identical to the
 *      data which was sent.
 *
 *      Each time data is sent to the RT it is changed.
 *
 *      The display indicates that the data was successfully wrapped by the
 *      RT back to us, by counting the number of messages which were
 *      successfully wrapped, and the number of words which did not compare.
 *      Additionally, the current message errors are displayed.
 *
 *      This program demonstrates basic functionality of the MS1553PC-x as
 *      a Bus Controller, in Temporary Frame Mode, and is intended to be
used
 *      with RTxx.EXE, which is assumed to be running on another board which
 *      is connected via 1553 to this board.
 *
 *      To start the program first start RTxx.EXE at the RT and then this
 *      program.  The main channel is channel A, this program only uses
 *      channel B if the user has specified to use it.
 *
 *      If the two host machines are not the same speed, than the faster
 *      machine should be running the RT simulation.
 *
 *      When running this program as a WINDOWS executable, remember that
 *      "MS1553.DLL" or "MST1553.DLL" must be accessable by WINDOWS (e.g.,
in
 *      the default directory, \WINDOWS, or in a directory in your PATH).

*============================================================================*
/
/* $Revision:  4.20 Release $ */
```

```
/*---------------------------------------------------------------------------*
 *                      INCLUDES, DEFINES and GLOBALS                        *
 *---------------------------------------------------------------------------
*/

#include <conio.h>
#include <dos.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#include "ms1553.h"

/* Set the compiler type */

#define MICROSOFT_C  0    /* Internal identification of Microsoft "C" */
#define TURBO_C      1    /* Internal identification of Borland   "C" */

#ifdef __TURBOC__
#define COMPILER_TYPE   TURBO_C
#else
#define COMPILER_TYPE   MICROSOFT_C
#endif

int  main(int argc, char **argv );

#define WORDS_PER_MESSAGE    2    /* Words in each message */
#define NUMBER_OF_MESSAGES   4    /* Number of messages to send/receive */
#define RT_NUM               1    /* The RT number                      */

/* This one contains the data received by the RT */
BT_U16BIT rt_rcv[NUMBER_OF_MESSAGES][WORDS_PER_MESSAGE];

/* This array contains the data to be transmitted  the RT */
BT_U16BIT rt_xmt[NUMBER_OF_MESSAGES][WORDS_PER_MESSAGE];

/* The error structure */
struct status_info bc_status[NUMBER_OF_MESSAGES],
bc_err[NUMBER_OF_MESSAGES];

/* Structure for bc */
struct initial_info   ist;

/* This one contains the subaddress/count array */
struct sa_list sal[NUMBER_OF_MESSAGES] = { {20, WORDS_PER_MESSAGE},
                                           {21, WORDS_PER_MESSAGE},
                                           {22, WORDS_PER_MESSAGE},
                                           {23, WORDS_PER_MESSAGE} };

static void  scr_clr(void);
static void  scr_clr_eol(void);
static void  scr_rowcol( int row, int col );

static BT_UINT  cardnum = 0;            // The number of the installed board

/*****************************************************************************
*
*
*  PROCEDURE - my_sleep()
*
*  FUNCTION
*     This procedure pauses for a specified number of milliseconds.
*
*  PARAMETERS
*     int wait - number of milliseconds to pause.
*
*****************************************************************************
/
void my_sleep(int wait)
{
#ifdef __WIN32__
   Sleep(wait);     /* Use the WIN32 function if available */
#else
   int         i, j;    // Loop indexes

   // Loop for the specified number of milliseconds
   for ( i = 0; i < wait; i++ ) {
      // Use ISA bus accesses to provide a processor-speed-independent
      //  high-resolution clock, at least sort of...
      for ( j = 0; j < 650; j++ ) {
```

```
         // Reading the board takes around 1.5 microseconds.  The function
         //  takes some time depending on the host processor speed.
         ms1553m_register_read(cardnum, 0);
      }
   }
#endif
}

/*=========================================================================*
 * ENTRY POINT:                S I G N O N
 *=========================================================================*
 *
 * FUNCTION:    Display the signon banner
 *
 * DESCRIPTION:
 *

*=========================================================================*/
static void signon(void)
{
    /* Clear the screen */
    scr_clr();

    scr_rowcol(1,10);
    printf("BC.C: MS1553 Example Bus Controller program.\n");
    scr_rowcol(2,10);
    printf("Version 4.20  Copyright (c) Condor Engineering, 1997-2000\n\n");
}

/*********************************************************************
*
*  static void CheckAndReportStatus(UINT status);
*
*  Check the return code and report any errors.
*
*********************************************************************/

static void CheckAndReportStatus(BT_UINT status)
{
   if ( status != 0 )
   {
      if ( status & INIT_ILLEGAL_IRUPT_NUMBER )
         printf("   IRQ is < 2 or > 15.\r\n");
      if ( status & INIT_MEMORY_FULL )
         printf("   Specified number of data/command "
                "blocks exceed the available board memory.\r\n");
      if ( status & INIT_MEMORY_SIZE_ERROR )
         printf("   Memory size specified by jumpers does "
                "not respond or does not exist.\r\n");
      if ( status & INIT_MEMORY_ALLOC_ERROR )
         printf("   MALLOC of memory failed.\r\n");
      if ( status & INIT_NOT_RESPONDING )
         printf("   Board/chip is not responding, check board
address.\r\n");
      if ( status & INIT_ILLEGAL_MODE )
         printf("   Mode is not RT_MODE, BC_MODE, or M_MODE.\r\n");
      if ( status & INIT_NOT_BCRTM )
         printf("   Chip does not support monitor mode.\r\n");
      if ( status & INIT_CANT_MAP_ADDR )
         printf("   Cannot map physical board address into "
                "protected address space.\r\n");
      printf("MS1553_INIT() returned error code %X (hex) as decoded
above.\r\n",
               status);
      ms1553m_bcrtm_exit(cardnum);
      exit(1);
   }
}

/*=========================================================================
=*
 * ENTRY POINT:         M A I N

*=========================================================================
*
 *
 * FUNCTION:    Main control
 *
```

```
*=====================================================================
*/

int main(int argc, char **argv )
{
    long    nMessages = 0;        // Number of messages sent/received.
    long    nDataDifferent = 0;   // Number of times RT data not consistant.
    long    nNoResponse = 0;      // Number of no response from the RT.
    int     nBus;                 // Bus code for A, B or Both
    char    CurrentBus;           // Current bus, either Primary or
Secondary
    int     BadMsg;               // Message number which has an error.
    int     BadWrd;               // Word within message which is in error.
    int     ReportError;        // Flag indicating an error should be
reported
    BT_U16BIT  irq;             // Interrupt number
    int     i, j;               // Loop indexes
    char    input_string[50];   // User input
    int     result;
    union {
        struct {
            BT_U16BIT  ofst;        // Offset, we set it to zero.
            BT_U16BIT  seg;         // Segment address from user.
        } so;
        BT_U16BIT  *bd_address;
    } baso;

    signon();      /* Display the signon banner */

#ifndef __WIN32__
    /* If the parameters are there, convert them, else ask the user */
    if ( argc != 4 ) {
       printf("Enter the 4-digit HEX segment address\n");
       printf(" of the MS1553 Board (ex: D000): ");
       fgets(input_string, sizeof(input_string), stdin);
       result = sscanf(input_string, "%x", &i);
       if ( result != 1 )
          i = 0xD000;
       printf("Enter the 1- or 2-digit DECIMAL interrupt number\n");
       printf(" of the MS1553 Board (ex: 15): ");
       fgets(input_string, sizeof(input_string), stdin);
       result = sscanf(input_string, "%i", &irq);
       if ( result != 1 )
          irq = 15;
       printf("Enter \'P\' to use the primary bus, \'S\' for secondary"
              " bus, \'B\' for both: ");
       fgets(input_string, sizeof(input_string), stdin);
       if ( (input_string[0] == 'P') || (input_string[0] == 'p') )
          nBus = 0;     // Use the primary bus only.
       else if ( (input_string[0] == 'S') || (input_string[0] == 's') )
          nBus = 1;     // Use the secondary bus only.
       else
          nBus = -1;    // Use both busses.
    }
    else {
       /* Now convert the params */
       sscanf( argv[1], "%x", &i );
       sscanf( argv[2], "%i", &irq );
       if ( (argv[3][0] == 'P') || (argv[3][0] == 'p') )
          nBus = 0;     // Use the primary bus only.
       else if ( (argv[3][0] == 'S') || (argv[3][0] == 's') )
          nBus = 1;     // Use the secondary bus only.
       else
          nBus = -1;    // Use both busses.
    }
    /* Convert to Segment:Offset format.  Could have used FP_SEG macro */
    baso.so.ofst = 0;
    baso.so.seg  = i;
#else
    if ( argc != 3 ) {
       printf("Enter the board number [0-3]: ");
       fgets(input_string, sizeof(input_string), stdin);
       result = sscanf(input_string, "%x", &cardnum);
       if ( result != 1 )
          cardnum = 0;
       printf("Enter \'P\' to use the primary bus, \'S\' for secondary"
              " bus, \'B\' for both: ");
       fgets(input_string, sizeof(input_string), stdin);
       if ( (input_string[0] == 'P') || (input_string[0] == 'p') )
```

```
            nBus = 0;     // Use the primary bus only.
         else if ( (input_string[0] == 'S') || (input_string[0] == 's') )
            nBus = 1;     // Use the secondary bus only.
         else
            nBus = -1;    // Use both busses.
      }
      else {
         /* Now convert the params */
         sscanf( argv[1], "%x", &cardnum );
         if ( (argv[2][0] == 'P') || (argv[2][0] == 'p') )
            nBus = 0;     // Use the primary bus only.
         else if ( (argv[2][0] == 'S') || (argv[2][0] == 's') )
            nBus = 1;     // Use the secondary bus only.
         else
            nBus = -1;    // Use both busses.
      }
      irq = 2;
      /* Convert to Segment:Offset format.  Could have used FP_SEG macro */
      baso.so.ofst = 0;
      baso.so.seg  = 0;
#endif

      /* Set the data for the init */
      if ( nBus <= 0 ) {
         CurrentBus = 'P';
         ist.bc.control = CHANNEL_SELECT_A;   /* no retries and channel A  */
      }
      else {
         CurrentBus = 'S';
         ist.bc.control = CHANNEL_SELECT_B;   /* no retries and channel B  */
      }
      ist.bc.max_data_buffers = 40; /* set the number of data buffers to 40 */
                                    /* unique rec/xmit data buffers         */
      ist.bc.hi_pri_irupts = 0;     /* no high-priority interrupts needed   */
      ist.bc.max_mc_buffers = 20;   /* 20 mode code data words (not used)   */

      /* Init the memory and the card */
      printf( "Initializing the board!\n" );
      i = ms1553m_init(cardnum, BC_MODE, &ist, (struct mode_code *)NULL,
                       irq, baso.bd_address, INTERRUPT);
      CheckAndReportStatus(i);
      printf("MS1553_INIT() Successful!\n");
      printf("Hit any key to stop.\n" );

      /* Now we are ready to go.  Generate some data, send it to the RT, then
*/
      /*  read it back and compare what we got back with what we sent.
*/
      nMessages = 0;
      ReportError = 0;
      while( !kbhit() ) {  // While the user has not hit any key we will
run...
         // Create some new data.
         for ( i = 0; i < NUMBER_OF_MESSAGES; i++ )
            for ( j = 0; j < WORDS_PER_MESSAGE; j++ )
               rt_rcv[i][j] = (BT_U16BIT)(nMessages + i + j);

SEND_AGAIN:
         scr_rowcol(15, 1);
         /* Switch busses if we are transmitting on both busses... */
         nMessages += 1;
         if ( nBus < 0 ) {
            if ( nMessages & 1 ) {
               CurrentBus = 'P';
               ms1553m_register_write(cardnum, CONTROL_REGISTER,
                       ms1553m_register_read(cardnum, CONTROL_REGISTER)
                                             | CHANNEL_SELECT_A);
            }
            else {
               CurrentBus = 'S';
               ms1553m_register_write(cardnum, CONTROL_REGISTER,
                       ms1553m_register_read(cardnum, CONTROL_REGISTER)
                                             & ~CHANNEL_SELECT_A);
            }
         }
         /* Send the Receive Data messages to the RT, and wait until complete
*/
         ms1553m_bc_tr_grp_data(cardnum, RT_NUM, RBIT, sal,
NUMBER_OF_MESSAGES,
                                &rt_rcv[0][0], bc_status, TRUE, 20);
```

```
       /* If the RT is not responding, print a message and don't try to  */
       /*  get the data back...                                          */
       if ( bc_status[2].error ) {
          if ( bc_status[2].runtime_error & BC_NO_RESPONSE ) {
             nNoResponse++;
             scr_rowcol(20, 1);
             printf("No Response from RT %d, %ld times...",
                    RT_NUM, nNoResponse);
             scr_clr_eol();
             if ( kbhit() )
                break;
             goto SEND_AGAIN;
          }
       }

       // Wait for the RT to copy the data it just received from its
       //  receive buffers into its transmit buffers.
       my_sleep(20);     // Delay for 20 milliseconds

       ms1553m_bc_tr_grp_data(cardnum, RT_NUM, TBIT, sal,
NUMBER_OF_MESSAGES,
                              &rt_xmt[0][0], bc_status, TRUE, 20);
       if ( bc_status[2].error ) {
          nNoResponse++;
          ReportError = 0;
          scr_rowcol(20, 1);
          if ( bc_status[2].runtime_error & BC_NO_RESPONSE )
             printf("No Response from RT %d, %ld times...",
                    RT_NUM, nNoResponse);
          scr_rowcol(21, 1);
          if ( bc_status[2].runtime_error & BC_MESSAGE_ERROR )
             printf("Message Error from RT %d, 1553 Status Word %4.4X...",
                    RT_NUM, bc_status[2].status1);
          scr_clr_eol();
       }
       else {
          // Test the data from the RT against what we sent.
          for ( i = 0; i < NUMBER_OF_MESSAGES; i++ ) {
             for ( j = 0; j < WORDS_PER_MESSAGE; j++ ) {
                if ( rt_xmt[i][j] != rt_rcv[i][j] ) {
                   nDataDifferent++;
                   BadMsg = i;
                   BadWrd = j;
                   ReportError = 1;
                }
             }
             if ( ReportError ) {
                scr_rowcol(16+BadMsg, 1);
                printf("Msgs %5ld Data Errs %4ld No Resp %ld at Msg %d Wrd
%d"
                       " is %4.4X exp %4.4X Bus %c\r",
                      nMessages, nDataDifferent, nNoResponse, BadMsg,
BadWrd,
                      rt_xmt[BadMsg][BadWrd], rt_rcv[BadMsg][BadWrd],
                      CurrentBus);
                ReportError = 0;
             }
          }
          scr_rowcol(15, 1);
          printf("RT %d, Messages %5ld, Errors %4ld, No Resp %ld, SA %d,"
                 "Data[0] = %4.4X\r",
                 RT_NUM, nMessages, nDataDifferent, nNoResponse,
                 sal[0].s_adr, rt_rcv[0][0]);
       }
    }
    ms1553m_bcrtm_exit(cardnum);
    getch();
    scr_rowcol(23, 1);
#ifdef _Windows         /* WINDOWS */
    printf( "\nSelect CLOSE (ALT+F4) to terminate program." );
#else
    printf("\nHit <Enter> to terminate program.\n");
    getch();
#endif
    return(0);
}

/*=========================================================================*
 * ENTRY POINT:                scr_clr
```

```
 *=========================================================================*
 *
 * FUNCTION:    Clear the screen
 *
 * DESCRIPTION:
 *
*=========================================================================*/

void scr_clr(void)
{
#if COMPILER_TYPE == MICROSOFT_C     /* Microsoft C compiler */
    _clearscreen( _GCLEARSCREEN );
#else
    clrscr();
#endif
}

/*=========================================================================*
 * ENTRY POINT:                 scr_clr_eol
 *=========================================================================*
 *
 * FUNCTION:    Clear from the current cursor position to the end of line.
 *
 * DESCRIPTION:
 *
*=========================================================================*/

void scr_clr_eol(void)
{
#if COMPILER_TYPE == MICROSOFT_C     /* Microsoft C compiler */
#else
    clreol();
#endif
}

/*=========================================================================*
 * ENTRY POINT:                 scr_rowcol
 *=========================================================================*
 *
 * FUNCTION:    Position the cursor to the row and column specified.
 *
 * DESCRIPTION:
 *              row     = row number from 1 to 25
 *              column  = column number from 1 to 80
 *

*=========================================================================*/
#ifdef __TURBOC__
#pragma argsused
#endif
void scr_rowcol( int row, int col )
{
#if COMPILER_TYPE == MICROSOFT_C     /* Microsoft C compiler */
    _settextposition( row, col );
#else
    gotoxy( col, row );
#endif
```

RT.C

```
/*=========================================================================
=*
 * FILE:                        R T . C
*=========================================================================
=*
 *
 * COPYRIGHT (C) 1994 - 2000 BY
 *          CONDOR ENGINEERING, INC., SANTA BARBARA, CALIFORNIA
 *          ALL RIGHTS RESERVED.
 *
 *      THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND
 *      COPIED ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH
```

```
 *      THE INCLUSION OF THE ABOVE COPYRIGHT NOTICE.  THIS SOFTWARE OR
ANY
 *      OTHER COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE
 *      AVAILABLE TO ANY OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF THE
 *      SOFTWARE IS HEREBY TRANSFERRED.
 *
 *          THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT
 *          NOTICE AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY CONDOR
 *          ENGINEERING.
 *
*============================================================================
*
 *
 * FUNCTION:    Demonstrate RT functionality.
 *
 * ENTRY POINTS:
 *
 * DESCRIPTION:
 *      This routine will uses the link routine for RT 1 at subaddresses
 *      20, 21, 22 and 23 and will receive and transmit data from the
same
 *      subaddresses.
 *
 *   This program is intended to be used with BCxx.EXE, and demonstrates
 *   setting up one MS1553-xxx board as a Remote Terminal and another
board
 *   as a Bus Controller.
 *
 *      To demonstrate the program first start RTxx.EXE at the RT board
and
 *      then BCxx.EXE on the Bus Controller board.
 *
 *   If the two host machines are not the same speed, than the faster
 *   machine should be running the RT simulation.
 *
 *      When running this program as a "Windows" program, remember that
the
 *      "MS1553.DLL" or "MST1553.DLL" file must be accessible by WINDOWS
 *      (e.g., in default directory, \WINDOWS, or PATH).
*============================================================================
*/

/*----------------------------------------------------------------------------
-*
 *                       INCLUDES, DEFINES and GLOBALS
 *----------------------------------------------------------------------------
*/

/* $Revision:  4.00 Release $ */
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <dos.h>

#include "ms1553.h"

#define RT_NUM               1    /* The RT number
*/

#define WORDS_PER_MESSAGE    2    /* Words in each message */
#define NUMBER_OF_MESSAGES   4    /* Number of messages to send/receive
*/

/* This contains the received data, which is then transmitted */
BT_U16BIT sd[NUMBER_OF_MESSAGES][WORDS_PER_MESSAGE];
INIT_INFO   ist;
MODE_CODE   mcst;
STAT_INFO   rcv_x_err[NUMBER_OF_MESSAGES], xmt_x_err[NUMBER_OF_MESSAGES];

/* The subaddress/count list */
struct sa_list sal[NUMBER_OF_MESSAGES] = { {20, WORDS_PER_MESSAGE},
                                           {21, WORDS_PER_MESSAGE},
                                           {22, WORDS_PER_MESSAGE},
                                           {23, WORDS_PER_MESSAGE} };
```

```
BT_UINT cardnum = 0;    // Board number

/*****************************************************************
*
*  static void CheckAndReportStatus(UINT status);
*
*  Check the return code and report any errors.
*
*****************************************************************/

static void CheckAndReportStatus(BT_UINT status)
{
   if ( status != 0 )
   {
      if ( status & INIT_ILLEGAL_IRUPT_NUMBER )
         printf("   IRQ is < 2 or > 15.\r\n");
      if ( status & INIT_MEMORY_FULL )
         printf("   Specified number of data/command "
                "blocks exceed the available board memory.\r\n");
      if ( status & INIT_MEMORY_SIZE_ERROR )
         printf("   Memory size specified by jumpers does "
                "not respond or does not exist.\r\n");
      if ( status & INIT_MEMORY_ALLOC_ERROR )
         printf("   MALLOC of memory failed.\r\n");
      if ( status & INIT_NOT_RESPONDING )
         printf("   Board/chip is not responding, check board"
                " address.\r\n");
      if ( status & INIT_ILLEGAL_MODE )
         printf("   Mode is not RT_MODE, BC_MODE, or M_MODE.\r\n");
      if ( status & INIT_NOT_BCRTM )
         printf("   Chip does not support monitor mode.\r\n");
      if ( status & INIT_CANT_MAP_ADDR )
         printf("   Cannot map physical board address into "
                "protected address space.\r\n");
      printf("MS1553_INIT() returned error code %X (hex) as decoded"
             " above.\r\n", status);
      ms1553m_bcrtm_exit(cardnum);
      exit(1);
   }
}

/*=========================================================================
*
 * ENTRY POINT:                M A I N

*=========================================================================*
 *
 * FUNCTION:    Main control
 *
 * DESCRIPTION:
 *
*=========================================================================
*/

int main(int argc, char **argv )
{
    BT_U16BIT   irq;
    BT_UINT     nError;
    BT_U16BIT   first_word;
    #if !defined(__WIN32__)
    int      i;
    #endif
    char     input_string[50];     // User input
    int      result;
    long     nErrors;
    long     nMessages;
    union {
        struct {
            BT_U16BIT  ofst;
            BT_U16BIT  seg;
        } so;
        BT_U16BIT  *bd_address;
    } baso;
```

```
    printf("    ------------------------------------------------------------
-----\n");
    printf("             RT.C: MS1553 Remote Terminal Example
Program.\n");
    printf("    Version 4.20  Copyright (c) Condor Engineering, Inc.
1998-2000\n");
    printf("    ------------------------------------------------------------
-----\n\n");

#ifndef __WIN32__
    /* If the parameters are there, convert them, else ask */
    if ( argc != 3 ) {
       printf("Enter the 4-digit HEX segment address\n");
       printf(" of the MS1553 Board (ex: D000): ");
       fgets(input_string, sizeof(input_string), stdin);
       result = sscanf(input_string, "%x", &i);
       if ( result != 1 )
          i = 0xD000;
       printf("Enter the 1- or 2-digit DECIMAL interrupt number\n");
       printf(" of the MS1553 Board (ex: 5): ");
       fgets(input_string, sizeof(input_string), stdin);
       result = sscanf(input_string, "%i", &irq);
       if ( result != 1 )
          irq = 5;
    }
    else {
       /* Now convert the params */
       sscanf( argv[1], "%x", &i );
       sscanf( argv[2], "%i", &irq );
    }
    /* Convert to segoff format */
    baso.so.ofst = 0;
    baso.so.seg = i;

#else
    if ( argc != 3 ) {
       printf("Enter the board number [0-3]: ");
       fgets(input_string, sizeof(input_string), stdin);
       result = sscanf(input_string, "%x", &cardnum);
       if ( result != 1 )
          cardnum = 0;
    }
    else {
       /* Now convert the params */
       sscanf( argv[1], "%x", &cardnum );
    }
    irq = 2;
    /* Convert to Segment:Offset format.  Could have used FP_SEG macro */
    baso.so.ofst = 0;
    baso.so.seg  = 0;
#endif

    /* set the data for the init */
    ist.rt.rt_address = RT_NUM;
    ist.rt.receive_illegal_broadcast = 0xffffffffL;
    ist.rt.receive_sub_addr = 0xff87ffffL;
    ist.rt.transmit_sub_addr = 0xff87ffffL;
    ist.rt.control = CHANNEL_A_ENABLE | CHANNEL_B_ENABLE;
    ist.rt.hi_pri_irupts = 0;

    /* Number of messages per subaddress */
    ist.rt.rnum_messages[19] = 10;        // SA 20
    ist.rt.rnum_messages[20] = 10;        // SA 21
    ist.rt.rnum_messages[21] = 10;        // SA 22
    ist.rt.rnum_messages[22] = 10;        // SA 23

    ist.rt.tnum_messages[19] = 1;         // SA 20
    ist.rt.tnum_messages[20] = 1;         // SA 21
    ist.rt.tnum_messages[21] = 1;         // SA 22
    ist.rt.tnum_messages[22] = 1;         // SA 23

    /* msg length.  Ignored since we are using the link function. */
    ist.rt.rmsg_length[19] = 32;        // SA 20
    ist.rt.rmsg_length[20] = 32;        // SA 21
    ist.rt.rmsg_length[21] = 32;        // SA 22
```

```
    ist.rt.rmsg_length[22] = 32;        // SA 23
    ist.rt.tmsg_length[19] = 32;        // SA 20
    ist.rt.tmsg_length[20] = 32;        // SA 21
    ist.rt.tmsg_length[21] = 32;        // SA 22
    ist.rt.tmsg_length[23] = 32;        // SA 23

    /* Now set up the mode codes */
    /* no interrupts on reception of mode code (without data word) */
    mcst.int_on_reception = 0;
    /* All mode codes are Illegal.... */
    mcst.illegalize_broadcast_no_word = 0xffff;
    mcst.illegalize_mode_code_no_word = 0xffff;
    mcst.illegalize_broadcast_with_word = 0xffff;
    mcst.illegalize_transmit_with_word = 0xffff;
    mcst.illegalize_receive_with_word = 0xffff;
    /* no interrupts on reception of mode code (with data word) */
    mcst.int_when_mode_code_transacted = 0;

    /* Init the memory and the card */
    clrscr();
    printf( "Initializing the board\n" );
    printf( "Hit any key to stop.\n" );
    nError = ms1553m_init(cardnum, RT_MODE, &ist, &mcst, irq,
                          baso.bd_address, INTERRUPT);
    CheckAndReportStatus(nError);

    /* Setup link routines which link the four RECEIVE subaddresses
*/
    ms1553m_rt_tr_link(cardnum, RBIT, &sal[0], NUMBER_OF_MESSAGES,
                       &sd[0][0], &rcv_x_err[0]);

    /* Setup link routines which link the four TRANSMIT subaddresses
*/
    ms1553m_rt_tr_link(cardnum, TBIT, &sal[0], NUMBER_OF_MESSAGES,
                       &sd[0][0], &xmt_x_err[0]);

    /* Now we are ready to go */
    nErrors = 0;
    first_word = 0;
    nMessages = 0;
    printf("\n\n\n\n\n\n\n\n\n\n\n\n");
    while( !kbhit() ) {
        gotoxy(1, 14);
        printf ("SA 20 W0 %4.4X msgs %ld ",sd[0][0], nMessages);

        rcv_x_err[0].completed = FALSE;
        xmt_x_err[0].completed = FALSE;

        // We only need to wait for any one of the receive messages.
        while (!(rcv_x_err[0].completed) ) {
           if ( kbhit() )
               break;
        }
        nMessages++;
        // Update the TRANSMIT link with the data we just RECEIVED.
        // This is necessary because the link functions move the new data
        // to the transmit buffers AFTER the previous data has been read.
        // This will cause the TRANSMIT data to lag by one message update
        // the data which has been received.
        ms1553m_rt_t_update(cardnum, &sal[0], NUMBER_OF_MESSAGES,
                            &sd[0][0]);
        if ( (nMessages > 1) && (first_word == sd[0][0]) )
        {
           if ( nErrors < 10 )
              gotoxy(1,(int)(15+nErrors));
           else
              gotoxy(1,24);
           printf("Message %ld, first word = %4.4X same as previous"
                  " message", nMessages, first_word);
           nErrors++;
        }
        first_word = sd[0][0];
    }
    ms1553m_bcrtm_exit(cardnum);
    getch();
```

```
    gotoxy(1,24);
#ifdef _Windows         /* WINDOWS */
    printf( "\nSelect CLOSE (ALT+F4) to terminate program." );
#else
    printf("\nHit <Enter> to terminate program.");
    getch();
#endif
    return(0);
}
```

MON.C

```
/*=========================================================================
=*
 * FILE:                          M O N . C
*=========================================================================
=*
 *
 * COPYRIGHT (C) 1994 - 2000 BY
 *          CONDOR ENGINEERING, INC., SANTA BARBARA, CALIFORNIA
 *          ALL RIGHTS RESERVED.
 *
 *      THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND
 *      COPIED ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH
 *      THE INCLUSION OF THE ABOVE COPYRIGHT NOTICE.  THIS SOFTWARE OR
ANY
 *      OTHER COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE
 *      AVAILABLE TO ANY OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF THE
 *      SOFTWARE IS HEREBY TRANSFERRED.
 *
 *          THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT
 *          NOTICE AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY CONDOR
 *          ENGINEERING.
 *
*=========================================================================
*
 *
 * FUNCTION:    Monitor bus activity on channels A and B.
 *
 * DESCRIPTION:
 *    This program will monitor bus activity on channels A and B.
 *    Each message type is displayed on a dedicated area of the screen.
 *

*=========================================================================
*/

/*-------------------------------------------------------------------------
*
 *                      INCLUDES, DEFINES and GLOBALS
 *-------------------------------------------------------------------------
*/

/* $Revision:  4.20 Release $
   Date        Revision
  ----------   ------------------------------------------------------------
--
  08/26/2000   Updated to display more of the 1553 data words, and to use
               the new simplified ms1553m_m_get_msg() function.V4.20.ajh
 */
#include <stdio.h>
#include <dos.h>
#include <stdlib.h>
#include <conio.h>
#include <string.h>

#include "ms1553.h"             /* must include this header file */

/* Set the compiler type */

#define MICROSOFT_C  0    /* Internal identification of Microsoft "C" */
#define TURBO_C      1    /* Internal identification of Borland   "C" */
```

```
#ifdef __TURBOC__
#define COMPILER_TYPE   TURBO_C
#else
#define COMPILER_TYPE   MICROSOFT_C
#endif

#if COMPILER_TYPE == MICROSOFT_C     /* Microsoft C compiler */
#include <graph.h>
#endif

/*  These are the constants for defining the message types. */

#define MF_CMD_BC_RT            0
#define MF_CMD_RT_BC            1
#define MF_CMD_RT_RT            2
#define MF_CMD_BROADCAST        3
#define MF_CMD_MODE_CODE        4
#define MF_CMD_BROADCAST_RT_RT  5
#define MF_CMD_BROADCAST_MC     6

char *message_types[] = {   /* Name the message types */
    "BC->RT",
    "RT->BC",
    "RT->RT",
    "Broadcast",
    "Mode Code",
    "Broadcast RT->RT",
    "Broadcast Mode Code"
};

/* Monitor mode data structures */
INIT_INFO        mon_init;    /* Setup this structure to init the board
*/
MON_COMMAND_DATA cmd_info;    /* Local copy of the current BM message
*/

/*  These are the internal routines. */

static void  dsp_data(int row, int col, long msg_num);
static void  scr_clr(void);
static void  scr_clr_eol(void);
static void  scr_rowcol( int row, int col );

/*=========================================================================
=*
 * ENTRY POINT:                S I G N O N

*=========================================================================
*
 *
 * FUNCTION:    Display the signon banner
 *
 * DESCRIPTION:
 *

*=========================================================================
*/
static void signon(void)
{
    /* Clear the screen */
    scr_clr();

    scr_rowcol(1,10);
    printf("                MS1553 Bus Monitor Demo Program\r\n");
    scr_rowcol(2,13);
    printf("Version 4.20  Copyright (c) Condor Engineering, 1997-
2000\n\n");
}

/*************************************************************
*
*  static void CheckAndReportStatus(UINT status);
*
*  Check the return code and report any errors.
*
```

```
*****************************************************************/

static void CheckAndReportStatus(BT_UINT cardnum, BT_UINT status)
{
   if ( status != 0 )
   {
      if ( status & INIT_ILLEGAL_IRUPT_NUMBER )
         printf("   IRQ is < 2 or > 15.\r\n");
      if ( status & INIT_MEMORY_FULL )
         printf("   Specified number of data/command "
                "blocks exceed the available board memory.\r\n");
      if ( status & INIT_MEMORY_SIZE_ERROR )
         printf("   Memory size specified by jumpers does "
                "not respond or does not exist.\r\n");
      if ( status & INIT_MEMORY_ALLOC_ERROR )
         printf("   MALLOC of memory failed.\r\n");
      if ( status & INIT_NOT_RESPONDING )
         printf("   Board/chip is not responding, check board"
                " address.\r\n");
      if ( status & INIT_ILLEGAL_MODE )
         printf("   Mode is not RT_MODE, BC_MODE, or M_MODE.\r\n");
      if ( status & INIT_NOT_BCRTM )
         printf("   Chip does not support monitor mode.\r\n");
      if ( status & INIT_CANT_MAP_ADDR )
         printf("   Cannot map physical board address into "
                "protected address space.\r\n");
      printf("MS1553_INIT() returned error code %X (hex) as decoded"
             " above.\r\n", status);
      ms1553m_bcrtm_exit(cardnum);
      exit(1);
   }
}

/*=========================================================================
=*
 * ENTRY POINT:                M A I N

*=========================================================================
*
 *
 * FUNCTION:    Main control routine
 *

*=========================================================================*
/

int main( int argc, char **argv )
{
    BT_U16BIT  irq, i;
    BT_UINT    cardnum = 0;          // The number of the installed board
    long       msg_num = 0;
    char       szMsg[100];
    int        status;

    /* Set structure for board segment */
    union
    {
        BT_U16BIT *ptr;
        struct
        {
            BT_U16BIT  offset;
            BT_U16BIT  seg;
        } so;
    } bd;

    signon();      /* Display the signon banner */

    /* If the parameters are there, convert them, else ask */
#ifndef __WIN32__
    if ( argc != 3 )
    {
       printf("Enter the 4-digit HEX segment address\n");
       printf(" of the MS1553 Board (Default: D000): ");
       gets(szMsg);
       status = sscanf(szMsg, "%X", &i);
```

```
        if ( status != 1 ) i = 0xD000;
        printf("Enter the 1- or 2-digit DECIMAL interrupt number\n");
        printf(" of the MS1553 Board (Default: 5): ");
        gets(szMsg);
        status = sscanf(szMsg, "%i", &irq);
        if ( status != 1 ) irq = 5;
    }
    else
    {
        /* Just convert the command line params */
        sscanf( argv[1], "%x", &i );
        sscanf( argv[2], "%i", &irq );
    }
    /* Convert to segment:offset format */
    bd.so.offset = 0;
    bd.so.seg = i;
#else
    if ( argc != 2 )
    {
        printf("Enter the board number\n");
        printf(" of the MS1553 Board (Default: 0): ");
        gets(szMsg);
        status = sscanf(szMsg, "%d", &cardnum);
        if ( status != 1 ) cardnum = 0;
    }
    else
    {
      /* Just convert the command line params */
       sscanf( argv[1], "%x", &cardnum );
    }
    bd.so.offset = 0;
    bd.so.seg = 0;
    irq = 0;
#endif
    signon();      /* Re-Display the signon banner */
#ifdef _Windows         /* WINDOWS */
    scr_rowcol(3,10);
    printf("CONTROL-BREAK to stop, select CLOSE (ALT-F4) to exit." );
#else
    scr_rowcol(3,27);
    printf("Hit any key to exit." );
#endif
    scr_rowcol(5,20);
    printf("Bus Monitor Message Display Screen");

    /* Setup monitor mode structure */
    mon_init.monitor.control     = ( CHANNEL_SELECT_A | CHANNEL_SELECT_B
);
    /* Select terminals to monitor  */
    /* Note that if only one RT of an RT to RT message is monitored,
*/
    /* BCRTM will incorrectly record only a RT to BC or BC to RT
message!*/
    mon_init.monitor.mon_control = MONITOR_SELECT_TERMINALS;
    mon_init.monitor.mon_control = MONITOR_ALL_TERMINALS;

    /* Initialize board for monitor mode */
    i = ms1553m_init(cardnum, M_MODE, &mon_init, NULL, irq, bd.ptr,
                     INTERRUPT);
    CheckAndReportStatus(cardnum, i);

    ms1553m_m_select_rt(cardnum, 0x07L); /* Selected RT addresses 0,1,2
*/
    ms1553m_m_start(cardnum);            /* Start cardnum monitor mode
*/

    while( !kbhit() )               /* Stay in loop until a key is hit
*/
    {
        /* Get Bus Monitor information */
        if (ms1553m_m_get_msg(cardnum, &cmd_info) == 0)
        {
            /* Now display the (copied) monitored information */
            switch (cmd_info.mci_cmd)
            {
```

```
                    case MF_CMD_BC_RT:
                        dsp_data( 7, 1, msg_num++);
                        break;

                    case MF_CMD_RT_BC:
                        dsp_data(10, 1, msg_num++);
                        break;

                    case MF_CMD_RT_RT:
                        dsp_data(13, 1, msg_num++);
                        break;

                    case MF_CMD_BROADCAST:
                        dsp_data(16, 1,msg_num++);
                        break;

                    case MF_CMD_BROADCAST_RT_RT:
                        dsp_data(19, 1, msg_num++);
                        break;

                    case MF_CMD_MODE_CODE:
                        dsp_data(22, 1, msg_num++);
                        break;

                    case MF_CMD_BROADCAST_MC:
                        dsp_data(22, 30, msg_num++);
                        break;

                    default:
                        /* Cannot happen, display garbage! */
                        dsp_data(20, 10, msg_num++);
                        break;
                }
            }
        }

        /* Stop monitor function */
        ms1553m_m_stop(cardnum);

        scr_rowcol(25,1);                   /* set the row/column position */

        ms1553m_bcrtm_exit(cardnum);        /* exit from the bcrtm chip */
        return 0;                           /* exit from the program */
    }

    /*=========================================================================
    =*
     * ENTRY POINT:                D S P _ D A T A

    *=========================================================================
    *
     *
     * FUNCTION:    Display the data collected
     *
     * DESCRIPTION:
     *

    *=========================================================================*
    /

    void dsp_data(int row, int col, long msg_num)
    {
       int   i, j;

       scr_rowcol(row, col);           /* set the row/column position */
       printf("-%s: ", message_types[cmd_info.mci_cmd]);
       printf("RT %d SA %d WC %d", cmd_info.mci_src_rt,
                                   cmd_info.mci_src_sa,
                                   cmd_info.mci_wc_mc);
       if ( (cmd_info.mci_cmd == MF_CMD_RT_RT) ||
            (cmd_info.mci_cmd == MF_CMD_BROADCAST_RT_RT) )
          printf(" To: RT %d SA %d", cmd_info.mci_dest_rt,
                                     cmd_info.mci_dest_sa);
       printf(" Message number %ld", msg_num);
       if ( cmd_info.mci_error )
```

```
      printf(" Error/No Response");
   scr_clr_eol();                  /* clear old information from line */

   scr_rowcol(row+1, col);         /* set the row/column position */
   if ( cmd_info.mci_cmd == MF_CMD_MODE_CODE )
      printf("data: %04x", cmd_info.data_buf[0]);
   else
   {
      printf("data: ");
      j = cmd_info.mci_wc_mc;
      if ( j > 14 ) j = 14;
      for ( i = 0; i < j; i++ )
         printf("%04x ", cmd_info.data_buf[i]);
      if ( j != cmd_info.mci_wc_mc )
         printf("...");
   }
   scr_clr_eol();                  /* clear old information from line */
}

/*=========================================================================
*
 * ENTRY POINT:                 scr_clr

*=========================================================================*
 *
 * FUNCTION:    Clear the screen
 *
 * DESCRIPTION:
 *

*=========================================================================*
/

void scr_clr(void)
{
#if COMPILER_TYPE == MICROSOFT_C     /* Microsoft C compiler */
    _clearscreen( _GCLEARSCREEN );
#else
    clrscr();
#endif
}

/*=========================================================================
*
 * ENTRY POINT:                 scr_clr_eol

*=========================================================================*
 *
 * FUNCTION:    Clear from the current cursor position to the end of
line.
 *
 * DESCRIPTION:
 *

*=========================================================================*
/

void scr_clr_eol(void)
{
#if COMPILER_TYPE == MICROSOFT_C     /* Microsoft C compiler */
#else
    clreol();
#endif
}

/*=========================================================================
*
 * ENTRY POINT:                 scr_rowcol

*=========================================================================*
 *
 * FUNCTION:    Position the cursor to the row and column specified.
 *
 * DESCRIPTION:
 *              row     = row number from 1 to 25
```

```
 *              column  = column number from 1 to 80
 *

*=========================================================================*
/
#ifdef __TURBOC__
#pragma argsused
#endif
void scr_rowcol( int row, int col )
{
#if COMPILER_TYPE == MICROSOFT_C     /* Microsoft C compiler */
    _settextposition( row, col );
#else
    gotoxy( col, row );
#endif
}
```

APPENDIX B

VMECFG Source File

Source Code

This appendix presents the source code for the vmecfg file. Basic test routines for VME board level testing of VME memory locations for 1553 boards start at 0x7D00 0000 and end at 0x7DFF FFFF. SW4 on the board sets A23 through A16. These switches allow the board to map the 0x0000 through 0xFFFF memory area available on board to any even 64KW boundary location.

Memory may not be mapped in all of 64KW area. Main CPU onboard RAM starts at 0x0000 0000 and is at least 1 Mg 0x000FFFFF. This setups a RT10 and waits for a message packet of 32 words on subaddress 1.

```
/*************************************************************************
  Possession of this source file requires a LICENSE AGREEMENT.
  Copyright (c) 1990-1995
  Condor Engineering Incorporated, Santa Barbara, California USA
                      ALL RIGHTS RESERVED

*************************************************************************/
/* $Revision$ */

#include <stdio.h>
#include <strings.h>

#include "ms1553.h"

#define UPPER_MEMORY_SA0x2440                /* upper memory starting location in words */
#define MEM_SIZE_8KW          0x2000             /* 8KW ram size words */
#define MEM_SIZE_16KW         0x4000             /* 16KW ram size words */
#define MEM_SIZE_32KW         0x8000             /* 32KW ram size words */
#define UTMCR          0x2000         /* chip registers */
#define BD_SWITCH      0x2400                /* PC interface, BCRTM, MEMORY switch */

/* #define getchar()    getc(stdin) */

unsigned short memsize, bcrtm_chip;
unsigned short *memptr, *dataptr, memory[2048];
```

```
unsigned short *base, *bcrtm, *sa;
unsigned short pass_flag, pat, count;
unsigned long pass_count = 0;
unsigned long fail_count = 0;
char temp[132];

/*  for the ctrlr test we will make it easier by creating a 16KB memory array and will
    string move it to and from the board. This can later be dynamic.
*/
unsigned short    ctrlr_buf[MEM_SIZE_8KW];
unsigned short    *cbuf;

struct RTDESC {
    unsigned short ctrl;
    unsigned short msg_ptr;
    unsigned short data_ptr;
    unsigned short reserved;
};

struct RTDESC *rd;
struct RTDESC *td;
struct RTDESC *mcd;

struct {
    unsigned short cmd;
    unsigned short word1;
}cmds[3000];

union {
    struct {
        unsigned long segment;
        unsigned long offset;
    }so;
    unsigned short *ptr;
}bd_address;

/*********************************************************************/

main( argc, argv)
unsigned short argc;
unsigned short **argv;
{
    /* memory test */
    int result, i;
    unsigned short j, l, k, *msg, *dat;
    unsigned short irq, seg, irqs;

    bd_address.so.offset = 0x0000;            /* set the base offset address */

    if (argc != 3) {
        do {
        puts( "\nWhat is the segment address selected for your board?\n" );
        puts( "   A.  7d7f0000" );
```

```
    puts( "   B.  7d7f8000" );
    puts( "   C.  7d800000" );
    puts( "   D.  7d808000" );
    puts( "   E.  7d810000" );
    puts( "   F.  7d818000" );
    puts( "\nEnter the letter of your choice? " );

        /* get the answer */
        seg = tolower( getchar() );
    } while( (seg < 'a') || (seg > 'f') );

    /* set the registers */
    bd_address.so.segment = 0x7d7f0000 + ((seg - 'a') * 0x8000);
    getchar();                                  /* dummy read to clear the buffer */

    do {
    puts( "\nWhat is the IRQ line selected? \n" );
    puts( "   A.  0" );
    puts( "   B.  1" );
    puts( "   C.  2" );
    puts( "   D.  3" );
    puts( "   E.  4" );
    puts( "   F.  5" );
    puts( "   G.  6" );
    puts( "   H.  7" );
    puts( "Enter the letter of your choice? " );

    /* get the IRQ */
    irq = tolower( getchar() );
    } while( (irq < 'a') || (irq > 'h') );

    /* set the IRQ variable */
    irq = (irq - 'a');
}
else {
    sscanf(argv[1], "%x", &bd_address.so.segment); /* get the address */
    sscanf(argv[2], "%x", &irqs );                 /* get the IRQ value */
    irq = irqs;
}

/* setup the 1553 chip base address */
base = bd_address.ptr;                             /* set the base address */
printf("Selected segment = %04X, IRQ = %1X\n",base,irq);

i = ms1553m_config_sw(0);                          /* get the switch configuration */
puts( "\nThe configuration switches are set as follows:" );

j=0;
switch( CONFIG_MEMSIZE_MASK & i )
{
case CONFIG_MEMSIZE_8K :
        j=MEM_SIZE_8KW;
        puts( "    Memory size = 8KW");
```

```
        break;
case CONFIG_MEMSIZE_16K :
        j=MEM_SIZE_16KW;
        puts( "    Memory size = 16KW");
        break;
case CONFIG_MEMSIZE_32K :
        j=MEM_SIZE_32KW;
        puts( "    Memory size = 32KW");
        break;
}

if ( (CONFIG_CHIP_MASK & i) == CONFIG_BCRTM )
    puts("    Chip type   = BCRTM" );
else
    puts("    Check switch settings: SW2-4 must be OFF" );

if ( (CONFIG_SABTECH_MASK & i) == CONFIG_SABTECH_AVAIL)
    puts("    Check switch settings: SW2-1 must be ON\n");
else
    puts("    PC Interface\n");

puts( ">>> Running the memory diagnostic ***" );
diag_memory_test( j );                              /* performing the memory test */
if ( fail_count ) {
    puts( "***** The memory does not pass, check your configuration" );
    exit(2);
}

puts( "\n>>> Running the controller test ***" );
diag_ctrlr_test();                                  /* perform the controller test */
if ( fail_count ) {
    puts( "***** The controller does not pass, check your configuration" );
    exit(2);
}

puts("setting the vme1553 IRQ");
set_irq(irq);                                       /* setting the IRQ */

puts("setting up the 1553 chip registers");
set_rt_reg();                                       /* setup 1553 Chip registers */

puts("setting the descriptor space");
puts("RT 10 xmit subaddress 1, Receive subaddress 1");
set_rt_desc_ptr();                                  /* setup the descriptors space */

st_rec_msg();                                        /* now receive a message */
puts("receive data");
for ( i = 0; i < 32; i++)
    printf("%04x ",memptr[i]);

for (i = 0; i < 50000; I++)
    ;
```

```
    bcrtm[0] = 0;                                       /* clear the control register */
    puts("**** Tests complete all operations passed ****");
    free(memory);
}

/*********************************************************************/

int ms1553m_config_sw(unsigned short board)
{
        unsigned short switches;
        int i;

    switches = (unsigned short) base[BD_SWITCH];  /* get the switch value */
    switch( switches & 6 )                        /* set the memory size first */
    {
    case 6 :
                /* switch 6 & 7 off  64kb_window and 64kb_ram set */
                i = CONFIG_MEMSIZE_32K;
                memsize = MEM_SIZE_32KW;
                break;
        case 2 :
                /* switch 6 on 7 off 32kb_window and 64kb_ram */
                i = CONFIG_MEMSIZE_16K;
                memsize = MEM_SIZE_16KW;
                break;
        default :
                /* switch 6 & 7 on  32kb_window and 16kb ram */
                i = CONFIG_MEMSIZE_8K;
                memsize = MEM_SIZE_8KW;
                break;
    }

    if ( switches & 1 ) {                                       /* now the chip type */
        i |= CONFIG_BCRTM;
            bcrtm_chip = TRUE;
    }
    else {
            i &= ~CONFIG_BCRTM;
            bcrtm_chip = FALSE;
    }

    /* and finally the sabtech info */
    i = ( switches & 8 ) ? i | CONFIG_SABTECH_AVAIL : i | CONFIG_PC_ONLY;
    return( i );
}

/*********************************************************************/

set_rt_reg()
{
    bcrtm = (unsigned short *) &base[UTMCR];      /* setup the 1553 chip */
    bcrtm[0]  = 0;                                /* clear out the control reg */
    bcrtm[12] = 0;                                        /* reset the chip */
```

```
    bcrtm[10] = 0x2a;                           /* set it to RT 10 */
    bcrtm[2]  = 0;                                      /* set descriptor adr to zero */
    bcrtm[7]  = 0;                                      /* now clear the interrupts */
    bcrtm[8]  = 0;
}

/*********************************************************************/

set_rt_desc_ptr()
{
    unsigned short i;
    puts("filling the descriptor space");
    for ( i = 0; i < 320; i++)
        base[i] = 0xffff;

    /* setup the pointers to the descriptor arrays */
    rd = (struct RTDESC *) base;                        /* set receive start address */
    td = rd+32;                                         /* set transmit start address */
    mcd = td+32;                                        /* set mode code start address */

    /* setup subaddress stuff for receive on 1 */
    (rd + (1 - 1) )->ctrl = 0;                          /* clear the status word */
    (rd + (1 - 1) )->msg_ptr = 0x800;                   /* set the message pointer */
    (rd + (1 - 1) )->data_ptr = 0x1000;                 /* set the data pointer */

    /* setup subaddress stuff for transmit on 1 */
    (td + (1 - 1) )->ctrl = 0;                          /* clear the status word */
    (td + (1 - 1) )->msg_ptr = 0x0810;                  /* set the message pointer */
    (td + (1 - 1) )->data_ptr = 0x1000;                 /* set the data pointer */
}

/*********************************************************************/

set_irq(irq)
unsigned short irq;
{
    base[(BD_SWITCH + 1)] = (irq << 8); /* display the IRQ value */
}

/*********************************************************************/

st_rec_msg()                                                    /* now receive a message */
{
    unsigned short k, cmd;
    unsigned short i = 0;
    bcrtm[0] = 0x181;                                   /* now start the RT */

    puts("receiving messages");
    k = bcrtm[1];
    while( (k & 0x4000) == 0 )                          /* wait for a command */
        k = bcrtm[1];

    cmds[i].cmd = bcrtm[5];                             /* save the command */
```

```
    /* see which SA */
    cmd = (unsigned short)(rd + (((cmds[i].cmd >> 5) & 0x1f)-1) );
    dataptr = &base[base[cmd + 2]];/* set the data pointer */

    k = bcrtm[1];
    while( (k & 0x4000) != 0 )                    /* wait for a command to complete */
        k = bcrtm[1];

    memptr = &memory[0];
    memwcpy( memptr, dataptr, 32);            /* move the data */
    cmds[0].word1 = *dataptr;                     /* move first word to cmds structure */
}

/*********************************************************************/
diag_memory_test( kw )
unsigned short kw;
{
    pass_count = fail_count = 0;                  /* set the count values */
    sa = base;                                    /* set the base address */
    count = MEM_SIZE_8KW;                         /* set the count to 8KW size */
    pat = 0xff00;                                 /* set the pattern to no xor */
    pass_flag = TRUE;                             /* clear the pass flag */
    do_mem_test();                                /* perform the memory test */

    pat = 0xff;                                   /* complement the test pattern */
    do_mem_test();                                /* perform the memory test */
    if ( pass_flag )                              /* increment the count */
            pass_count++;
    else
            fail_count++;

    sprintf(temp,"Lower Memory test----> pass = %8ld   fail = %8ld",
                pass_count, fail_count);
    puts( temp );                                 /* print the error message */

    /* only do the upper part if there is memory there */
    if ( kw > MEM_SIZE_8KW )  {               /* over 8KW memory ? */
                /* set the UPPER address */
            sa = (unsigned short *) &base[UPPER_MEMORY_SA * 2];

            if ( kw = MEM_SIZE_16KW )          /* is memory over 8KW ? */
                    count = MEM_SIZE_16KW - UPPER_MEMORY_SA;
            else
                    count = MEM_SIZE_32KW - UPPER_MEMORY_SA;

            pat = 0xff00;                         /* set the pattern to no xor */
            pass_flag = TRUE;                     /* clear the pass flag */
            do_mem_test();                        /* perform the memory test */

            pat = 0xff;                           /* complement test pattern */
            do_mem_test();                        /* perform the memory test */
            if ( pass_flag )                      /* increment the count */
```

```
                pass_count++;
            else
                fail_count++;

            sprintf(temp,"Upper Memory test----> pass = %8ld   fail = %8ld",
                        pass_count, fail_count);
            puts( temp );                           /* print the error message */
    }
}

/*****************************************************************/

do_mem_test()
{
    unsigned short x, i;
    x = 0x1234;                                         /* set test pattern */
    for( i = 0; i < count; i++ ) {
        sa[i] = (x ^ pat);                          /* write the patterns */
        x += 0x3333;                                /* increment the pattern */
    }
    x = 0x1234;                                         /* set test pattern */

    for( i = 0; i < count ; i++ ) {                 /* set counter */
        if ( sa[i] != (x ^ pat) ) {                 /* read the data */
                pass_flag = FALSE ;                 /* set the pass flag to fail */
                sprintf(temp,"Memory Error---> adr = %04X exp = %04X ret = %04X",
                            sa[i],(x^pat),sa[i]);
                puts( temp );                       /* print the error message */
            }
            x += 0x3333;                            /* increment the pattern */
    }
}

/*****************************************************************/

diag_ctrlr_test()
{
    pass_count = fail_count = 0;                    /* set the count values */
    sa = base;                                      /* set the base address value */
    count = MEM_SIZE_8KW;                           /* set count to 8K words */

    cbuf = ctrlr_buf;                               /* set ptr for scratch buffer */

    puts( "Performing string read/write to/from dual port RAM" );
    pat = 0xff00;                                   /* set the pattern to no xor */
    pass_flag = TRUE;                               /* clear the pass flag */
    do_ctrlr_test();                                /* do the memory test */

    pat = 0xff;                                     /* now the complement */
    do_ctrlr_test();                                /* perform the controller test */

    puts( "Performing memory-memory string move within dual port RAM" );
    memwcpy( sa, &sa[1], count-1 );           /* shift everything down 1 wrd */
```

```
    memwcpy( &cbuf[1], sa, count-1 );          /* move to array for testing */

    chk_ctrlr_data();                              /* compare the data are ok */
    if ( pass_flag )                               /* increment the count */
            pass_count++;
    else
            fail_count++;

    sprintf(temp,"Memory controller Test----> pass = %8ld fail = %8ld",
                pass_count, fail_count);
    puts( temp );                                  /* print the error message */
}

/*********************************************************************/
do_ctrlr_test()
{
    unsigned short x, i;
    x = 0x1234;                                            /* set the pattern */
    for( i = 0; i < count; i++ )  {
            cbuf[i] = (x ^ pat);                   /* write the patterns */
            x += 0x3333;                           /* increment the pattern */
    }
    memwcpy( sa, cbuf, count );                            /* now send them to the board */
    memwcpy( cbuf, sa, count );                            /* retreive it from the board */
    chk_ctrlr_data();                                      /* check the data */
}

/*********************************************************************/
chk_ctrlr_data()
{
    unsigned short x, i;

    x = 0x1234;                                            /* set the pattern */
    for( i = 0; i < count; i++ ) {                 /* setup counter */
            if (cbuf[i] != (x ^ pat)) {            /* compare the data */
                pass_flag = FALSE ;                /* set the pass flag to fail */
                    sprintf(temp,"Memory Error---> adr = %04X exp = %04X ret = %04X",
                             sa, x ^ pat, cbuf[i]);
            puts( temp );                          /* print the error message */
            }
            x += 0x3333;                           /* increment the pattern */
    }
}
/*********************************************************************/
memwcpy( dest, src, cnt)
unsigned short *dest;                                  /* destation data pointer */
unsigned short *src;                                   /* source data pointer */
unsigned short cnt;                                    /* count value */
{
    unsigned short z;
    for (z = 0; z < cnt; z++)                      /* setup counter */
            dest[z] = src[z];                  /* compare the data */
}
```

APPENDIX C

Cabling and Connectors

Introduction

This appendix provides information on cabling and connections.

MIL-STD-1553 Modules

The MS1553 boards are designed to be a part of a dual-redundant 1553 bus system. This means that there are two independent buses that must be cabled and terminated separately.

The board is designed to be connected to a multi-node 1553 bus. The two extreme ends of each channel, A and B, must be terminated with a 70-ohm termination. You must use a Tee connector on each board so that both connections are to cable, or one connection is to cable and the other is the terminator. If you don't terminate the bus, you receive a large number of data errors, or in many cases, the bus may not work at all.

You may order cables, bus couplers, tees and terminators from Condor Engineering or directly from a manufacturer.

MIL-STD-1773 Modules

The MS1773 boards are connected to a fiber-optic bus with 850 nanometer, 200/230 micron step-index optical cable, and SMA low-profile style connectors or ST style connectors. Four cables are required per board. You may order these cables from Condor Engineering (please specify type) or directly from a manufacturer.

Other wavelength fiber-optic drivers are available, contact Condor Engineering for availability.

APPENDIX D

Power Requirements

Requirements

This appendix provides information about the power requirements for the various MS1553 configurations.

	+5V (mA)	+12V (mA)	-12V (mA)
MS1553PC-3/4	445	10	140
MS1773PC-3/4	530	0	0
MS1553PC104	525	0	0
MS1553VME	550	10	140
MS1553VMX	550	0	0

Artisan Technology Group is an independent supplier of quality pre-owned equipment

Gold-standard solutions

Extend the life of your critical industrial, commercial, and military systems with our superior service and support.

We buy equipment

Planning to upgrade your current equipment? Have surplus equipment taking up shelf space? We'll give it a new home.

Learn more!

Visit us at **artisantg.com** for more info on price quotes, drivers, technical specifications, manuals, and documentation.

Artisan Scientific Corporation dba Artisan Technology Group is not an affiliate, representative, or authorized distributor for any manufacturer listed herein.

We're here to make your life easier. How can we help you today?

(217) 352-9330 | sales@artisantg.com | artisantg.com

ARTISAN® TECHNOLOGY GROUP