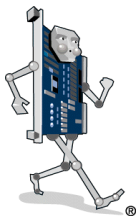
VMIOMAX-8451
Embedded PC-Based Controller

Product Manual

A GE Fanuc Company

12090 South Memorial Parkway
Huntsville, Alabama 35803-3308, USA
(256) 880-0444 ◆ (800) 322-3616 ◆ Fax: (256) 882-0859
500-318451-000 Rev. A

COPYRIGHT AND TRADEMARKS



(I/O man figure)

(IOWorks man figure)

Table of Contents

List of Figures

List of Tables

Overview

Introduction

VMIC's VMIOMAX-8451 is a low power, full-featured x86-based PC/104-*Plus* expansion (passively cooled) CPU utilizing the advanced technology of National Semiconductor's Geode GX1 processor. PC-Based Controller System uses standard communication software and hardware technologies. Instead of a dedicated, closed architecture implemented in PLCs, a PC-based controller distributes basic controller functions and responsibilities among standard PC "plug-and-play" components. The VMIOMAX-8451 allows the flexibility to upgrade your environment as well as mix-and-match products to use the most current and cost-effective technologies. See Figure 2 on page 20 for a functional block diagram of the VMIOMAX-8451.

The VMIOMAX-8451 provides features typically found on desktop systems such as:

- Up to 256 Mbyte PC-133 SDRAM
- Integrated 2D graphics (16 bpp at 1,280 x 1,024)
- On-board 10BaseT and 100BaseTx interfaces
- On-board Ultra DMA/33 IDE and floppy drive interfaces
- On-board CompactFlash socket (up to 512 Mbyte CompactFlash)
- Two high-performance 16550-compatible serial ports
- Two USB ports
- PS/2-style keyboard and mouse connectors
- PC/104-*Plus* expansion
- Passive heat sink

Real-time features include:

- Watchdog Timer
- Three 16-bit user programmable timers
- 32 Kbyte of nonvolatile SRAM
- Real-time clock

The VMIOMAX-8451 utilizes the Geode CS5530A I/O companion multi-function south bridge, designed to work in conjunction with the Geode GX1 processor. The CS5530A I/O companion is a PCI-to-ISA bridge, that provides AT/ISA style functionality. The VMIOMAX-8451 is capable of executing many of today's desktop operating systems such as Microsoft's Windows NT 4.0, Windows 2000 and a wide

variety of Linux based operating systems. The standard desktop features of the VMIOMAX-8451 are described in Chapter 2 of this manual.

Additionally, the VMIOMAX-8451 is capable of executing many of today's embedded operating systems such as VxWorks, Linux, and Microsoft's Windows NT/Windows 2000. The embedded features of the VMIOMAX-8451 are described in Chapter 3 of this manual.

The VMIOMAX-8451 is suitable for use in applications ranging from telecommunications, simulation, instrumentation, industrial control, process control and monitoring, factory automation, automated test systems, data acquisition systems and anywhere that the highest performance processing power is desired.

Additional Features

The additional features provided by the VMIOMAX-8451 are described below.

- **Pre-loaded with IOWorks.** You can order the VMIOMAX-8451 with an IOWorks system pre-loaded at the factory. The IOWorks PC-based control software provides the project management, programming environment and run-time tools necessary for PLC-like functionality.
- **Integrated solution and support.** You can purchase the target controller, software, remote I/O and other add-ons from a single vendor, VMIC.
- **Plug and play installation**. For an operational system, simply (1) connect the common PC support peripherals such as the keyboard and mouse; and (2) connect the controller to your local area network (LAN).
- **Standard network communication**. Ethernet and Fast Ethernet are the supported network topologies. Networked users, at all levels of organization in a plant, can get access to real-time data. In addition, data can be transferred between any connected PLCs and the controller simultaneously.
- **Hard real-time deterministic solutions**. Time-critical data exchange can be built with the Wind River® VxWorks operating system running on your controller.
- **Open IOMAX standard for remote I/O**. VMIC offers one of the widest selections of I/O boards in the industry. In addition, the VMIOMAX-8451 enables you to connect third-party I/O systems and other fieldbuses.
- **Flexible programming options**. Use Ladder Logic and Function Block IEC 61131-3 languages, or C\C++ languages to create control applications.

Basic Concepts

As a PC-based controller, the VMIOMAX-8451 can be used as a remote target controller in a distributed system. In a distributed system, a *host* CPU connects to your VMIOMAX-8451 CPU via Ethernet. A host CPU contains the program development tools and runs under the Windows NT 4.0 or Linux operating systems. The host encapsulates the utility functions of the controller as listed below.

- Provides the user interface for program development, configuration and control
- Contains compiler to build programs
- Supports multiple logical ports

Target controllers represent the run-time functionality of a controller to execute the control programs developed on the host. The *target controller* in a distributed system provides the program execution and run-time functionality. A remote target controller off-loads computing responsibilities from the host CPU.

A few of the remote target controller functions are listed below.

- Maintains the internal file system for target controller configuration, startup and recovery
- Manages real-time target controller data
- Enables on-line editing
- Schedules program execution for real-time control

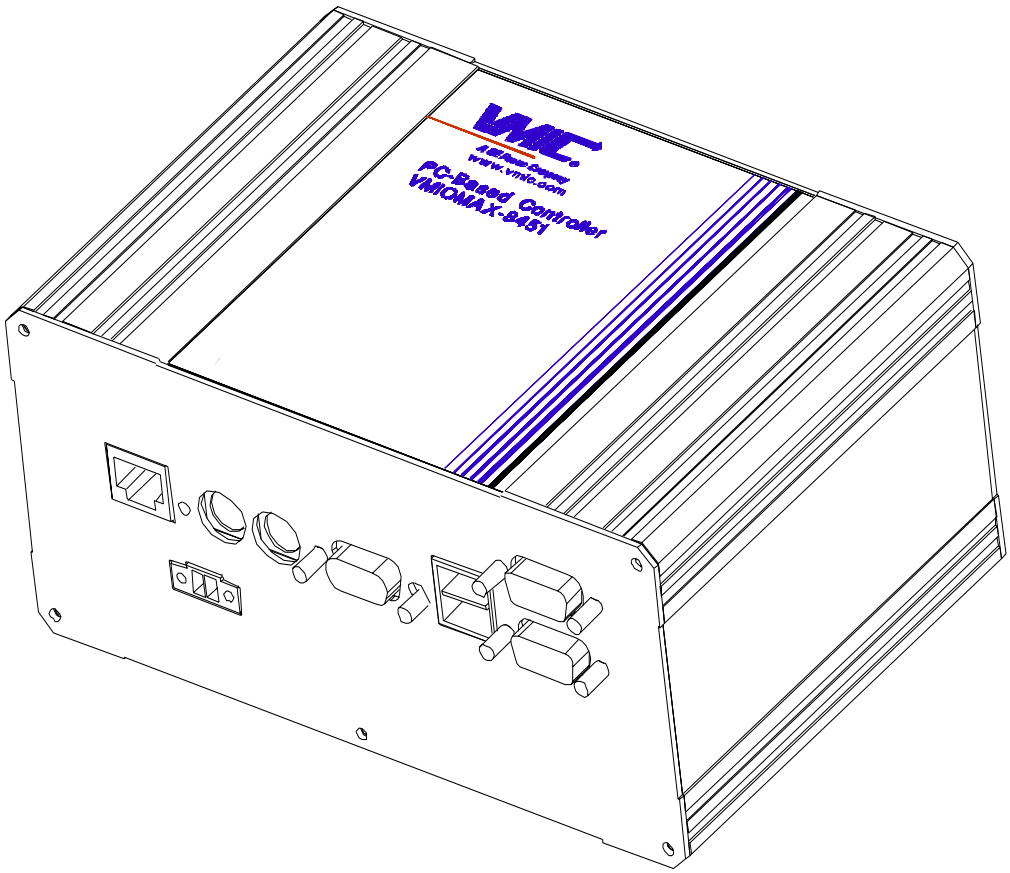


Figure 1 VMIOMAX-8451 Embedded PC

Geode GX1 Processor

The VMIOMAX-8451 utilizes the National Semiconductor Low Power x86-based Geode GX1 processor. The Geode GX1 processor, fabricated in 0.18 micron CMOS, is available with a typical average power consumption of 1.4 watts for the 333 MHz processor, making it the industry's lowest power consumption for an x86 processor at this performance level. Operating at lower voltages improves power consumption and thermal characteristics. The intelligent integration of the Geode GX1 processor features key system elements such as the video and memory controller embedded directly into the silicon.

Key Features of the Geode GX1 Processor

- Packaging:
  - 352-Terminal Ball Grid Array (BGA)
- 0.18-micron four-layer metal CMOS process
- Split-rail design
  - Available 2.2V core
  - 3.3 V I/O interface
- Fully static design
- Low typical power consumption
  - 1.4W @ 2.2V/333 MHz

NOTE: Typical power consumption is defined as an average, measured running browser at 80% active idle (Suspend-on-Halt) with a display resolution of 800 x 600 x 8 bpp @ 75 Hz.

- Speed: 333 MHz
- Unified memory architecture
  - Frame buffer and video memory reside in main memory
  - Minimizes PCB area requirements
  - Reduces system cost
- Compatible with multiple Geode I/O and graphics companion devices provided by National Semiconductor

32-Bit x86 Processor

- Supports the MMX™ instruction set extension for the acceleration of multimedia applications
- 16 Kbyte unified L1 cache
- 6-stage, pipelined integer unit
- Integrated Floating Point Unit (FPU)
- Memory Management Unit (MMU) adheres to standard paging mechanisms and optimizes code fetch performance:
  - Load-store reordering gives priority to memory reads
  - Memory-read bypassing eliminates unnecessary or redundant memory reads
- Re-entrant System Management Mode (SMM)

PCI Host Controller

- Synchronous to CPU core
- Allows external PCI master accesses to main memory concurrent with CPU accesses to L1 cache

Virtual Systems Architecture (VSA) Technology

- Innovative architecture allowing OS independent (software) virtualization of hardware functions
- Provides XpressGRAPHICS subsystem
  - High-performance legacy VGA core compatibility
  Note: GUI (Graphical User Interface) acceleration is pure hardware.

2D Graphics Accelerator

- Accelerates BitBLTs, line draw, text
  - Bresenham vector engine
- Supports all 256 Microsoft-defined Raster Operations (ROPs)
- Supports transparent BLTs and page flipping for DirectDraw®
- Runs at core clock frequency
- Full VGA and VESA mode support
- Special "driver level" instructions utilize internal scratchpad for enhanced performance

Display Controller

- Display Compression Technology (DCT) greatly reduces memory bandwidth consumption of display refresh
- Supports a separate video buffer and data path to enable video acceleration in Geode I/O
- Internal palette RAM for gamma correction
- Direct interface to Geode I/O for CRT support eliminating the need for an external RAMDAC
- Hardware cursor
- Supports up to 1280x1024 at 8 bpp and 1024x768 at 16 bpp

XpressROM Subsystem

- SDRAM interface tightly coupled to CPU core and graphics subsystem for maximum efficiency
- 64-Bit wide memory bus
- Support for:
  - One 144-pin unbuffered SODIMM (small outline dual inline memory module)
  - 16-byte reads (burst length of two)
  - Up to 256 Mbyte total memory supported

Geode CS5530A I/O Companion Multi-Function South Bridge

The VMIOMAX-8451 incorporates the Geode CS5530A I/O Multi-Function South Bridge. The CS5530A I/O companion is designed to work in conjunction with the Geode GX1 processor. The CS5530A I/O companion is a PCI-to-ISA bridge chipset that provides AT/ISA style functionality. The CS5530A provides a PCI bus interface that is both a slave for PCI cycles initiated by the CPU or other PCI bus masters. The chip is also a standard PCI master for the IDE controller.

The following functionality is provided in the CS5530A I/O companion:

- PCI bus master/slave interface
- ISA bus interface
- AT compatibility logic
- IDE controller
- Video display (includes MPEG accelerator)
- USB controller

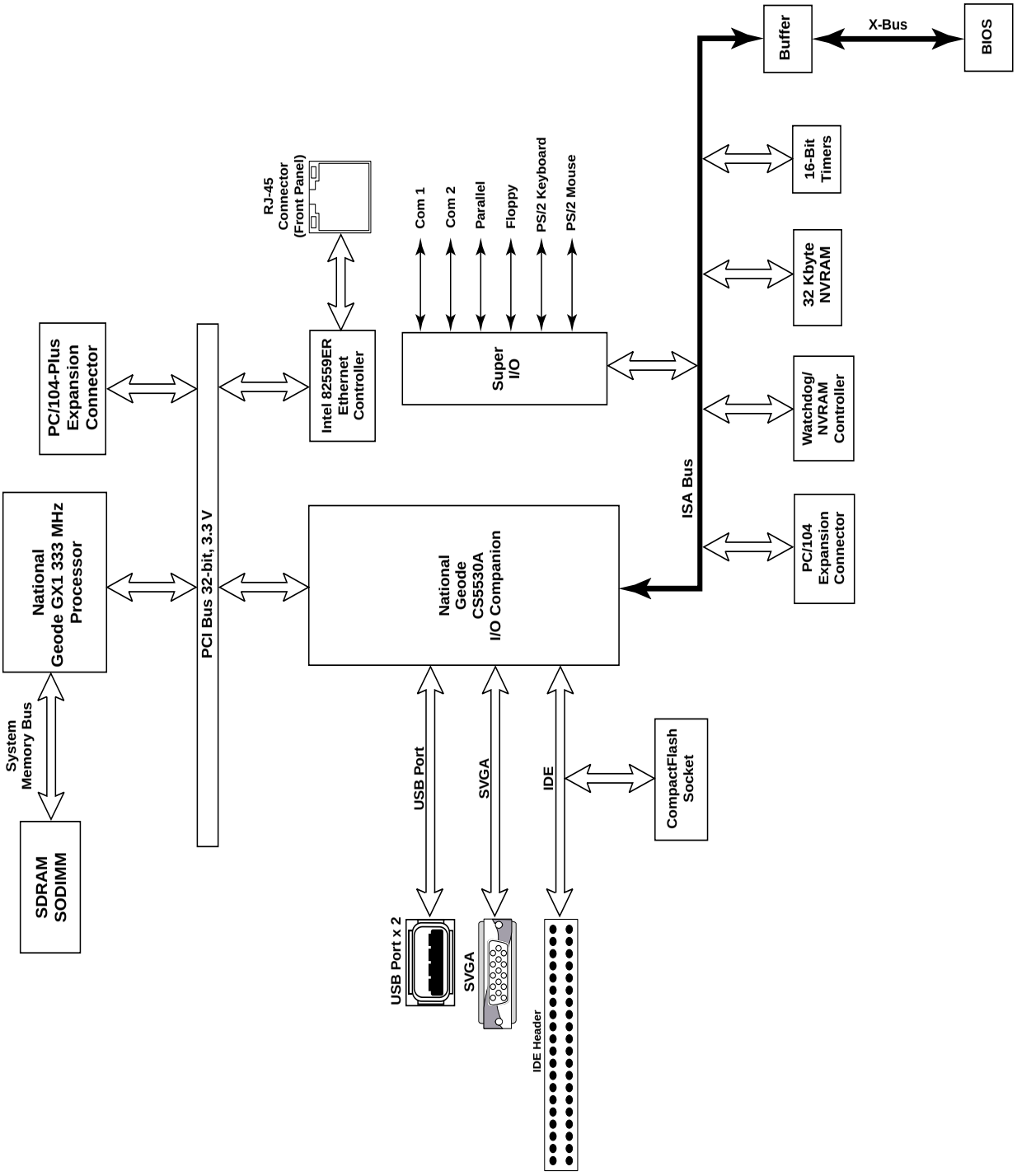


Figure 2 VMIOMAX-8451 Block Diagram

Organization of the Manual

This manual is composed of the following chapters and appendices:

Chapter 1 - Installation and Setup describes unpacking, inspection, hardware jumper settings, connector definitions, installation, system setup and operation of the VMIOMAX-8451.

Chapter 2 - Standard Features describes the unit design in terms of the standard PC memory and I/O maps, along with the standard interrupt architecture.

Chapter 3 - Embedded PC/RTOS Features describes the unit features that are beyond standard functions.

Chapter 4 - Maintenance provides information relative to the care and maintenance of the unit.

Appendix A - Connector Pinouts illustrates and defines the connectors included in the unit's I/O ports.

Appendix B - Software Installation provides details for installing drivers under Windows 2000, Windows NT and Linux 8.0.

Appendix C - Embedded Systems BIOS describes the menus and options associated with the Embedded (system) BIOS.

Appendix D - Sample C Software provides example code to use with the VMIOMAX-8451.

References

National Semiconductor Geode GX1 Processor

National Semiconductor
2900 Semiconductor Dr.
P.O. Box 58090
Santa Clara, CA
USA 95052-8090
(800) 272-9959
(800) 272-9959
www.national.com

National Semiconductor Geode CS5530A I/O Companion Multi-Function South Bridge

National Semiconductor
2900 Semiconductor Dr.
P.O. Box 58090
Santa Clara, CA 95052-8090
(800) 272-9959
(800) 737-7018 (FAX)
www.national.com

PCI Local Bus Specification, Rev. 2.1

PCI Special Interest Group
P.O. Box 14070
Portland, OR 97214
(800) 433-5177 (U.S.)
(503) 797-4207 (International)
(503) 234-6762 (FAX)

PC/104 Specification - contains the specifics on the PC/104. Available from:

PC/104 Embedded Consortium
1060 North Fourth St.
San Jose, CA 95112
(650) 903-8304
FAX: (408) 999-0344
E-mail info@pc104.org
Internet: www.pc104.org

Physical Description and Specifications

Refer to Product Specification, 800-318451-000 available from the following source:

VMIC
12090 South Memorial Parkway
Huntsville, Al 35803-3308, USA
(256) 880-0444
(800) 322-3616
FAX: (256) 882-0859
www.vmic.com

The following is useful information related to remote Ethernet booting of the VMIOMAX-8451:

Microsoft Windows NT Server Resource Kit
Microsoft Corporation
ISBN: 1-57231-344-7
www.microsoft.com

For more information on the Embedded Systems BIOS, refer to "*Embedded BIOS 2000*" Technical Reference Manual available from the following:

General Software, Inc.
11000 NE 33rd Place, Suite 102
Bellevue, Washington 98004
425-576-8300
800-850-5755
425-576-8334 fax
support@gensw.com
www.gensw.com

Safety Summary

The following general safety precautions must be observed during all phases of the operation, service and repair of this product. Failure to comply with these precautions or with specific warnings elsewhere in this manual violates safety standards of design, manufacture and intended use of this product.

VMIC assumes no liability for the customer's failure to comply with these requirements.

Ground the System

To minimize shock hazard, the chassis and system cabinet must be connected to an electrical ground. A three-conductor AC power cable should be used. The power cable must either be plugged into an approved three-contact electrical outlet or used with a three-contact to two-contact adapter with the grounding wire (green) firmly connected to an electrical ground (safety ground) at the power outlet.

Do Not Operate in an Explosive Atmosphere

Do not operate the system in the presence of flammable gases or fumes. Operation of any electrical system in such an environment constitutes a definite safety hazard.

Keep Away from Live Circuits

Operating personnel must not remove product covers. Component replacement and internal adjustments must be made by qualified maintenance personnel. Do not replace components with power cable connected. Under certain conditions, dangerous voltages may exist even with the power cable removed. To avoid injuries, always disconnect power and discharge circuits before touching them.

Do Not Service or Adjust Alone

Do not attempt internal service or adjustment unless another person capable of rendering first aid and resuscitation is present.

Do Not Substitute Parts or Modify System

Because of the danger of introducing additional hazards, do not install substitute parts or perform any unauthorized modification to the product. Return the product to VMIC for service and repair to ensure that safety features are maintained.

Dangerous Procedure Warnings

Warnings, such as the example below, precede only potentially dangerous procedures throughout this manual. Instructions contained in the warnings must be followed.

WARNING: Dangerous voltages, capable of causing death, are present in this system. Use extreme caution when handling, testing and adjusting.

Warnings, Cautions and Notes

STOP informs the operator that a practice or procedure should not be performed. Actions could result in injury or death to personnel, or could result in damage to or destruction of part or all of the system.

WARNING denotes a hazard. It calls attention to a procedure, practice or condition, which, if not correctly performed or adhered to, could result in injury or death to personnel.

CAUTION denotes a hazard. It calls attention to an operating procedure, practice or condition, which, if not correctly performed or adhered to, could result in damage to or destruction of part or all of the system.

NOTE denotes important information. It calls attention to a procedure, practice or condition which is essential to highlight.

CHAPTER
1

Installation and Setup

Contents

Introduction

This chapter describes the hardware jumper settings, connector descriptions, installation, system setup and operation of the VMIOMAX-8451.

Unpacking Procedures

Any precautions found in the shipping container should be observed. All items should be carefully unpacked and thoroughly inspected for damage that might have occurred during shipment. The board(s) should be checked for broken components, damaged printed circuit board(s), heat damage and other visible contamination. All claims arising from shipping damage should be filed with the carrier and a complete report sent to VMIC Customer Service along with a request for advice concerning the disposition of the damaged item(s).

CAUTION: Components installed on VMIC's products are sensitive to electrostatic discharge and damage will occur on boards that are subjected to a high energy electrostatic field. When the board is placed on a bench for configuring, etc., it is suggested that conductive material be inserted under the board to provide a conductive shunt. Unused boards should be stored in the same protective boxes in which they were shipped.

Hardware Setup

The VMIOMAX-8451 is factory populated with user-specified options as part of the VMIOMAX-8451 ordering information. The memory size and storage media size are not user-upgradable. To change memory size or storage media size, contact customer service to receive a Return Material Authorization (RMA).

VMIC Customer Care is available at: 1-800-240-7782.

Or E-mail us at customer.service@vmic.com

The VMIOMAX-8451 is tested for system operation before being shipped from the factory. The physical location of the front panel connectors for the single board CPU are illustrated in Figure 1-1 on page 30. The definitions of the CPU board connectors are included in Table 1-1 on page 31.

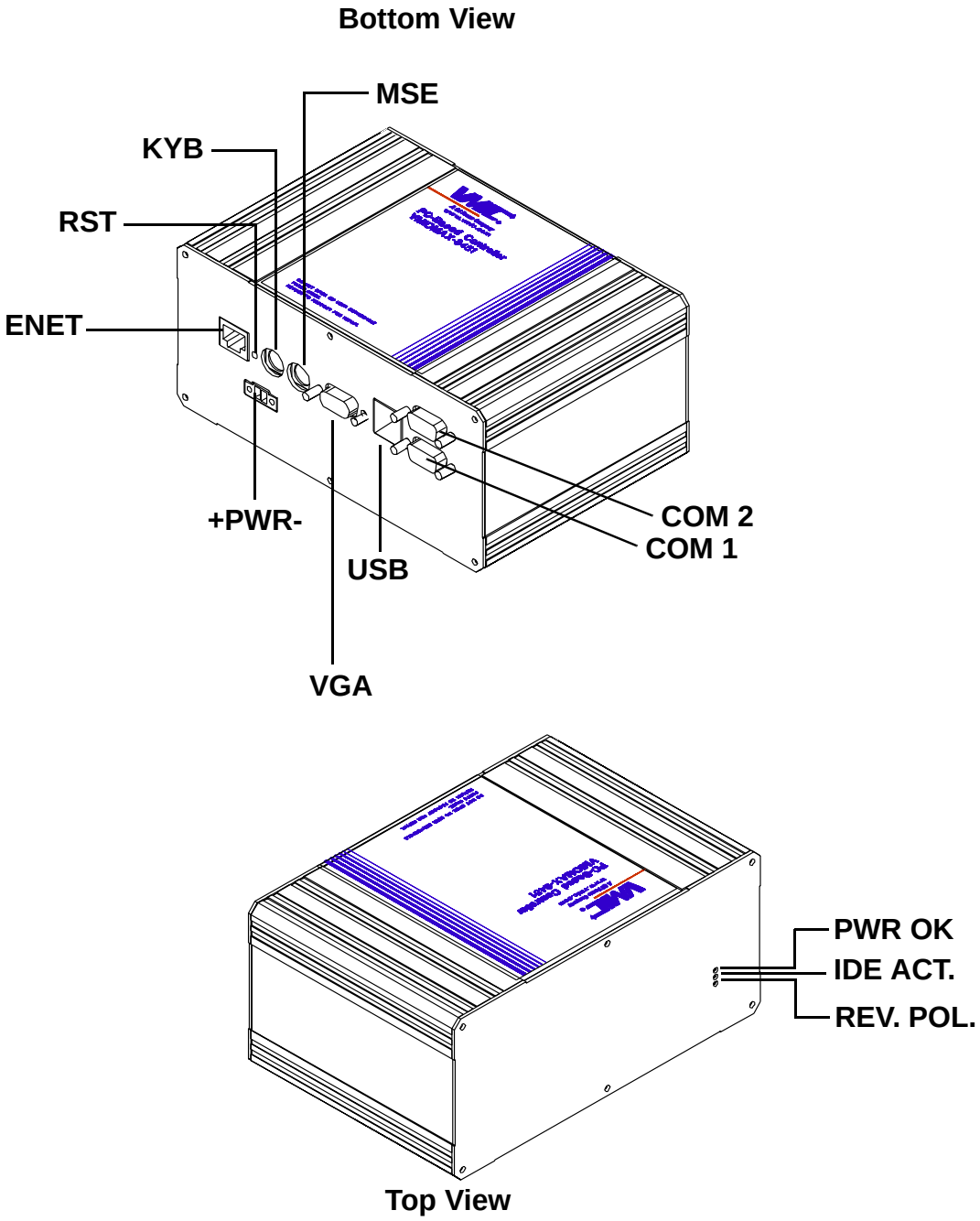


Figure 1-1 VMIOMAX-8451 Embedded PC-Based Controller

Front Panel Connectors

The VMIOMAX-8451 provides front-panel access for the SVGA connector, the 10/100 Ethernet connector, USB connector, two serial ports, reset switch and the mouse and keyboard connectors. A drawing of the VMIOMAX-8451 front-panel is shown in Figure 1-1 on page 30. The front-panel connectors are labeled as follows:

- ENET 10/100 Mbit Ethernet connector
- VGA SVGA video connector
- RST Manual reset switch
- MSE PS/2 mouse connector
- KYB PS/2 keyboard connector
- USB Dual USB connector (two USB ports)
- COM1 Serial port 1
- COM2 Serial port 2
- +PWR- Input power connector

The front panel connectors, including connector pinouts and orientation, for the VMIOMAX-8451 are defined in Appendix A. Table 1-1 below lists the board connectors and functions.

Table 1-1 CPU Board Connectors

Connector	Function
J1	2.5 inch Hard Drive Connector
E3	CMOS Clear
E5	Power Connector (Front Panel)
E6	IDE (PRI)
E7	Floppy Drive
E8	Parallel Port
E9	LED Output
J2	Dual USB
J3	Keyboard
J4	Mouse
J5	PC/104-Plus
J6	PC/104
J8	Dual SCSI
J9	Ethernet
P4, E10	COM 1, COM 2

Power Requirements

Input voltage range: 8 to 28 VDC
Output voltage/current: +5 VDC with 5 A (Supply Capability)

Installation

Connect all needed peripherals to the front panel. Each connector's pinouts are detailed in Appendix A. Minimally, a keyboard and a monitor are required if the user has not previously configured the system.

1. Connect the 8 to 28 volt VDC power source to the front panel of the VMIOMAX-8451.
2. Several messages are displayed on the screen, including names, versions and copyright dates for the various BIOS modules on the VMIOMAX-8451.
3. The VMIOMAX-8451 features a Flash Disk resident on the board. Refer to *Local IDE Drive* on page 83 for set up details.
4. If an IDE drive is installed, the BIOS Setup program must be run to configure the drive types. See *Embedded Systems BIOS* on page 103 to properly configure the system.
5. If a drive is present, install the operating system according to the manufacturer's instructions.

See Appendix B for instructions on installing VMIOMAX-8451 peripheral driver software during operating system installation.

BIOS Setup

The VMIOMAX-8451 has an on-board BIOS Setup program that controls many configuration options. These options are saved in a special non-volatile memory chip and are collectively referred to as the board's 'CMOS Configuration'. The CMOS configuration controls many details concerning the behavior of the hardware from the moment power is applied.

The VMIOMAX-8451 is shipped from the factory with hard drive type configuration set to AUTO in the CMOS.

Details of the VMIOMAX-8451 BIOS setup program are included in Appendix C.

CMOS Configuration

To clear the CMOS:

1. Turn off power to the unit.
2. Place a jumper on E3.
3. Power up the unit.
4. Power down the unit and remove the jumper from E3.

When power is reapplied to the unit, the CMOS password will be cleared.

Master/Slave Configuration

The hard drive for the VMIOMAX-8451 can be configured as either master or slave. There are two jumpers that are used: E4 is for the 2.5 inch IDE drive and E7 for the 1.8 inch drive. The following procedure is used to configure the drive.

Master/Slave Configuration Procedure

Use the following procedure for instructions on opening and closing the VMIOMAX-8451.

1. Turn off power and remove the AC power adapter from the front panel power receptacle. Disconnect all cables connected to the front panel (i.e., monitor, keyboard and mouse). Using a #1 Phillips head screwdriver, remove the six screws holding the front panel to the case. Remove four screws on the bottom of the case that hold the boards inside the case. Ensure that the standoffs remain on the board.

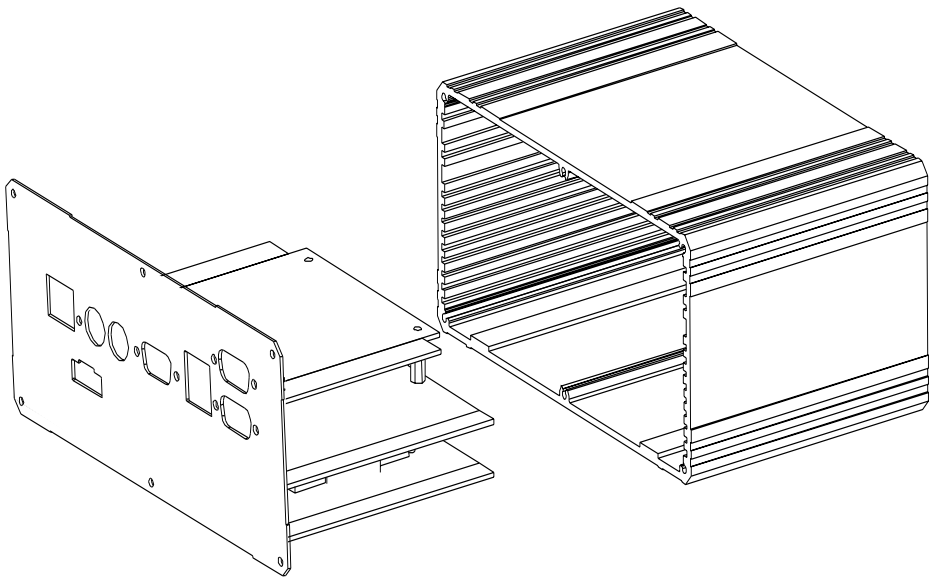


Figure 1-2 Removing the VMIOMAX from the Case

2. Remove the four standoffs from the top board.
3. Gently pry the top board away from the bottom board, being careful not to bend any of the pins on the three connectors.

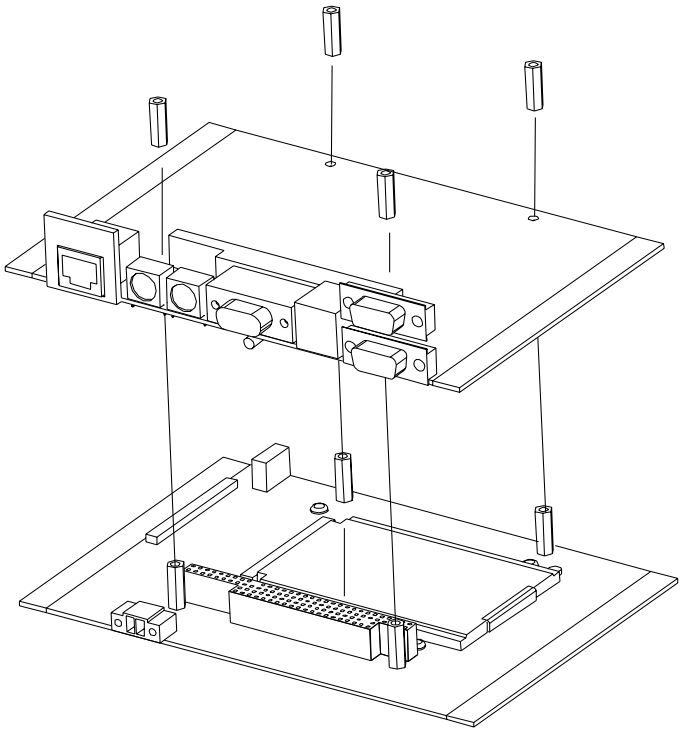


Figure 1-3 Removing Standoffs and Disconnecting the Top and Bottom Boards

4. Place the VMIOMAX-8451 on a non-conductive surface. Check the type of drive installed. If the VMIOMAX-8451 has a 2.5” IDE drive (connected to J1), jumper E4 is used to configure master or slave. If the VMIOMAX-8451 is equipped with a 1.8” (E7) or a 2.5” (E4) IDE drive, refer to Table 1-2 below.
5. Locate the jumper depending on drive type (see Figure 1-4 on page 35 for location of jumpers). Use Table 1-2 below to configure the VMIOMAX-8451 for the desired mode.

Table 1-2 Master/Slave Configuration (Jumper E4 and E7)

Pins	Description
NO Jumper	Master
2 and 4	Slave on Fujitsu
1 and 2	Slave on IBM and Seagate
3 and 4	Slave on Toshiba

NOTE: When configuring the VMIOMAX-8451 for master / slave mode, the drive manufacturer will determine the jumper settings (i.e., IBM, Seagate or Toshiba).

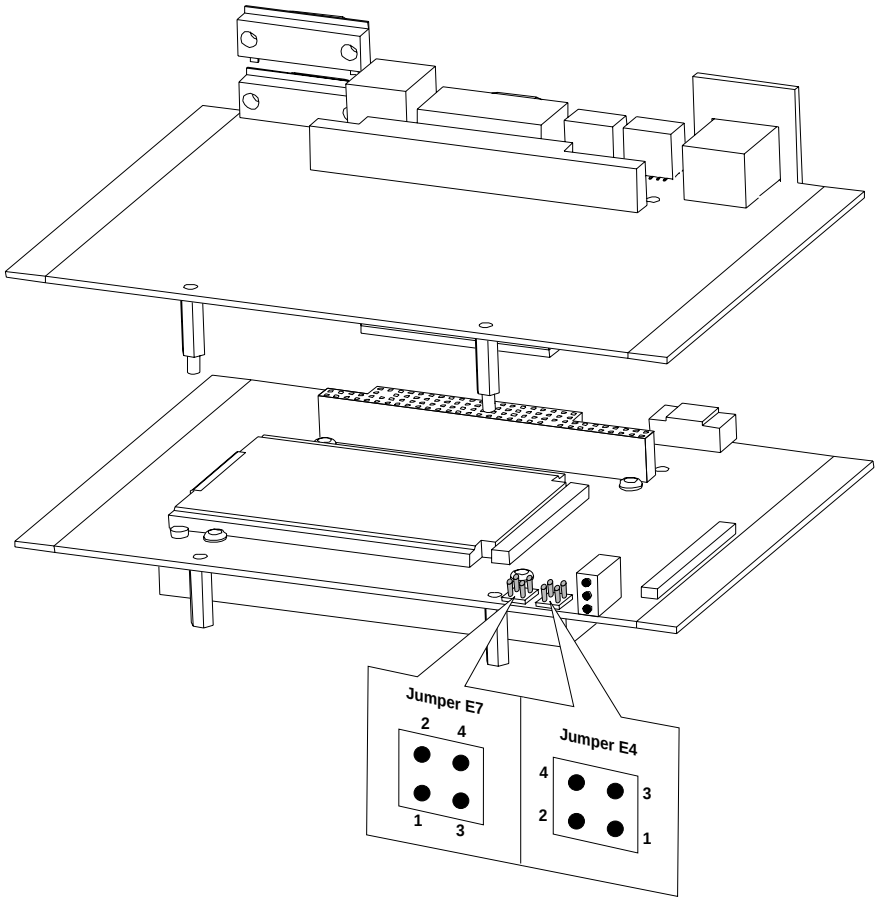


Figure 1-4 Location of User Configurable Jumpers (E4 and E7)

6. When the VMIOMAX-8451 has been jumpered for the desired mode, mount the top board on the bottom board and replace the standoffs on the top board. Place the VMIOMAX-8451 back in the case and secure it using the screws removed in step 1. Also re-connect all cables that were removed in step 1.

CAUTION: Do not over tighten the four bottom screws that hold the standoffs in place.

7. Setup is finished and the VMIOMAX-8451 can be powered up.

After configuring the drive for either master or slave, you may need to boot up the VMIOMAX-8451 and proceed to the BIOS to ensure the correct setup for your particular need. See “Embedded Systems BIOS” on page 103.

Parallel Port (LPT1) Configuration and Setup

You can connect a device to the parallel port using a 26-pin to 25-pin adapter cable (supplied with the VMIOMAX-8451). The top board of the VMIOMAX-8451 has a 26-pin header (E8). Using header E8 and the adapter cable, parallel port signals are routed externally. Any device that is normally connected to a parallel port can be connected to this cable. The VMIOMAX-8451 has to be removed from the case in order to connect the adapter cable to the header.

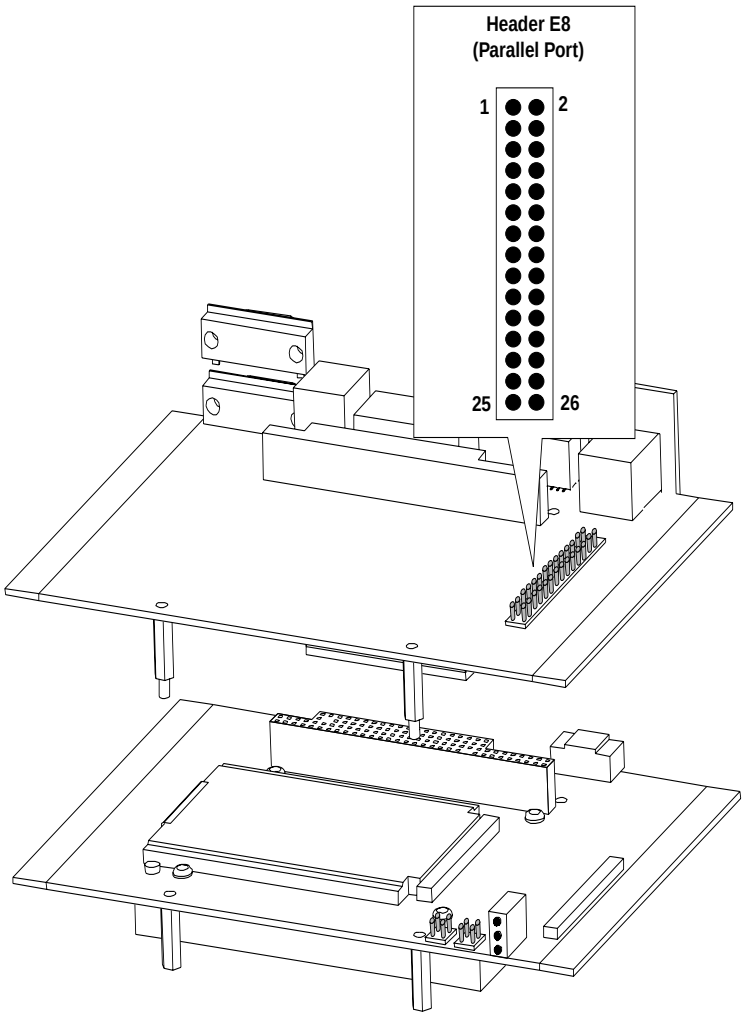


Figure 1-5 Location of Header E8 (Parallel Port Header)

Parallel Port Adapter Cable Installation Procedure:

1. Turn off power and remove the AC power adapter from the front panel power

receptacle. Disconnect all cables connected to the front panel (i.e., monitor, keyboard and mouse). Using a #1 Phillips head screwdriver remove the six screws holding the front panel to the case. Remove the four screws on the bottom of the case that hold the boards inside the case.

WARNING: Ensure that the standoffs remain on the board.

2. Remove the VMIOMAX-8451 from the case. See Figure 1-5 on page 36 for location on Header.
3. Place the VMIOMAX-8451 on a non-conductive surface. Re-connect the keyboard and mouse. Connect the adapter cable to header E8 as shown in Figure 1-6 below.
4. Connect the parallel port device. Connect the 8 to 28 VDC power source to the front panel.
5. Installation is complete.

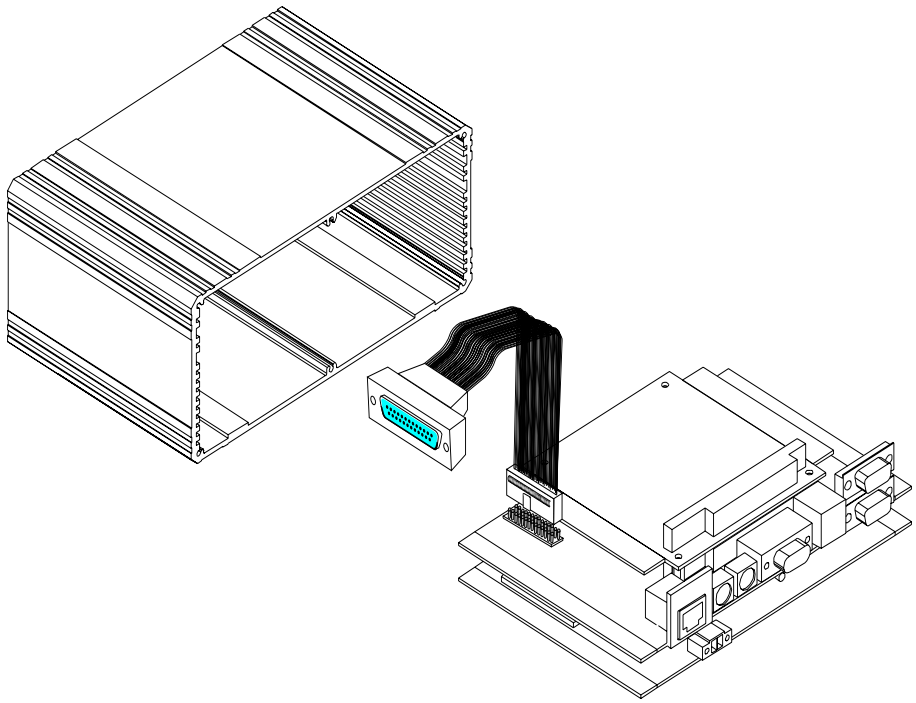


Figure 1-6 Installation of the Parallel Port Adapter Cable

After installing the parallel port adapter and cabling the device, you will need to boot up the VMIOMAX-8451 and proceed to the BIOS. See “Embedded Systems BIOS” on page 103. Also, you may need to load the driver for that particular device.

Installing a Floppy Disk Drive

The VMIOMAX-8451 allows for the connection of a floppy disk drive, using an on-board header (E7). An industry standard 34-pin ribbon cable is needed to interconnect with the floppy drive (not supplied by VMIC). When installing the 34-pin ribbon cable ensure that pin 1 of the cable mates up to pin 1 of the on-board header. The strip on the ribbon cable denotes pin 1. To install the ribbon cable, remove the VMIOMAX-8451 from the case and attach the cable. See Figure 1-7 below for an illustration of the header and cable.

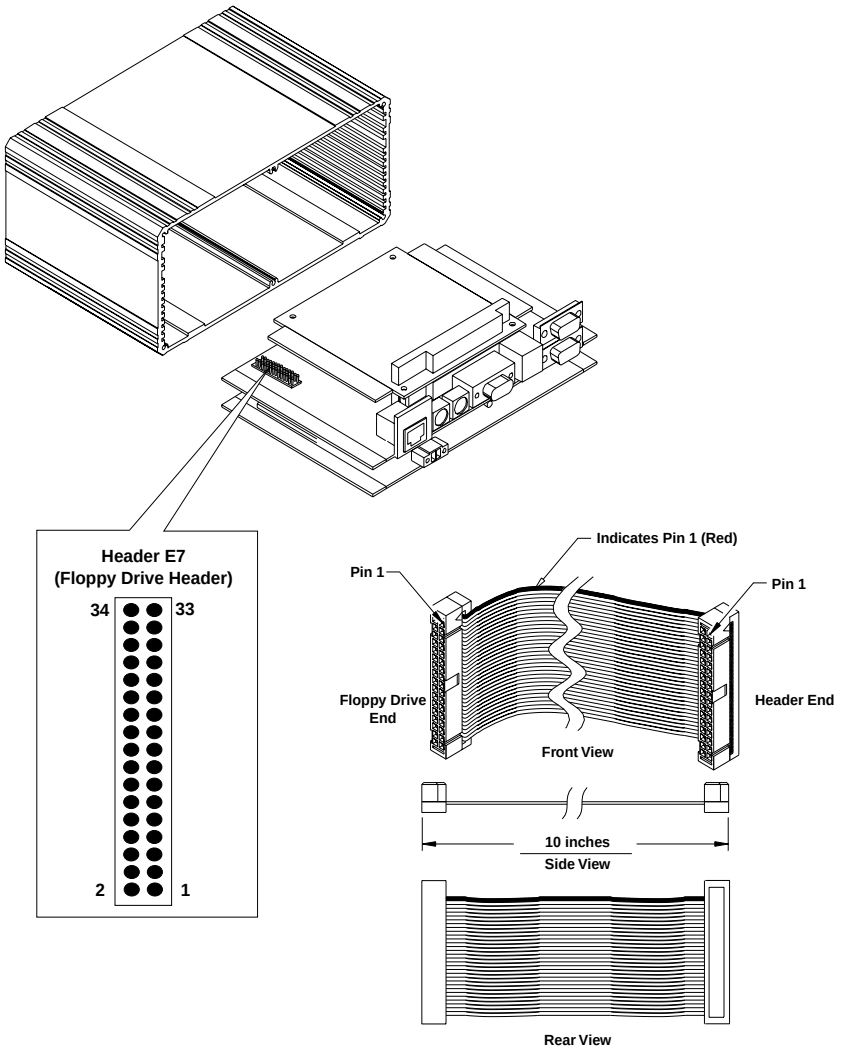


Figure 1-7 Location of Floppy Drive Header and Ribbon Cable

Installing a CD-ROM Drive

The VMIOMAX-8451 uses an IDE adapter (Figure 1-8) to connect to an on-board header (E6) allowing for the installation of a CD-ROM drive. When installing the IDE adapter, ensure that correct pin assignments are observed (i.e., pin 1 on the adapter to pin 1 on the board).

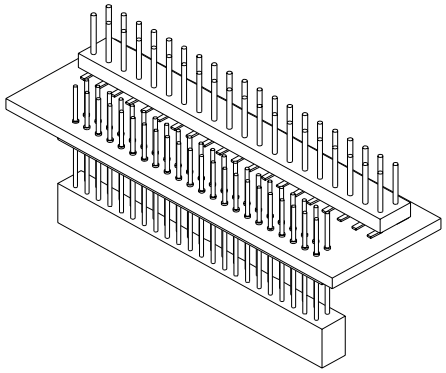


Figure 1-8 VMIOMAX-8451 IDE Adapter

IDE Adapter Installation Procedure:

1. Turn off power and remove the AC power adapter from the front panel power receptacle. Disconnect all cables connected to the front panel (i.e., monitor, keyboard and mouse). Using a #1 Phillips head screwdriver remove the six screws holding the front panel to the case. Remove the four screws on the bottom of the case that hold the boards inside the case. Ensure that the standoffs remain on the board.
2. Remove the VMIOMAX-8451 from the case. See Figure 1-9 on page 40 for location on Header.
3. Place the VMIOMAX-8451 on a non-conductive surface. Re-connect the keyboard and mouse. Connect the IDE adapter to header E6 as shown in Figure 1-9 on page 40. Ensure that pin 1 on the IDE adapter and pin 1 on header E6 mate up. The IDE adapter has a silk screen on the PCB showing the location of pin 1.
4. Using a ribbon cable, connect the CD-ROM to the IDE adapter. Connect the AC power adapter to the front panel. Ensure that pin 1 on the IDE adapter, CD-ROM and the cable, which normally has a strip indicating pin 1, mate up.
5. Installation is complete.

After installing the IDE adapter and cabling the CD-ROM, you will need to boot up the VMIOMAX-8451 and proceed to the BIOS. See “Embedded Systems BIOS” on page 103. Also, you may need to load the driver for that particular device.

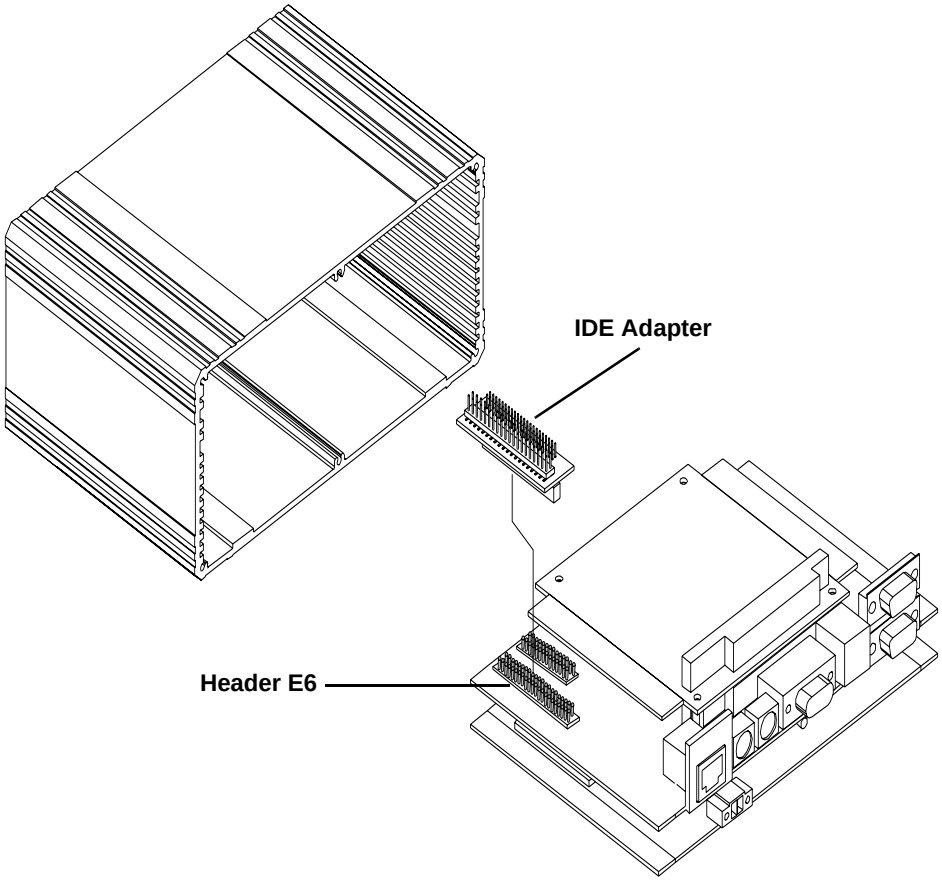


Figure 1-9 Location of Header E6 and the IDE Adapter Installation

Installing PC/104 and PC/104-*Plus* Cards

The VMIOMAX-8451's modular, scalable architecture provides easy system expansion via PC / 104-*Plus*. This expansion capability allows third-party devices to be used with the VMIOMAX-8451. The device is installed on two connectors on the top board depending on whether it's a PC / 104 or PC / 104-Plus. Connector J5 is the PC / 104-Plus and J6 is the PC / 104 connector. PC / 104-Plus cards will use both the J5 and J6 connectors.

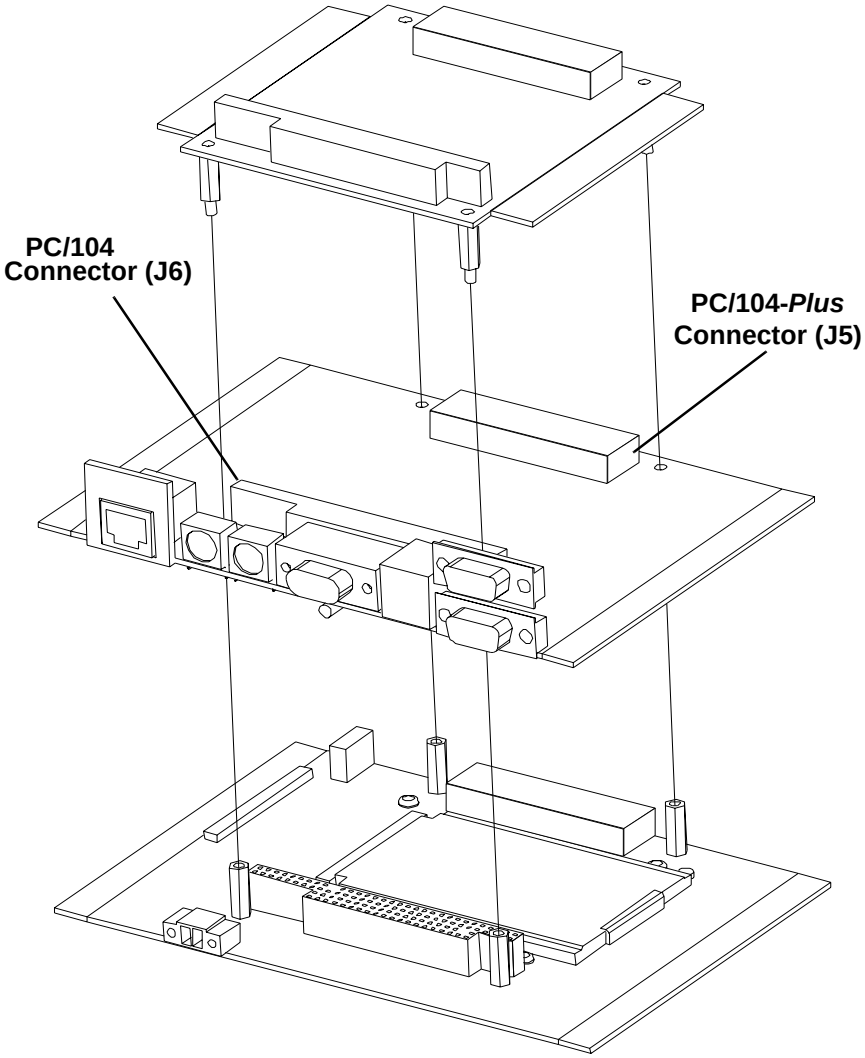


Figure 1-10 Installation of the PC/104-Plus Devices

CHAPTER
2

Standard Features

Contents

Introduction

The VMIOMAX-8451 is a PC-based embedded controller compatible with modern industry-standard desktop systems. The VMIOMAX-8451 therefore retains industry standard memory and I/O maps, along with a standard interrupt architecture. The integrated peripherals described in this section (such as serial ports, USB ports, IDE drives, video controller and Ethernet controller) are all memory mapped the same as similarly equipped desktop systems, ensuring compatibility with modern operating systems.

The following sections describe the standard features of the VMIOMAX-8451.

CPU

The VMIOMAX-8451's CPU is a 333 MHz Geode GX1 x-86-based processor. The RAM/local IDE type and size are user specified as part of the VMIOMAX-8451 ordering information.

To change memory size or local IDE type and size contact Customer Care to receive a Return Material Authorization (RMA).

VMIC Customer Care is available at: 1-800-240-7782, or E-mail us at customer.service@vmic.com.

Physical Memory

The VMIOMAX-8451 provides Synchronous DRAM (SDRAM) as on-board system memory. Memory can be accessed as bytes, words or longwords.

The VMIOMAX-8451 accepts one 144-pin SDRAM SODIMM for a maximum capacity of 256 Mbytes.

NOTE: When using the Configure utility of VMIC's IOWorks Access with Windows NT 4.0 to configure RAM, do not request more than 25 percent of the physical RAM. Exceeding the 25 percent limit may result in a known Windows NT bug that causes unpredictable behavior during the Windows NT boot sequence, and requires the use of an emergency repair disk to restore the computer. The bug is present in Windows NT 4.0 service pack level 3. It is recommended that an emergency repair disk be kept up-to-date and easily accessible.

The VMIOMAX-8451 includes 32 Kbyte of non-volatile SRAM which can be accessed by the CPU at any time, and is used to store system data that must not be lost during power-off conditions.

NOTE: Memory capacity may be extended as parts become available.

Memory and Port Maps

Memory Map

The memory map for the VMIOMAX-8451 is shown in Table 2-1. All systems share this same memory map, although a VMIOMAX-8451 with less than the full 1 Gbyte of SDRAM does not fill the entire space reserved for On-Board Extended Memory.

Table 2-1 VMIOMAX-8451, Interface Memory Address Map

MODE	MEMORY ADDRESS RANGE	SIZE	DESCRIPTION
PROTECTED MODE	$FFC0 8000 - $FFFF FFFF	4 MBytes - 32 KBytes	ROM BIOS Image
	$FFC0 0020 - $FFC0 7FFF	32 KBytes - 32 Bytes	Non-volatile SRAM
	$FFC0 001E - $FFC0 001F	2 Bytes	Device ID: 6504
	$FFC0 001C - $FFC0 001D	2 Bytes	Board ID: 8451
	$FFC0 0000 - $FFC0 001B	28 Bytes	RT / Watchdog Timer Control Registers
	$0400 0000 - $FFBF FFFF	3.9 GBytes	Unused*
	$0010 0000 - $0FFF FFFF	255 MBytes	Reserved for On-Board Extended Memory** (not filled on all systems)
REAL MODE	$D8000 - $FFFFF	160 Kbytes	ROM BIOS
	$C8000 - $D7FFF	64 Kbytes	Reserved
	$C0000 - $C7FFF	32 Kbytes	Video ROM
	$A0000 - $BFFFF	128 Kbytes	Video RAM
	$00000 - $9FFFF	640 Kbytes	User RAM / DOS RAM

* This space can be used to set up protected mode PCI-to-CompactPCI bus windows (also referred to as PCI slave images).

** This space can be allocated as shared memory (for example, between the processor-based CPU and CompactPCI bus Master). Note that if a PMC board is loaded, the expansion BIOS may be placed in this area.

I/O Port Map

Like a desktop system, the VMIOMAX-8451 includes special input/output instructions that access I/O peripherals residing in I/O addressing space (separate and distinct from memory addressing space). Locations in I/O address space are referred to as *ports*. When the CPU decodes and executes an I/O instruction, it produces a 16-bit I/O address on lines A00 to A15 and identifies the I/O cycle with the M/I/O control line. Thus, the CPU includes an independent 64 Kbyte I/O address space, which is accessible as bytes, words or longwords.

Standard hardware circuitry reserves only 1,024 byte of I/O addressing space from I/O $000 to $3FF for peripherals. All standard PC I/O peripherals, such as serial and parallel ports, hard and floppy drive controllers, video system, real-time clock, system timers and interrupt controllers are addressed in this region of I/O space. The BIOS initializes and configures all these registers properly; adjusting these I/O ports directly is not normally necessary.

The assigned and user-available I/O addresses are summarized in the I/O Address Map, Table 2-2.

Table 2-2 VMIOMAX-8451 I/O Address Map

I/O ADDRESS RANGE	SIZE IN BYTES	HW DEVICE	PC/AT FUNCTION
$000 - $00F	16		DMA Controller 1 (Intel 8237A Compatible)
$010 - $01F	16		Reserved
$020 - $021	2		Master Interrupt Controller (Intel 8259A Compatible)
$022 - $03F	30		Reserved
$040 - $043	4		Programmable Timer (Intel 8254 Compatible)
$044 - $05F	30		Reserved
$060 - $064	5		Keyboard, Speaker, System Configuration (Intel 8042 Compatible)
$065 - $06F	11		Reserved
$070 - $071	2		Real-Time Clock
$072 - $07F	14		Reserved
$080 - $08F	16		DMA Page Registers
$090 - $091	2		Reserved
$092	1		Alt. Gate A20/Fast Reset Register
$093 - $09F	11		Reserved
$0A0 - $0A1	2		Slave Interrupt Controller (Intel 8259A Compatible)

Table 2-2 VMIOMAX-8451 I/O Address Map (Continued)

I/O ADDRESS RANGE	SIZE IN BYTES	HW DEVICE	PC/AT FUNCTION
$0A2 - $0BF	30		Reserved
$0C0 - $0DF	32		DMA Controller 2 (Intel 8237A Compatible)
$0E0 - $16F	142		Reserved
$170 - $177	8		Secondary Hard Disk Controller
$178 - $1EF	120		User I/O
$1F0 - $1F7	8		Primary Hard Disk Controller
$1F8 - $277	128		User I/O
$278 - $27F	8	Super I/O Chip*	LPT2 Parallel I/O*
$280 - $2E7	104		Reserved
$2E8 - $2EE	7	UART*	COM4 Serial I/O*
$2EF - $2F7	9		User I/O
$2F8 - $2FE	7	Super-I/O Chip	COM2 Serial I/O (16550 Compatible)
$2FF - $36F	113		Reserved
$370 - $377	8	Super-I/O Chip	Secondary Floppy Disk Controller*
$378 - $37F	8	Super-I/O Chip*	LPT1 Parallel I/O*
$380 - $3E7	108		Reserved
$3E8 - $3EE	7		COM3 Serial I/O*
$3F0 - $3F7	8	Super-I/O Chip	Primary Floppy Disk Controller
$3F8 - $3FE	7	Super-I/O Chip	COM1 Serial I/O (16550 Compatible)
$3FF - $4FF	256		Reserved
$500 - CFF	2048		Reserved

* While these I/O ports are reserved for the listed functions, they are not implemented on the VMIOMAX-8451. They are listed here to make the user aware of the standard PC usage of these ports.

Interrupts

System Interrupts

In addition to an I/O port address, an I/O device has a separate hardware interrupt line assignment. Assigned to each interrupt line is a corresponding interrupt vector in the 256-vector interrupt table at $00000 to $003FF in memory. The 16 maskable interrupts are listed in Table 2-3 along with their functions. Table 2-4 on page 50 details the vectors in the interrupt vector table. The interrupt number in HEX and decimal are also defined for real and protected mode in Table 2-4 on page 50.

The interrupt hardware implementation on the VMIOMAX-8451 is standard for computers built around the PC architecture, which evolved from the IBM PC/XT. In the IBM PC/XT computers, only eight interrupt request lines exist, numbered from IRQ0 to IRQ7 at the PIC. The IBM PC/AT computer added eight more IRQx lines, numbered IRQ8 to IRQ15, by cascading a second slave PIC into the original master PIC. IRQ2 at the master PIC was committed as the cascade input from the slave PIC.

To maintain backward compatibility with PC/XT systems, IBM chose to use the new IRQ9 input on the slave PIC to operate as the old IRQ2 interrupt line on the PC/XT Expansion Bus. Thus, in AT systems, the IRQ9 interrupt line connects to the old IRQ2 pin (pin B4) on the AT Expansion Bus (or ISA bus).

Table 2-3 PC Hardware Interrupt Line Assignments

IRQ	AT FUNCTION	COMMENTS
0	System Timer	Set by BIOS Setup
1	Keyboard	Set by BIOS Setup
2	Duplexed to IRQ9	
3	COM2	
4	COM1	
5	Unused	
6	Floppy Controller	
7	LPT1	
8	Real-Time Clock	
9	Old IRQ2	SVGA or Network I/O
10	Not Assigned	Determined by BIOS
11	Not Assigned	Determined by BIOS
12	Mouse	

Table 2-3 PC Hardware Interrupt Line Assignments (Continued)

IRQ	AT FUNCTION	COMMENTS
13	Math Coprocessor	
14	AT Hard Drive	
15	Flash Drive	

Table 2-4 PC Interrupt Vector Table

INTERRUPT NO.		IRQ LINE	REAL MODE	PROTECTED MODE
HEX	DEC			
00	0		Divide Error	Same as Real Mode
01	1		Debug Single Step	Same as Real Mode
02	2			
03	3		Debug Breakpoint	Same as Real Mode
04	4		ALU Overflow	Same as Real Mode
05	5		Print Screen	Array Bounds Check
06	6			Invalid OpCode
07	7			Device Not Available
08	8	IRQ0	Timer Tick	Double Exception Detected
09	9	IRQ1	Keyboard Input	Coprocessor Segment Overrun
0A	10	IRQ2	BIOS Reserved	Invalid Task State Segment
0B	11	IRQ3	COM2 Serial I/O	Segment Not Present
0C	12	IRQ4	COM1 Serial I/O	Stack Segment Overrun
0D	13	IRQ5	Timer	Same as Real Mode
0E	14	IRQ6	Floppy Disk Controller	Page Fault
0F	15	IRQ7	LPT1	Same as Real Mode
10	16		BIOS Video I/O	Coprocessor Error
11	17		System Configuration Check	Same as Real Mode
12	18		Memory Size Check	Same as Real Mode
13	19		XT Floppy/Hard Drive	Same as Real Mode
14	20		BIOS Comm I/O	Same as Real Mode
15	21		BIOS Cassette Tape I/O	Same as Real Mode
16	22		BIOS Keyboard I/O	Same as Real Mode
17	23		BIOS Printer I/O	Same as Real Mode

Table 2-4 PC Interrupt Vector Table (Continued)

INTERRUPT NO.		IRQ LINE	REAL MODE	PROTECTED MODE
HEX	DEC			
18	24		ROM BASIC Entry Point	Same as Real Mode
19	25		Bootstrap Loader	Same as Real Mode
1A	26		Time of Day	Same as Real Mode
1B	27		Control/Break Handler	Same as Real Mode
1C	28		Timer Control	Same as Real Mode
1D	29		Video Parameter Table Pntr	Same as Real Mode
1E	30		Floppy Parm Table Pntr	Same as Real Mode
1F	31		Video Graphics Table Pntr	Same as Real Mode
20	32		DOS Terminate Program	Same as Real Mode
21	33		DOS Function Entry Point	Same as Real Mode
22	34		DOS Terminate Handler	Same as Real Mode
23	35		DOS Control/Break Handler	Same as Real Mode
24	36		DOS Critical Error Handler	Same as Real Mode
25	37		DOS Absolute Disk Read	Same as Real Mode
26	38		DOS Absolute Disk Write	Same as Real Mode
27	39		DOS Program Terminate, Stay Resident	Same as Real Mode
28	40		DOS Keyboard Idle Loop	Same as Real Mode
29	41		DOS CON Dev. Raw Output	Same as Real Mode
2A	42		DOS 3.x+ Network Comm	Same as Real Mode
2B	43		DOS Internal Use	Same as Real Mode
2C	44		DOS Internal Use	Same as Real Mode
2D	45		DOS Internal Use	Same as Real Mode
2E	46		DOS Internal Use	Same as Real Mode
2F	47		DOS Print Spooler Driver	Same as Real Mode
30-60	48-96		Reserved by DOS	Same as Real Mode
61-66	97-102		User Available	Same as Real Mode
67-6F	103-111		Reserved by DOS	Same as Real Mode
70	112	IRQ8	Real Time Clock	

Table 2-4 PC Interrupt Vector Table (Continued)

INTERRUPT NO.		IRQ LINE	REAL MODE	PROTECTED MODE
HEX	DEC			
71	113	IRQ9	Redirect to IRQ2	
72	114	IRQ10	Not Assigned	
73	115	IRQ11	Not Assigned	
74	116	IRQ12	Mouse	
75	117	IRQ13	Math Coprocessor	
76	118	IRQ14	AT Hard Drive	
77	119	IRQ15	Flash Drive	
78-7F	120-127		Reserved by DOS	Same as Real Mode
80-F0	128-240		Reserved for BASIC	Same as Real Mode
F1-FF	241-255		Reserved by DOS	Same as Real Mode

PCI Interrupts

Interrupts on Peripheral Component Interconnect (PCI) Local Bus are optional and defined as “level sensitive,” asserted low (negative true), using open drain output drivers. The assertion and de-assertion of an interrupt line, INTx#, is asynchronous to CLK. A device asserts its INTx# line when requesting attention from its device driver. Once the INTx# signal is asserted, it remains asserted until the device driver clears the pending request. When the request is cleared, the device de-asserts its INTx# signal.

PCI defines one interrupt line for a single function device and up to four interrupt lines for a multifunction device or connector. For a single function device, only INTA# may be used while the other three interrupt lines have no meaning.

Any function on a multifunction device can be connected to any of the INTx# lines. The Interrupt Pin register defines which INTx# line the function uses to request an interrupt. If a device implements a single INTx# line, it is called INTA#; if it implements two lines, they are called INTA# and INTB#; and so forth. For a multifunction device, all functions may use the same INTx# line, or each may have its own (up to a maximum of four functions), or any combination thereof. A single function can never generate an interrupt request on more than one INTx# line.

Integrated Peripherals

The VMIOMAX-8451 incorporates a National Semiconductor Super I/O (SIO) chip. The SIO provides the VMIOMAX-8451 with a standard floppy drive controller, two 16550 UART-compatible serial ports, and keyboard and mouse ports.

The IDE interface is provided by the Geode CS5530A I/O companion. The IDE interface supports two channels known as the primary and secondary channels. The primary channel is routed to the optional on-board IDE drive. This secondary channel can support two drives, a master and slave. The IDE interface on the VMIOMAX-8451 supports ATA-33 drives and automatically determines the proper operating mode based on the type of drive used.

Ethernet Controller

The network capability is provided by the Intel 82559ER Ethernet Controller. This Ethernet controller is PCI bus based and is software configurable. The VMIOMAX-8451 supports 10BaseT and 100BaseTx Ethernet.

10BaseT

A network based on the 10BaseT standard uses unshielded twisted-pair cables, providing an economical solution to networking. An RJ-45 connector is used with the 10BaseT standard. 10BaseT has a maximum length of 100 meters.

100BaseTx

The VMIOMAX-8451 also supports the 100BaseTx Ethernet. A network based on a 100BaseTx standard uses unshielded twisted-pair cables and a RJ-45 connector. 100BaseTx has a maximum length of 100 meters.

Video Graphics Adapter

The SVGA port on the VMIOMAX-8451 is controlled by the National Semiconductor GX1 / 5530 chipset. The GX1 / 5530 is hardware and BIOS compatible with the industry EGA and SVGA standards supporting both VESA high-resolution and extended video modes. Table 2-5 shows the graphic video modes supported by the video controller.

Table 2-5 Supported Graphics Video Resolutions

Screen Resolution	Maximum Colors	Maximum Refresh Rates (Hz)
640 x 480	64 K	85
800 x 600	64 K	85
1024 x 768	64 K	85
1280 x 1024	64 K	75

Not all SVGA monitors support resolutions and refresh rates beyond 640 x 480 at 85 Hz. Do not attempt to drive a monitor to a resolution or refresh rate beyond its capability.

CHAPTER

3

Embedded PC/RTOS Features

Contents

Introduction

VMIC's VMIOMAX-8451 features additional capabilities beyond those of a typical desktop computer system. The unit provides three software-controlled, general-purpose timers along with a programmable Watchdog Timer for synchronizing and controlling multiple events in embedded applications. The VMIOMAX-8451 also provides 32 Kbyte of non-volatile SRAM and an optional local IDE drive.

PC/104-Plus Bus Interface

The PC/104-Plus bus is an extension of the standard PC/104 bus. It incorporates all standard features of the basic PC/104, but has been enhanced to provide faster speeds.

The following is a list of the PC/104-Plus' main features:

- A PC/104-Plus bus slot which is compatible with PCI version 2.1 specifications
- An integrated PCI arbitration interface (32 bits wide, 3.3V)
- A 33 MHz PCI clock
- Translates PCI cycles to the ISA bus
- Translates ISA master initiated cycles to PCI
- Supports burst read/write from the PCI master

The PC/104-Plus bus supports signal levels at 3.3V only. Thus, any adapter boards built around the PC/104-Plus bus platform must be 3.3V-compatible. Signal levels of 5V are **not** allowed.

The following is a list of slot restrictions applicable to the PC/104-Plus bus:

- SLOT 0 — No known restrictions.
- SLOT 1 — Restricted to extension boards which will **not** use the bus master function. If an extension board with bus master function is used, the VMIOMAX-8451 system may lock up while booting.

NOTE: The VMIOMAX-8451 motherboard supports one bus master extension board in SLOT 0 only.

The PC/104-Plus bus specification is available from the PC/104 Consortium (www.PC104.org).

Embedded PCI Functions

The VMIOMAX-8451 provides non-volatile RAM (NVRAM), Timers and a Watchdog Timer. These functions are required for embedded and real time applications. These embedded functions are located in 32 Kbytes of memory space starting at address $D8000. The first 32 bytes of this space is utilized by the Watchdog and Timer registers. It also contains the board ID. The remaining space starting at $D8020 is utilized by the NVRAM.

FPGA Timers

The VMIOMAX-8451 incorporates three 16-bit Timers and a single Watchdog Timer in an FPGA. The FPGA base address is at $D8000.

Timers

Each of the three 16-bit timers has a Control and Status register (CSR) used to select one of four base clock frequencies, enable generation of an interrupt, and monitor time out status. All three Timers share interrupt 5.

The Timers share a common enable register that allows starting and stopping all three Timers with a single access. The enable register also controls a common read latch bit that allows all three Timer's Current Count registers to be latched for reading when Timer one's Current Count register is read. The Timer Load register utilizes immediate update mode allowing a new count to be loaded while the Timer is running. The new count will be used on the next clock cycle after loading the Count register.

If the Timers are configured to generate an interrupt, and the interrupt is enabled in the Programmable Interrupt Controller, the Interrupt Status bit in the the CSR of the interrupting Timer must be cleared in the interrupt service routine. This can be accomplished two different ways. Clearing the status bit in the CSR of the interrupting timer while preserving the other bits, or writing zero to the Interrupt Clear register of the interrupting timer.

Table 3-1 Timer Enable Register Bit Map

Timer Enable Register (TMRENA): Offset $00, Read-Only							
Bit 07	**Bit 06**	**Bit 05**	**Bit 04**	**Bit 03**	**Bit 02**	**Bit 01**	**Bit 00**
Reserved				TMR Latch	TMR3 Enable	TMR2 Enable	TMR1 Enable

Watchdog Timer Registers

The independent Watchdog Timer can be programmed to one of eight time outs. The Watchdog Control and Status register is located at offset $02 from the base address of the FPGA. The Watchdog Keepalive register is located at offset $07 from the base address of the FPGA. Timeouts range from 2.048 milliseconds to 67.1 seconds. A byte write of any data to the Watchdog Keepalive register resets the time out counter to the selected value and prevents the Watchdog Timer from driving reset.

Watchdog Control Status Register (WCSR)

The Watchdog Timer is controlled and monitored by the Watchdog Control Status Register (WCSR) which is located at offset $02 from the base address of the FPGA ($D8000). The mapping of the bits in this register are show in Table 3-2 on page 61.

Table 3-2 Watchdog Control and Status Register Bit Map

Watchdog Control and Status Register (WCSR): Offset $02, Read/Write, Byte							
Bit 07	**Bit 06**	**Bit 05**	**Bit 04**	**Bit 03**	**Bit 02**	**Bit 01**	**Bit 00**
Reserved	WCSR TO_Select			TMR RSVD	TMR3 RSVD	TMR2 RSVD	WCSR Enable

The “WCSR Timeout Select” field is used to select the timeout value of the Watchdog Timer as follows:

Table 3-3 Watchdog Timeout Values

Timeout	WCSR[10]	WCSR[9]	WCSR[8]
67.1s	0	0	0
33.6s	0	0	1
2.1s	0	1	0
524ms	0	1	1
262ms	1	0	0
131ms	1	0	1
32.768ms	1	1	0
2.048ms	1	1	1

The “WCSR Enable” bit is used to enable the Watchdog Timer function. This bit must be set to “1” in order for the Watchdog Timer to function. Note that since all registers default to zero after reset, the Watchdog Timer is always disabled after a reset. The Watchdog Timer must be re-enabled by the application software after reset in order for the Watchdog Timer to continue to operate. Once the Watchdog Timer is enabled, the application software must refresh the Watchdog Timer within the selected timeout period to prevent a reset from being generated. The Watchdog Timer is refreshed by performing a write to the Watchdog Keepalive register. The data written is irrelevant.

Watchdog Keepalive Register

When enabled, the Watchdog Timer is prevented from resetting the system by writing to the Watchdog Keepalive register located at offset $07 from the base address of the FPGA ($D8000) within the selected timeout period. The data written to this location is irrelevant. Table 3-4 shows the bit definitions for the Watchdog Keepalive register.

Table 3-4 Watchdog Keepalive Register Bit Map

Watchdog Keepalive Register: Offset $07, Read/Write, Byte							
Bit 07	**Bit 06**	**Bit 05**	**Bit 04**	**Bit 03**	**Bit 02**	**Bit 01**	**Bit 00**
N / A							

Timer Control and Status Registers

Table 3-5 Timer Control and Status Register 1 Bit Map

Timer Control and Status Register 1 (TCSR1): Offset $04, Read/Write, Byte							
Bit 07	**Bit 06**	**Bit 05**	**Bit 04**	**Bit 03**	**Bit 02**	**Bit 01**	**Bit 00**
TCSR1 IRQSTAT	Reserved				TCSR1 CK_Select		TCSR1 IRQEnable

Table 3-6 Timer Control and Status Register 2 Bit Map

Timer Control and Status Register 2 (TCSR2): Offset $05, Read/Write, Byte							
Bit 07	**Bit 06**	**Bit 05**	**Bit 04**	**Bit 03**	**Bit 02**	**Bit 01**	**Bit 00**
TCSR2 IRQSTAT	Reserved				TCSR2 CK_Select		TCSR2 IRQEnable

Table 3-7 Timer Control and Status Register 3 Bit Map

Timer Control and Status Register 3 (TCSR3): Offset $06, Read/Write, Byte							
Bit 07	**Bit 06**	**Bit 05**	**Bit 04**	**Bit 03**	**Bit 02**	**Bit 01**	**Bit 00**
TCSR3 IRQSTAT	Reserved				TCSR3 CK_Select		TCSR3 IRQEnable

Timer Load Count Registers

Table 3-8 Timer 1 Load Count Register LSB Bit Map

Timer 1 Load Count Register LSB: Offset $08, Read/Write, Word							
Bit 07	**Bit 06**	**Bit 05**	**Bit 04**	**Bit 03**	**Bit 02**	**Bit 01**	**Bit 00**

Table 3-9 Timer 1 Load Count Register MSB Bit Map

Timer 1 Load Count Register MSB: Offset $09, Read/Write, Word							
Bit 15	**Bit 14**	**Bit 13**	**Bit 12**	**Bit 11**	**Bit 10**	**Bit 09**	**Bit 08**

Table 3-10 Timer 2 Load Count Register LSB Bit Map

Timer 2 Load Count Register LSB: Offset $0A, Read/Write, Word							
Bit 07	**Bit 06**	**Bit 05**	**Bit 04**	**Bit 03**	**Bit 02**	**Bit 01**	**Bit 00**

Table 3-11 Timer 2 Load Count Register MSB Bit Map

Timer 2 Load Count Register MSB: Offset $0B, Read/Write, Word							
Bit 15	**Bit 14**	**Bit 13**	**Bit 12**	**Bit 11**	**Bit 10**	**Bit 09**	**Bit 08**

Table 3-12 Timer 3 Load Count Register LSB Bit Map

Timer 3 Load Count Register LSB: Offset $0C, Read/Write, Word							
Bit 07	**Bit 06**	**Bit 05**	**Bit 04**	**Bit 03**	**Bit 02**	**Bit 01**	**Bit 00**

Table 3-13 Timer 3 Load Count Register MSB Bit Map

Timer 3 Load Count Register MSB: Offset $0D, Read/Write, Word							
Bit 15	**Bit 14**	**Bit 13**	**Bit 12**	**Bit 11**	**Bit 10**	**Bit 09**	**Bit 08**

Timer Current Count Registers

Table 3-14 Timer 1 Current Count Register LSB Bit Map

Timer 1 Current Count Register LSB: Offset $10, Read/Write, Word							
Bit 07	**Bit 06**	**Bit 05**	**Bit 04**	**Bit 03**	**Bit 02**	**Bit 01**	**Bit 00**

Table 3-15 Timer 1 Current Count Register MSB Bit Map

Timer 1 Current Count Register MSB: Offset $11, Read/Write, Word							
Bit 15	**Bit 14**	**Bit 13**	**Bit 12**	**Bit 11**	**Bit 10**	**Bit 09**	**Bit 08**

Table 3-16 Timer 2 Current Count Register LSB Bit Map

Timer 1 Current Count Register LSB: Offset $12, Read/Write, Word							
Bit 07	**Bit 06**	**Bit 05**	**Bit 04**	**Bit 03**	**Bit 02**	**Bit 01**	**Bit 00**

Table 3-17 Timer 2 Current Count Register MSB Bit Map

Timer 1 Current Count Register MSB: Offset $13, Read/Write, Word							
Bit 15	**Bit 14**	**Bit 13**	**Bit 12**	**Bit 11**	**Bit 10**	**Bit 09**	**Bit 08**

Table 3-18 Timer 3 Current Count Register LSB Bit Map

Timer 1 Current Count Register LSB: Offset $14, Read/Write, Word							
Bit 07	**Bit 06**	**Bit 05**	**Bit 04**	**Bit 03**	**Bit 02**	**Bit 01**	**Bit 00**

Table 3-19 Timer 3 Current Count Register MSB Bit Map

Timer 1 Current Count Register MSB: Offset $15, Read/Write, Word							
Bit 15	**Bit 14**	**Bit 13**	**Bit 12**	**Bit 11**	**Bit 10**	**Bit 09**	**Bit 08**

Timer Clear IRQ Status Registers

Table 3-20 Timer 1 Clear IRQ Status Register Bit Map

Timer 1 Clear IRQ Status Register: Offset $18, Read/Write, Byte							
Bit 07	**Bit 06**	**Bit 05**	**Bit 04**	**Bit 03**	**Bit 02**	**Bit 01**	**Bit 00**
0	0	0	0	0	0	0	0

Table 3-21 Timer 2 Clear IRQ Status Register Bit Map

Timer 2 Clear IRQ Status Register: Offset $19, Read/Write, Byte							
Bit 07	**Bit 06**	**Bit 05**	**Bit 04**	**Bit 03**	**Bit 02**	**Bit 01**	**Bit 00**
0	0	0	0	0	0	0	0

Table 3-22 Timer 3 Clear IRQ Status Register Bit Map

Timer 3 Clear IRQ Status Register: Offset $1A, Read/Write, Byte							
Bit 07	**Bit 06**	**Bit 05**	**Bit 04**	**Bit 03**	**Bit 02**	**Bit 01**	**Bit 00**
0	0	0	0	0	0	0	0

Board ID Registers

The Board ID registers are located at offsets $1C and $1E and are read-only. Offset $1C has $7301 and offset $1E has $FF46.

Table 3-23 Board ID Register Bit Map

Board ID LSB: Offset $1C, Read-Only, ($01)							
Bit 07	**Bit 06**	**Bit 05**	**Bit 04**	**Bit 03**	**Bit 02**	**Bit 01**	**Bit 00**
0	0	0	0	0	0	0	1

Board ID MSB: Offset $1D, Read-Only, ($73)							
Bit 15	**Bit 14**	**Bit 13**	**Bit 12**	**Bit 11**	**Bit 10**	**Bit 09**	**Bit 08**
0	1	1	1	0	0	1	1

Board ID LSB: Offset $1E, Read-Only, ($46)							
Bit 07	**Bit 06**	**Bit 05**	**Bit 04**	**Bit 03**	**Bit 02**	**Bit 01**	**Bit 00**
0	1	0	0	0	1	1	0

Board ID MSB: Offset $1F, Read-Only, ($FF)							
Bit 15	**Bit 14**	**Bit 13**	**Bit 12**	**Bit 11**	**Bit 10**	**Bit 09**	**Bit 08**
1	1	1	1	1	1	1	1

Bit Definitions for the FPGA Timers and Watchdog Timer

The following code is used to set the FPGA Timers and the Watchdog Timer.

```
/* register offsets from FPGA base address */
#define TMRENA_A       0x00    /* timer 1,2,3 enable + read latch bits */
#define WCSR_A         0x02    /* watchdog CSR */
#define TCSR1_A        0x04    /* timer 1 CSR */
#define TCSR2_A        0x05    /* timer 2 CSR */
#define TCSR3_A        0x06    /* timer 3 CSR */
#define WKPA_A         0x07    /* watchdog keepalive */
#define TMRLCR1_A      0x08    /* timer 1 load count register */
#define TMRLCR2_A      0x0A    /* timer 2 load count register */
#define TMRLCR3_A      0x0C    /* timer 3 load count register */
#define TMRCCR1_A      0x10    /* timer 1 current count register */
#define TMRCCR2_A      0x12    /* timer 2 current count register */
#define TMRCCR3_A      0x14    /* timer 3 current count register */
#define TMR1IC_A       0x18    /* timer 1 IRQ clear register */
#define TMR2IC_A       0x19    /* timer 2 IRQ clear register */
#define TMR3IC_A       0x1A    /* timer 3 IRQ clear register */
#define BRDID1_A       0x1C    /* board ID bits 15-00 */
#define BRDID2_A       0x1E    /* board ID bits 31-16 */

/* Timer enable bits and latch control bit (latch all on timer 1 current count read) */
#define TMR1_EN        0x01    /* timer 1 run enable */
#define TMR2_EN        0x02    /* timer 2 run enable */
#define TMR3_EN        0x04    /* timer 4 run enable */
#define TMR_LATCH      0x08    /* latch all timers on read of timer 1 */

/* timer CSR bits */
#define TCSR_IRQSTAT   0x80    /* timer IRQ STATUS flag */
#define TCSR_CK250K    0x06    /* timer clock select 250 KHz */
#define TCSR_CK500K    0x04    /* timer clock select 500 KHz */
#define TCSR_CK1M      0x02    /* timer clock select 1 Mhz */
#define TCSR_CK2M      0x00    /* timer clock select 2 Mhz */
#define TCSR_IRQEN     0x01    /* timer IRQ enable */

/* watchdog CSR bits */
#define WCSR_TO2M      0x70    /* timeout = 2.048 Ms */
#define WCSR_TO32M     0x60    /* timeout = 32.768 Ms */
#define WCSR_TO131M    0x50    /* timeout = 131 Ms */
#define WCSR_TO262M    0x40    /* timeout = 262 Ms */
#define WCSR_TO524M    0x30    /* timeout = 524 Ms */
#define WCSR_TO2S      0x20    /* timeout = 2.1 S */
#define WCSR_TO33S     0x10    /* timeout = 33.6 S */
#define WCSR_TO67S     0x00    /* timeout = 67.1 S */
#define WCSR_EN        0x01    /* watchdog timeout enable */
```

Example Code

Nonvolatile SRAM

```
/*
**  FILE: NVRAM.C
**
**
*/

#include  <stdlib.h>
#include  <stdio.h>
#include  <dos.h>

#include  "8451.h"

int main( int argc, char * argv[] )
{
  unsigned char byte_data;
  unsigned long temp_dword, cpuid;
  unsigned char far * nvram_ba;
  unsigned short far * timer_ba;

  timer_ba = (unsigned short far *) MK_FP( 0xD800, 0x0000 );
  nvram_ba = (unsigned char far *) MK_FP( 0xD800, 0x0020 );

  cpuid = 0;

  temp_dword = *( timer_ba + (BRDID1_A / 2));
  cpuid = *( timer_ba + (BRDID2_A / 2) );
  cpuid = cpuid << 16;
  cpuid = cpuid | temp_dword;

  printf("ID: %.8lX\n", cpuid );
  printf("NVRAM base address  : %p\n", nvram_ba );

  *nvram_ba = 0x55;
  byte_data = *nvram_ba;

  return( 0 );

} /* end main */
```

PIC.H

```
/* FILE: PIC.H                                                              */
/*      Bit definitions for the PC/AT programable interrupt controller (PIC) */
/*      sjl     5/15/95                                                     */
/*                                                                          */

#define         PIC1_A          0x20    /* I/O port addr of PIC 1 */
#define         PIC1_MASK_A     0x21    /* I/O port addr of PIC 1 mask reg */
#define         PIC2_A          0xA0    /* I/O port addr of PIC 2 (AT only) */
#define         PIC2_MASK_A     0xA1    /* I/O port addr of PIC 2 mask reg */

#define         IRQ0_MASK       0x01    /* mask off int 0 - timer 0 sys tic */
#define         IRQ1_MASK       0x02    /* mask off int 1 - keyboard */
#define         IRQ2_MASK       0x04    /* mask off int 2 - rsvd XT; 8-15 AT */
#define         IRQ3_MASK       0x08    /* mask off int 3 - com port */
#define         IRQ4_MASK       0x10    /* mask off int 4 - com port */
#define         IRQ5_MASK       0x20    /* mask off int 5 - HD XT; LPT AT */
#define         IRQ6_MASK       0x40    /* mask off int 6 - Floppy Disk */
#define         IRQ7_MASK       0x80    /* mask off int 7 - LPT */

#define         IRQ8_MASK       0x01    /* mask off int 8 - RTC */
#define         IRQ9_MASK       0x02    /* mask off int 9 - Re-directed IRQ2 */
#define         IRQ10_MASK      0x04    /* mask off int 10 - Unassigned */
#define         IRQ11_MASK      0x08    /* mask off int 11 - Unassigned */
#define         IRQ12_MASK      0x10    /* mask off int 12 - Unassigned */
#define         IRQ13_MASK      0x20    /* mask off int 13 - Co-processor */
#define         IRQ14_MASK      0x40    /* mask off int 14 - HD AT */
#define         IRQ15_MASK      0x80    /* mask off int 15 - Unassigned */

#define         PIC_EOI         0x20    /* PIC End Of Interrupt */

#define         IRQ2            0x0A    /* Int. No. for hardware int. 2 */
#define         IRQ3            0x0B    /* Int. No. for hardware int. 3 */
#define         IRQ4            0x0C    /* Int. No. for hardware int. 4 */
#define         IRQ5            0x0D    /* Int. No. for hardware int. 5 */
#define         IRQ6            0x0E    /* Int. No. for hardware int. 6 */
#define         IRQ7            0x0F    /* Int. No. for hardware int. 7 */
#define         IRQ8            0x70    /* Int. No. for hardware int. 8 */
#define         IRQ9            0x71    /* Int. No. for hardware int. 9 */
#define         IRQA            0x72    /* Int. No. for hardware int. A */
#define         IRQB            0x73    /* Int. No. for hardware int. B */
#define         IRQC            0x74    /* Int. No. for hardware int. C */
#define         IRQD            0x75    /* Int. No. for hardware int. D */
#define         IRQE            0x76    /* Int. No. for hardware int. E */
#define         IRQF            0x77    /* Int. No. for hardware int. F */
```

Timers.C

```
/*
**  FILE: TIMERS.C
**
**
*/

#include  <stdlib.h>
#include  <stdio.h>
#include  <string.h>
#include  <conio.h>
#include  <ctype.h>
#include  <dos.h>
#include  <time.h>

#include  "pic.h"
#include  "8451.h"

/* function prototypes */
void interrupt irq_rcvd( void );
void init_int( void );
void init_pic_rtc( void );           /* initialize pic rtc and install isr */
void restore_orig_int( void );
void reset_pic_rtc( void );          /* restore pic rtc and isr to original */
int test_timers( void );             /* test timers write/read, latch all, run */

/* globals */
unsigned char pic2_org;
unsigned char pic1_org;
unsigned short int1_status, int2_status, int3_status, scratch;
void far interrupt (* old_vect)(void);
unsigned char int_line = 5;
char user[80];
unsigned long to_count;
int t1_isr_count;
int t2_isr_count;
int t3_isr_count;
unsigned char far * timer_ba;

int main( int argc, char * argv[] )
{
  int test_int;
  unsigned long temp_dword, cpuid;
```

```
  timer_ba = (unsigned char far *) MK_FP( 0xD800, 0x0000 );
  temp_dword = *( (unsigned short far *)timer_ba + (BRDID1_A / 2));
  cpuid = *( (unsigned short far *)timer_ba + (BRDID2_A / 2) );
  cpuid = cpuid << 16;
  cpuid = cpuid | temp_dword;

  printf("\nIRQ: %.2X\n", int_line );
  printf("ID: %.8lX\n", cpuid );
  printf("TIMER base address  : %p\n", timer_ba );

  /* setup RTC for 976.5625 US tics */
  init_int();
  test_timers();
  restore_orig_int();
  return( 0 );
} /* end main */

/*********************************************************************/
/*  test_timers()                                                    */
/*                                                                   */
/*  purpose: Test the fpga on board timers                           */
/*                                                                   */
/*                                                                   */
/*********************************************************************/
/*  parameters: none                                                 */
/*********************************************************************/
/*  return value: TRUE - Passed                                      */
/*                FALSE - Failed                                     */
/*********************************************************************/
/*  GLOBALS: timer_ba - timer base address                           */
/*                                                                   */
/*********************************************************************/
int test_timers( void )
{
  unsigned char far * timer_bab;
  unsigned short far * timer_baw;
  timer_bab = timer_ba;
  timer_baw = (unsigned short far *)timer_ba;
  /* TEST TIMERS GENERATE INTERRUPT */
  printf("Testing Timer 1 gen int.............................");
  /* disable all timers */
  *( timer_bab + TMRENA_A ) = 0;
  int1_status = 0;
  to_count = 100;
```

```
  /* load timer 0xFFFF and 2 Mhz clock (0xFFFF = 32.768 Ms) */
  *( timer_baw + (TMRLCR1_A / 2) ) = 0xFFFF;
  *( timer_bab + TCSR1_A ) = (TCSR_CK2M | TCSR_IRQEN);
  /* enable timer */
  *( timer_bab + TMRENA_A ) = TMR1_EN;
  do
  {
    if( int1_status & TCSR_IRQSTAT ) break;
    delay( 1 );
    to_count--;
  } while( to_count );
  /* disable timers */
  *( timer_bab + TMRENA_A ) = 0;
  if( !to_count )
  {
    printf("FAILED\nTimed out waiting for interrupt from Timer 1\n");
    return( 0 );
  }
  else
    printf("PASSED\n");

/*************************************************************************/

  printf("Testing Timer 2 gen int.............................");
  /* disable all timers */
  *( timer_bab + TMRENA_A ) = 0;
  int2_status = 0;
  to_count = 100;
  /* load timer 0xFFFF and 2 Mhz clock (0xFFFF = 32.768 Ms) */
  *( timer_baw + (TMRLCR2_A / 2) ) = 0xFFFF;
  *( timer_bab + TCSR2_A ) = (TCSR_CK2M | TCSR_IRQEN);
  /* enable timer */
  *( timer_bab + TMRENA_A ) = TMR2_EN;
  do
  {
    if( int2_status & TCSR_IRQSTAT ) break;
    delay( 1 );
    to_count--;
  } while( to_count );
  /* disable timers */
  *( timer_bab + TMRENA_A ) = 0;
  if( !to_count )
  {
    printf("FAILED\nTimed out waiting for interrupt from Timer 2\n");
```

```
    return( 0 );
  }
  else
    printf("PASSED\n");

/*****************************************************************************/

  printf("Testing Timer 3 gen int.............................");
  /* disable all timers */
  *( timer_bab + TMRENA_A ) = 0;
  int3_status = 0;
  to_count = 100;
  /* load timer 0xFFFF and 2 Mhz clock (0xFFFF = 32.768 Ms) */
  *( timer_baw + (TMRLCR3_A / 2) ) = 0xFFFF;
  *( timer_bab + TCSR3_A ) = (TCSR_CK2M | TCSR_IRQEN);
  /* enable timer */
  *( timer_bab + TMRENA_A ) = TMR3_EN;
  do
  {
    if( int3_status & TCSR_IRQSTAT ) break;
    delay( 1 );
    to_count--;
  } while( to_count );
  /* disable timers */
  *( timer_bab + TMRENA_A ) = 0;
  if( !to_count )
  {
    printf("FAILED\nTimed out waiting for interrupt from Timer 3\n");
    return( 0 );
  }
  else
    printf("PASSED\n");
  return( 1 );
} /* end test_timers */

/*****************************************************************/
/*  init_int()                                                   */
/*                                                               */
/*  purpose: Using the interrupt assigned, the original vector is  */
/*           saved and the vector to the new ISR is installed. The */
/*           programmable-interrupt-controller (PIC) is enabled.   */
/*                                                               */
/*****************************************************************/
/*  parameters: none                                             */
```

```
/*****************************************************************/
/*  return value: none                                           */
/*****************************************************************/
void init_int( void )
{
  disable();
  /* Read 8259 master/slave Programmable Interrupt controller */
  pic2_org = inp(PIC2_MASK_A) & 0xFF; /* slave mask bits */
  pic1_org = inp(PIC1_MASK_A) & 0xFF; /* master mask bits */
  switch( int_line )
  {
    case 0x2:
      old_vect = getvect( IRQ2 ); /* save vector for IRQ 02 */
      setvect( IRQ2, irq_rcvd );
      /* enable interrupt 2 */
      outp(PIC1_MASK_A, (pic1_org & (~IRQ2_MASK)) );
    break;
    case 0x3:
      old_vect = getvect( IRQ3 ); /* save vector for IRQ 03 */
      setvect( IRQ3, irq_rcvd );
      /* enable interrupt 3 */
      outp(PIC1_MASK_A, (pic1_org & (~IRQ3_MASK)) );
    break;
    case 0x4:
      old_vect = getvect( IRQ4 ); /* save vector for IRQ 04 */
      setvect( IRQ4, irq_rcvd );
      /* enable interrupt 4 */
      outp(PIC1_MASK_A, (pic1_org & (~IRQ4_MASK)) );
    break;
    case 0x5:
      old_vect = getvect( IRQ5 ); /* save vector for IRQ 05 */
      setvect( IRQ5, irq_rcvd );
      /* enable interrupt 5 */
      outp(PIC1_MASK_A, (pic1_org & (~IRQ5_MASK)) );
    break;
    case 0x6:
      old_vect = getvect( IRQ6 ); /* save vector for IRQ 06 */
      setvect( IRQ6, irq_rcvd );
      /* enable interrupt 6 */
      outp(PIC1_MASK_A, (pic1_org & (~IRQ6_MASK)) );
    break;
    case 0x7:
      old_vect = getvect( IRQ7 ); /* save vector for IRQ 07 */
      setvect( IRQ7, irq_rcvd );
```

```
  /* enable interrupt 7 */
  outp(PIC1_MASK_A, (pic1_org & (~IRQ7_MASK)) );
break;
case 0x8:
  old_vect = getvect( IRQ8 ); /* save vector for IRQ 08 */
  setvect( IRQ8, irq_rcvd );
  /* enable interrupt 8 */
  outp(PIC2_MASK_A, (pic2_org & (~IRQ8_MASK)) );
break;
case 0x9:
  old_vect = getvect( IRQ9 ); /* save vector for IRQ 09 */
  setvect( IRQ9, irq_rcvd );
  /* enable interrupt 9 */
  outp(PIC2_MASK_A, (pic2_org & (~IRQ9_MASK)) );
break;
case 0xa:
  old_vect = getvect( IRQA ); /* save vector for IRQ 10 */
  setvect( IRQA, irq_rcvd );
  /* enable interrupt 10 */
  outp(PIC2_MASK_A, (pic2_org & (~IRQ10_MASK)) );
break;
case 0xb:
  old_vect = getvect( IRQB ); /* save vector for IRQ 11 */
  setvect( IRQB, irq_rcvd );
  /* enable interrupt 11 */
  outp(PIC2_MASK_A, (pic2_org & (~IRQ11_MASK)) );
break;
case 0xc:
  old_vect = getvect( IRQC ); /* save vector for IRQ 12 */
  setvect( IRQC, irq_rcvd );
  /* enable interrupt 12 */
  outp(PIC2_MASK_A, (pic2_org & (~IRQ12_MASK)) );
break;
case 0xd:
  old_vect = getvect( IRQD ); /* save vector for IRQ 13 */
  setvect( IRQD, irq_rcvd );
  /* enable interrupt 13 */
  outp(PIC2_MASK_A, (pic2_org & (~IRQ13_MASK)) );
break;
case 0xe:
  old_vect = getvect( IRQE ); /* save vector for IRQ 14 */
  setvect( IRQE, irq_rcvd );
  /* enable interrupt 14 */
  outp(PIC2_MASK_A, (pic2_org & (~IRQ14_MASK)) );
```

```
    break;
    case 0xf:
      old_vect = getvect( IRQF ); /* save vector for IRQ 15 */
      setvect( IRQF, irq_rcvd );
      /* enable interrupt 15 */
      outp(PIC2_MASK_A, (pic2_org & (~IRQ15_MASK)) );
    break;
  } /* end switch */
  enable();
} /* init_int */

/*****************************************************************/
/*  restore_orig_int()                                           */
/*                                                               */
/*  purpose: Using the interrupt assigned, the original vector is  */
/*           restored and the programmable-interrupt-controller    */
/*           is disabled.                                        */
/*                                                               */
/*  Prerequisite: The interrupt line to be used must have        */
/*                already been loaded in the global variable.    */
/*                                                               */
/*****************************************************************/
/*  parameters: none                                             */
/*****************************************************************/
/*  return value: none                                           */
/*****************************************************************/
void restore_orig_int( void )
{
  disable();
  outp(PIC2_MASK_A, pic2_org);
  outp(PIC1_MASK_A, pic1_org);
  switch( int_line )
  {
    case 0x2:
      setvect( IRQ2, old_vect );
    break;
    case 0x3:
      setvect( IRQ3, old_vect );
    break;
    case 0x4:
      setvect( IRQ4, old_vect );
    break;
    case 0x5:
      setvect( IRQ5, old_vect );
```

```
    break;
    case 0x6:
      setvect( IRQ6, old_vect );
    break;
    case 0x7:
      setvect( IRQ7, old_vect );
    break;
    case 0x8:
      setvect( IRQ8, old_vect );
    break;
    case 0x9:
      setvect( IRQ9, old_vect );
    break;
    case 0xa:
      setvect( IRQA, old_vect );
    break;
    case 0xb:
      setvect( IRQB, old_vect );
    break;
    case 0xc:
      setvect( IRQC, old_vect );
    break;
    case 0xd:
      setvect( IRQD, old_vect );
    break;
    case 0xe:
      setvect( IRQE, old_vect );
    break;
    case 0xf:
      setvect( IRQF, old_vect );
    break;
  } /* end switch */
  enable();
} /* restore_orig_int */


/*****************************************************************/
/*  irq_rcvd()                                                   */
/*                                                               */
/*  purpose: Interrupt service routine used to service any of the  */
/*           interrupts generated.                               */
/*           (INTA handler)                                      */
/*                                                               */
/*****************************************************************/
```

```
/*  parameters: none                                                    */
/************************************************************************/
/*  return value: none                                                  */
/************************************************************************/
void interrupt irq_rcvd( void )
{
  disable();
  /* SERVICE TIMERS */
  int1_status = *( timer_ba + TCSR1_A );
  int2_status = *( timer_ba + TCSR2_A );
  int3_status = *( timer_ba + TCSR3_A );
  if( int1_status & TCSR_IRQSTAT )
  {
    t1_isr_count++;
    *( timer_ba + TMR1IC_A ) = 0;
  }
  if( int2_status & TCSR_IRQSTAT )
  {
    t2_isr_count++;
    *( timer_ba + TMR2IC_A ) = 0;
  }
  if( int3_status & TCSR_IRQSTAT )
  {
    t3_isr_count++;
    *( timer_ba + TMR3IC_A ) = 0;
  }
  /* Non specific end of interrupt to master & slave PIC */
  outp(PIC1_A, PIC_EOI);  /* Master end of irq command IRQ7 - IRQ0 */
  outp(PIC2_A, PIC_EOI);  /* Slave end of irq command IRQ15 - IRQ8 */

  enable();

}
```

Watchdog.C

```
/*
**  FILE: WATCHDOG.C
**
**
*/

#include  <stdlib.h>
#include  <stdio.h>
#include  <string.h>
#include  <dos.h>

#include  "8451.h"
/* function prototype */
int test_wd( int );
/* global variable */
unsigned short far * timer_ba;
int main( int argc, char * argv[] )
{
  char user[80];
  int test_int;
  unsigned long temp_dword, cpuid;
  timer_ba = (unsigned short far *) MK_FP( 0xD800, 0x0000 );
  temp_dword = *( timer_ba + (BRDID1_A / 2));
  cpuid = *( timer_ba + (BRDID2_A / 2) );
  cpuid = cpuid << 16;
  cpuid = cpuid | temp_dword;
  printf("ID: %.8lX\n", cpuid );
  printf("TIMER base address  : %p\n", timer_ba );
  /* TEST WATCHDOG */
  printf("\n\n");
  printf("Select Watch Dog test\n");
  printf("1 - Feed the dog\n");
  printf("2 - 2.048 Msec\n");
  printf("3 - 32.768 Msec\n");
  printf("4 - 131.0 Msec\n");
  printf("5 - 262.0 Msec\n");
  printf("6 - 524.0 Msec\n");
  printf("7 - 2.1 Sec\n");
  printf("8 - 33.6 Sec\n");
  printf("9 - 67.1 Sec\n");
  printf("[>> ");
  gets( user );
```

```
  sscanf( user, "%i", &test_int );
  test_wd( test_int );
 return( 0 );
} /* end main */
/*****************************************************************/
/*  test_wd()                                                    */
/*                                                               */
/*  purpose: Test the fpga on board watch dog                    */
/*                                                               */
/*                                                               */
/*****************************************************************/
/*  parameters: none                                             */
/*****************************************************************/
/*  return value: TRUE - Passed                                  */
/*                FALSE - Failed                                 */
/*****************************************************************/
/*  GLOBALS: timer_ba - timer base address                       */
/*                                                               */
/*****************************************************************/
int test_wd( int tst )
{
unsigned long to_count;
unsigned char far * timer_bab;
unsigned short far * timer_baw;

const char ticratestr[8][20] =
{
  "2.048 Msec......\0",
  "32.768 Msec.....\0",
  "131.0 Msec......\0",
  "262.0 Msec......\0",
  "524.0 Msec......\0",
  "2.1 Sec.........\0",
  "33.6 Sec........\0",
  "67.1 Sec.........\0"
};
const unsigned long tocnt[8] =
{
  10,
  50,
  150,
  300,
  600,
  2200,
```

```
  35000,
  140000
};
const unsigned short ticrate[8] =
{
  WCSR_TO2M,
  WCSR_TO32M,
  WCSR_TO131M,
  WCSR_TO262M,
  WCSR_TO524M,
  WCSR_TO2S,
  WCSR_TO33S,
  WCSR_TO67S
};
  timer_bab = (unsigned char far *) timer_ba;
  timer_baw = timer_ba;
  /* disable watchdog */
  *(timer_baw + (WCSR_A / 2)) = 0;
  switch( tst )
  {
    case 1:
      /* TEST WATCHDOG TO RESET FEED THE DOG */
      printf("Testing Watchdog feed the dog.............");
      to_count = 100;
      /* enable watchdog 32.768 Ms */
      *( timer_baw + (WCSR_A / 2) ) = (WCSR_RST | WCSR_TO32M | WCSR_EN);
      do
      {
        *( timer_bab + WKPA_A ) = 0; /* feed the dog */
        to_count--;
        delay( 1 );
      } while( to_count );
      /* disable watchdog */
      *( timer_baw + (WCSR_A / 2) ) = 0;
      printf("PASSED\n");
    break;
    case 2:
    case 3:
    case 4:
    case 5:
    case 6:
    case 7:
    case 8:
    case 9:
```

```
      /* TEST WATCHDOG TO RESET VALIDATE CLOCK TIC */
      printf("Testing Watchdog to RESET tic = %s", ticratestr[tst - 2] );
      to_count = tocnt[tst - 2];
      *( timer_baw + (WCSR_A / 2) ) = (WCSR_RST | WCSR_EN | ticrate[tst - 2]);
      do
      {
        delay( 1 );
        to_count--;
      } while( to_count );
      /* disable watchdog */
      *(timer_baw + (WCSR_A / 2)) = 0;
      printf("FAILED\nTimed out waiting for reset from watchdog\n");
      return( 0 );
    break;
    default: printf("\nInvalid Entry\n");
  } /* end switch */
  return( 1 );
} /* end test_wd */
```

NVSRAM

The VMIOMAX-8451 provides 32 Kbytes of non-volatile SRAM. This memory is mapped in 32K of address space starting at the base address of the FPGA ($D8020). This memory is available at any time and supports byte, short word and long word accesses from the PCI bus. The contents of this memory is retained when the power to the board is removed.

Local IDE Drive

Configuration

The Flash Disk and hard drive reside on the same VMIOMAX-8451 secondary IDE bus. The default setting in the BIOS 'STANDARD CMOS SETUP' screen is the 'AUTO' setting. In the BIOS 'PERIPHERAL SETUP' screen, the secondary PCI IDE interface must be enabled for these disks to be functional. Refer to Appendix C for additional details.

The Primary and Secondary PCI IDE Interfaces are controlled (enabled or disabled) in the Integrated Peripheral Setup screen of the BIOS. The First Boot Device is selected in the BIOS Features Setup screen.

A bootable device is one on which an operating system has been installed, or formatted as a system disk using MS-DOS.

Functionality

The Flash Disk performs identically to a standard IDE hard drive. Reads and writes to the device are performed using the same methods, utilizing DOS command line entries or the file managers resident in the chosen operating system.

Advanced Configuration

The previous discussion is based on using the IDE disk devices formatted as one large partition per device. Some applications may require the use of multiple partitions. The following discussion of these partitions includes special procedures that must be followed to create multiple partitions on the VMIOMAX-8451 IDE disk devices (including the resident Flash Disk).

Partitions may be either a primary or an extended partition. An extended partition may be subdivided farther into logical partitions. Each device may have up to four main partitions; one of which may be an extended partition. However, if multiple primary partitions are created, only one partition may be active at a time. Data in the non-active partitions are not accessible.

Following the creation of the partitioning scheme, the partitions can be formatted to contain the desired file system.

As discussed earlier, a typical system consists of the VMIOMAX-8451 with its resident Flash Disk configured as the Secondary IDE device, a hard drive attached to the Primary IDE interface, and a floppy drive attached to the floppy interface.

Using this configuration, it may be desirable to have a logical device on either IDE device, configured as a bootable device, allowing the selection of the first boot device by way of the Advanced CMOS Setup screen. Using this capability, a user could have a system configured with multiple operating systems that would be selectable by assigning the IDE logical device as the boot device.

The DOS utility FDISK is commonly used to configure the partition structure on a hard drive. Comments on the following page pertain to partitioning efforts using FDISK.

CAUTION: Deleting a partition will erase all the data previously stored in that partition.

The Flash Disk will be configured as a single partition device as delivered from the factory. The following sample sequence illustrates a proven method for creating two 8 Mbyte partitions, with one as an active primary partition. Take note of the instructions to exit FDISK. This has been shown to be an important step in a successful partitioning effort.

1. Power up the VMIOMAX-8451 and enter the CMOS set-up.
2. Set IDE HDD Master to “Not Installed”.
3. Set Flash Disk Master to “AUTO”.
4. Set boot device to floppy.
5. Boot DOS from the floppy, and verify that the System Configuration Screen shows only the Flash Disk.
6. Run FDISK.
7. Delete all current partitions (any data currently stored in the partitions will be lost).
8. Exit FDISK (this will cause a reboot), then run FDISK again.
9. Create an 8 Mbyte primary partition.
10. Create an 8 Mbyte extended partition.
11. Set-up a logical device for the 8 Mbyte extended partition.
12. Set the Primary partition as an active partition.
13. Exit FDISK.

If an operating system has been installed on the Flash Disk that modifies the Master Boot Record (MBR), the following steps are required to rewrite the MBR for DOS.

14. Run FDISK / MBR.
15. Run FORMAT C: (use the extension /s option if you want the Flash Disk as a bootable DOS device).
16. Format D: (this is only required if two partitions were created).
17. Reset the CPU and enter the CMOS set-up.
18. Set Primary Master to “AUTO”.
19. Set boot device to desired boot source.

Understanding the order the operating system assigns drive letters is necessary for these multiple partition configurations. The operating system assigns drive letter C to the active primary partition on the first hard disk (the boot device). Drive D is assigned to the first recognized primary partition on the next hard disk. The operating system will continue to assign drive letters to the primary partitions in an alternating fashion between the two drives. The next logical partitions will be assigned drive letters starting on the first hard drive, lettering each logical device sequentially, until all are assigned a drive letter. The system will then perform the same sequential lettering of each logical partition on the second hard disk.

NOTE: Drive letter changes caused by adding an additional drive or changing the initial partitioning scheme may cause difficulties with an operating system installed prior to the changes. Plan your configuration prior to installing the operating system to minimize difficulties.

Maintenance

Maintenance

This section provides information relative to the care and maintenance of VMIC's products. If the product malfunctions, verify the following:

- Software
- System configuration
- Electrical connections
- Jumper or configuration options
- Boards are fully inserted into their proper connector location
- Connector pins are clean and free from contamination
- No components of adjacent boards are disturbed when inserting or removing the board from the chassis
- Quality of cables and I/O connections

If the product must be returned, contact VMIC for a Return Material Authorization (RMA) Number. **This RMA Number must be obtained prior to any return**.

Contact VMIC Customer Care at 1-800-240-7782, or
E-mail: customer.service@vmic.com .

Maintenance Prints

User level repairs are not recommended. The drawings and tables in this manual are for reference purposes only.

Appendix

A

Connector Pin Definitions

Contents

Introduction

This appendix describes the pin definitions for the connectors on the VMIOMAX-8451. Wherever possible, the VMIOMAX-8451 uses connectors and pinouts typical for any desktop PC. This ensures maximum compatibility with a variety of systems.

Power Connector Pin Definition

Connector type: External power connector (Green Plug) located on the front panel.

Table A-1 Power Connector Pin Definition (E5)

Pin	Signal Name
1	VCC*
2	GND

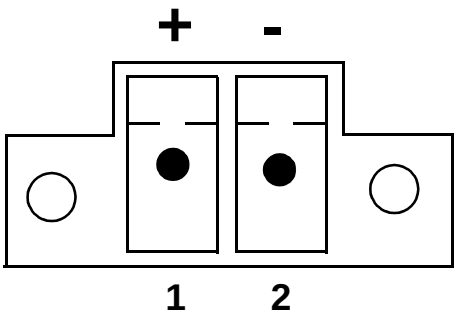


Figure A-1 Front Panel Input Power Connector

* The internal power supply will successfully accept from 8VDC to 28VDC as input power.

COM1 and COM2 Serial Port Connectors and Pin Definitions

Connector type: COM1 and COM2 are male DB9 connectors located on the front panel.

Table A-2 COM1 and COM2 Serial Port Connector Pin Definitions

Pin	Signal Name	Pin	Signal Name
1	DCD	2	RXD
3	TXD	4	DTR
5	GND	6	DSR
7	RTS	8	CTS
9	RI	10	GND

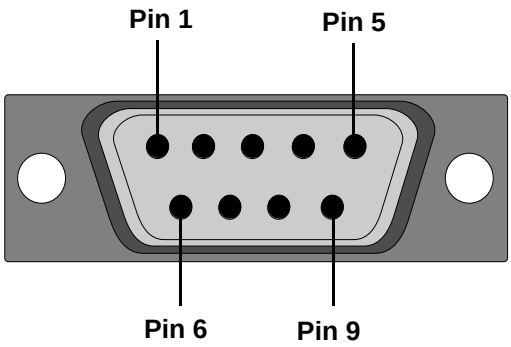
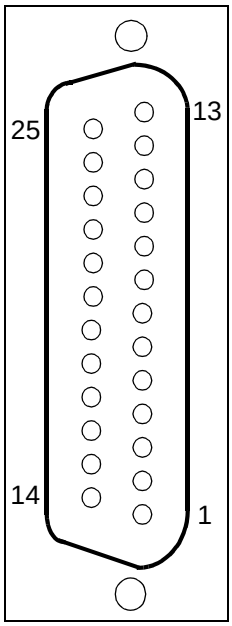


Figure A-2 COM 1 and COM 2 DB9 Connector

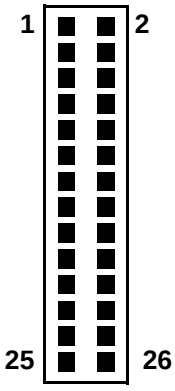
Parallel Port LPT1 (E8) Connector Pin Definition

The parallel port is an internal 26-pin header located on the motherboard. The parallel port uses an adapter to convert the header to a standard DB25 female connector typical of any PC / AT system (VMIC part number 360-000068-012).

Adapter DB25 Connector



Internal 26-Pin Header (Motherboard)

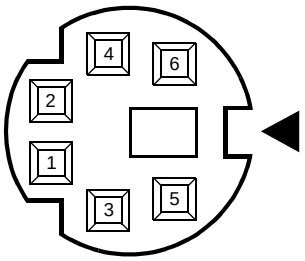


Parallel Port Connector Pinout		
Pin	**Direction**	**Function**
1	In / Out	Data Strobe
2	In / Out	Bidirectional Data D0
3	In / Out	Bidirectional Data D1
4	In / Out	Bidirectional Data D2
5	In / Out	Bidirectional Data D3
6	In / Out	Bidirectional Data D4
7	In / Out	Bidirectional Data D5
8	In / Out	Bidirectional Data D6
9	In / Out	Bidirectional Data D7
10	In	Acknowledge
11	In	Device Busy
12	In	Out of Paper
13	In	Device Selected
14	Out	Auto Feed
15	In	Error
16	Out	Initialize Device
17	In	Device Ready for Input
18		Signal Ground
19		Signal Ground
20		Signal Ground
21		Signal Ground
22		Signal Ground
23		Signal Ground
24		Signal Ground
25		Signal Ground
Shield		Chassis Ground

Figure A-3 Parallel Port Connector Pinout

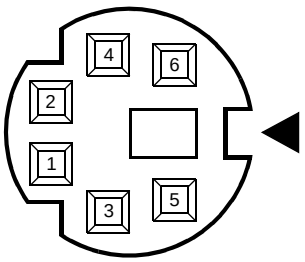
Keyboard and Mouse Connectors and Pin Definitions

The keyboard and mouse connectors are standard 6-pin female mini-DIN PS/2 connectors as shown in Figure A-4.



Keyboard (J3) Connector		
Pin	**DIR**	**Function**
1	In/Out	Data
2		Reserved
3		Ground
4		+5 V
5	Out	Clock
6		Reserved
Shield		Chassis Ground

Figure A-4 Keyboard Connector and Pinout

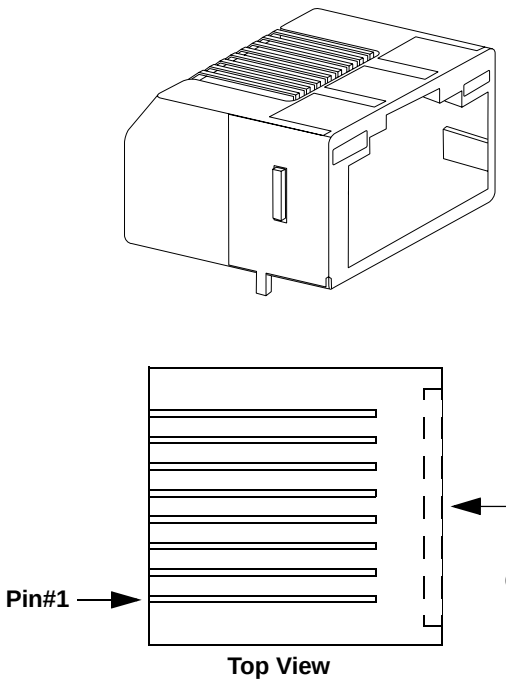


Mouse (J4) Connector		
Pin	**DIR**	**Function**
1	In/Out	Data
2		Reserved
3		Ground
4		+5 V
5	Out	Clock
6		Reserved
Shield		Chassis Ground

Figure A-5 Mouse Connector and Pinout

Ethernet Connector (J9) and Pinout

The pinout diagram for the Ethernet 10BaseT and 100BaseTx connector is shown in Figure A-6.



Ethernet Connector J9 (10BaseT, 100BaseTx)		
PIN	**Signal Name**	
1	TD+	Transmit Data
2	TD-	Transmit Data
3	RD+	Receive Data
4	TX_CT_OUT	Transmit Center Tap Out
5	TX_CT_OUT	Transmit Center Tap Out
6	RD-	Receive Data
7	RX_CT_OUT	Receive Center Tap Out
8	RX_CT_OUT	Receive Center Tap Out

Figure A-6 Ethernet Connector and Pinout

EIDE Connector (E6) and Pin Definition

Connector type: 44-pin header located on the motherboard.

Internal 44-Pin Header (Motherboard)

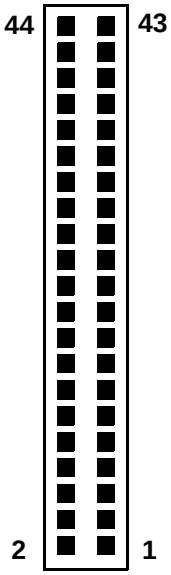


Table A-3 EIDE Connector Pin Definition

Pin	Signal Name	Pin	Signal Name
1	Reset	2	GND
3	Data7	4	Data8
5	Data6	6	Data9
7	Dat5a	8	Data10
9	Data4	10	Data11
11	Data3	12	Data12
13	Data2	14	Data13
15	Data1	16	Data14
17	Data0	18	Data15
19	GND	20	NC
21	DRQ0	22	GND
23	Write	24	GND
25	Read	26	GND
27	Ready	28	PU0
29	DACK0	30	GND
31	IRQ	32	/CS16
33	Address1	34	GND
35	Address0	36	Address2
37	CS1	38	CS3
39	LED	40	GND
41	+5V	42	+5V
43	GND	44	NC

Floppy Connector (E7) and Pin Definition

Connector type: 34 pin Header, 2x17, located on the motherboard.

Internal 34-Pin Header (Motherboard)

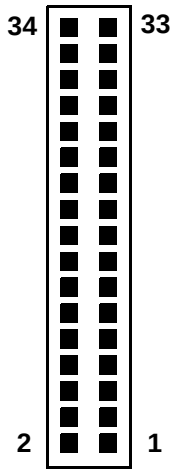


Table A-4 Floppy Connector Pin Definition

Pin	Signal Name	Pin	Signal Name
1	GND	2	Drive Density 0
3	GND	4	N/C
5	GND	6	Drive Density 1
7	GND	8	Index
9	GND	10	Motor Enable B
11	GND	12	Drive Select B
13	GND	14	Drive Select A
15	GND	16	Motor Enable B
17	GND	18	Step Motor Dir
19	GND	20	Step Pulse
21	GND	22	Write Data
23	GND	24	Write Gate
25	GND	26	Track 0
27	GND	28	Write Protect
29	GND	30	Read Data
31	GND	32	Select Head 1
33	GND	34	Disk Change

PC/104-Plus Bus Connector (J5) Pin Definition

NOTES: All VI/O pins are connected to 3.3V by default.
-12V is not supported.

PC/104-Plus Connector (J5)

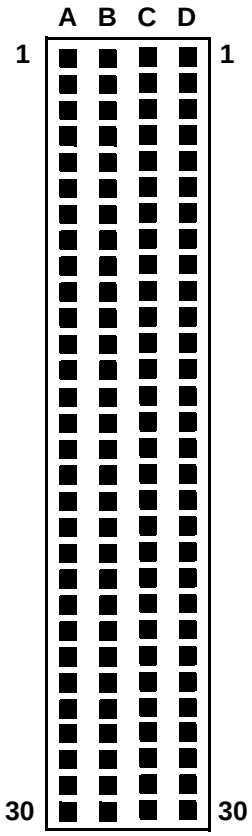


Table A-5 PC/104-Plus Bus (J5) Pin Definition

Pin	Signal A	Signal B	Signal C	Signal D
1	GND	Reserved	+5V	AD00
2	VI/O	AD02	AD01	+5V
3	AD05	GND	AD04	AD03
4	C/BE0	AD07	GND	AD06
5	GND	AD09	AD08	GND
6	AD11	VI/O	AD10	M66EN
7	AD14	AD13	GND	AD12
8	+3.3V	C/BE1	AD15	+3.3V
9	SERR	GND	SB0	PAR
10	GND	PERR	+3.3V	SDONE
11	STOP	+3.3V	LOCK	GND
12	+3.3V	TRDY	GND	DEVSEL
13	FRAME	GND	TRDY	+3.3V
14	GND	AD16	+3.3V	C/BE2
15	AD18	+3.3V	AD17	GND
16	AD21	AD20	GND	AD19
17	+3.3V	AD23	AD22	+3.3V
18	DSEL0	GND	DSEL	DSEL2
19	AD24	C/BE3	VI/O	DSEL3
20	GND	AD26	AD25	GND
21	AD29	+5V	AD28	AD27
22	+5V	AD30	GND	AD31
23	REQ0	GND	REQ1	VI/O
24	GND	REQ2	+5V	GNT0
25	GNT1	VI/O	GNT2	GND
26	+5V	CLK0	GND	CLK1
27	CLK2	+5V	CLK3	GND
28	GND	INTD	+5V	RST
29	+12V	NTA	INTB	INTC
30	+12V	Reserved	Reserved	GND

PC/104 Bus Connector (J6) Pin Definition

NOTE: -5V and -12V are not supported.

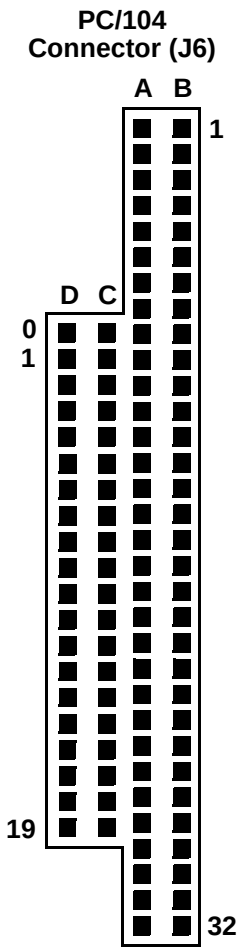


Table A-6 PC/104 Bus Pin Definition (J6)

Pin	Signal A	Signal B	Signal C	Signal D
0			GND	GND
1	IOCHCK	GND	SBHE	MEMCS16
2	D7	RSTDRV	LA23	IOCS16
3	D6	+5V	LA22	IRQ10
4	D5	IRQ9	LA21	IRQ11
5	D4	-5V	LA20	IRQ12
6	D3	DRQ2	LA19	IRQ13
7	D2	-12V	LA18	IRQ14
8	D1	ENDXFR	LA17	DACK0
9	D0	+12V	MEMR	DRQ0
10	IOCHRDY	GND / KEY	MEMW	DACK5
11	AEN	SMEMW	SD8	DRQ5
12	A19	SMEMR	SD9	DACK6
13	A18	IOW	SD10	DRQ6
14	A17	IOR	SD11	DACK7
15	A16	DACK3	SD12	DRQ7
16	A15	DRQ3	SD13	+5V
17	A14	DACK1	SD14	MASTER
18	A13	DRQ1	SD15	GND
19	A12	REFRESH	GND	GND
20	A11	SYSCLK		
21	A10	IRQ7		
22	A9	IRQ6		
23	A8	IRQ5		
24	A7	IRQ4		
25	A6	IRQ3		
26	A5	DACK2		
27	A4	TC		
28	A3	BALE		
29	A2	+5V		
30	A1	OSC		
31	A0	GND		
32	GND	GND		

Appendix

B

Software Installation

Contents

Introduction

The VMIOMAX-8451 provides video and Local Area Network (LAN) access by means of on-board PCI-based adapters, integrated hardware, and associated software drivers. To optimize performance of each of the PCI-based subsystems, install the driver software located on the distribution disc provided with the unit. Detailed instructions for installation of the drivers during the installation of Windows NT 4.0, Windows 2000 Professional, and Red Hat Linux 8.0 operating systems are provided in the following sections.

Microsoft Windows NT 4.0 Software Driver Installation

Ethernet Adapter Driver Disk Preparation

The LAN driver for Windows NT 4.0 is provided on the Windows Drivers CD-ROM. Before beginning the installation of one Windows NT, the appropriate driver files must be copied to a blank, user-supplied floppy disk.

To create a Windows NT LAN driver disk, browse to the \Lan directory on the Windows Drivers CD-ROM. Copy all files in the \Lan directory to a blank floppy disk. Label the disk 'Intel(R) 8255xER Fast Ethernet Adapter'.

Ethernet Adapter Driver Installation

1. Follow the normal Windows NT 4.0 installation until the 'Windows NT Setup' window that states 'Windows NT needs to know how this computer should participate on a network.'
2. Ensure 'This computer will participate on a network' is selected and click 'Next'.
3. Click 'Select from list'. Click 'Have Disk'. Insert the 'Intel(R) 8255xER Fast Ethernet Adapter' driver disk, ensure 'A:\' is entered in the 'Insert Disk' window text box, and click 'OK'.
4. Ensure 'Intel(R) 8255xER Fast Ethernet Adapter' is selected and click 'Next'.
5. Continue through remaining network setup screens, providing data relative to the network configuration.
6. Continue through this portion of Setup. Allow the computer to restart and boot NT.
7. Log on as 'Administrator', using the password provided during setup.
8. Install Service Pack 6a from the CD-ROM provided. Allow the computer to restart and boot NT.
9. Log on as 'Administrator'.

Video Driver Installation

1. If not already present, insert the Windows Drivers CD-ROM.
2. Click 'Start', 'Settings', 'Control Panel'. Double-click on 'Display'.
3. Click on the 'Settings' tab. Click 'Display Type', 'Change', 'Have Disk', 'Browse'. In the 'Look in' pull-down selection menu, select the Windows Drivers CD-ROM. Double-click on the 'Video_NT' folder. Double-click on 'gx_nt40' and click 'OK'.
4. At the 'Change Display' window, click 'OK'. At the 'Third-party Drivers' window, click 'Yes'. Click 'OK'.
5. Click 'Close', 'Close'. Click 'Yes' to restart the computer.

Microsoft Windows 2000 Software Driver Installation

Complete the installation of Windows 2000, following the instructions provided by the Windows 2000 manual. Please note that the computer will restart several times during the Windows 2000 installation process. Leave the Windows 2000 installation CD-ROM in the CD-ROM drive during the entire setup process as additional files may need to be copied.

Ethernet Adapter Driver Installation

1. Insert the Windows Drivers CD-ROM.
2. Right-click on 'My Computer' and select 'Properties'. Click on the 'Hardware' tab. Click 'Device Manager'.
3. Double-click on 'Ethernet Controller' (found under 'Other devices').
4. In the 'Ethernet Controller Properties' window, click on the 'Driver' tab. Click 'Update Driver'. Click 'Next' twice. Ensure 'Specify a location' is the only search location selected (checked) and click 'Next'.
5. In the 'Upgrade Device Driver Wizard' window, click 'Browse'. In the 'Look in' pull-down selection menu, select the Windows Drivers CD-ROM. Double-click on the 'LAN' folder. Double-click on 'oemsetup'. Click 'OK'. Click 'Next'.
6. At the 'Digital Signature Not Found' window, click 'Yes'. After files have been copied, click 'Finish'.
7. At the 'Intel(R) 8255xER PCI Adapter Properties' window, click 'Close'.

Video Driver Installation

1. In 'Device Manager', double-click 'Video Controller (VGA Compatible)', found under 'Other devices'.
2. In the 'Video Controller (VGA Compatible) Properties' window, click on the 'Driver' tab. Click 'Update Driver'. Click 'Next' twice. Ensure 'Specify a location' is the only search location selected (checked) and click 'Next'.
3. In the 'Upgrade Device Driver Wizard' window, click 'Browse'. In the 'Look in' pull-down selection menu, select the Windows Drivers CD-ROM. Double-click on the 'Video_2K' folder. Double-click on 'gx_win2k'. Click 'OK'. Click 'Next'.
4. At the 'Digital Signature Not Found' window, click 'Yes'. At this point, the video display may blink a few times. After files have been copied, click 'Finish'.
5. At the 'National Semiconductor Corporation Win2K Graphics Driver Properties' window, click 'Close'. Click 'X' to close Device Manager. Click 'X' to close the 'System Properties' window.

Red Hat Linux 8.0 Installation

1. Create an installation boot floppy by doing one of the following:

   A. From an existing Windows computer: Insert Disc 1 of the Red Hat Linux 8.0 media set into the CD-ROM drive. Insert a blank floppy into the floppy drive. Open a Command Prompt window. Type 'D:\dosutils\rawrite' and press ENTER. At the 'Enter disk image source file name:' prompt, type 'D:\images\boot.img' and press ENTER. Please note that if the CD-ROM is assigned a drive letter other than D, enter that drive letter in place of 'D'. At the 'Enter target diskette drive:' prompt, type 'A:' and press Enter. At the 'Please insert a formatted diskette into drive A: and press -ENTER-:' prompt, press ENTER to create the Red Hat Linux 8.0 installation boot disk.

   B. From an existing Linux computer: Insert Disc 1 of the Red Hat Linux 8.0 media set into the CD-ROM drive. Insert a blank floppy into the floppy drive. Run the following commands at a shell prompt:

      mount /mnt/cdrom

      dd if=/mnt/cdrom/images/boot.img of=/dev/fd0

      umount /mnt/cdrom

2. Insert Disc 1 and the boot floppy created earlier. Boot from the floppy and begin the installation per the Red Hat Linux 8.0 documentation. Please note that depending on the amount of memory installed on the VMIOMAX-8451, installation may take place in text mode.
3. At the 'Graphical Interface (X) Configuration' screen, the Cyrix MediaGX video adapter with 4MB video RAM should be selected. Click 'Next'.
4. At the 'Monitor Configuration' screen, select a monitor from the list and click 'Next'.
5. At the 'Customize Graphics Configuration' screen, select the desired color depth and resolution. Select 'Text' login type and click 'Next'. The XFree86 will have to be manually configured later before the X server can run properly.
6. Click 'Exit' at the end of the installation. Allow the system to reboot.
7. Login as 'root' using the password provided during installation.
8. Using 'vi' or another text-mode editor, edit /etc/X11/XF86Config as follows:

   A. In the mouse "InputDevice" section, change the line that reads

      Option "Device" "/dev/psaux"

      to

      Option "Device" "/dev/mouse"

   B. In the "Device" section, change the line that reads

      Driver "cyrix"

      to

      Driver "vesa"

   C. Start X with the 'startx' command.

Appendix

C

Embedded Systems BIOS

Introduction

The VMIOMAX-8451 utilizes the BIOS (Basic Input / Output System) in the same manner as other PC / AT compatible computers. This appendix describes the menus and options associated with the VMIOMAX-8451 BIOS.

Entering SETUP

The SETUP screen system is invoked after POST has completed, in response to the following:

- During POST's memory countup display, the console user can press the <DEL> key. This causes the remainder of the memory test to be quickly performed, and then the SETUP screen takes over.

SETUP cannot be entered during normal system operation with a special key combination, because SETUP requires low memory to use for scratch space.

SETUP Screens

A fully-configured SETUP main menu looks like the following:

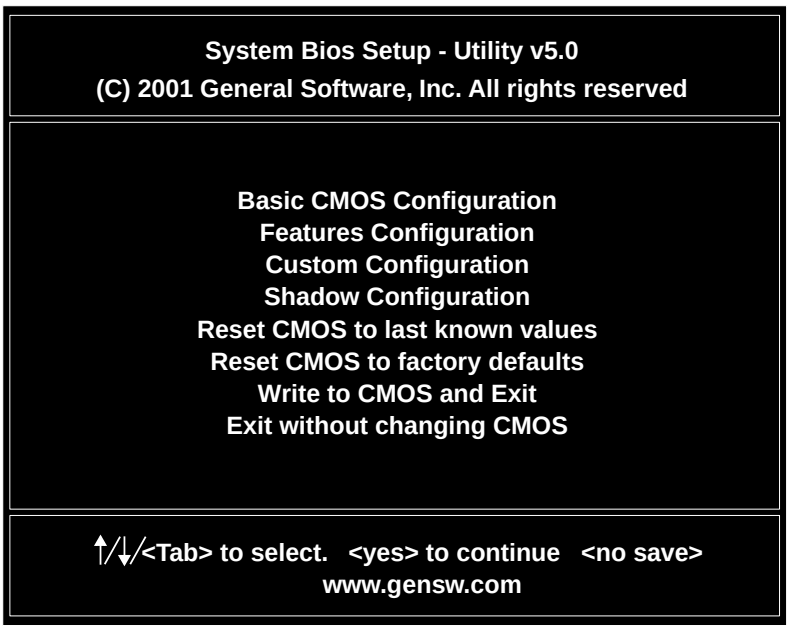


Figure C-1 The Embedded BIOS Setup Menu

NOTE: Getting around in the SETUP screens is normally accomplished by using the arrow keys, pressing ENTER, or even using the TAB or SHIFT-TAB keys.

Basic CMOS Configuration Screen

The system's drive types, boot activities, and POST optimizations are configured from the Basic Setup Screen (Figure C-2). In order to use disk drives with your system, you must select appropriate assignments of drive types in the left-hand column. Then, if you are using true floppy and IDE drives (not memory disks that emulate these drives), you need to configure the drive types themselves in the Floppy Drive Types and IDE Drive Geometry sections. Finally, you'll need to configure the boot sequence in the middle of the screen. Once these selections have been made, your system is ready to use.

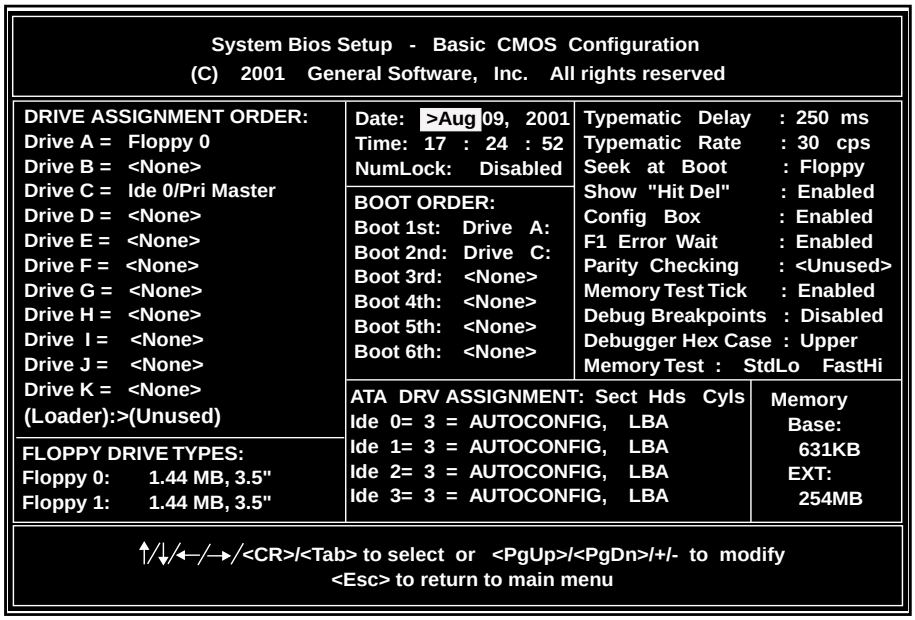


Figure C-2 The Embedded BIOS Basic Setup Screen

Configuring Drive Assignments

Embedded BIOS allows the user to map a different file system to each drive letter. The BIOS allows file systems for each floppy (Floppy0 and Floppy1), each IDE drive (Ide0, Ide1, Ide2, and Ide3), and memory disks when configured (Flash0, ROM0, RAM0, etc.) Figure C-2 on page 105 shows how the first floppy drive (Floppy0) is assigned to drive A in the system, and then how the first IDE drive (Ide0) is assigned to drive C in the system.

CAUTION: Take care to not skip Drive A when making Floppy Disk assignments, as well as Drive C when making hard disk assignments. The first floppy should be drive A, and the first hard drive should be C. Also, ***DO NOT*** assign the same file system to more than one drive letter. Thus, Floppy0 should not be used for both drive A and B. The BIOS permits this to allow embedded devices to alias drives, but desktop operating systems may not be able to maintain cache coherency with such a mapping in place.

Configuring Floppy Drive Types

If true floppy drive file systems (and not their emulators, such as ROM, RAM, or Flash disks) are mapped to drive letters, then the floppy drives themselves must be configured in this section. Floppy0 refers to the first floppy disk drive on the drive ribbon cable (normally drive A), and Floppy1 refers to the second drive (drive B).

Configuring IDE Drive Types

If true IDE disk file systems (and not their emulators, such as ROM, RAM, or Flash disks) are mapped to drive letters, then the IDE drives themselves must be configured in this section. The following table shows the drive assignments for Ide0-Ide3:

File System Name	Controller	Master/Slave
Ide0	Primary	Master
Ide1	Primary	Slave
Ide2	Secondary	Master
Ide3	Secondary	Slave

To use the primary master IDE drive in your system (the typical case), just configure Ide0 in this section, and map Ide0 to drive C in the Configuring Drive Assignments section.

The IDE Drive Types section lets you select the type for each of the four IDE drives: None, User, Physical, LBA, or CHS.

The User type allows you to select the maximum cylinders, heads and sectors per track associated with the IDE drive. This method is rarely used since LBA is now in common use.

The Physical type instructs the BIOS to query the drive's geometry from the controller on each POST. No translation on the drive's geometry is performed, so this type is limited to drives of 512MB or less. Commonly, this is used with embedded ATA PC Cards.

The LBA type instructs the BIOS to query the drive's geometry from the controller on each POST, but then translate the geometry according to the industry-standard LBA convention. This supports up to 16GB drives. ***Use this method for all new drives***.

The CHS type instructs the BIOS to query the drive's geometry from the controller on each POST, but then translate the geometry according to the CHS convention.

CAUTION: Using this type on a drive previously formatted with LBA or Physical geometry might show data as being missing or corrupted.

Configuring Boot Actions

EMBEDDED BIOS supports up to six different user-defined steps in the boot sequence. When the entire system has been initialized, POST executes these steps in order until an operating system successfully loads. In addition, other pre-boot features can be run before, after, or between operating system load attempts. The following actions are supported:

Drive A - D	Boot operating system from specified drive. If "Loader" is set to "BootRecord" or "Unused", then the standard boot record will be invoked, causing DOS, Windows95, Windows 98, Windows ME, Windows 2000, Windows NT, Windows XP, Linux, or other industry-standard operating system to load.
CDROM	Boot from the first IDE CDROM found that contains an El Torito bootable CDROM.
Debugger	Launch the Integrated BIOS Debugger. To return exit the debugger environment, type "G" at the debugger prompt and press ENTER.
MFGMODE	Initiate Manufacturing Mode, allowing the system to be configured remotely via an RS232 connect to a host computer. Not supported on the VMIOMAX-8451.
DOS in ROM	Execute a ROM-resident copy of DOS, if available. This feature is not applicable unless an XIP copy of DOS has been stored in the BIOS boot ROM. Not supported on the VMIOMAX-8451.

Alarm	Generate an alarm by beeping the speaker and sending a signal to a Firmware Application running in the Firmbase environment. The application can perform whatever processing is necessary to handle the alarm, including taking local action, interacting with other tightly-coupled computers, or even notifying other systems on the network, for example. Not supported on the VMIOMAX-8451.
RAS	Enter remote access mode by sending a signal to a Firmware Application running in the Firmbase environment. The application can perform whatever processing is necessary to implement the RAS mode, which is largely defined to mean some state where the system accepts remote connections for normal operation; not specifically for field maintenance. Not supported on the VMIOMAX-8451.
Power Off	Cause the target to switch off its power with a "soft off" feature, and signal Firmware Applications running in the Firmbase environment that power is going down. Not supported on the VMIOMAX-8451.
Reboot	Reboot the target, and send a signal to a Firmware Application running in the Firmbase environment indicating that the target is rebooting. Not supported on the VMIOMAX-8451.
CLI	Enter command line mode by calling a special Board Module Function (BoardPostControl) that can be used to implement an OEM-defined Command Language Interpreter. Not supported on the VMIOMAX-8451.
None	No action; POST proceeds to the next activity in the sequence.

Features Configuration Setup Screen

EMBEDDED BIOS has many high-level features that may be selectively enabled or disabled at run-time in the Features Setup Screen (Figure C-3). This screen contains fields defined for the features that are enabled in the core BIOS. The following features have fields to enable or disable them in this screen:

- POST Memory Manager
- System Management BIOS (SMBIOS / DMI)

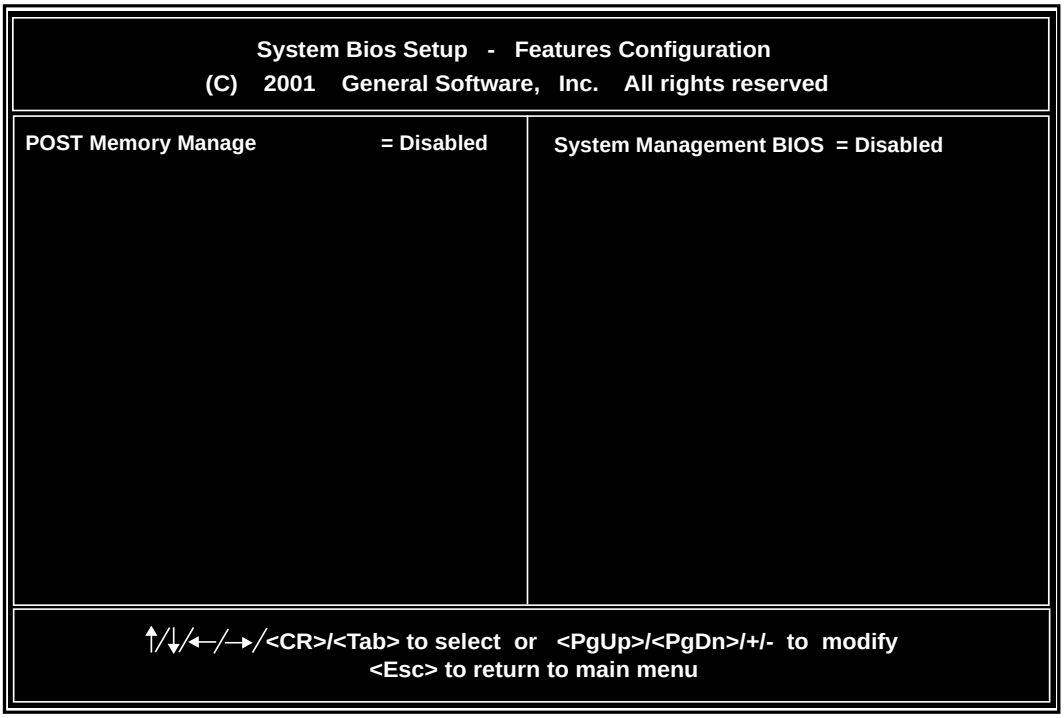


Figure C-3 Embedded BIOS Features Setup Screen

Custom Configuration Setup Screen

The system's hardware-specific features are configured with the Custom Setup Screen (Figure C-4). This screen is for the OEM to define.

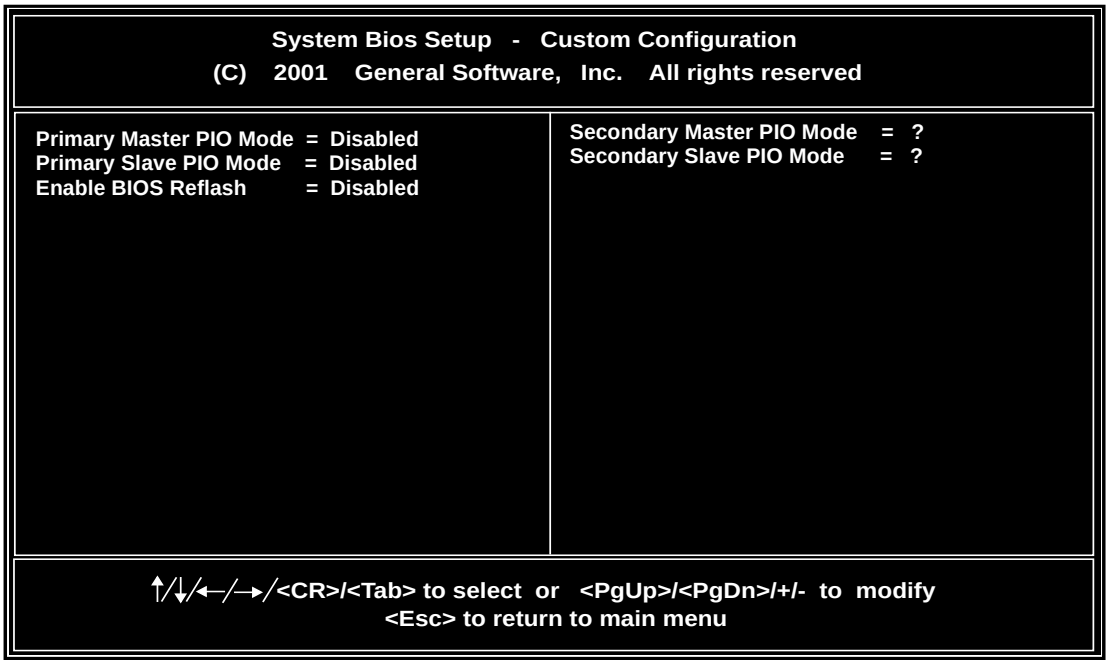


Figure C-4 Embedded BIOS Custom Setup Screen (Custom Configuration)

Shadow Configuration Setup Screen

The system's Shadow Configuration Setup Screen (Figure C-5) allows enabling and disabling of shadowing in 16KB sections, except for the top 64KB of the BIOS ROM, which is shadowed as a unit. Normally, shadowing should be enabled at C000/C400 to enhance VGA ROM BIOS performance, and E000-F000 should be shadowed to maximize system ROM BIOS performance.

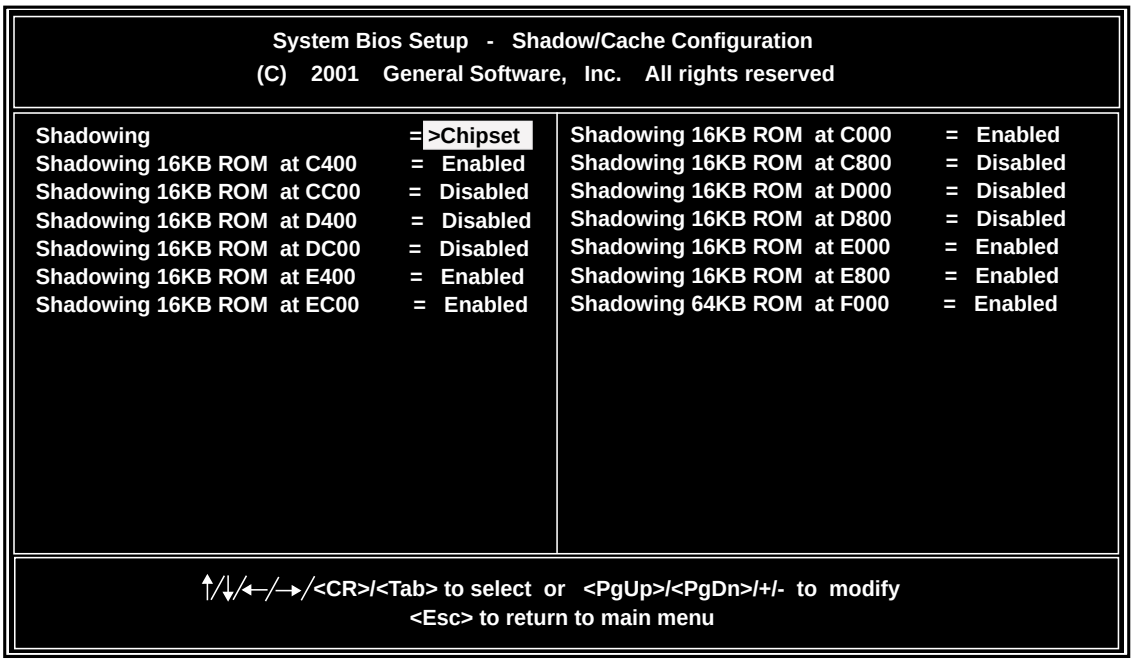


Figure C-5 Embedded BIOS Shadow Setup Screen (ROM Shadowing)

Other Pre-Boot Setup Screens

Embedded BIOS provides other Setup screens to the OEM as well. The following are available:

- RESET CMOS TO LAST KNOWN VALUES - This option causes SETUP to restore the values it had prior to any edits performed during the current SETUP session.
- RESET CMOS TO FACTORY DEFAULTS - This option causes SETUP to reset CMOS with the values that are defined in INC\CONFIG.INC as factory defaults.
- WRITE TO CMOS AND EXIT - This option causes SETUP to save the current edits to CMOS and reboot the target, causing the new edits to take effect.
- EXIT WITHOUT CHANGING CMOS - This option causes SETUP to reboot the target without saving any changes made during the SETUP session.

Appendix

D

Sample C Software

Introduction

This appendix provides listings of a library of sample code that the programmer may utilize to build applications. These files are provided in the directory "\sample code" on CD 320-500077-000, labeled "Windows Drivers", included with the VMIOMAX-8451.

Directory \Include

This directory contains common files required to compile several of the sample code applications.

Directory \Support

This directory contains memory and PCI access routines used by many of the sample code applications.

Directory \Timers

This directory contains code used to test the functions of the VMIC-designed FPGA such as timers, SRAM controller and Watchdog Timer.