XVME-983

Software Support Library

P/N 74983-001C

Printed in the United States of America

Xycom Revision Record

Revision	*Description*	*Date*
A	Manual Released	11/90
B	Manual Updated	5/92
C	Manual Updated	11/93

xycom
Technical Publications Department
750 North Maple Road
Saline, MI 48176-1292
734-429-4971 (phone)
734-429-1010 (fax)

TABLE OF CONTENTS

LIST OF TABLES

1.1 MANUAL STRUCTURE

This manual is designed to help you understand and use the Xycom XVME-983 MS-DOS Software Support Library. Chapter 1 gives an overview of the package and directions on how to start using the library. Chapters 2-8 each describe a set of routines for a specific category of XVME boards.

This outline of the manual structure will help you find the specific sections containing information relevant to your system needs:

1.2 OVERVIEW

The XVME-983 MS-DOS Software Support Library (MS-DOS SSL) is a collection of routines designed to provide a consistent, easy-to-use tool for developing application programs for Xycom VMEbus board products. The routines in this library contain the low-level coding necessary to configure the supported hardware modules and perform I/O. You spend less time and effort programming since you do not need to write any module-specific codes or program any routines.

The XVME-983 MS-DOS SSL is distributed on one 3.5-inch, 1.44 Mbyte MS-DOS diskette. It runs on Xycom VMEbus PC/AT processor modules (XVME-674, 677, 678 and 688). The README.DOC file describes the contents of the disk and the location of specific files.

In addition to XVME module-specific routines, the SSL contains general purpose routines which, when run on PC/AT processor modules, can access either Xycom or non-Xycom hardware**.** Real and Protected mode routines provide access to VMEbus address spaces. There are also routines to set up, enable, and disable interrupts and their respective handlers.

The SSL includes source code for all routines. Most routines in this package are written in C; a few are written in assembly language. You can tailor these routines to the specific needs of your system.

The SSL diskette contains:

- Object code libraries to be linked with application programs
- Device drivers for Xycom VME communication boards
- Sample programs for every VME board supported
- Source code and make files for all programs and libraries

The **object code libraries** contain routines needed to communicate with Xycom VME boards. The object code libraries are available in all memory models supported by Microsoft C (version 7.0), Microsoft QuickC (version 2.5), and Borland Turbo C++ (version 3.0). You can simply link in the memory model library corresponding to your configuration to develop an application.

The **device drivers** for Xycom VME communication boards are compatible with MS-DOS version 3.0 or higher. The drivers provide all the necessary facilities to use these boards as additional serial I/O modules.

The sample programs in the Software Support Library (SSL) use each of the defined library routines. The SSL specifies all source files and make files needed to build any of the sample programs. You can use these source files and make files to integrate the SSL into application programs written for your Xycom products.

Most sample programs in this manual were compiled using Microsoft C 7.0. The \EXAMPLES\QCEXAMPL files explain how to use these routines with the Microsoft QuickC compiler; the \EXAMPLES\TCEXAMPL files pertain to the Borland Turbo C++ compiler (Section 1.2.1). If you want to recompile the example programs using Borland Turbo C++, you must refer to your compiler manual and the information in the appropriate directory to achieve the same results.

1.2.1 VMEbus Boards Supported with C Language Subroutines

PC/AT Processors:

XVME-674	33 MHz 80486DX or 66 MHz 80486DX2 processor; 4, 16, or 32 Mbytes DRAM; IDE hard disk and floppy disk controllers; two serial ports and one parallel port
XVME-677	33 M;Hz 80486SX; 4, 16, or 32 Mbytes DRAM; IDE hard disk and floppy disk controllers; Super VGA graphics controller; two serial ports; and one parallel port.
XVME-678	25 MHz Cyrix 486SLC/e processor; 1 or 4 Mbytes DRAM; two RS-232C serial ports; Super VGA controller; and 16-bit IDE hard drive controller
XVME-688	25 MHz 80386SX processor; 1 or 4 Mbytes DRAM; two serial ports; one parallel port; Super VGA controller; and 80387SX math co-processor site

Digital I/O:

XVME-200/290	32-channel DIO with interrupts
XVME-201	48-channel DIO without interrupts
XVME-202	PAMUX controller
XVME-212	32-channel isolated digital input module with change-of-state detection
XVME-220	32-channel isolated digital output module
XVME-240	80-channel digital TTL I/O module
XVME-244	64-channel isolated digital I/O module
XVME-260	32-channel relay output module

Analog I/O:

XVME-500/590	16SE/8DI-channel analog input
XVME-505/595	4-channel analog output
XVME-530	8-channel isolated analog output module
XVME-540	32/16-channel analog input, 4-channel analog output, 12-bit A/D
XVME-560	64/32-channel analog input module
XVME-566	100 KHz, 32/16-channel analog input module

Counter Modules:

XVME-203/293	10-channel counter
XVME-230	16-channel intelligent counter module

1.2.2 VMEbus Boards Supported with MS-DOS Drivers

Communication Modules:

XVME-400/490	4-channel RS-232C serial I/O
XVME-401/491	4-channel RS-422A/RS-485 serial I/O
XVME-428	Intelligent Peripheral Controller Module

1.3 LIBRARY ORGANIZATION

Each library module name references the type and memory model of the compiler which it is designed to support. The library names are all set up in this format:

XVME{compiler type}{memory model}.{library extension}

For example, XVMEMSCC.LIB is the XVME compact memory model library for the Microsoft C 7.0 compiler. The memory models and library extensions are compiler specific, as described below:

For Microsoft C 7.0

XVMEMSCS.LIB for small memory model
XVMEMSCC.LIB for compact memory model
XVMEMSCM.LIB for medium memory model
XVMEMSCL.LIB for large memory model
XVMEMSCH.LIB for huge memory model

***For Microsoft QuickC** 2.5*

XVMEMQCS.LIB for small memory model
XVMEMQCC.LIB for compact memory model
XVMEMQCM.LIB for medium memory model
XVMEMQCL.LIB for large memory model

For Borland Turbo C++ 3.0

XVMEBTCT.LIB for tiny memory model
XVMEBTCS.LIB for small memory model
XVMEBTCC.LIB for compact memory model
XVMEBTCM.LIB for medium memory model
XVMEBTCL.LIB for large memory model
XVMEBTCH.LIB for huge memory model

1.4 INSTALLATION

> **NOTE**
> The information in the remainder of Chapter 1 is also found in a README.DOC file on the diskette.

You do not have to load the SSL files onto the hard disk to run the routines, but we suggest you install the files as directed in this chapter.

The directories on the XVME-983 SSL diskette are:

\EXAMPLES
\INCLUDE
\LIB
\SERIAL

The INSTALL.BAT file on the SSL diskette can be used to create these directories (and their subdirectories) in a target directory and copy the files from the SSL diskette into these directories.

To install XVME983 on a hard disk directory, follow these steps (user input shown in boldface):

1. At the C:> prompt, type **MD XVME983** to create the XVME983 directory.

2. Type **CD XVME983** to get into the directory.

3. Put diskette into drive A.

4. Type **A:INSTALL** to run the install batch file.

Read the note in the section describing the \INCLUDE directory for additional installation instructions.

1.4.1 \EXAMPLES DIRECTORY

The \EXAMPLES directory contains example programs for Xycom XVME modules. This directory is divided into the following subdirectories:

\EXAMPLES\ANALOG

ANALOG.xxx	Menu-driven program that allows users to call each of the routines. Users are prompted for the parameters for the routines. Help screens are provided to describe the routines and their parameters. The routines physically read and write to the XVME-5xx boards in the VMEbus.
X5xxDISP.xxx	Modules linked into ANALOG.xxx for each of the XVME-5xx boards.
X5xxTEXT.H	Include files for X5xxDISP.C files.
ALOG.MAK	Make file for ANALOG.EXE.
X5xxINT.xxx	Programs that show how to use the routines/boards with interrupts.
AINT.MAK	Make and Link file for all X5xxINT.EXE programs.

\EXAMPLES\BITBUS

BITBUS.xxx	A program that shows how to use the XVME-402 with interrupts.
BITPOLL.xxx	A program that shows how to use the XVME-402 in polled mode.
BIT.MAK	Make and Link file for BITBUS and BITPOLL.

\EXAMPLES\COUNTER

COUNTER.xxx	Menu-driven program that lets users call each of the routines. Users are prompted for the routine parameters. Help screens are provided to describe the routines and their parameters. The routines physically read and write to the XVME-2xx boards on the VMEbus.
X2xxDISP.xxx	Modules linked into COUNTER.xxx for each of the XVME-2xx boards.
X2xxTEXT.H	Include files for X2xxDISP.C files.
COUNT.MAK	Make file for COUNTER.EXE.
X2xxINT.xxx	Programs that show how to use the routines/boards with interrupts.
CINT.MAK	Make and Link file for all X2xxINT.EXE programs.

\EXAMPLES\DIGITAL

DIGITAL.xxx	Menu-driven program that lets users call each of the routines. Users are prompted for the routine parameters. Help screens are provided to describe the routines and their parameters. The routines physically read and write to the XVME-2xx boards in the VME bus.
X2xxDISP.xxx	Modules linked into DIGITAL.xxx for each of the XVME-2xx boards.
X2xxTEXT.H	Include files for X2xxDISP.C files.
DIG.MAK	Make file for DIGITAL.EXE.
X2xxINT.xxx	Programs that show how to use the routines/boards with interrupts.
DINT.MAK	Make and Link file for all X2xxINT.EXE programs.

\EXAMPLES\EDITMEM

EDITMEM.XXX	Menu-driven program that utilizes the general purpose memory transfer routines to read and/or write data anywhere in memory.

\EXAMPLES\INCLUDE

This directory contains include files used by the utility routines for the example programs. In order for the make files and C programs to compile, the environmental variable INCLUDE should be set up to include this directory. This is done by executing the following command (as long as XVME983 was created in the root directory):

```
SET INCLUDE=%INCLUDE%;c:\XVME983\EXAMPLES\INCLUDE
```

\EXAMPLES\INT

ANMI.xxx	A program that shows how to handle NMI interrupts.
BERR.xxx	A program that shows how to handle BUS ERROR interrupts.
WDTIMER.xxx	A program showing how to handle interrupts from the watch-dog-timer.
DUALPORT.xxx	A program showing how to handle dual-port interrupts.

\EXAMPLES\QCEXAMPL\IDE (compiled with Microsoft QuickC 2.5)

QCEXAM.BAT A batch file that shows how to invoke the QuickC interactive environment when integrating the XVME-983 software support library.

\EXAMPLES\QCEXAMPL\CMDLINE

QCEXAM.BAT A batch file that shows how to make a QuickC program in the non-interactive environment when integrating the XVME-983 software support library.

\EXAMPLES\TCEXAMPL (compiled with Borland Turbo C++ 3.0)

TCEXAM.BAT A batch file that shows how to invoke the Turbo C interactive environment when integrating the XVME-983 software support library.

\EXAMPLES\UTILS

This directory contains files which provide general support routines (i.e., window display, data entry) for the example programs.

1.4.2 \INCLUDE DIRECTORY

The \INCLUDE directory contains include files used by the library routines and the example programs. In order for the make files and C programs to compile, the environmental variable INCLUDE should be set up to include this directory. This is done by or executing the following command (as long as XVME983 was created in the root directory):

```
SET INCLUDE=%INCLUDE%;c:\XVME983\INCLUDE
```

1.4.3 \LIB DIRECTORY

The \LIB directory contains the XVME-983 software support library. Included in the \LIB\SOURCES subdirectory is the source code and make files necessary to recreate the libraries. This subdirectory contains the following files:

BTCLIBS.BAT	Batch file that builds the XVMEBTCx.LIB library files.
BUILDALL.BAT	Batch file that builds all the XVMExxxx.xxx library files.
MQCLIBS.BAT	Batch file that builds the XVMEMQCx.LIB library files.
MSCLIBS.BAT	Batch file that builds the XVMEMSCx.LIB library files.

The \LIB\SOURCES directory is further divided into the following subdirectories:

\LIB\SOURCES\ANALOG

XVME5xx.xxx	Support routines for each board.
\BTCLIB\ANALLIB.MAK	Make file for the support routines (Turbo C).
\MQCLIB\ANALLIB.MAK	Make file for the support routines (QuickC).
\MSCLIB\ANALLIB.MAK	Make file for the support routines (MSC).

\LIB\SOURCES\COUNTER

XVME2xx.xxx	Support routines for each board.
\BTCLIB\COUNTLIB.MAK	Make file for the support routines (Turbo C).
\MQCLIB\COUNTLIB.MAK	Make file for the support routines (QuickC).
\MSCLIB\COUNTLIB.MAK	Make file for the support routines (MSC).

\LIB\SOURCES\DIGITAL

XVME2xx.xxx	Support routines for each board.
\BTCLIB\DIGITLIB.MAK	Make file for the support routines (Turbo C).
\MQCLIB\DIGITLIB.MAK	Make file for the support routines (QuickC).
\MSCLIB\DIGITLIB.MAK	Make file for the support routines (MSC).

\LIB\SOURCES\XVME

XVME.C	VMEbus general purpose routines.
P286MT.ASM	80286 protected mode memory transfer routine.
P386MT.ASM	80386 protected mode memory transfer routine.
NONANSI.ASM	Rewritten C library routines that were not ANSI standard.
\BTCLIB\XVMELIB.MAK	Make file for the support routines (Turbo C).
\MQCLIB\XVMELIB.MAK	Make file for the support routines (QuickC).
\MSCLIB\XVMELIB.MAK	Make file for the support routines (MSC).

1.4.4 **\SERIAL DIRECTORY**

The \SERIAL directory contains the files for the XVME-400 and 428 device drivers. This directory is divided into subdirectories:

\SERIAL\X40X

X40X.SYS	Device driver for XVME-400/401 boards.
\SOURCE	Subdirectory containing C and assembly source code and a make file for the driver.

\SERIAL\X42X

X42X.SYS	Device driver for XVME-428
\SOURCE	Subdirectory containing C and assembly source code and a make file for the driver.

1.5 **DEVELOPMENT NOTES**

Development was performed on a 80486 AT machine using DOS 5.0.

Development tools were as follows:

For Microsoft (R) C:	Those supplied with version 7.0. CL.EXE, LINK.EXE, LIB.EXE, MAKE.EXE, and MASM 5.1
For Microsoft (R) QUICKC:	Those supplied with version 2.5. QC.EXE, QCL.EXE, LINK.EXE, LIB.EXE, MAKE.EXE, and MASM 5.1
For Borland Turbo C++:	Those supplied with Turbo C++ 3.0. TC.EXE, TCC.EXE, TLINK.EXE, TLIB.EXE, MAKE.EXE, and TASM 2.0

2.1 INTRODUCTION

These general purpose routines are designed to let users easily program Xycom's VMEbus PC/AT processor modules (see Section 1.2.1). This chapter deals with two areas: transferring data to and from the VMEbus address space and supporting VMEbus interrupts. These routines form the basis of all of the specialized routines found in the XVME-983 MS-DOS Software Support Library.

2.2 REAL MODE WINDOW

Some of the general purpose routines use the Real Mode Window (RMW). The RMW provides a mechanism for addressing the entire VMEbus memory space without the need to run the CPU in protected mode. It is 64 Kbytes long and resides at addresses 0E0000-0EFFFF. The window can be configured, via software, to address one of the following: VMEbus Short I/O, VMEbus Standard Address Space, VMEbus IACK Space, or EPROM. You can also configure some PC/AT modules to address VMEbus Extended Address Space. Refer to your PC/AT module manual to determine if you can access Extended Address Space.

When the RMW is configured for VMEbus Short I/O, the 64 Kbyte Short I/O Address Space may be accessed through the 64 Kbyte window. Any references to the RMW will map directly into the VMEbus Short I/O Space.

When the RMW is configured for VMEbus Standard Address Space, the 64 Kbyte window may be used to access any 64 Kbyte block of the VMEbus Standard Address space. In this mode, the 16 Mbyte Standard Address Space is logically divided into 256 64 Kbyte blocks that are configured through software.

When the RMW is configured for VMEbus IACK, a byte read from specific locations in the RMW will cause the PC/AT processor to perform a VMEbus IACK cycle. The data returned from the byte read will be the status ID vector returned from the responding interrupter.

When the RMW is configured for EPROM, the lower 64 Kbytes of the PC/AT's EPROM will appear in the window. This is the mode selected after reset. This mode is compatible with the IBM PC/AT architecture.

When the RMW is configured for VMEbus Extended Address Space, the 64 Kbyte window may be used to access any 64 Kbyte block of the VMEbus Extended Address Space. In this mode, the 4 Gbyte Extended Address Space is logically divided into 65,536 64 Kbyte blocks which are configured through software.

2.3 INTERRUPTS

All Xycom VMEbus PC/AT processor modules are capable of handling interrupts on all seven VMEbus interrupt levels. The XVME-674 and 677 also contain a VMEbus interrupter circuit. This local interrupter allows the local CPU to generate a VMEbus interrupt on any of the seven VMEbus interrupt levels.

2.4 GENERAL PURPOSE ROUTINES

The routines in this section are designed to either process interrupts or transfer memory on any of Xycom VMEbus PC/AT processor modules. Functions unique to a specific CPU are called out.

The parameters you use when implementing the general purpose routines must match the type expected by the routine. If you pass an invalid parameter, the routine will operate erratically.

Parameters:

AccessType	=	1 byte	Address	=	4 bytes
Block64K	=	2 bytes	BlockSize	=	2 bytes
Buffptr	=	Pointer to 1st byte of data buffer	CPU	=	2 bytes
EnableFlag	=	1 byte	EndianType	=	1 byte
IntHandler	=	4 bytes	IRQLevel	=	1 byte
Level	=	1 byte	TimeOut	=	1 byte
TransType	=	1 byte	Vector	=	1 byte
VMELock	=	1 byte			

Certain parameters for the general purpose routines have reserved constants, as shown in Table 2-1.

Table 2-1. Reserved Constants

Parameter	Constants	
AccessType	IACK_ACCESS SHORT_IO_ACCESS PROM_ACCESS EXTEND_ADDR_ACCESS	Interrupt acknowledge space Standard address space XVME-CPU ROM address space Extended address space
EnableFlag	DISABLE ENABLE	
EndianType	BIGENDIAN LITTLEENDIAN	680X0 byte ordering 80X86 byte ordering
TransType	TRANSFER8 TRANSFER16 TRANSFER32	8 bits 16 bits 32 bits
VMELock	LOCK UNLOCK	Keep the VMEbus for entire blocksize transfer Release and then re-acquire the VMEbus for each TransType transfer within the blocksize

These constants are defined in the file \INCLUDE\XVMEDEFS.H.

Source Code Location: \LIB\SOURCES\XVME\XVME.C

2.4.1 Initial XVME Library

Syntax:

AutoInitLib()

Function:

Dynamically determines which XVME-CPU module is being used and initializes the XVME SSL. This routine or the InitLib routine **must** be called initially for the library to function correctly.

Return Value:

Type of CPU module (674, 677, 678, or 688)

2.4.2 Initialize XVME Library

Syntax:

InitLib(CPU)

Where:

CPU = Type of XVME CPU module (674, 677, 678, or 688)

Function:

This routine initializes the XVME SSL for a particular XVME CPU model. It **must** be called initially with the appropriate CPU value for the library to function correctly.

Return Value:

None

2.4.3 Release the VMEbus

Syntax:

ReleaseVMEBus()

Function:

This routine releases the VMEBus from CPU control.

Return Value:

None

2.4.4 Access the VMEbus

Syntax:

LockVMEBus(TimeOut)

Where:

TimeOut = Amount of time to wait for the bus

Function:

This routine tries to gain access to the VMEbus within the specified time. The TimeOut value has a resolution of 55 milliseconds (i.e., a value of 18 is approximately equal to one second). If the routine receives a value of zero, it waits infinitely for the bus. A zero value is returned if the CPU has successfully accessed the VMEbus. A non-zero value is returned if the TimeOut value expires before gaining access to the VMEbus.

Return Value:

Zero if VMEbus has been accessed; non-zero if not

2.4.5 Set Real Mode Window

Syntax:

Set_RM_Window(AccessType,Block64K)

Where:

AccessType	=	Type of VMEbus access desired IACK_ACCESS - interrupt acknowledge space SHORT_IO_ACCESS - short I/O space STAND_ADDR_ACCESS - standard address space PROM_ACCESS - XVME-CPU ROM address space EXTEND_ADDR_ACCESS - extended address space (XVME-674, 677)
Block64	=	Which 64 Kbyte block to map in if standard or extended access is desired.

Function:

This routine maps the Real Mode Window (0xE0000 - 0xEFFFF) to the desired VMEbus address space.

Return Value:

None

2.4.6 **Read VMEbus Memory Through The Real Mode Window**

Syntax:

ReadVMEBusMemoryRM(Buffptr,TransType,EndianType,BlockSize,AccessType,Address, VMELock)

Where:

Buffptr	=	A byte pointer to the local storage area
TransType	=	Type of data transfer TRANSFER8 - 8 bits TRANSFER16 - 16 bits TRANSFER32 - 32 bits
EndianType	=	BIGENDIAN(680x0) or LITTLEENDIAN(80x86)
BlockSize	=	Number of bytes to read (1 to 64K)
AccessType	=	Type of VMEbus access IACK_ACCESS - Interrupt Acknowledge Space SHORT_IO_ACCESS - Short I/O Space STAND_ADDR_ACCESS - Standard Address Space PROM_ACCESS - XVME-CPU Local EPROM Space EXTEND_ADDR_ACCESS - Extended Address Space
Address	=	Specifies starting address for the transfer (32-bit address)
VMELock	=	Set TRUE to lock the VMEbus for the entire BlockSize transfer. Else the VMEBus will be locked and then released for each transfer within BlockSize.

Function:

This routine reads a block of memory on the VMEbus through the Real Mode Window (0xE0000 - 0xEFFFF). This routine **cannot** be used to read the local dual-port memory on the PC/AT processor.

Return Value:

Zero if successful; non-zero if not

2.4.7 **Write VMEbus Memory Through The Real Mode Window**

Syntax:

WriteVMEBusMemoryRM(Buffptr,TransType,EndianType,
BlockSize,AccessType,Address,VMELock)

Where:

Buffptr	=	A byte pointer to the local storage area
TransType	=	Type of data transfer TRANSFER8 - 8 bits TRANSFER16 - 16 bits TRANSFER32 - 32 bits
EndianType	=	BIGENDIAN(680x0) or LITTLEENDIAN(80x86)
BlockSize	=	Number of bytes to write (1 to 64K)
AccessType	=	Type of VMEbus access IACK_ACCESS - Interrupt Acknowledge space SHORT_IO_ACCESS - Short I/O Space STAND_ADDR_ACCESS - Standard Address Space PROM_ACCESS - XVME-CPU EPROM Address Space EXTEND_ADDR_ACCESS - Extended Address Space
Address	=	Specifies starting address for the transfer (32-bit address)
VMELock	=	Set TRUE to lock the VMEbus for the entire BlockSize transfer. Else the VMEBus will be locked and then released for each transfer within BlockSize.

Function:

This routine writes a block of memory out on the VMEbus through the Real Mode Window (0xE0000 - 0xEFFFF). This routine **cannot** be used to write to local dual port memory.

Return Value:

Zero if successful; non-zero if not

2.4.8 **Read VMEbus Memory In Protected Mode**

Syntax:

ReadVMEBusMemoryPM(Buffptr,TransType,EndianType,BlockSize,Address,VMELock)

where:

Buffptr	=	A byte pointer to the local storage area
TransType	=	Type of data transfer TRANSFER8 - 8 bits TRANSFER16 - 16 bits TRANSFER32 - 32 bits
EndianType	=	BIGENDIAN(680x0) or LITTLEENDIAN(80x86)
BlockSize	=	Number of bytes to read (1 to 64K)
Address	=	Specifies starting address for the transfer (32-bit address)
VMELock	=	Set TRUE to lock the VMEbus for the entire BlockSize transfer. Else the VMEBus will be locked and then released for each transfer within BlockSize.

Function:

This routine reads a block of memory out on the VMEbus with the CPU in Protected Mode. This routine **can** be used to read from local dual-port memory.

Return Value:

Zero if successful; non-zero if not

NOTE

This routine is not available for the XVME-678 or 688. These modules cannot access the VMEbus in protected mode.

2.4.9 Write VMEbus Memory In Protected Mode

Syntax:

WriteVMEBusMemoryPM(Buffptr,TransType,EndianType,BlockSize,Address,VMELock)

Where:

Buffptr	=	A byte pointer to the local storage area
TransType	=	Type of data transfer TRANSFER8 - 8 bits TRANSFER16 - 16 bits TRANSFER32 - 32 bits
EndianType	=	BIGENDIAN(680x0) or LITTLEENDIAN(80x86)
BlockSize	=	Number of bytes to write (1 to 64K)
Address	=	Specifies starting address for the transfer (32-bit address)
VMELock	=	Set TRUE to lock the VMEbus for the entire BlockSize transfer. Else the VMEBus will be locked and then released for each transfer within BlockSize.

Function:

This routine writes a block of memory out on the VMEbus with the CPU in Protected Mode. This routine **can** be used to write to local dual-port memory.

Return Value:

Zero if successful; non-zero if not

NOTE

This routine is not available for the XVME-678 or 688. These modules cannot access the VMEbus in protected mode.

2.4.10 Set Interrupt Vector

Syntax:

SetIntVect(IRQLevel, IntHandler)

Where:

IRQLevel = AT IRQ (0-15) level whose interrupt vector is to be replaced.
IntHandler = Address of user-defined interrupt handler

Function:

This routine sets the desired IRQ (0 - 15) vector to point to the user-defined interrupt handler. It returns the IRQ vector that is being replaced so it can be restored at a later time if desired.

Return Value:

Address of original IRQ vector

2.4.11 Read Interrupt Vector

Syntax:

ReadIntVect(IRQLevel)

Where:

IRQLevel = AT IRQ (0-15) level whose interrupt vector is to be read

Function:

This routine returns the current interrupt vector for the desired IRQ (0 - 15) level.

Return Value:

Address of IRQ vector

2.4.12 Mask the Interrupt Controller

Syntax:

Mask8259(IRQLevel, EnableFlag)

Where:

IRQLevel = AT IRQ (8-15) level to mask.
EnableFlag = Flag (ENABLE/DISABLE) to enable or disable.

Function:

This routine sends the appropriate mask to the slave 8259 interrupt controller to enable or disable the desired IRQ level.

Return Value:

None

2.4.13 **Disable VME Interrupts**

Syntax:

DisableVMEInterrupts()

Function:

This routine disables the AT Auxiliary Maskable Interrupts used for VME interrupt levels 1-7 and the dual-port interrupt.

Return Value:

None

2.4.14 **Enable VME Interrupts**

Syntax:

EnableVMEInterrupts()

Function:

This routine enables the AT Auxiliary Maskable Interrupts used for VME interrupt levels 1-7 and the dual-port interrupt.

Return Value:

None

2.4.15 **Set NMI Interrupt Vector**

Syntax:

SetNMIVect(IntHandler)

Where:

IntHandler = Address of user-defined interrupt handler.

Function:

This routine sets the NMI vector to point to the user-defined interrupt handler. It returns the NMI vector that is being replaced so it can be restored at a later time if desired.

Return Value:

Address of the original NMI vector

2.4.16 **Read NMI Vector**

Syntax:

ReadNMIVect()

Function:

This routine returns the current NMI interrupt vector.

Return Value:

Address of the current NMI vector

2.4.17 **Enable Auxiliary NMI**

Syntax:
EnableNMIInt()

Function:
This routine enables Auxiliary NMIs.

Return Value:
None

2.4.18 **Disable Auxiliary NMI**

Syntax:
DisableNMIInt()

Function:
This routine disables Auxiliary NMIs.

Return Value:
None

2.4.19 **Generate VMEbus Interrupt**

Syntax:
GenVMEBusInt(Level,Vector)

Where:
Level = VMEbus interrupt level (1-7)
Vector = Interrupt vector (0-255)

Function:
This routine generates a VMEbus interrupt on the specified level. An error value will be returned if the XVME-CPU board cannot generate a VMEbus interrupt.

Return Value:
Zero if successful; non-zero if not

NOTE
This routine is not available on the XVME-678 or 688, as they do not have a VMEbus Interrupter.

2.4.20 **Enable the Watchdog Timer**

Syntax:

EnableWDTimer()

Function:

This routine enables the Watchdog Timer.

Return Value:

None

> **NOTE**
> This routine is not available on the XVME-678 or 688, as they do not have a Watchdog timer.

2.4.21 **Disable the Watchdog Timer**

Syntax:

DisableWDTimer()

Function:

This routine disables the Watchdog Timer.

Return Value:

None

> **NOTE**
> This function is not available on the XVME-678 or 688.

2.4.22 **Strobe the Watchdog Timer**

Syntax:

StrobeWDTimer()

Function:

This routine retriggers the Watchdog Timer. This routine must be executed at least once every 150 ms to keep the Watchdog Timer from generating an Auxiliary NMI (if enabled).

Return Value:

None

> **NOTE**
> This function is not available on the XVME-678 or 688.

2.4.23 **Reset the Watchdog Timer**

Syntax:

ResetWDTimer()

Function:

This routine resets the Watchdog Timer after it has timed out.

Return Value:

None

> **NOTE**
> This function is not available on the XVME-678 or 688.

3.1 INTRODUCTION

Table 3-1. Digital I/O Routine Matrix

	MODULE							
Routine	**200/290**	**201**	**202**	**212**	**220**	**240**	**244**	**260**
CPLoad	•	•						
Init	•	•	•	•				
InitDisable				•				
Read	•	•	•	•	•	•	•	•
ReadAll					•		•	•
ReadChannel				•	•		•	
ReadPort								
Read2Ports								
Read4Ports								
ReadScan				•				
ReadWord				•	•	•	•	•
ReadWordScan				•				
Reset			•		•	•	•	
SetPortDir		•						
SetPortADir	•							
SetPortBDir	•							
SetPortCDir		•						
SetSubModeA	•							
SetSubModeB	•							
Write	•	•	•	•	•	•	•	•
WriteAll					•		•	•
WriteChannel					•		•	•
WriteWord				•	•	•	•	•
	200/290	**201**	**202**	**212**	**220**	**240**	**244**	**260**

3.2 XVME-200/290 DIGITAL I/O MODULE

The XVME-200 is a Digital I/O Module consisting of a VMEbus interface, two 68230 PI/T chips, and TTL buffers. The two PI/T chips provide 32 bits of digital I/O. The XVME-200 provides VMEbus interrupts and makes the PI/T timer available to users. The XVME-290 is a 6U version of the XVME-200 with the I/O routed to the VMEbus P2 connector.

The parameters used in the XVME-200/290 routines must match the type expected by the routine. If you pass an invalid parameter, the routine will operate erratically.

Parameters:

ByteData	=	1 byte	Direction	=	1 byte
PITNum	=	1 byte	Register	=	2 byte
SubMode	=	1 byte	TimerVal	=	4 bytes
X200Base	=	2 bytes			

Source Code Location: LIB\SOURCES\DIGITAL\XVME200.C

The routines in the following sections are available to the user in the XVME-983 Software Support Library.

3.2.1 Initialize

Syntax:

X200Init(X200Base)

Where:

X200Base = board address in VMEbus short I/O space

Function:

This function initializes Port C and Data Direction Registers on both PI/T chips to prevent unintentional interrupts.

Return Value:

None

3.2.2 Set the Counter Pre-Load Register

Syntax:

X200CPLoad(X200Base,PITNum,TimerVal)

Where:

X200Base	=	Board address in VMEbus short I/O space
PITNum	=	Number of PI/T chip (0 or 1)
TimerVal	=	Value to write into register (24 bit)

Function:

This routine writes a value between 0 and 16,777,215 to the Counter Pre-Load Register of the indicated PI/T chip.

Return Value:

None

3.2.3 Set Port A Direction

Syntax:

X200PortADir(X200Base,PITNum,Direction)

Where:

X200Base	=	Board address in VMEbus short I/O space
PITNum	=	Number of PI/T chip (0 or 1)
Direction	=	Sets port to either IN (0) or OUT (1)

Function:

This routine sets the data direction for Port A, as well as the transceiver direction, on the desired PI/T chip.

Return Value:

None

3.2.4 Set Port B Direction

Syntax:

X200PortBDir(X200Base,PITNum,Direction)

Where:

X200Base	=	Board address in VMEbus short I/O space
PITNum	=	Number of PI/T chip (0 or 1)
Direction	=	Sets port to either IN (0) or OUT (1)

Function:

This routine sets the Port B Data Direction, as well as the transceiver direction, on the desired PI/T chip.

Return Value:

None

3.2.5 Set Port A Sub Mode

Syntax:

X200SubModeA(X200Base,PITNum,SubMode)

Where:

X200Base	=	Board address in VMEbus short I/O space
PITNum	=	Number of PI/T chip (0 or 1)
SubMode	=	Sets Sub Mode 0 = 0 or 0X 1 = 1 2 = 1X

Function:

This routine sets the Port A Sub Mode in the Port A control Register on the desired PI/T chip.

Return Value:

None

3.2.6 **Set Port B Sub Mode**

Syntax:

X200SubModeB(X200Base,PITNum,SubMode)

Where:

X200Base	=	Board address in VMEbus short I/O space
PITNum	=	Number of PI/T chip (0 or 1)
SubMode	=	Sets Sub Mode 0 = 0 or 0X 1 = 1 2 = X

Function:

This routine sets the Port B Sub Mode in the Port B control register on the desired PI/T chip.

Return Value:

None

3.2.7 **Read a Byte**

Syntax:

X200Read(X200Base,PITNum,Register)

Where:

X200Base	=	Board address in VMEbus short I/O space
PITNum	=	Number of PI/T chip (0 or 1)
Register	=	Address offset of register to Read

Function:

This routine reads and returns a byte from a desired register in the chosen PI/T chip. The XVME-200/290 manual contains all of the register definitions and corresponding addresses.

Return Value:

Byte read from register

3.2.8 **Write a Byte**

Syntax:

X200Write(X200Base,PITNum,Register,ByteData)

Where:

X200Base	=	Board address in VMEbus short I/O space
PITNum	=	Number of PI/T chip (0 or 1)
Register	=	Address offset of register to Read
ByteData	=	The byte value to be written

Function:

This routine writes a given byte to a desired register in the chosen PI/T chip. The XVME-200/290 manual contains all of the register definitions and corresponding addresses.

Return Value:

None

3.2.9 **Interrupt Example**

There are two types of interrupts on the XVME-200/290: port and timer. For your system to acknowledge either type of interrupt, you must set your jumpers and/or switches as detailed below. You can also refer to the XVME-200/290 manual for further details.

An example program for each type of interrupt can be found in the library. The file for the port interrupt example is:

\EXAMPLES\DIGITAL\X200PINT.EXE

The file for the timer interrupt example is:

\EXAMPLES\DIGITAL\X200TINT.EXE

Port Interrupt Jumpers*:*

XVME-200: J1 and J2 are IN (handshake line H2 is an INPUT)
JA1 - JA3 configured to enable one of the VMEbus interrupt levels.

XVME-290: J1 and J2 are IN (handshake line H2 is an INPUT)
JA1 - JA3 configured to enable one of the VMEbus interrupt levels.

To cause an interrupt, ground line H2. An Interrupt Occurred message will appear on screen.

Timer Interrupt Jumpers*:*

XVME-290: JA1 - JA3 configured to enable on VMEbus interrupt level

XVME-200: JA1 - JA3 configured to enable on VMEbus interrupt level

An interrupt will occur approximately every 0.5 sec. A Timer Int n message will appear on the screen.

3.3 XVME-201 DIGITAL I/O MODULE

The XVME-201 is a Digital I/O Module consisting of a VMEbus interface, two 68230 PI/T chips, and TTL buffers. The two PI/T chips provide 48 bits of digital I/O. The XVME-201 also makes the PI/T timer available to the user. The XVME-201 is not capable of generating interrupts.

The parameters used in the routines for the XVME-201 must match the type expected by the routine. If you pass an invalid parameter, the routine will operate erratically.

Parameters:

ByteData	=	1 byte	Direction	=	1 byte
PITNum	=	1 byte	Register	=	2 bytes
TimerVal	=	4 bytes	X201Base	=	2 bytes

Source Code Location: \LIB\SOURCES\DIGITAL\XVME201.C

The routines in the following sections are available to the user in the XVME-983 Software Support Library.

3.3.1 **Initialize**

Syntax:

X201Init(X201Base)

Where:

X201Base = Board address in VMEbus short I/O space

Function:

This function initializes the two PI/T chips to Mode 0, Submode 1X, and sets Port C by disabling Timer and Port Interrupts.

Return Value:

None

3.3.2 **Set Port Direction**

Syntax:

SetPortDir(X201Base,PITnum,Direction)

Where:

X201Base = Board address in VMEbus short I/O space
PITNum = Number of PI/T chip (0 or 1)
Direction = Sets port to either IN or OUT

Function:

This routine sets the data direction for Port A, as well as the transceiver direction, on the desired PI/T chip.

If Direction = 0 (IN): all pins on Ports A and B are set up as inputs, then the transceiver set to input.

If Direction = 1 (OUT): the transceiver is set to output, then all pins on Ports A and B are set up as outputs.

Return Value:

None

3.3.3 **Set Port C Direction**

Syntax:

X201PortCDir(X201Base,PITNum,Direction)

Where:

X201Base	=	Board address in VMEbus short I/O space
PITNum	=	Number of PI/T chip (0 or 1)
Direction	=	Sets port to either IN or OUT

Function:

This routine sets the Port C pins as either all inputs or all outputs.

If Direction = 0 (IN): All pins on Port C are set up as inputs and the transceiver set to input.
If Direction = 1 (OUT): The transceiver is set to output, then all pins on Port C are set up as outputs.

Return Value:

None

3.3.4 **Set the Counter Pre-Load Register**

Syntax:

X201CPLoad(X201Base,PITNum,TimerVal)

Where:

X201Base	=	Board address in VMEbus short I/O space
PITNum	=	Number of PI/T chip (0 or 1)
TimerVal	=	Value to write into register (24 bit)

Function:

This routine writes a value between 0 - 16,777,215 to the Counter Pre-Load Register of the indicated PI/T chip.

Return Value:

None

3.3.5 **Read a Byte**

Syntax:

X201Read(X201Base,PITNum,Register)

Where:

X201Base	=	Board address in VMEbus short I/O space
PITNum	=	Number of PI/T chip (0 or 1)
Register	=	Address offset of register to Read

Function:

This routine reads and returns a byte from a desired register in the chosen PI/T chip. The XVME-201 manual contains all of the register definitions and corresponding addresses.

Return Value:

Byte read from register

3.3.6 **Write a Byte**

Syntax:

X201Write(X201Base,PITNum,Register,ByteData)

Where:

X201Base	=	Board address in VMEbus short I/O space
PITNum	=	Number of PI/T chip (0 or 1)
Register	=	Address offset of register to Read
ByteData	=	The byte value to be written

Function:

This routine writes a given byte to a desired register in the chosen PI/T chip. The XVME-201 manual contains all of the register definitions and corresponding addresses.

Return Value:

None

3.4 XVME-202 PAMUX CONTROLLER

The XVME-202 PAMUX Interface Module is a single-high VMEbus compatible board that allows a VMEbus master to communicate with a PAMUX I/O subsystem.

The parameters used in the routines for the XVME-202 must match the type expected by the routine. If you pass an invalid parameter, the routine will operate erratically.

Parameters:

Bank = 1 byte
ByteData = 1 byte
X202Base = 2 bytes

Source Code Location: \LIB\SOURCES\DIGITAL\XVME202.C

The routines in the following sections are available to the user in the XVME-983 Software Support Library.

3.4.1 Initialize

Syntax:

X202Init (X202Base)

Where:

X202Base = Board address in VMEbus short I/O space.

Function:

This routine deactivates the reset line to enable the module.

Return Value:

None

3.4.2 **Read a Byte**

Syntax:

X202Read (X202Base,Bank)

Where:

X202Base	=	Board address in VMEbus short I/O space
Bank	=	Bank register to read from (0-63)

Function:

This routine reads a byte value from the desired PAMUX bank. The XVME-202 manual contains information on bank register parameters.

Return Value:

Byte read from bank

3.4.3 **Write a Byte**

Syntax: X202Write (X202Base,Bank,ByteData)

Where:

X202Base	=	Board address in VMEbus short I/O space
Bank	=	Bank register to write to
ByteData	=	The byte value to be written

Function:

This routine writes a byte value to the desired PAMUX bank. The XVME-202 manual contains information on bank register definitions.

Return Value:

None

3.4.4 **Reset**

Syntax:

X202Reset (X202Base)

Where:

X202Base	=	Board address in VMEbus short I/O space

Function:

This routine asserts the reset line on the XVME-202 which in turn will reset the attached PAMUX units.

Return Value:

None

3.5 XVME-212 DIGITAL INPUT MODULE

The XVME-212 is a Digital Input Module consisting of a VMEbus interface with 32 isolated digital input channels. The XVME-212 features a programmable scanner which can detect a change of state on any input line and generate a VMEbus interrupt on any level when a change of state is detected.

The parameters used in the routines for the XVME-212 must match the type expected by the routine. If you pass an invalid parameter, the routine will operate erratically.

Parameters:

Channel	=	1 byte	CRStatus	=	Pointer to 1st byte of data buffer
ByteData	=	1 byte	DRStatus	=	Pointer to 1st byte of data buffer
Port	=	1 byte	Register	=	2 bytes
RegisterSet	=	1 byte	StartPort	=	1 byte
WordData	=	2 bytes	X212Base	=	2 bytes

Source Code Location: \LIB\SOURCES\DIGITAL\XVME212.C

The routines in the following sections are available to the user in the XVME-983 Software Support Library.

3.5.1 Initialize

Syntax:

X212Init(X212Base)

Where:

X212Base = Board address in VMEbus short I/O space

Function: This function will clear all of the Change Registers by reading all of the corresponding Data Registers. The FAIL LED will go out and the PASS LED will light.

Return Value: None

3.5.2 **Read a Byte**

Syntax:

X212Read(X212Base,Register)

Where:

X212Base	=	Board address in VMEbus short I/O space
Register	=	Address offset of register to Read

Function:

This routine reads and returns a byte from a desired register. The XVME-212 manual contains all of the register definitions and corresponding addresses.

> **NOTE**
> Reading any Change Register stops the XVME-212 scanner.

Return Value:

The byte read at the desired register

3.5.3 **Write a Byte**

Syntax:

X212Write(X212Base,Register,ByteData)

Where:

X212Base	=	Board address in VMEbus short I/O space
Register	=	Address offset of register to Read
ByteData	=	The byte value to be written

Function:

This routine writes a given byte to a desired register. The XVME-212 manual contains all of the register definitions and corresponding addresses.

Return Value: None

3.5.4 **Read a Word**

Syntax:

X212ReadWord(X212Base,Register)

Where:

X212Base	=	Board address in VMEbus short I/O space
Register	=	Address offset of first register to Read

Function:

This routine reads and returns a word (2 byte) value from two consecutive registers. The XVME-212 manual contains all of the register definitions and corresponding addresses.

Return Value:

The word value read starting at the specified register

3.5.5 **Write a Word**

Syntax:

X212WriteWord(X212Base,Register,WordData)

Where:

X212Base	=	Board address in VMEbus short I/O space
Register	=	Address offset of register
WordData	=	The word value to be written

Function:

This routine writes a given word (two bytes) to a desired register. The XVME-212 manual contains all of the register definitions and corresponding addresses.

Return Value:

None

3.5.6 **Read Scan**

Syntax:

X212ReadScan(X212Base,CRStatus,DRStatus,Port)

Where:

X212Base	=	Board address in VMEbus short I/O space
CRStatus	=	Pointer to a byte Where Change Register status can be stored
DRStatus	=	Pointer to a byte Where Data Register status can be stored
Port	=	One of four ports
	0 =	Port 1
	1 =	Port 2
	2 =	Port 3
	3 =	Port 4

Function:

This routine first reads the Change Register status for the specified port. This causes the XVME-212 Scanner to stop. The routine then reads that same port's Data Register status. Reading the Data Register status restarts the Scanner.

> **NOTE**
> All interrupts should be disabled during this routine to minimize the time that the scanner is off.

Return Value:

None

3.5.7 **Read Word Scan**

Syntax:

X212ReadWordScan(X212Base,CRStatus,DRStatus,StartPort)

Where:

X212Base	=	Board address in VMEbus short I/O space
CRStatus	=	Pointer to a word Where Change Register status can be stored
DRStatus	=	Pointer to a word Where Data Register status can be stored
StartPort	=	Identifies the first of four ports

If StartPort is:	Data is read from:
0	Ports 1 and 2
1	Ports 2 and 3
2	Ports 3 and 4

Function:

This routine performs the same function as ReadScan, except that it reads two bytes.

3.5.8 **Interrupt Disable**

Syntax:

X212IntDisable(X212Base)

Where:

X212Base	=	Board address in VMEbus short I/O space

Function:

This function will disable the Interrupt Enable bit in the Status Control Register.

Return Value:

None

3.5.9 **Read a Channel**

Syntax:

X212ReadChannel(X212Base,Channel,RegisterSet)

Where:

X212Base	=	Board address in VMEbus short I/O space
Channel	=	Channel containing status information
RegisterSet	=	Indicates which register to read channel from
		0 = Change Register
		1 = Data Register

Function:

This routine reads a channel from a Data or Change Register and returns:
0 = Channel is low
1 = Channel is high

Return Value:

0 or 1

3.5.10 **Interrupt Example**

For your system to acknowledge interrupts generated by the board, you must set your jumpers and or switches as detailed below. Please refer to the XVME PC/AT manual and the XVME-212 manual for further details.

An example program for the use of these interrupts can be found in the library. The file for the example is as follows:

\EXAMPLES\DIGITAL\X212INT.EXE

Jumpers:

XVME-212: Switch SW2 configured to enable one of the VMEbus interrupt levels.

An Interrupt Occurred message will appear on screen each time a change of state occurs on any one of the 32 input channels.

3.6 **XVME-220 DIGITAL OUTPUT MODULE**

The XVME-220 is a Digital Output Module consisting of a VMEbus interface with 32 isolated digital output channels. All outputs are software readable and disabled whenever the module is reset. Open collector outputs are provided to 30VDC and 100mA. Transient and Reverse bias protection is available on all output channels.

The parameters used in the routines for the XVME-220 must match the type expected by the routine. If you

pass an invalid parameter, the routine will operate erratically.

Parameters:

ByteData	=	1 byte	Channel	=	1 byte
DWordData	=	4 bytes	Register	=	2 bytes
Start	=	2 bytes	WordData	=	2 byte
X220Base	=	2 bytes			

Source Code Location: \LIB\SOURCES\DIGITAL\XVME220.C

The routines in the following sections are available to the user in the XVME-983 Software Support Library.

3.6.1 **Read a Channel**

Syntax:

X220ReadChannel(X220Base,Channel)

Where:

X220Base	=	Board address in VMEbus short I/O space
Channel	=	Channel containing status information (0-31)

Function:

This routine reads a specified channel and returns:

0	=	Channel is low
1	=	Channel is high

Return Value:

0 or 1

3.6.2 **Read a Byte**

Syntax:

X220Read(X220Base,Register)

Where:

X220Base	=	Board address in VMEbus short I/O space
Register	=	Address offset of register or port to Read

Function:

This routine reads and returns a byte value from a register or port. The XVME-220 manual contains all of the register definitions and corresponding addresses.

Return Value:

Byte value read from desired register

3.6.3 **Read a Word**

Syntax::

X220ReadWord(X220Base,Start)

Where:

X220Base	=	Board address in VMEbus short I/O space
Start	=	Address offset of first register or port to Read

Function:

This routine reads and returns a word value from two consecutive registers or ports. The XVME-220 manual contains all of the register definitions and corresponding addresses.

Return Value:

The word value read starting at the specified register

3.6.4 **Read All**

Syntax:

X220ReadAll(X220Base)

Where:

X220Base	=	Board address in VMEbus short I/O space

Function:

This routine reads and returns a four byte (2 word) value from all four output status registers. The XVME-220 manual contains all of the register definitions and corresponding addresses.

Return Value:

The double word status of all four ports

3.6.5 **Write a Channel**

Syntax:

X220WriteChannel(X220Base,Channel,ByteData)

Where:

X220Base	=	Board address in VMEbus short I/O space
Channel	=	Channel to write status information (0-31)
ByteData	=	Information to configure output Channel
	0 =	Channel set low
	1 =	Channel set high

Function:

This routine sets a specified channel to the desired polarity

Return Value:

None

3.6.6 **Write a Byte**

Syntax:

X220Write(X220Base,Register,ByteData)

Where:

X220Base	=	Board address in VMEbus short I/O space
Register	=	Address offset of register or port to write
ByteData	=	The byte value to be written

Function:

This routine writes a given byte to a desired register or port. The XVME-220 manual contains all of the register and port definitions and corresponding addresses.

Return Value:

None

3.6.7 **Write a Word**

Syntax:

X220WriteWord(X220Base,Start,WordData)

Where:

X220Base	=	Board address in VMEbus short I/O space
Start	=	Address offset of first register or port to write
WordData	=	The word value to be written

Function:

This routine writes a given word (two bytes) to two consecutive registers or ports. The XVME-220 manual contains all of the register and port definitions and corresponding addresses.

Return Value:

None

3.6.8 **Write All**

Syntax:

X220WriteAll(X220Base,DWordData)

Where:

X220Base	=	Board address in VMEbus short I/O space
DWordData	=	The double word value to be written

Function:

This routine writes two words (four bytes) to all four configuration ports, configuring all 32 output channels.

Return Value:

None

3.6.9 **Reset**

Syntax:

X220Reset(X220Base)

Where:

X220Base	=	Board address in VMEbus short I/O space

Function:

This routine sets all of the output port configuration registers to 0.

Return Value:

None

3.7 XVME-240 DIGITAL INPUT/OUTPUT MODULE

The XVME-240 is a Digital Input/Output Module that provides 80 TTL-level channels. Sixty-four of these channels are organized as eight ports of 8 bits each. All of the ports can be programmed individually as either input or output. The XVME-240 also provides eight edge-selectable inputs that can be programmably masked to generate VMEbus interrupts. Eight flag outputs are also provided.

The parameters used in the routines for the XVME-240 must match the type expected by the routine. If you pass an invalid parameter, the routine will operate erratically.

Parameters:

ByteData	=	1 byte	Register	=	2 bytes
Start	=	2 bytes	WordData	=	2 bytes
X240Base	=	2 bytes			

Source Code Location: \LIB\SOURCES\DIGITAL\XVME240.C

The routines in the following sections are available to the user in the XVME-983 Software Support Library.

3.7.1 Reset

Syntax:

X240Reset(X240Base)

Where:

X240Base = Board address in VMEbus short I/O space

Function:

This routine sets the interrupt mask to 0, sets the output flags as low, clears the interrupt latches, and sets all ports as inputs. The PASS LED will be ON and the FAIL LED will be OFF

Return Value:

None

3.7.2 **Read a Byte**

Syntax:

X240Read(X240Base,Register)

Where:

X240Base	=	Board address in VMEbus short I/O space
Register	=	Address offset of register or port to Read

Function:

This routine reads and returns a byte value from a register or port. The XVME-240 manual contains all of the register definitions and corresponding addresses.

Return Value:

Byte value read from specified port or register

3.7.3 **Read a Word**

Syntax:

X240ReadWord(X240Base,Start)

Where:

X240Base	=	Board address in VMEbus short I/O space
Start	=	Address offset of first register or port to Read

Function:

This routine reads and returns a word (two byte) value from two consecutive registers or ports. The XVME-240 manual contains all of the register definitions and corresponding addresses.

Return Value:

The word value read starting at the specified port or register

3.7.4 **Write a Byte**

Syntax:

X240Write(X240Base,Register,ByteData)

Where:

X240Base	=	Board address in VMEbus short I/O space
Register	=	Address offset of register or port to write
ByteData	=	The byte value to be written

Function:

This routine writes a given byte to a desired register or port. The XVME-240 manual contains all of the register and port definitions and corresponding addresses.

Return Value:
None

3.7.5 **Write a Word**

Syntax:
X240WriteWord(X240Base,Start,WordData)

Where:

X240Base	=	Board address in VMEbus short I/O space
Start	=	Address offset of first register or port to write
WordData	=	The word value to be written

Function:
This routine writes a given word (two bytes) to two consecutive registers or ports. The XVME-240 manual contains all of the register and port definitions and corresponding addresses.

Return Value:
None

3.7.6 **Interrupt Example**

The XVME-240 board will generate an interrupt whenever one of the eight edge-selectable inputs is toggled.

For your system to acknowledge interrupts generated by the board, you must set your jumpers and or switches as detailed below. Please refer to the XVME PC/AT manual and the XVME-240 manual for further details.

An example program for the use of interrupts can be found in the Library. The file for the example is:

\EXAMPLES\DIGITAL\X240INT.EXE

Jumpers:

XVME-240: One of the interrupts must be enabled via switch S2.

An Interrupt Occurred message will appear on screen each time a change of state occurs on any one of the edge-selectable inputs.

3.8 XVME-244 DIGITAL OUTPUT MODULE

The XVME-244 is a Digital Output Module consisting of a VMEbus interface with 64 isolated digital I/O channels, 32 input channels, and 32 output channels. The inputs are arranged in four groups of eight each. Each group is optically isolated from the others.

The parameters used in the routines for the XVME-244 must match the type expected by the routine. If you pass an invalid parameter, the routine will operate erratically.

Parameters:

ByteData	=	1 byte	STATUS_REG =	129 (0x81)	
DWordData	=	4 bytes	OUTPUT_REG =	130 (0x82)	
Start	=	2 bytes	FILTERED_REG	=	160 (0xA0)
X244Base	=	2 bytes	INPUT_REG	=	192 (0xC0)
Channel	=	1 byte			
Register	=	2 bytes			
WordData	=	2 byte			

Source Code Location: \LIB\SOURCES\DIGITAL\XVME244.C

The routines in the following sections are available to the user in the XVME-983 Software Support Library.

3.8.1 Read a Channel

Syntax:

X244ReadChannel(X244Base,Channel,Type)

Where:

X244Base	=	Board address in VMEbus short I/O space
Channel	=	Channel containing status information (0-31)
Type	=	Register type (INPUT_REG, FILTERED_REG, or OUTPUT_REG)

Function:

This routine reads a specified channel and returns:

0	=	Channel is low
1	=	Channel is high

Return Value:

0 or 1

3.8.2 Read a Byte

Syntax:

X244Read(X244Base,Register,Type)

Where:

X244Base	=	Board address in VMEbus short I/O space
Register	=	Address offset of register or port to Read
Type	=	Register type (STATUS_REG, INPUT_REG, FILTERED_REG, or OUTPUT_REG)

Function:

This routine reads and returns a byte value from a register. The XVME-244 manual contains all of the register definitions and corresponding addresses.

Return Value:

Byte value read from desired register

3.8.3 **Read a Word**

Syntax::

X244ReadWord(X244Base,StartReg,Type)

Where:

X244Base	=	Board address in VMEbus short I/O space
StartReg	=	Address offset of first register or port to Read
Type	=	Register type (INPUT_REG, FILTERED_REG, or OUTPUT_REG)

Function:

This routine reads and returns a word value from two consecutive registers. The XVME-244 manual contains all of the register definitions and corresponding addresses.

Return Value:

The word value read starting at the specified register

3.8.4 **Read All**

Syntax:

X244ReadAll(X244Base,Type)

Where:

X244Base	=	Board address in VMEbus short I/O space
Type	=	Register type (INPUT_REG, FILTERED_REG, or OUTPUT_REG)

Function:

This routine reads and returns a four byte (2 word) value from all four output status registers. The XVME-244 manual contains all of the register definitions and corresponding addresses.

Return Value:

The double word status of all four ports

3.8.5 **Write a Channel**

Syntax:

X244WriteChannel(X244Base,Channel,Data)

Where:

X244Base	=	Board address in VMEbus short I/O space
Channel	=	Channel to write status information (0-31)
Data	=	Information to configure output Channel
	0 =	Channel set low
	1 =	Channel set high

Function:

This routine sets a specified channel to the desired polarity

Return Value:

None

3.8.6 **Write a Byte**

Syntax:

X244Write(X244Base,Register,Type,Data)

Where:

X244Base	=	Board address in VMEbus short I/O space
Register	=	Address offset of register to write (0-3)
Type	=	Register type (STATUS_REG or OUTPUT_REG)
Data	=	The byte value to be written

Function:

This routine writes a given byte to a desired register or port. The XVME-244 manual contains all the register definitions and corresponding addresses.

Return Value:

None

3.8.7 **Write a Word**

Syntax:

X244WriteWord(X244Base,StartReg,WordData)

Where:

X244Base	=	Board address in VMEbus short I/O space
StartReg	=	Address offset of first register to write (0 or 2)
WordData	=	The word value to be written

Function:

This routine writes a given word (two bytes) to two consecutive output registers. The XVME-244 manual contains all of the register and port definitions and corresponding addresses.

Return Value:

None

3.8.8 **Write All**

Syntax:

X244WriteAll(X244Base,LongData)

Where:

X244Base	=	Board address in VMEbus short I/O space
LongData	=	The double word value to be written

Function:

This routine writes two words (four bytes) to all four output registers, configuring all 32 output channels.

Return Value:

None

3.8.9 **Reset**

Syntax:

X244Reset(X244Base)

Where:

X244Base	=	Board address in VMEbus short I/O space

Function:

This routine sets all of the output registers to 0.

Return Value:

None

3.9 XVME-260 DIGITAL OUTPUT MODULE

The XVME-260 is a Digital Relay Output Module that provides 32 channels of relay output, 300 VDC isolation, and transient suppression. The outputs are capable of switching either AC or DC loads. All outputs are software readable and disabled whenever the module is set to run diagnostics. All relays return to a Normally Open (NO) state when the module is reset for standard operation.

The parameters used in the routines for the XVME-260 must match the type expected by the routine. If you pass an invalid parameter, the routine will operate erratically.

Parameters:

ByteData	=	1 byte	Channel	=	1 byte
DWordData	=	4 bytes	Register	=	2 bytes
Start	=	2 bytes	WordData	=	2 bytes
X260Base	=	2 bytes			

Source Code Location: \LIB\SOURCES\DIGITAL\XVME260.C

The routines in the following sections are available to the user in the XVME-983 Software Support Library.

3.9.1 Read a Channel

Syntax:

ReadChannel(X260Base,Channel)

Where:

X260Base	=	Board address in VMEbus short I/O space
Channel	=	Channel containing status information (0-31)

Function:

This routine reads a specified channel and returns:

0	=	Channel relay is open
1	=	Channel relay is closed

Return Value:

0 or 1

3.9.2 **Read a Byte**

Syntax:

X260Read(X260Base,Register)

Where:

X260Base	=	Board address in VMEbus short I/O space
Register	=	Address offset of port register to Read

Function:

This routine reads and returns a byte value from a port register. The XVME-260 manual contains all of the register definitions and corresponding addresses.

> **NOTE**
> This routine will invert the bit read from the port register, showing the actual state of the port.

Return Value:

The inverted byte value read at the specified port register

3.9.3 **Read a Word**

Syntax:

X260ReadWord(X260Base,Start)

Where:

X260Base	=	Board address in VMEbus short I/O space
Start	=	Address offset of first register or port to Read

Function:

This routine reads and returns a word value from two consecutive port registers. The XVME-260 manual contains all of the register definitions and corresponding addresses.

NOTE
This routine will invert the bits read from the port register, showing the actual state of the port.

Return Value:
The inverted word value read starting at the specified port register

3.9.4 **Read All**

Syntax:
X260ReadAll(X260Base)

Where:
X260Base = Board address in VMEbus short I/O space

Function:
This routine reads and returns a four byte (2 word) value from all four port status registers. The XVME-260 manual contains all of the register definitions and corresponding addresses.

NOTE
This routine will invert the bits read from the port register, showing the actual state of the port.

Return Value:
The inverted double word value of all four port registers

3.9.5 **Write a Channel**

Syntax:

X260WriteChannel(X260Base,Channel,ByteData)

Where:

X260Base	=	Board address in VMEbus short I/O space
Channel	=	Channel to write status information
ByteData	=	Information to configure output channel (0-31)
		0 = Channel set low
		1 = Channel set high

Function:

This routine sets a specified channel relay to the desired position.

Return Value:

None

3.9.6 **Write a Byte**

Syntax:

X260Write(X260Base,Register,ByteData)

Where:

X260Base	=	Board address in VMEbus short I/O space
Register	=	Address offset of register or port to write
ByteData	=	The byte value to be written

Function:

This routine writes a given byte to a desired register or port. The XVME-260 manual contains all of the register and port definitions and corresponding addresses.

Return Value:

None

3.9.7 **Write a Word**

Syntax:

X260WriteWord(X260Base,Start,WordData)

Where:

X260Base	=	Board address in VMEbus short I/O space
Start	=	Address offset of first register or port to write
WordData	=	The word value to be written

Function:

This routine writes a given word (two bytes) to two consecutive registers or ports. The XVME-260 manual contains all of the register and port definitions and corresponding addresses.

Return Value:

None

3.9.8 **Write All**

Syntax

WriteAll(X260Base,DWordData)

Where:

X260Base	=	Board address in VMEbus short I/O space
DWordData	=	The double word value to be written

Function:

This routine writes two words (four bytes) to all four configuration ports, configuring all 32 output channels. The XVME-260 manual contains all of the register and port definitions and corresponding addresses.

Return Value:

None

4.1 INTRODUCTION

Table 4-1. Analog I/O Routine Matrix

	MODULE					
Routine	**500/595**	**505/595**	**530**	**540**	**560**	**566**
ADRead	•			•	•	
ChanOut			•	•		
Diag						
ForceAD	•			•	•	
Int	•			•	•	
Output		•		•	•	
Read	•			•	•	•
ReadGain	•			•		
Reset	•			•	•	•
SetCK						•
SetGain	•			•	•	
SetMode	•			•	•	
SetSampCK						•
Wait	•			•	•	•
WordWrite						•
Write	•			•	•	•
	500/595	**505/595**	**530**	**540**	**560**	**566**

4.2 XVME-500/590 ANALOG INPUT MODULE

The XVME-500 is a complete, single-high VMEbus compatible Analog Input Module with 16 analog input channels expandable to 32 channels. An programmable gain option is available. The XVME-590 is a 6U version of the XVME-500, with the I/O routed to the VMEbus P2 connector.

The parameters used in the routines for the XVME-500/590 must match the type expected by the routine. If you pass an invalid parameter, the routine will operate erratically.

Parameters:

ByteData	= 1 byte	Channel	= 1 byte
GainFactor	= 1 byte	IntFlag	= 1 byte
Mode	= 1 byte	Register	= 2 bytes
X500Base	= 2 bytes		

Source Code Location: \LIB\SOURCES\ANALOG\XVME500.C

The routines in the following sections are available to the user in the XVME-983 Software Support Library.

4.2.1 Read a Byte

Syntax:

X500Read(X500Base,Register)

Where:

X500Base = Board address in VMEbus short I/O space
Register = Address offset of register to Read

Function:

This routine reads and returns a byte from a desired register. The XVME-500/590 manual contains all of the register definitions and corresponding addresses.

Return Value:

The byte read from the desired register

4.2.2 **Write a Byte**

Syntax:

X500Write(X500Base,Register,ByteData)

Where:

X500Base	=	Board address in VMEbus short I/O space
Register	=	Address offset of register to write
ByteData	=	The byte value to be written

Function:

This routine writes a given byte to a desired register. The XVME-500/590 manual contains all of the register definitions and corresponding addresses.

Return Value:

None

Source Code\XVME500.C

4.2.3 **Wait**

Syntax:

X500Wait(X500Base)

Where:

X500Base	=	Board address in VMEbus short I/O space

Function:

This routine waits until the current analog-to-digital (A/D) conversion is completed before allowing the program to continue.

Return Value:

None

Source Code\XVME500.C

4.2.4 Force an A/D Conversion

Syntax:

X500ForceAD(X500Base)

Where:

X500Base = Board address in VMEbus short I/O space

Function:

This routine sets the A/D busy bit in the Status Control Register to initiate an analog-to-digital conversion on the currently selected channel.

Return Value:

None

4.2.5 Set Interrupt

Syntax:

X500Int(X500Base,IntFlag)

Where:

X500Base = Board address in VMEbus short I/O space
IntFlag = Interrupt status
0 = Interrupts disabled
1 = Interrupts enabled

Function:

This routine sets the interrupt bit in the Status Control Register to either enable or disable the interrupts.

Return Value:

None

4.2.6 **Set Conversion Mode**

Syntax:

X500SetMode(X500Base,Mode)

Where:

X500Base = Board address in VMEbus short I/O space
Mode = Conversion mode
0 = Set to Single Channel Mode
1 = Set to Sequential Channel Mode
2 = Set to Random Channel Mode
3 = Set to External Trigger Mode

Function:

This routine sets the module to one of four analog conversion modes.

Return Value:

None

4.2.7 **Reset**

Syntax:

X500Reset(X500Base)

Where:

X500Base = Board address in VMEbus short I/O space

Function:

This routine performs a software reset on the module. The A/D busy and the interrupt pending bits in the Status/Control Register are reset.

Return Value:

None

4.2.8 **Analog to Digital**

Syntax:

X500ADRead(X500Base)

Where:

X500Base = Board address in VMEbus short I/O space

Function:

This routine reads a word (two byte) value containing the A/D reading for a channel.

> **NOTE**
> If the module is in Single Channel or Sequential Mode, a new A/D conversion will be initiated every time this routine is called.

Return Value:

The word value containing the A/D reading for a channel

4.2.9 Set Gain Factor

> **NOTE**
> This routine is applicable to the XVME-500/590-2 and the XVME-500/590-3 **only**. The gain factor on the XVME-500/590-1 is **not** programmable.

Syntax:

X500SetGain(X500Base,Channel,GainFactor)

Where:

X500Base = Board address in VMEbus short I/O space
Channel = Target of new gain setting (0-31)
GainFactor = Gain factor applied to converted signal

	Range 1	**Range 2**	**Range 3**
0 = 1		4	10
1 = 2		8	20
2 = 5		20	50
3 = 10		40	100

> **NOTE**
> Select the range via jumpers on the board. Refer to the XVME-500/590 manual for details.

Function:

This routine sets the module to one of four gain factors in the current range setting.

Return Value:

None

4.2.10 **Read Gain Factor**

> **NOTE**
> This routine is applicable to XVME-500/590-2 and XVME-500/590-3 **only**. The gain factor on the XVME-500/590-1 is hardware selectable.

Syntax:

X500ReadGain(X500Base,Channel)

Where:

X500Base = Board address in VMEbus short I/O space
Channel = Channel number to read gain from (0-31)

> **NOTE**
> If the module is in Random Mode, an A/D conversion will be initiated for the channel whose gain is to be read.

Function:

This routine reads the module gain register and returns one of four factors in the current range setting.

Return Value:

A byte containing the gain mode value (0-3)

4.2.11 **Interrupt Example**

For your system to acknowledge interrupts generated by the board, you must set your jumpers and or switches as detailed below. Please refer to the XVME PC/AT manual and the specific Analog I/O manual for further details.

An example program for the use of interrupts can be found in the library. The file for the example is:

\EXAMPLES\ANALOG\X500INT.EXE

Jumpers:

XVME-500: J10-12 configured to enable one of the VMEbus interrupt levels.

4.3 XVME-505/595 4-CHANNEL ANALOG OUTPUT MODULE

The XVME-505 is a powerful, single-high VMEbus-compatible I/O module that can perform 12-bit digital-to-analog (D/A) conversions. The module can output D/A conversions over any of four independently configured output channels. The XVME-595 is functionally the same as the XVME-505 in a 6U form factor, with the I/O routed to the VMEbus P2 connector.

Unipolar ranges include 0-5 VDC and 0-10 VDC. Bipolar voltages include ±5 VDC and ±10 VDC, and the current range is 4-20 mA. All outputs go to 0 V (4 mA) state when the module is reset.

The parameters used in the routines for the XVME-505/595 must match the type expected by the routine. If you pass an invalid parameter, the routine will operate erratically.

Parameters:

Channel = 1 byte DacValue = 2 bytes
X505Base = 2 bytes

Source Code Location: LIB\SOURCES\ANALOG\XVME505.C

The following routine is available to the user in the XVME-983 Software Support Library.

4.3.1 Channel Output

Syntax:

X505Output(X505Base,Channel,DacValue)

Where:

X505Base = Board address in VMEbus short I/O space
Channel = Channel number whose output to program (0-3)
DacValue = 12 bit output value to write to channel output register

Function:

This routine writes a 12-bit value into a channel output register, causing a voltage output that corresponds to the 12-bit value.

Return Value:

None

4.4 XVME-530 ANALOG OUTPUT MODULE

The XVME-530 provides VMEbus systems with eight channels of digital-to-analog (D/A) conversion capability. The eight analog outputs are optically isolated from the VMEbus system. The resolution of each D/A conversion is 12 bits.

The parameters used in the routines for the XVME-530 must match the type expected by the routine. If you pass an invalid parameter, the routine will operate erratically.

Parameters:

ByteData	= 1 byte	Channel	= 1 byte
DacValue	= 2 bytes	Register	= 2 bytes
X530Base	= 2 bytes		

Source Code Location: \LIB\SOURCES\ANALOG\XVME530.C

The routines in the following sections are available to the user in the XVME-983 Software Support Library.

4.4.1 Read a Byte

Syntax:

X530Read(X530Base,Register)

Where:

X530Base = Board address in VMEbus short I/O space
Register= Address offset of register to Read

Function:

This routine reads and returns a byte from a desired register. The XVME-530 manual contains all of the register definitions and corresponding addresses.

Return Value:

The byte read from the desired register

4.4.2 **Write a Byte**

Syntax:

X530Write(X530Base,Register,ByteData)

Where:

X530Base = Board address in VMEbus short I/O space
Register= Address offset of register to write
ByteData = The byte value to be written

Function:

This routine writes a given byte to a desired register. The XVME-530 manual contains all of the register definitions and corresponding addresses.

Return Value:

None

4.4.3 **Reset**

Syntax:

X530Reset(X530Base)

Where:

X530Base = Board address in VMEbus short I/O space

Function:

This routine sets all eight channel outputs to 0 VDC (or 4 mA if in current mode) and turns the green LED to ON and the red LED to OFF.

Return Value:

None

4.4.4 **Wait**

Syntax:

X530Wait(X530Base)

Where:

X530Base = Board address in VMEbus short I/O space

Function:

This routine waits until the current D/A conversion is completed before allowing the program to continue.

Return Value:

None

4.4.5 **Channel Output**

Syntax:

X530ChanOut(X530Base,Channel,DacValue)

Where:

X530Base = Board address in VMEbus short I/O space
Channel= Which output channel to program (0-7)
DacValue = 12-bit value determining what voltage value will be set

Function:

This routine sets the output channel to a specified voltage value

Return Value:

None

4.5 XVME-540 ANALOG INPUT/OUTPUT MODULE

The XVME-540 is a powerful VMEbus-compatible I/O module that can perform 12-bit A/D and D/A conversions. The XVME-540 allows the dual capability of inputting A/D conversions on up to 32 different input channels and outputting D/A conversions over any of four different output channels. The input signal gain is programmable over three ranges, and all outputs can be in the form of either voltage or current.

The parameters used in the routines for the XVME-540 must match the type expected by the routine. If you pass an invalid parameter, the routine will operate erratically.

Parameters:

ByteData	= 1 byte	Channel	= 1 byte
DacValue	= 2 bytes	GainFactor	= 1 byte
IntFlag	= 1 byte	Mode	= 1 byte
Register	= 2 bytes	X540Base	= 2 bytes

Source Code Location: LIB\SOURCES\ANALOG\XVME540.C

The routines in the following sections are available to the user in the XVME-983 Software Support Library.

4.5.1 Read a Byte

Syntax:

X540Read(X540Base,Register)

Where:

X540Base = Board address in VMEbus short I/O space
Register = Address offset of register to Read

Function:

This routine reads and returns a byte from a desired register. The XVME-540 manual contains all of the register definitions and corresponding addresses.

Return Value:

The byte read from the desired register

4.5.2 **Write a Byte**

Syntax:

X540Write(X540Base,Register,ByteData)

Where:

X540Base	=	Board address in VMEbus short I/O space
Register	=	Address offset of register to Read
ByteData	=	The byte value to be written

Function:

This routine writes a given byte to a desired register. The XVME-540 manual contains all of the register definitions and corresponding addresses.

Return Value:

None

4.5.3 **Wait**

Syntax:

X540Wait(X540Base)

Where:

X540Base	=	Board address in VMEbus short I/O space

Function:

This routine waits until the current A/D conversion is completed before allowing the program to continue.

Return Value:

None

4.5.4 **Force an A/D Conversion**

Syntax:

X540ForceAD(X540Base)

Where:

X540Base = Board address in VMEbus short I/O space

Function:

This routine sets the A/D busy bit in the Status Control Register to initiate an A/D conversion on the present channel.

Return Value:

None

4.5.5 **Set Interrupt**

Syntax:

X540Int(X540Base,IntFlag)

Where:

X540Base = Board address in VMEbus short I/O space
IntFlag = Interrupt status
0 = Interrupts disabled
1 = Interrupts enabled

Function:

This routine sets the interrupt bit in the Status Control Register to enable or disable the interrupts.

Return Value:

None

4.5.6 Set Conversion Mode

Syntax:

X540SetMode(X540Base,Mode)

Where:

X540Base = Board address in VMEbus short I/O space
Mode = Interrupt status
0 = Set to Single Channel Mode
1 = Set to Sequential Channel Mode
2 = Set to Random Channel Mode
3 = Set to External Channel Mode

Function:

This routine sets the module to one of four analog conversion modes.

Return Value:

None

4.5.7 Reset

Syntax:

X540Reset(X540Base)

Where:

X540Base = Board address in VMEbus short I/O space

Function:

This routine performs a software reset on the module. It also resets the A/D busy bit and the interrupt pending bits in the Status/Control Register, turns the green LED to ON, and turns the red LED to OFF.

Return Value:

None

4.5.8 **Analog to Digital**

Syntax:

X540ADRead(X540Base)

Where:

X540Base = Board address in VMEbus short I/O space

Function:

This routine reads a word (two byte) value containing the A/D reading for a channel.

Return Value:

The word value containing the A/D reading for a channel

4.5.9 **Set Gain Factor**

Syntax:

X540SetGain(X540Base,Channel,GainFactor)

Where:

X540Base = Board address in VMEbus short I/O space
Channel = Target of new gain setting (0-31)
GainFactor = Gain factor applied to converted signal

Range 1	**Range 2**	**Range 3**
0 = 1	4	10
1 = 2	8	20
2 = 5	20	50
3 = 10	40	100

NOTE
Select the range via jumpers on the board. Refer to the XVME-540 manual for details.

Function:

This routine sets the module to one of four gain factors in the current range setting.

Return Value:

None

4.5.10 **Read Gain Factor**

Syntax:

X540ReadGain(X540Base,Channel)

Where:

X540Base = Board address in VMEbus short I/O space
Channel = Channel number to read gain from (0-31)

> **NOTE**
> An A/D conversion will be initiated for the channel whose gain is to be read.

Function:

This routine reads the module gain RAM and returns one of four factors in the current range setting.

Return Value:

A byte containing the gain mode value (0-3)

4.5.11 **Channel Output**

Syntax:

X540ChanOut(X540Base,Channel,DacValue)

Where:

X540Base = Board address in VMEbus short I/O space
Channel = Which output channel to program (0-3)
DacValue = 12-bit value to write to channel output register

Function:

This routine writes a 12-bit value into a channel output register, causing a voltage output that corresponds to the 12-bit value.

Return Value:

None

4.6 XVME-560 ANALOG INPUT/OUTPUT MODULE

The XVME-560 is a powerful VMEbus-compatible I/O module that can perform 12-bit A/D conversions. The XVME-560 lets you input A/D conversions on up to 64 different input channels with programmable gain.

The parameters used in the routines for the XVME-560 must match the type expected by the routine. If you pass an invalid parameter, the routine will operate erratically.

Parameters:

Channel	= 1 byte	Data	= 1 byte
GainFactor	= 1 byte	IntFlag	= 1 byte
Mode	= 1 byte	Register	= 2 bytes
X560Base	= 2 bytes		

Source Code Location: LIB\SOURCES\ANALOG\XVME560.C

The routines in the following sections are available to the user in the XVME-983 Software Support Library.

4.6.1 **Read a Byte**

Syntax:

X560Read(X560Base,Register)

Where:

X560Base	=	Board address in VMEbus short I/O space
Register	=	Address offset of register to Read

Function:

This routine reads and returns a byte from a desired register. The XVME-560 manual contains all of the register definitions and corresponding addresses.

Return Value:

The byte value read from the desired register

4.6.2 **Write a Byte**

Syntax:

X560Write(X560Base,Register,ByteData)

Where:

X560Base	=	Board address in VMEbus short I/O space
Register	=	Address offset of register to write
ByteData	=	The byte value to be written

Function:

This routine writes a given byte to a desired register. The XVME-560 manual contains all of the register definitions and corresponding addresses.

Return Value:

None

4.6.3 **Wait**

Syntax:

X560Wait(X560Base)

Where:

X560Base = Board address in VMEbus short I/O space

Function:

This routine waits until the current A/D conversion is completed before allowing the program to continue.

Return Value:

None

4.6.4 **Force an A/D Conversion**

Syntax:

X560ForceAD(X560Base)

Where:

X560Base = Board address in VMEbus short I/O space

Function:

This routine sets the A/D busy bit in the Status Control Register to initiate an A/D conversion on the present channel.

Return Value:

None

4.6.5 Set Interrupt

Syntax:

X560Int(X560Base,IntFlag)

Where:

X560Base	=	Board address in VMEbus short I/O space
IntFlag	=	Interrupt status 0 = Interrupts disabled 1 = Interrupts enabled

Function:

This routine sets the interrupt bit in the Status Control Register to enable or disable interrupts.

Return Value:

None

4.6.6 Set Conversion Mode

Syntax:

X560SetMode(X560Base,Mode)

Where:

X560Base	=	Board address in VMEbus short I/O space
Mode	=	Conversion mode 0 = Set to Single Channel Mode 1 = Set to Sequential Channel Mode 2 = Set to Random Channel Mode 3 = Set to External Trigger Mode

Function:

This routine sets the module to one of four analog conversion modes.

Return Value:

None

4.6.7 **Reset**

Syntax:

X560Reset(X560Base)

Where:

X560Base = Board address in VMEbus short I/O space

Function:

This routine performs a software reset on the module. It also resets the A/D busy bit and the interrupt pending bits in the Status/Control Register, turns the green LED to ON, and turns the red LED to OFF.

Return Value:

None

4.6.8 **Analog to Digital**

Syntax:

X560ADRead(X560Base)

Where:

X560Base = Board address in VMEbus short I/O space

Function:

This routine reads a word (two byte) value containing the A/D reading for a channel.

Return Value:

The word value containing the A/D reading for a channel

4.6.9 **Set Gain Factor**

Syntax:

X560SetGain(X560Base,Channel,GainFactor)

Where:

X560Base	=	Board address in VMEbus short I/O space
Channel	=	Target of new gain setting
GainFactor	=	Gain factor applied to converted signal
		0 = 1
		1 = 2
		2 = 4
		3 = 8

Function:

This routine sets the module to one of four gain factors.

Return Value:

None

4.6.10 **Interrupt Example**

For your system to acknowledge interrupts generated by the board, you must set your jumpers and/or switches as detailed below. Please refer to the XVME PC/AT manual and the specific Analog I/O manual for further details.

An example program for the use of interrupts can be found in the Library. The file for the example is:

\EXAMPLES\ANALOG\X560INT.EXE

Jumpers:

XVME-560: Switch bank 1 configured to enable one of the VMEbus interrupt levels.

4.7 XVME-566 HIGH-PERFORMANCE ANALOG INPUT MODULE

The XVME-566 is a high-performance VMEbus-compatible Analog Input Module. It converts data on 32 single-ended or 16 differential analog input channels and provides 12-bit resolution. These conversions are performed at a rate of 100 KHz using a dual sample and hold architecture. The module provides 64 Kbytes of dual-ported RAM. Gain RAM and a programmable sample clock are provided. Sample sequences can be triggered by a trigger clock, an external trigger, or a software trigger.

The parameters used in the routines for the XVME-566 must match the type expected by the routine. If you pass an invalid parameter, the routine will operate erratically.

Parameters:

ByteData	= 1 byte	ClockNum	= 1 byte
ControlMode	= 2 bytes	HoldReg	= 2 bytes
LoadReg	= 2 bytes	Period	= 2 bytes
Register	= 2 bytes	WordData	= 2 bytes
X566Base	= 2 bytes		

Source Code Location: LIB\SOURCES\ANALOG\XVME566.C

The routines in the following sections are available to the user in the XVME-983 Software Support Library.

4.7.1 Read a Byte

Syntax:

X566Read(X566Base,Register)

Where:

X566Base = Board address in VMEbus short I/O space
Register = Address offset of register to read

Function:

This routine reads and returns a byte from a desired register. The XVME-566 manual contains all of the register definitions and corresponding addresses.

Return Value:

The byte value read from the desired register

4.7.2 **Write a Byte**

Syntax:

X566Write(X566Base,Register,ByteData)

Where:

X566Base	=	Board address in VMEbus short I/O space
Register	=	Address offset of register to write
ByteData	=	The byte value to be written

Function:

This routine writes a given byte to a desired register. The XVME-566 manual contains all of the register definitions and corresponding addresses.

Return Value:

None

4.7.3 **Write a Word**

Syntax:

X566WordWrite(X566Base,Register,WordData)

Where:

X566Base	=	Board address in VMEbus short I/O space
Register	=	Address offset of register to Read
WordData	=	The word (two byte) value to be written

Function:

This routine writes a given word to a desired register. The XVME-566 manual contains all of the register definitions and corresponding addresses.

Return Value:

None

4.7.4 Set Sample Clock

Syntax:

X566SetSampCK(X566Base,Period)

Where:

X566Base	=	Board address in VMEbus short I/O space
Period	=	The period (usec) that determines the sampling frequency

Function:

This routine programs the sample clock (STC counter 4) period that controls the A/D conversion frequency.

CAUTION

The period must be $\geq$ 10 usec for 12-bit conversions and $\geq$ 7 usec for eight-bit conversions.

Return Value:

None

4.7.5 Set Clock

Syntax:

X566SetCK(X566Base,ClockNum,ControlMode,LoadReg,HoldReg)

Where:

X566Base	=	Board address in VMEbus short I/O space
ClockNum	=	Identifies which of the five counter channels to program
ControlMode	=	Word (two byte) value used to program the counter mode
LoadReg	=	Word (two byte) value to store in counter's load register
HoldReg	=	Word (two byte) value to store in counter's hold register

Function:

This routine programs one of five counters by writing the user-supplied Control Mode into the counter control register and by writing the desired values to the counter load and hold registers.

Return Value:

None

4.7.6 **Reset**

Syntax:

X566Reset(X566Base)

Where:

X566Base = Board address in VMEbus short I/O space

Function:

This routine performs a software reset on the module, resetting the sequence controller. This module is set to:

Continuous Mode
Sequential Mode
VMEbus Interrupts Disabled
Red LED to OFF
Green LED to ON
STC Channels 2, 4, 5 disabled

Return Value:

None

4.7.7 **Interrupt Example**

For your system to acknowledge interrupts generated by the board, you must set your jumpers and/or switches as detailed below. Please refer to the XVME PC/AT manual and the specific Analog I/O manual for further details.

An example program for the use of interrupts can be found in the Library. The file for the example is:

\EXAMPLES\ANALOG\X566INT.EXE

Jumpers:

XVME-566: Switch bank 1 configured to enable one of the VMEbus interrupt levels.

5.1 INTRODUCTION

Table 5-1. Counter Command Matrix

	MODULE	
Routine	**203/293**	**230**
BldCmdBlk		•
BldCmdBlkBuf		•
ExecCmd		•
IntInit	•	
Read	•	•
Reset	•	
Read	•	
SetCK	•	
Write	•	•
	203/293	**230**

5.2 XVME-203/293 COUNTER MODULE

The XVME-203 Counter Module is a single-high, VMEbus-compatible board using two AM9513A timer/counter devices to provide a total of ten 16-bit counting channels. The timer/counter devices are fully programmable and are capable of counting at a rate of up to 5 MHz. The XVME-203 can provide a complete interrupt structure via eight interrupt channels. This interrupt structure allows for the selection of a fixed interrupt vector for all eight channels or a separate vector for each individual channel.

The XVME-203 can also be used for quadrature detection. Advanced encoding circuitry permits the use of up to four quadrature transducers simultaneously.

The parameters used in the routines for the XVME-203/293 must match the type expected by the routine. If you pass an invalid parameter, the routine will operate erratically.

Parameters:

ClockNum	=	1 byte	CtrlReg	=	2 bytes
ByteData	=	1 byte	HoldReg	=	2 bytes
IntMask	=	1 byte	IREQVectors	=	Pointer to 1st byte of data buffer
LoadReg	=	2 bytes	Register	=	2 bytes
X203Base	=	2 bytes			

Source Code Location: \LIB\SOURCES\COUNTER\XVME203.C

The routines in the following sections are available to the user in the XVME-983 Software Support Library.

5.2.1 Read a Byte

Syntax:

X203Read(X203Base,Register)

Where:

X203Base = Board address in VMEbus short I/O space
Register = Address offset of register to Read

Function:

This routine reads and returns a byte from a desired register. The XVME-203 manual contains all of the register definitions and corresponding addresses.

Return Value:

The byte value read from the desired register

5.2.2 Write a Byte

Syntax:

X203Write(X203Base,Register,ByteData)

Where:

X203Base	=	Board address in VMEbus short I/O space
Register	=	Address offset of register to Write
ByteData	=	The byte value to be written

Function:

This routine writes a given byte to a desired register. The XVME-203 manual contains all of the register definitions and corresponding addresses.

Return Value:

None

5.2.3 **Reset**

Syntax:

X203Reset(X203Base)

Where:

X203Base	=	Board address in VMEbus short I/O space

Function:

This routine resets all counters on STC A and B, enables eight-bit access to counters, turns off quadrature detect circuitry, and resets the AM9519 interrupt controller

Return Value:

None

5.2.4 **Set Clock**

Syntax:

X203SetCK(X203Base,ClockNum,CtrlReg,LoadReg,HoldReg)

Where:

X203Base	=	Board address in VMEbus short I/O space
ClockNum	=	Identifies which of the clock channels to program (1-10)
CtrlReg =		Word (two byte) value used to program the counter mode
LoadReg	=	Word (two byte) value to store in the counter's load register
HoldReg	=	Word (two byte) value to store in the counter's hold register

Function:

This routine programs the one of ten counters by writing the user-supplied Control Mode into the

counter control register and writing the desired values to the counter load and hold registers.

Return Value:
None

5.2.5 **Initialize Interrupt**

Syntax:
X203IntInit(X203Base,IntMask,IREQVectors)

Where:

X203Base	= Board address in VMEbus short I/O space
IntMask=	The byte value that determines which interrupts are enabled
IREQVectors	= Array of eight 1 byte vectors corresponding to each interrupt line

Function:
This routine initializes the AM9519 interrupt controller as follows: mode register to individual vectors, fixed priority, GINT active high, IREQs active low, and chip armed. Also, this routine writes the IntMask value to the IMR and the IREQVectors array to the response memory.

Return Value:
None

5.2.6 **Interrupt Example**

For your system to acknowledge interrupts generated by the board, you must set your jumpers and/or switches as detailed below. Please refer to the XVME PC/AT manual and the XVME-203 manual for further details.

An example program for the use of interrupts can be found in the library. The file for the example is:

\EXAMPLES\COUNTER\X203INT.EXE

Jumpers:

XVME-203: Interrupts are enabled via the three least significant status control register bits in the software. See the XVME-203 manual for more information.

5.3 **XVME-230 INTELLIGENT COUNTER MODULE**

The XVME-230 Intelligent Counter Module is a VMEbus-compatible intelligent I/O Module with a variety of high-performance, high-level counting functions. The module's architecture features the Xycom 68000-based I/O kernel, which significantly enhances performance by relieving the host processor of many time-consuming functions.

The parameters used in the routines for the XVME-230 must match the type expected by the routine. If you pass an invalid parameter, the routine will operate erratically.

Parameters:

BlkOff	=	2 bytes	BufAddr	=	4 bytes
BufSize	=	2 bytes	ByteData	=	1 byte
Command	=	1 byte	ILevel	=	1 byte
IVect	=	1 byte	NextBlk	=	2 bytes
OpBuf	=	Pointer to 1st byte of data buffer	OpSize	=	1 byte
Register	=	2 bytes	X230Base	=	2 bytes

Source Code Location: LIB\SOURCES\COUNTER\XVME230.C

The routines in the following sections are available to the user in the XVME-983 Software Support Library.

5.3.1 **Read a Byte**

Syntax:

X230Read(X230Base,Register)

Where:

X230Base = Board address in VMEbus short I/O space
Register = Address offset of register to Read

Function:

This routine reads and returns a byte from a desired register. The XVME-230 manual contains all of the register definitions and corresponding addresses.

Return Value:

The byte read from the desired register

5.3.2 **Write a Byte**

Syntax:

X230Write(X230Base,Register,ByteData)

Where:

X230Base = Board address in VMEbus short I/O space
Register = Address offset of register to Write
ByteData = The byte value to be written

Function:

This routine writes the given byte to the desired register. The XVME-230 manual contains all of the register definitions and corresponding addresses.

Return Value:

None

5.3.3 **Build a Command Block**

Syntax:

X230BldCmdBlk(X230Base,BlkOff,Command,ILevel,IVect,NextBlk,OpSize,OpBuf)

Where:

X230Base = Board address in VMEbus short I/O space
BlkOff = Offset in short I/O memory Where block is to be built
Command = XVME-230 command to execute

ILevel	=	Interrupt level to use if interrupts enabled
IVect	=	Interrupt vector to use if interrupts enabled
NextBlk	=	Pointer to the next command block (Ø if none)
OpSize	=	Number of operands
OpBuf	=	Pointer to operand buffer to store within operand field

Function:

This routine builds a command block starting at the specified address, transfers the OpBuf contents to the operand field of this block, and sets the response flag to non-zero.

Return Value:

None

5.3.4 Build a Command Block with Operand Buffer

Syntax:

X230BldCmdBlkBuf(X230Base,BlkOff,Command,ILevel,IVect,NextBlk,BufSize,BufAddr)

Where:

X230Base	=	Board address in VMEbus short I/O space
BlkOff	=	Offset in short I/O memory Where block is to be built
Command	=	Command to execute
ILevel	=	Interrupt level to use if interrupts enabled
IVect	=	Interrupt vector to use if interrupts enabled
NextBlk	=	Pointer to the next command block
BufSize=		Size of operand buffer
BufAddr	=	Address of operand buffer to store within command block

Function:

This routine builds a command block with user-provided parameters starting at the specified address. It is used for commands with more than five bytes worth of operand data.

Return Value:

None

5.3.5 Execute Command Block

Syntax:

X230ExecCmd(X230Base,BlkAddr,Channel)

Where:

X230Base	=	Board address in VMEbus short I/O space
BlkAddr	=	Address of the command block to use
Channel	=	Channel to command

Function:
This routine sets up and executes a previously built command block.

Return Value:
None

5.3.6 **Interrupt Example**

For your system to acknowledge interrupts generated by the board, you must set your jumpers and/or switches as detailed below. Please refer to the XVME PC/AT manual and the XVME-230 manual for further details.

An example program for the use of interrupts can be found in the Library. The file for the example is:

\EXAMPLES\COUNTER\X230INT.EXE

This example starts a counter to count four seconds, then generates an interrupt, at which time the program terminates.

Jumpers:

XVME-230: User-programmable interrupt level and vector.

6.1 INTRODUCTION

This section describes the MS-DOS Communication Drivers included in the XVME-983 Software Support Library. This makes it easier for the XVME CPU to use the XVME-400, XVME-401, or XVME-428 as an additional serial I/O module.

The MS-DOS device drivers you need are in a file named **X40X.SYS** or **X42X.SYS**, depending on which module you use. This name is for identification only and the installed software requires a different naming convention. Details of the naming convention are found in Section 6.4.1.

The software is installed as part of the configuration information used by DOS and obtained from the DOS file **CONFIG.SYS**. Modifying this file is discussed in Section 6.4.

In order for the XVME CPU and the serial I/O module to communicate with each other, there must be some agreement as to the address modifiers that will be used. The pertinent settings are provided in the following sections. For further details on addressing and configuration, please consult the XVME-400/401/428 and XVME CPU manuals.

6.2 VMEbus BASE ADDRESS JUMPERS

Base address jumpers should be set to correspond to the base address field of the MS-DOS device command in the CONFIG.SYS file. The base address jumpers on the XVME-428 are J4, J5, J7, and J8. The base address jumpers on the XVME-400/401 are JA10 through JA15.

6.3 ADDRESS MODIFIER JUMPERS

Both the CPU and the Serial I/O Modules must be set to supervisor access mode (refer to the XVME CPU manual). If the XVME-400/401 is used, jumpers J1 and J7 will be removed. On the XVME-428, jumper J3 will be removed.

6.4 CONFIG.SYS

The CONFIG.SYS file on the system must be modified to contain the following line:

DEVICE=NAMEx.SYS bbbb,c,ss,p,d,q

Where:

NAMEx.SYS	=	File name
bbbb	=	Base address
c	=	Channel number
ss	=	Baud rate
p	=	Parity
d	=	Data bits
q	=	Stop bits

The various fields of this command line are explained in the following sections.

6.4.1 File Name Field

The field shown in the example above as **NAMEx.SYS** is the file name field. This is the actual name of the file that DOS will attempt to load as it reads the CONFIG.SYS file. The following rules govern the format of the file names. Be sure to copy the distribution file X42X.SYS or X40X.SYS to the new file name before editing the CONFIG.SYS file.

- The file name must be a text string followed by a one- or two-digit integer. Examples are **COM3.SYS**, **XPORT6.SYS**, and **KOM1.SYS**.

- Because of the way DOS treats device names such as **COM1** and **COM2**, it is necessary to provide alternate names for these installable files. If a file begins with the string **KOM**, it will be converted into the string **COM** and treated as a replacement for the standard DOS device.

- The name of the file, minus the .**SYS** extension, will be the name of the installed device.

- The trailing digits of the file name will be the **INT 14H** device number that the installed device driver will use. That means that the device driver named **KOM1.SYS** and installed as COM1 will respond to interrupts as device **0**, replacing the standard DOS COM1 service. If any name other than COM is used, the trailing digits are incremented in such a way that the standard DOS COM1 and COM2 drivers are unaffected and the new drivers are installed directly above them for INT 14H service. That means that the device driver installed as XPORT0 will respond to the device code of 2, leaving device codes 0 and 1 as they normally are.

6.4.2 Base Address Field (bbbb)

The base address corresponds directly to the base address that the XVME-42X or XVME-40X board is jumpered to respond to. It is necessary to tell the installable device driver the base address of the board, so that multiple boards may be installed in the system.

The 400/401 can be jumped from 0000H to FC00H on 1 Kbyte boundaries using jumpers JA10 through JA15. The 428 can be jumped from 0000H to 3C00H on 1 byte boundaries using jumpers J4, J5, J7, and J8.

6.4.3 Channel Number (c)

The XVME-400/401 each have four available serial data channels, while the XVME-428 has eight. This parameter specifies which data channel is to be associated with this configuration command. It is the responsibility of the user to make sure that each data channel is used only once. Channel numbers start at 0, so the XVME-428 will have channels 0 through 7.

6.4.4 Speed or Baud Rate (ss)

This parameter, and the following parameters, correspond directly to the parameters used in the DOS **MODE** command. For information about this command, see the DOS Commands section of the DOS manual.

This parameter specifies the baud rate at which this channel is initialized to run. The following baud rates are recognized: 110, 150, 300, 600, 1200, 2400, 4800, and 9600. Baud rates are represented by their first two digits only, so the values you can enter for this parameter are 11, 15, 30, 60, 12, 24, 48, and 96.

6.4.5 Parity (p)

This parameter may take the value **N** (none), **O** (odd) or **E** (even).

6.4.6 Databits (d)

This parameter may be **7** or **8**.

6.4.7 Stopbits (q)

This parameter may be **1** or **2**.

6.5 EXAMPLES

From the preceding, it can be seen that the following are examples of valid commands.

DEVICE=KOM1.SYS 3C00,0,9600,N,8,1

This command would cause the device driver to expect the board at short I/O address 3C00H. It would assign channel 0 on the board to the normal DOS COM1 functions and initialize the channel at 9600 baud, no parity, 8 data bits and 1 stop bit.

DEVICE=XPORT2.SYS 3000,2,4800,E,7,2

This command would assign channel 2 on the Serial I/O Module located at base address 3000H to the device XPORT2 using INT 14H device assignment 4. It would initialize the channel at 4800 baud, even parity, 7 data bits and 2 stop bits.

6.6 TROUBLESHOOTING

- Some versions of DOS do not make the distinction between a file named **COM1.SYS** and the **COM1:** device. Since DOS always has **COM1** and **COM2** loaded, you should use KOM1.SYS and KOM2.SYS so DOS can differentiate.

- In view of the above problem, **always** perform any file manipulation before editing the CONFIG.SYS file. Do not remove a previously used DEVICE line from the CONFIG.SYS file before making modifications.

- Each active I/O channel must have its own **DEVICE** line in the **CONFIG.SYS** file. Each **DEVICE** line must have its own **NAMEx.SYS** file to be loaded. Remember that each channel can only be used once.

7.1 INTRODUCTION

This chapter explains the following example programs available in the XVME-PC/AT software support library: Interrupt Service Routines to handle ABORT, ACFAIL, SYSFAIL, BERR, and WDT interrupts.

7.2 AUXILIARY NON-MASKABLE INTERRUPT

This sample program sets up an interrupt handler for NMI. After the handler is set up, it waits until the operator invokes an NMI by pressing the ABORT button, then the program ends. This program is in the ANMI.C file on your XVME-983 diskette.

```
<stdio.h>
#include <dos.h>
#include <xvmedefs.h>
#include <vmeext.h>

#define ABORT_BIT8
#define ACFAIL2
#define ANMI_ENABLE0x10
#define SYSFAIL4
#define TIMER_BIT0x80
#define WDT_ENABLE1
#define WDT_TRIG0x35

extern void Invoke_NMI();

ULONG OldNMI;
VOID (interrupt far *OldNMIHandler)();
UBYTE AbortFlag;
UBYTE FailFlag;
```

```
*****************************************************************************
*
************
************                    Function:     Anmi ISR
************                    Description:  Interrupt handler for Auxiliary NMI
************
*****************************************************************************
*
VOID interrupt  far  AnmiISR()
{
   UBYTE   Mask;
   UBYTE   Level;
   UBYTE   Status;

   Status = Inp(STATUS_1);

   if((Status & ABORT_BIT)) {
      AbortFlag++;
   }
   else if(Status & (ACFAIL | SYSFAIL)) {
      FailFlag++;
   }
   else {
     OldNMIHandler = (void (interrupt far *)())OldNMI;
     (*OldNMIHandler)();
   }
  }
```

```
*****************************************************************************
*
************
************                Function:       main
************                Description:    Sets up and enables the auxiliary NMI handler
************                                        and waits for the abort button to be presses
************
*****************************************************************************
*

main()
{
   UBYTE   Data;
   UBYTE   Status;

   AbortFlag = FailFlag = 0;
   printf("\nPush the abort button to exit");
   OldNMI = SetNMIVect((ULONG)AnmiISR);
   EnableNMIInt();
   while(!AbortFlag) {
      if(FailFlag) {
         printf("\n SYSFAIL or ACFAIL occurred");
         FailFlag = 0;
      }
   }
   DisableNMIInt();
   printf("\nAbort Button was pushed");
   SetNMIVect(OldNMI);
}
```

7.3 BERR

The program in this section sets up an interrupt handler for BERR. When the handler is set up, it tries to read data out on the VMEbus to an illegal address; this causes a bus error (BERR). The program will end when the bus error has been acknowledged. This program is in the BERR.C file on your XVME-983 diskette.

```
<stdio.h>
#include <dos.h>
#include <xvmedefs.h>
#include <vmeext.h>

#define BERR_BIT        1
#define BERR_RST        8
#define PORT_B          0x61

ULONG OldNMI;
VOID (interrupt far *OldNMIHandler)();
UBYTE BerrFlag=0;

*****************************************************************************
*
************
************            Function:    BerrISR
************            Description: Interrupt handler for Auxiliary NMI
************
*****************************************************************************
*
VOID  interrupt  far  BerrISR()
{
   UBYTE   Mask;
   UBYTE   Level;
   UBYTE   Status;

   Status = Inp(STATUS_1);

   if(Status & BERR_BIT) {
      BerrFlag = 1;
      Status = Inp(PORT_B);
      Outp(PORT_B, Status | BERR_RST);
      Outp(PORT_B, Status);
   }
   else {
     OldNMIHandler = (void (interrupt far *)())OldNMI;
     (*OldNMIHandler)();
   }
}
```

```
*****************************************************************************
*
************
************          Function:       main
************          Description:    Sets up and enables the Auxiliary NMI handler and
************                          then purposely generates a Bus Error on the VMEbus
*****************************************************************************
*
main()
{
   UBYTE   Data;
   UBYTE input[80];
   UBYTE *stopstr;

   printf("\n\nEnter the type of XVME CPU board (681,682,683,686): ");
   gets(input);
   switch((UINT)strtoul(input, &stopstr, 10)){
     case 681:
     case 682:
     case 683:
     case 686:
       InitLib((UINT)strtoul(input, &stopstr, 10));
       break;
     default:
       InitLib(682);
       break;
   }
   OldNMI = SetNMIVect((ULONG)BerrISR);
   EnableNMIInt();
   ReadVMEBusMemoryRM(&Data,TRANSFER8,0,1,SHORT_IO_ACCESS,0x500L,0);
   while(!BerrFlag) {
   ;}
   printf("\nBERR has occurred");
   DisableNMIInt();
   SetNMIVect(OldNMI);
}
```

7.4 WATCHDOG TIMER (WDT)

This example program sets up an interrupt handler for the watchdog timer, telling the timer to strobe until a key is hit. After a key is hit, it will wait until the timer "times-ou 25 times before ending the program. This program is in the WDTIMER.C file on your XVME-983 diskette.

```
<stdio.h>
#include <dos.h>
#include <xvmedefs.h>
#include <vmeext.h>

#define STATUS_2                0x33
#define TIMER_BIT               0x80
#define WDT_ENABLE              1
#define WDT_TRIG                0x35

UINT  far  *Head = (unsigned far *) 0x0000041A;
UINT  far  *Tail = (unsigned far *) 0x0000041C;

ULONG OldNMI;
VOID (interrupt far *OldNMIHandler)();
UBYTE TimerFlag = 0;
*****************************************************************************
*
************        Function:       TimerISR
************        Description:    Interrupt handler for watchdog timer.
*****************************************************************************
*
VOID  interrupt  far  TimerISR()
{
   UBYTE   Mask;
   UBYTE   Level;
   UBYTE   Status;

   DisableNMIInt();
   Status = inp(STATUS_2);

   if(!(Status & TIMER_BIT)) {
      TimerFlag++;
      ResetWDTimer();
   }
   else {
     OldNMIHandler = (void (interrupt far *)())OldNMI;
     (*OldNMIHandler)();
   }
   EnableNMIInt();
}
```

```
*****************************************************************************
*
************
************          Function:           main
************          Description:        Enables the watchdog timer and strobes it until a key
************                              is pressed.  At that time strobing will cease and the
************                              timer will be allowed to trigger 25 times before the
************                              program stops.  The interrupt handler will reset the
************                              timer after each trigger.
************
*****************************************************************************
*
main()
{
  UBYTE   Data;
  UBYTE   Status;

    printf("\nThe watchdog timer has been enabled and is being retriggered. Press");
    printf("\na key to quit retriggering the timer. The program will then stop after");
    printf("\nthe timer has triggered 25 times.\n");
    OldNMI = SetNMIVect((ULONG)TimerISR);
    ResetWDTimer();
    while(*Head == *Tail) {             /* While no keys are pressed */
        StrobeWDTimer();
    }
    while(TimerFlag < 25) {
    ;}
    DisableWDTimer();
    SetNMIVect(OldNMI);
    printf("\n25 watchdog timer interrupts have occurred since the key was pressed.");
    *Tail = *Head;                  /* Get rid of the key */
}
```

M

N

P

R

S

V

W

X