OPERATING MANUAL

MODEL 4284

TMS320C40 Digital Signal Processor
MIX Baseboard for VMEbus Systems

Pentek, Inc.
One Park Way
Upper Saddle River, NJ 07458
(201) 818-5900
http://www.pentek.com/

Manual Part No: 800.42840 Rev: F.2 – July 26, 2000

Pentek Model 4284 Operating Manual – Revision History

Date	Rev	Applicable Serial #'s	Comments
01/03/94	Pre.	9401003 – 9441043	Initial Product Release – Manual generated by Engineering Dept.
06/03/94	A	9401003 – 9441043	Complete rewrite, reformat and update. Changed pinout of TCK connector. Added Software examples and Appendices.
06/10/94	A.1	9401003 – 9441043	Added header file listings to sample 4284 setup code in Appendix A, Section A.1.
07/08/94	B	9401003 – 9441043	Listed Pentek part # for ordering Comm Port cables. Minor change to detail block diagram. Add section 4.8 on memory usage & table of transfer times.
10/11/94	B.1	9401003 – 9441043	Changes to Tables 4–8 to properly represent transfer timing for 50 MHz 'C40. Delete Table 4–9 & refer to 'C40 Manual for DMA timing info. Changes to Sections 2.11.4 and 3.2.1, regarding which signals are inputs and which are outputs.
11/11/94	B.2	9401003 – 9441042	Correct pin numbering error in Table 2–12. System Clock jumper installed between pins 1–2, not 11–12.
12/29/94	C	9441043 – Forward	New features: Flash EEPROM and Turbo–MIX. Properly identify jumper blocks SW1 and SW2 in Figure 2–2. Expanded discussion of Reset Operation. Add more transfer timing tables for 40 and 50 MHz parts, using block repeat, DMA and Turbo–MIX. Descriptions of these tables updated. Turbo–MIX example added to Appendix A. Flash EEPROM programming code added to Appen–dix A. Update Boot Code in Appendix B.
01/27/95	C.1	9441043 – Forward	Add info re:Option 010, in which A32 address is independent of A24 address, and Memory Window jumpers affect A24 window only. Correct Table 2–2 (A16 address 8700 is reserved). A24 inhibit bit also prevents A32 access. This change affects Section 2.4.3, Section 3.11.2.4, and Table 3–12.
01/05/96	C.2	9441043 – Forward	Mention FTL in Section 1.9 (Software Support). Add info on mating connector for front panel Serial and GPIO connectors (Sec. 2.11). Add Section 2.14, describing blinking of Front Panel LED on power–up. Correction to note in Section 4.2.1, reference to EPROM rev. should be Revision E, not B. Indicate that use of the Linker Command File shown in Sections 4.3.1 and A.6 requires a change to .main. Changes to Hex File Loader & Starter in Section A.7.
09/10/96	D	9441043 – Forward	Sec. 3.8.5: 'C40 Int. to VME Status bit in Transfer Mode Reg. is accessible to the 'C40 (not VME) for read only. Add Tables & Figures to Table of Contents. Update MIX Appendix to Rev. F.1.
07/02/97	D.1	9441043 – Forward	IACK Vector Register must be re–written to clear VME interrupts generated by 4284. This infor–mation added as a Note in Section 3.11.1, and to text in Section 4.6.1.
09/22/97	D.2	9441043 – Forward	Section 4.2.1 – The code that triggers a boot from Flash EEPROM is A55A 5AA5, not A55A A55A – this changes the last paragraph on pg. 52, the first paragraph on pg. 53, Table 4–4, and the first paragraph on page 57.
05/04/98	E	9441043 – Forward	Add Parallel C to Software Development Support section (1.9). Replaced board diagram with VSB ver–sion of board. Added section 3.12 on VSB. Modified Table 3–7 to include VSB addresses.
05/21/98	E.1	9441043 – Forward	Revisions to VSB section (3–12) – renamed Table 3–14 (VSB Control Register) and renamed some bits in the table. Add Tables 3–15 – 3–17 illustrating bit field usages in VSB Control Register. Add VSB interface to Detailed Block Diagram (Fig. 4–1). In SW Development Support section (1.9) Parallel C is now called Diamond.
7/23/98	E.2	9441043 – Forward	Correct & cross reference to Table 3–16 in Table 3–14.
10/12/98	E.3	9441043 – Forward	Correct address range for VME Master Access in Table 3–7 (0xB000 0000 – 0x**B**FFF FFFF)
10/22/98	F	9441043 – Forward	Regardless of DRAM Window size, DRAM must be mapped on an address bounmdary that is a multiple of the actual DRAM size. This changes text in Secs. 2.4.2 and 2.4.3, and Tables 2–5 – 2–7, and deletes Tables 2–8 & 2–9 (these tables all concerned base address vs. window size). Add note to Sec. 2.4.3 indi–cating that for DRAM access, each 'C40 address is equivalent to four VME addresses. Reinforce same concept in Sec. 4.10. Give VME address of DRAM in Sec. 3.10 & add a note to the 'C40 memory Map stat–ing that DRAM is shared w/ VME. Sec. 4.2.3 – Host Control Reset does NOT reset MIX or VSB. Change timer address in Sec. A.4 (1st executable line under main()) to 0x100020. Remove MIX Appendix & refer–ences to it (this info now included in 800.00001).
12/06/99	F.1	9441043 – Forward	Sec. 1.9 – Removed SPOX, FTL & Diamond from Software Support list. Corrections to Tables 2–5 thru 2–7, which were correct up until last Rev. Sec. 2.1.2.1 – Comm Port Cable is Model 2104, not 2014. Table 4–3: Removed extra '0' in last row of address column. Added NOTEs to Sec's 4.2.4 & 4.3.5 about initial–izing the Local Control Register if you do not boot from the factory EPROM. Remove Sec. 4.4.2 (Down–loading SPOX programs), and the sentence referencing SPOX in Sec. 4.4
7/26/00	F.2	9441043 – Forward	Sec. 1–12: delete reference to Opt. 060. Additional corrections to Tables 2–5 thru 2–7. Update Boot Code listing in Appendix B.

References made in this manual to the Texas Instruments **TMS320C4x User's Guide** are made specifically to Revision C of that publication, dated August 1993.

WARRANTY

Pentek warrants that all products manufactured by Pentek conform to published Pentek specifications and are free from defects in mate–rials and workmanship for a period of one year from the date of delivery when used under normal operating conditions and within the service conditions for which they were furnished.

The obligation of Pentek arising from a warranty claim shall be limited to repairing or at its option, replacing without charge, any product which in Pentek's sole opinion proves to be defective within the scope of the warranty.

Pentek must be notified in writing of the defect or nonconformity within the warranty period and the affected product returned to Pentek within thirty days after discovery of such defect or nonconformity.

Buyer shall prepay shipping charges, taxes, duties and insurance for products returned to Pentek for warranty service. Pentek shall pay for the return of products to buyer except for products returned from another country.

Pentek shall have no responsibility for any defect or damage caused by improper installation, unauthorized modification, misuse, neglect, inade–quate maintenance, accident or for any product which has been repaired or altered by anyone other than Pentek or its authorized representatives.

The warranty described above is buyer's sole and exclusive remedy and no other warranty, whether written or oral, is expressed or implied. Pentek specifically disclaims fitness for a particular purpose. Under no circumstances shall Pentek be liable for any direct, indirect, special, incidental or consequential damages, expenses, losses or delays (including loss of profits) based on contract, tort, or any other legal theory.

Table of Contents

Page

Chapter 1: Overview

Chapter 2: Installation and Connections

Table of Contents

Page

Chapter 2: Installation and Connections *(continued)*

Table of Contents

Page

Chapter 3: Memory Maps and Register Descriptions

Table of Contents

Table of Contents

Page

Chapter 4: Working with the Model 4284

Table of Contents

Page

Chapter 4: Working with the Model 4284 (continued)

Appendix A: Programming Examples

Appendix B: Boot EPROM Source Code Listing

Chapter 1: Overview

1.1 General Description

The Model 4284 MIX Baseboard for VMEbus systems is based on the Texas Instruments TMS320C40 Floating Point Digital Signal Processor. The 4284's 'C40 acts as both a VMEbus Master and a MIX bus Master. It serves as a powerful 40 or 50 MFlop Digital Signal Processor, as well as a complete DMA controller.

This manual will describe the installation and operation of the Model 4284 in several typical VMEbus environments.

1.2 MIX Bus Support

The Model 4284's MIX interface supports high−speed, multi−master control and data transfer to any of Pentek's MIX modules. Mapped directly into the 'C40's local bus, this 32−bit data channel includes support for full interrupt handling and generation, to ensure optimum real−time performance.

1.3 TMS320C40 Comm Port Interface

The six high−speed Communications Ports of the 'C40 are buffered, and brought out to convenient front panel connectors. Any MIX module stacked on the 4284 thus becomes Comm Port compatible. In this manner, the 4284 acts as a Comm Port based I/O controller for other 'C40 systems.

1.4 Flexible VMEbus Control

The Model 4284 has master, slave and system controller capabilities on the VMEbus. As a bus master, it can read from and write to the entire 32−bit address space of the VMEbus, thus accessing any external slave device. As a system controller, it can act as a bus arbiter for multiple−master systems.

1.5 Local and Global Static RAM

Two separate banks of SRAM are provided for the 'C40, one on its local bus and the other on the global bus. Each bank may be either 1 or 2 MBytes deep, depending upon the options ordered. These resources maximize the use of the 'C40's dual−bus architecture and its ability to conduct data and program cycles in parallel on the two busses. Both the local and global SRAMs operate with zero wait state performance.

1.6 Dual Access DRAM

A 4, 8 or 16 MByte dual−access DRAM provides an extremely powerful structure for passing data and programs between the 'C40 and the VMEbus. From the VMEbus, this resource appears as relocatable slave memory in A24 or A32 address space. From the 'C40, it is mapped directly onto the global bus.

1.7 Local EPROM

In order to support nonvolatile storage of programs or data, the Model 4284 is equipped with a user−programmable 32 kByte EPROM, located on the 'C40's local bus. Ideal for embedded systems, that self−boot from power−up, this memory is also used for system firmware.

1.8 Flash EEPROM Option

The Model 4284 may optionally be equipped with a 128 kByte (Option 002) or 512 kByte (Option 003) in−circuit programmable Flash EEPROM. The Flash EEPROM is mapped as a 'C40 Local Bus resource, and code programmed into it may be executed at boot time.

1.9 Software Development Support

No DSP hardware product offering is complete without a full complement of development software. The Model 4284 software products are described briefly below.

SwiftTools Development Environment

Pentek's SwiftTools is a complete software development environment for PC−AT and SUN workstations. It is tailored to work directly with Pentek's family of digital signal processing products which incorporate TMS320C40 and TMS320C30 digital signal processors. SwiftTools integrates all of the operations involved in a typical software development project and supports full C language source code generation and debugging with a comprehensive suite of powerful tools for a wide range of applications. SwiftTools includes an in−line assembler and disassembler for on−screen changes to object code in RAM. Registers can be examined and loaded, and memory regions can be examined, filled, and moved. Full file uploading and downloading routines are provided and provisions for EPROM byte manipulation are included.

SwiftNet Communications Protocols

Pentek's SwiftNet is a software product that supports a network of distributed VMEbus systems connected via Ethernet to a host computer, such as a SUN workstation or PC−AT. All software development tools are run on the host with remote target access provided transparently to the user. SwiftNet utilizes the industry standard TCP/IP interface for Ethernet, making it quite portable across many Operating System environments.

1.9 Software Development Support (continued)

Macro Assembler/ Linker The assembler translates assembly language source code into machine language object files in common object file format (COFF). The linker section combines the COFF object files into an executable object module. The archiver supports a macro library accessible by the assembler.

C Compiler The compiler is a full implementation of Kernighan and Ritchie C language, generating assembly language compatible with the Macro Assembler/Linker. Time critical assembly language routines are callable within the C program.

Code Composer Go–DSP's Code Composer is a fully integrated development environment with advanced features specifically tuned for DSP code Designers. It allows code designers to edit, build, manage projects, debug and profile from a single application.

XDS–510 Emulator The Texas Instruments XDS510 Emulator allows the user to access the 'C40 through a 'back door' port provided by a special 14–pin connector on the board. The emulator board is installed in a PC/AT computer where all software development can take place.

Bus Adapters Pentek offers bus adapters to provide a connecting link between the computer your 'C40 programs will be developed on and the VMEbus card cage where those programs will be run. Supported platforms include PC/AT compatibles, SUN SPARCStations and HP Workstations.

1.10 VSB Interface (Option 012)

The VME Subsystem Bus (VSB) is a secondary bus utilizing the outer rows of pins (rows A and C) on the VME P2 connector, which are unused in the standard VMEbus implementation. The Model 4284 operates as a VSB master and interrupt handler using the VSB1400A/B chip set from PLX Technologies.

1.11 Block Diagram

A simplified block diagram of the Model 4284 TMS320C40 DSP/MIX Baseboard for VMEbus systems is presented as Figure 1–1, below. More detail can be found in the diagram presented as Figure 4–1, on page 58 of this manual. The elements of these diagrams will be discussed in detail in Chapter 4 of this manual.

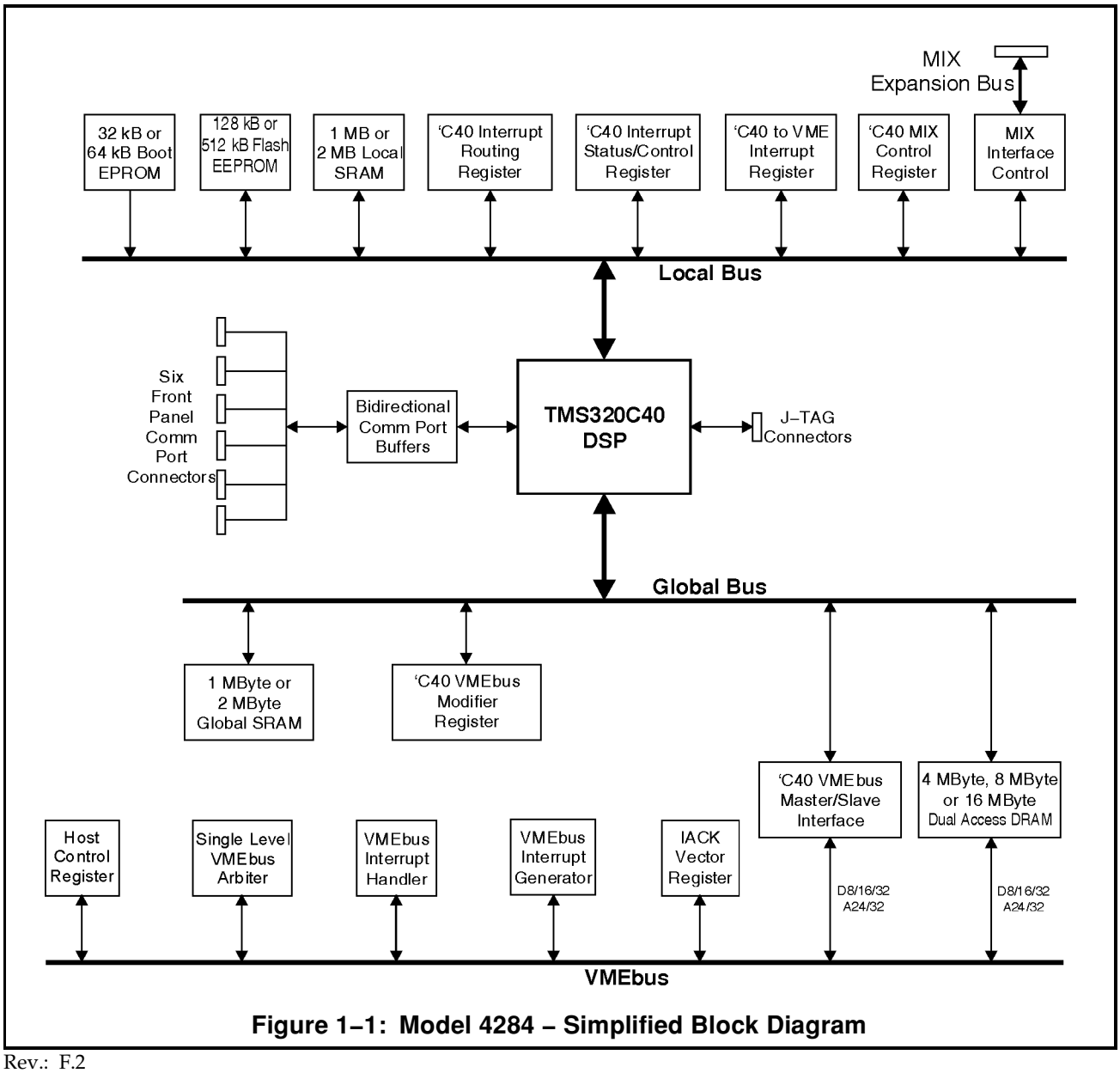

Figure 1–1: Model 4284 – Simplified Block Diagram

1.12 Specifications

Processor

Type:	Texas Instruments TMS320C40
Clock Speed	
Standard:	40 MHz
Option 015:	50 MHz
Address Bus:	32 Bits
Data Bus:	32 Bits

Dual–Access DRAM

Size	
Standard:	4 MBytes (1M x 32)
Option 007:	8 MBytes (2M x 32)
Option 008:	12 MBytes (3M x 32)
Option 009:	16 MBytes (4M x 32)
Arbitration:	Hardware, fully transparent
'C40 Access:	Memory mapped on Global bus, 3 wait states
VMEbus Access:	4, 8, 12 or 16 MBytes of VMEbus slave memory Relocatable on 1 MByte boundaries, 170 ns $\overline{\text{DTACK}}$ delay

Local Static RAM

Size	
Standard:	256k x 32 (1 MByte)
Option 005:	512k x 32 (2 Mbyte)
'C40 Access:	Memory mapped on local bus, 0 wait states

Global Static RAM

Size	
Standard:	256k x 32 (1 MByte)
Option 006:	512k x 32 (2 Mbyte)
'C40 Access:	Memory mapped on global bus, 0 wait states

EPROM

Size	
Standard:	32 kBytes
Option 016:	64 kBytes
'C40 Access:	Memory mapped on Local Bus, 1 wait state

1.12 Specifications (continued)

Flash EEPROM (optional)

Size

Option 002:	128 kBytes
Option 003:	512 kBytes
'C40 Access:	Memory mapped on Local Bus, 1 wait state

VMEbus Compliance

Bus Master:	D32 A32 I(1–7) IH(1–7)
Slave:	D32 A32
Slot 1 Controller:	Level 3 Arbiter, can drive SYSCLK and $\overline{\text{SYSRST}}$

MIX Bus Compliance

Type:	Conforms to Intel MIX Specifications, Master/Slave, Interrupt generator and handler
Width:	32–bits for data and address
Speed:	16 MBytes/sec typical

J–Tag XDS Support: 14–pin Connector for Texas Instruments XDS–510 J–Tag Emulator

Serial I/O Support: Front Panel 10–pin connector for RS–232 level signals.

Misc. I/O Support: Front Panel 10–pin connector for timers, flags, etc.

VME Subsystem Bus (VSB) Interface: (Option 012 ONLY!)

Master, Interrupt Handler

Address Bus Width:	32 bits
Data Bus Width:	32 bits

Physical

Dimensions:	6U Eurocard – (160 mm x 233.35 mm) Multilayer construction.
Power:	+ 5 V_{DC} @ 3A max. – 5 V_{DC} @ 0A (for MIX modules only) +12 V_{DC} @ 0A (for MIX modules only) –12 V_{DC} @ 0A (for MIX modules only)

Chapter 2: Installation and Connections

2.1 Inspection

After unpacking the unit, inspect it carefully for possible damage to connec–tors or components. If any damage is discovered, contact Pentek immediately at (201) 818–5900. Please save the original shipping container and packing material in case re–shipment is required.

2.2 Introduction

This section contains information and instructions for the configuration and various modes of operation of the Model 4284. These are achieved by setting jumper blocks on the board, before it is installed in the VMEbus card cage.

The Model 4284 supports up to three expansion modules, which may be attached to its MIX stacking connector. These modules and the MIX interface system are described in the manual entitled "MIX Tutorial for VMEbus Systems", included with your ship–ment. Prior to attaching any expansion modules you should successfully install the Model 4284 by itself. This makes access to the jumper blocks and diagnosis of installa–tion problems easier.

2.3 Jumper Block Locations

A drawing of the Model 4284's the circuit board, showing the location of jumper blocks referred to in the sections below, is shown in Figure 2–2, on the following page.

Figure 2–1, below, shows the pin numbering for double–row jumper blocks used in the Model 4284. Pin 1 of the jumper blocks is indicated by the line extending from the box that surrounds the header. All of the jumper blocks on the 4284 are right–angle head–ers, located near the edges of the board. This makes them accessible without removing the DRAM mezzanine card or any attached MIX modules.

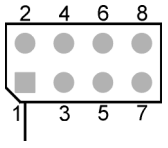

Figure 2–1: Jumper Block Pin Numbering

For most of the jumper blocks on the Model 4284, each jumper position represents a bit. Installing a jumper in a given bit's position sets that bit to a '0'. Leaving the jumper off sets the bit to a '1'.

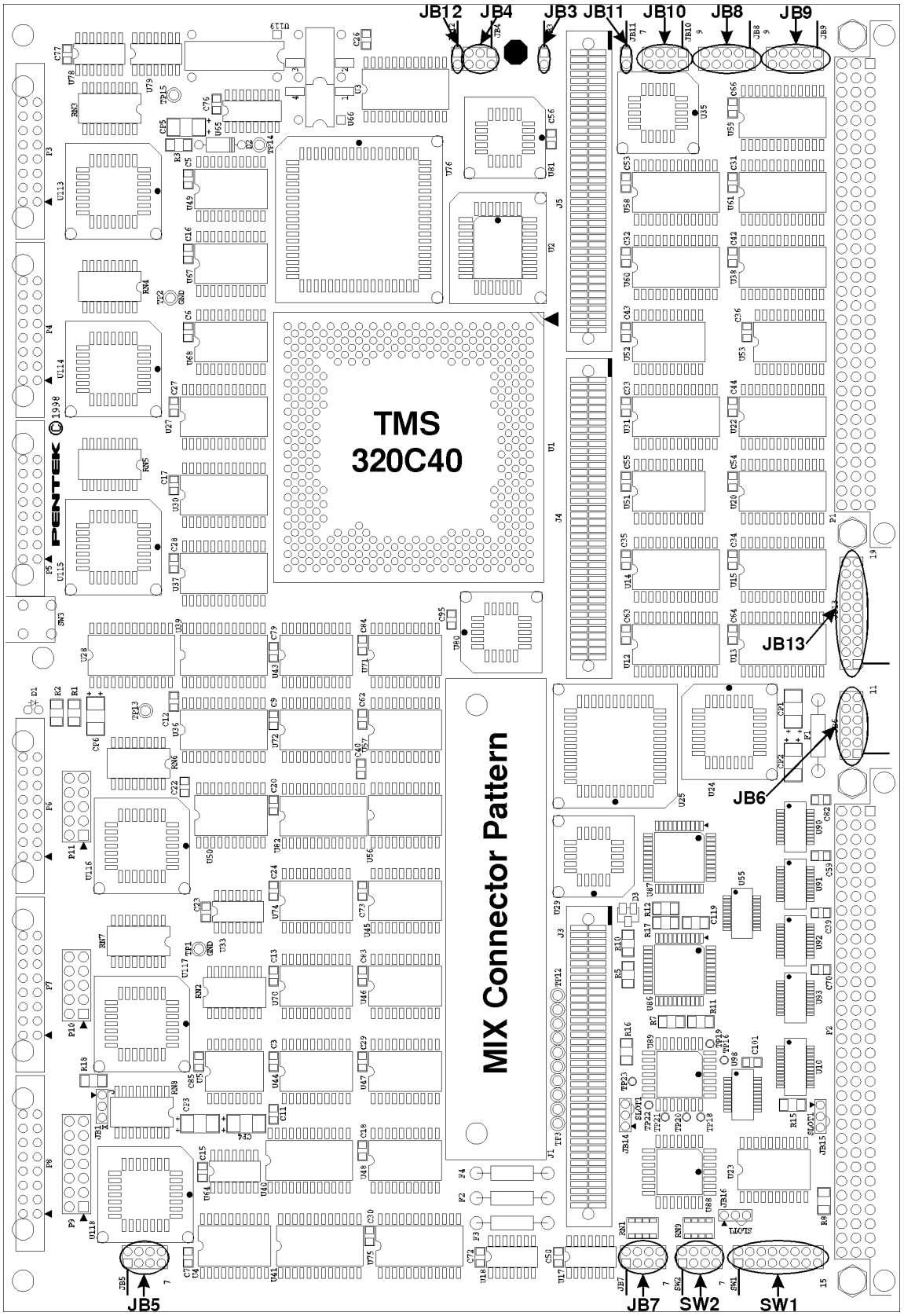

Figure 2-2: Model 4284 Circuit Board Showing Jumper Block Locations, Option -012 (VSB) shown.

2.4 VMEbus Slave Base Address Jumper Settings

The VMEbus Slave resources available on the Model 4284 are the dual–access DRAM (shared with the 'C40), accessible in A24 or A32 address space, and the Host Control Register and IACK Vector Register, both accessed in A16 space. The base addresses for these resources are determined by the placement of shorting jumpers on the jumper blocks described in the sections below.

2.4.1 **A16 Base Address** – Jumper Block JB6

The Host Control Register allows a VMEbus Bus Master to control the reset and interrupt lines into the 'C40. The IACK Vector Register is an 8–bit register loaded from the VMEbus which stores a vector that the Model 4284 sends in response to an interrupt acknowledge (IACK) cycle.

Both of these registers are mapped into the VMEbus 16–bit address space, for Address Modifier Codes 29 and 2D. The addresses at which they reside are selected by the placement of shorting jumpers on jumper block JB6. If there is more than one Model 4284 in a card cage, these jumpers can be set to uniquely configure the addresses for each board. The address set by this jumper block is referred to as the A16_base address.

Jumpers should ALWAYS be installed in positions 9–10 and 11–12 on jumper block JB6. For the remaining positions, pins 1–2 set the value of address bit A8, pins 3–4 set A9, pins 5–6 set A10, and pins 7–8 set the value of A15. Any A16 address on the VMEbus is compared with the settings of these jumpers to determine if this is the board to be accessed.

For any given board, the IACK Vector Register is mapped at the A16_base address and the Host Control Register is mapped at A16_base + 0x0004. Table 2–1, below, describes the address bits that are set by this jumper block, and their hexadecimal weights. Table 2–2, on the next page, lists the 16 available A16 Base Addresses, and the jumper settings used to obtain each of them. Note that one of the 16 possible combinations (all four jumpers removed) is not a valid selection.

Table 2–1: Model 4284 – A16 Base Address – Jumper Block JB6 Address Bits and Hexadecimal Weights of Jumper Positions

Pins 11 – 12	Pins 9 – 10	Pins 7 – 8	Pins 5 – 6	Pins 3 – 4	Pins 1 – 2
Jumpers must always be installed on these pins		A15 (0x8000)	A10 (0x0400)	A9 (0x0200)	A8 (0x0100)

2.4 VMEbus Slave Base Address Jumper Settings (continued)

2.4.1 A16 Base Address (continued)

Table 2–2: Model 4284 – A16 Base Address – Jumper Block JB6 Jumper Settings for Available Base Addresses

Hex Address	Pins 7 – 8	Pins 5 – 6	Pins 3 – 4	Pins 1 – 2
***0x0000**	ON	ON	ON	ON
0x0100	ON	ON	ON	OFF
0x0200	ON	ON	OFF	ON
0x0300	ON	ON	OFF	OFF
0x0400	ON	OFF	ON	ON
0x0500	ON	OFF	ON	OFF
0x0600	ON	OFF	OFF	ON
0x0700	ON	OFF	OFF	OFF
0x8000	OFF	ON	ON	ON
0x8100	OFF	ON	ON	OFF
0x8200	OFF	ON	OFF	ON
0x8300	OFF	ON	OFF	OFF
0x8400	OFF	OFF	ON	ON
0x8500	OFF	OFF	ON	OFF
0x8600	OFF	OFF	OFF	ON
0x8700	Reserved			
* – Factory deault setting				

2.4.2 VME DRAM Window Size Jumpers – Jumper Block JB7

Jumper block JB7 is used to set the size of the area occupied by the Model 4284's DRAM in VME A24 or A32 Address space. These jumpers are set at the factory for the memory installed in your unit. However, the installed memory size is an important parameter to consider when setting the Base Address. This is because the memory must be mapped on address boundaries that are multiples of the window size. For example, for the standard (4 MByte) DRAM size, only four A24 base addresses (0x00 0000, 0x40 0000, 0x80 0000 and 0xC0 0000) are valid.

If you have 16 Mbytes of DRAM in your Model 4284 (Option 009), and are restricted to the use of A24 address space (i. e., your card cage or cage controller has no VME P2 connectors), then the 4284's DRAM occupies ALL of your available memory. In this case, you may wish to reduce the VMEbus' window on the DRAM by rearranging the jumpers on JB7 for less apparent memory. This makes the upper portion of the DRAM inaccessible by VME, but it is still available to the 'C40.

2.4 VMEbus Slave Base Address Jumper Settings (continued)

2.4.2 VME DRAM Window Size Jumpers (continued)

For the 12 Mbyte DRAM size (Option 008), if all of DRAM is to be available to the VMEbus, set these jumpers for a 16 MByte window.

Table 2–3, below, tells which of JB7's jumper positions corresponds to which address bit, and summarizes the settings of the DRAM window size jumper block.

Table 2-3: Model 4284 - VMEbus DRAM Window Size - Jumper Block JB7

DRAM Window	Pins 7 – 8 (A23)	Pins 5 – 6 (A22)	Pins 3 – 4 (A21)	Pins 1 – 2 (A20)
1 Megabyte	ON	ON	ON	ON
2 Megabytes	ON	ON	ON	OFF
4 Megabytes	ON	ON	OFF	OFF
8 Megabytes	ON	OFF	OFF	OFF
16 Megabytes	OFF	OFF	OFF	OFF

NOTES:

1) Option 010 for the Model 4284 handles the Memory Window size issue differently. On units equipped with this option, jumper block JB7 sets the Window Size for A24 space ONLY. All of the DRAM is available in A32 space in these units, regardless of the settings of JB7.

2) Reducing the DRAM Window size does not change the requirement that the DRAM must be mapped on a VME address boundary that is a multiple of the full DRAM size. In A24 space, the standard (4 MByte) DRAM can only be mapped at addresses 0x00 0000, 0x40 0000, 0x80 0000 or 0xC0 0000. The Option 007 (8 MByte) DRAM must have a base address of 0x00 0000 or 0x80 0000, and the Option 008 (12 MByte) and 009 (16 MByte) DRAMs must be mapped at VME address 0x00 0000 in A24 space. See Section 2.4.3, which begins at the top of the next page, for information about setting the DRAM base address.

2.4 VMEbus Slave Base Address Jumper Settings (continued)

2.4.3 VME DRAM Base Address – Jumper Blocks SW1 and SW2

The Base Address used by VMEbus Masters when accessing the DRAM on the Model 4284 is determined by the settings of jumper blocks SW1 and SW2. SW2 sets the upper four bits (A20 – A23) of the A24 Base address. The upper eight bits of the A32 base address (A24 – A31) are set by SW1. Like the jumper blocks discussed above, an installed jumper in a given bit's position sets that bit to '0', and an absent jumper sets the bit to a '1'.

Table 2-4: Model 4284 - A24 DRAM Base Address Jumper Block SW2 - Address Bits and Hex Weights of Jumper Positions

Pins 7 – 8	Pins 5 – 6	Pins 3 – 4	Pins 2 – 1
A23 (0x80 0000)	A22 (0x40 0000)	A21 (0x20 0000)	A20 (0x10 0000)

Table 2–4, above, lists the address bits set by jumper block SW2 and their hex weights. Table 2–5, below, summarizes all valid A24 DRAM base addresses that may be set with SW2. Note that the DRAM **MUST** be mapped on an address boundary that is a multiple of the full size of the DRAM, regardless of the settings of the DRAM Window jumpers.

Table 2–5: Model 4284 – A24 Base Address – Jumper Block SW2 Jumper Settings for Valid Base Addresses – 4 MByte DRAM

Hex Address	Pins 7 – 8 (A23)	Pins 5 – 6 (A22)	Pins 3 – 4 (A21)	Pins 1 – 2 (A20)
***0x00 0000**	ON	ON	OFF	OFF
0x40 0000	ON	OFF	OFF	OFF
0x80 0000	OFF	ON	OFF	OFF
0xC0 0000	OFF	OFF	OFF	OFF
* – factory default setting				

2.4 VMEbus Slave Base Address Jumper Settings (continued)

2.4.3 VME DRAM Base Address (continued)

Some of the addresses listed in Table 2–5 are not valid for larger DRAM sizes. Table 2–6 and Table 2–7, below, list the jumper settings for valid VME DRAM base addresses when the DRAM size is greater than 4 MBytes.

Table 2–6: Model 4284 – A24 Base Address – Jumper Block SW2 Jumper Settings for Valid Base Addresses – 8 MByte DRAM

Hex Address	Pins 7 – 8 (A23)	Pins 5 – 6 (A22)	Pins 3 – 4 (A21)	Pins 1 – 2 (A20)
*0x00 0000	ON	OFF	OFF	OFF
0x80 0000	OFF	OFF	OFF	OFF
* – Factory Default Setting				

Table 2–7: Model 4284 – A24 Base Address – Jumper Block SW2 Jumper Settings for Valid Base Address – 12 or 16 MByte DRAM

Hex Address	Pins 7 – 8 (A23)	Pins 5 – 6 (A22)	Pins 3 – 4 (A21)	Pins 1 – 2 (A20)
*0x00 0000	OFF	OFF	OFF	OFF
* – Factory Default Setting				

Table 2–8, below, lists the bits and weights set by jumper block SW1, which set the A32 base address for the 4284's DRAM. All jumpers are installed on this block at the factory, configuring it for an A32 base address of 0x0000 0000.

Table 2–8: Model 4284 – A32 DRAM Base Address – Jumper Block SW1 Address Bits and Hexadecial Weights of Jumper Positions

Pins 15 - 16	Pins 13 - 14	Pins 11 - 12	Pins 9 - 10	Pins 7 - 8	Pins 5 - 6	Pins 3 - 4	Pins 1 - 2
A31 (0x8000 0000)	A30 (0x4000 0000)	A29 (0x2000 0000)	A28 (0x1000 0000)	A27 (0x0800 0000)	A26 (0x0400 0000)	A25 (0x0200 0000)	A24 (0x0100 0000)

2.4 VMEbus Slave Base Address Jumper Settings (continued)

2.4.3 VME DRAM Base Address (continued)

If any jumpers are removed from jumper block SW2, which sets the A24 base address, then the complete A32 base address is the sum of the two settings. For example, if the jumper between pins 7 and 8 of SW2 is removed, which sets the A24 base address to 0x80 0000 (this address is invalid if your DRAM window is set for 16 MBytes, see Note (2) on page 19, and Table 2–7), and the jumper between pins 15 and 16 of SW1 is also removed, then the A32 base address for the DRAM is 0x8080 0000. Another important point to keep in mind is that, in A32 space as well as in A24 space, the Model 4284's DRAM **MUST** be mapped on an address boundary that is a multiple of the memory size (see the first paragraph of Section 2.4.2, page 18.)

NOTE: While the 'C40 addresses the 4284's Dual–Port DRAM in Longword mode, VMEbus masters address all resources in byte mode. Consequently, each 'C40 address is equivalent to four VMEbus addresses. For example, if a VMEbus master wishes to read data that the 4284's 'C40 wrote to the DRAM at 'C40 address 0x8000 0020, the VME master would find that data at VME address VME_base+0x0000 0080.

In some applications, it may be necessary or convenient to disable VMEbus access to the Model 4284's DRAM. This can be accomplished by setting the DRAM_Inhibit bit in the Host Control Register (D4 at address A16_base+0x04) to the logic '1' state. See Section 3.11.2.4 for further details.

NOTE: For 4284s equipped with Option 010, the A32_base address is set by SW1 **ONLY**. The setting of the A24_base address jumper block (SW2) has no effect on the A32_base address in Option 010.

2.5 **VSB System Arbiter Jumpers** – JB14, JB15, and JB16 Mezzanine Board

The VSB interface on the Model 4284 Option 012 can be configured to include the system bus arbitration function. This is accomplished by placing shorting jumpers between pins 2 & 3 of JB14, JB15, and JB16 on the Model 4284's VSB Mezzanine board.

If you would rather have another device in your card cage handle VSBus arbitration, then the Model 4284's VSB arbiter functions should be diabled. To do this, place the shorting jumpers between pins 1 & 2 of JB14, JB15, and JB16 on the Model 4284's VSB Mezzanine board. This is the factory default setting.

The shorting jumpers should always be in the same positions of all three of these Jumper Blocks. Table 2–9, below, summarizes the settings.

Table 2–9: Model 4284, Option 012 – VSB System Controller Jumper Blocks – JB14, JB15, and JB16, Mezzanine Board

Jumper Position	VSB System Arbitration
*1 – 2	Disabled
2 – 3	Enabled
* – Factory default position	

2.6 Bus Request and Bus Grant Jumpers

In the Model 4284, jumper blocks JB8, JB9 and JB10 are used for the selection of the device's VME Bus Request level, and to receive and pass on Bus Grant signals. The sections below describe the functions of these blocks.

2.6.1 **Bus Request Jumpers** – Jumper Block JB8

The Bus Request signal generated by the Model 4284 is connected to pins 4, 6, 8, and 10 of jumper block JB8. Installing a jumper between pins 3 & 4 of JB8 connects the internal bus request signal to the VMEbus BREQ0 line, making the 4284 a level 0 bus requester. Similarly, the 4284 may request mastership of the VMEbus on level 1 (BREQ1) if a jumper is installed on JB8 between pins 5 & 6. Placing a jumper between pins 7 & 8 of JB8 allows the Model 4284 to drive the BREQ2 line as a level 2 requester, and the 4284 can be configured as a level 3 requester (BREQ3) by jumpering pins 9 & 10 of JB8 together (this is the factory default setting). Table 2–10, below, summarizes jumper settings for JB8.

Table 2–10: Model 4284 VMEbus Master Request and Grant Jumpers – Jumper Blocks JB8, JB9, and JB10

Bus Request Level	Request Pins JB8	$\overline{\text{BGIn}}$ Pins JB9	$\overline{\text{BGOut}}$ Pins JB10
$\overline{\text{BREQ0}}$	3 – 4	1 – 2	1 – 2
$\overline{\text{BREQ1}}$	5 – 6	3 – 4	3 – 4
$\overline{\text{BREQ2}}$	7 – 8	5 – 6	5 –6
*$\overline{\text{BREQ3}}$	9 – 10	7 – 8	7 – 8
* – Factory Default Setting			

One, and **only one**, of the four jumpers described in the table above should be installed on JB8 at any given time. If two or more of these jumpers are installed, the VMEbus' priority arbitration scheme is defeated by the shorting together of bus request lines. If no jumpers are installed, the 4284's VMEbus Master interface is unable to request the bus, effectively disabling the 4284 as a bus master.

2.6.2 **Bus Grant Jumpers** – Jumper Blocks JB9 and JB10

The four VME Bus Grant In signals ($\overline{\text{BGIn0}}$, $\overline{\text{BGIn1}}$, $\overline{\text{BGIn2}}$, and $\overline{\text{BGIn3}}$) are connected to pins 1, 3, 5, and 7 of jumper block JB9, respectively. After a bus requester level has been selected by JB8, the same level's Bus Grant In signal must be connected to the 4284's VME Master Interface by jumpering the appropriate odd–numbered pin of JB9 to the adjacent even–numbered pin. These connections are summarized in Table 2–10, above.

2.6 Bus Request and Buys Grant Jumpers (continued)

2.6.2 Bus Grant Jumpers (continued)

If the 4284 receives a Bus Grant In signal on the appropriate level, but it did not request the bus, VMEbus Master Interface will generate a Bus Grant Out signal, which is delivered to the even numbered pins (2, 4, 6 and 8) on jumper block JB10. This signal must be jumpered across the block and back out to the VMEbus to complete the Bus Grant daisy chain. Table 2–10, on the previous page, summarizes these settings. Once a requester level has been selected, ALL the pins listed in that row of the table, and **ONLY** the pins listed in that row of the table, should be installed on the jumper blocks indicated.

2.6.3 Unused Bus Grant Levels

The Bus Grant signals on the VMEbus are daisy chained, meaning that they are not simply connected across the bus to each module, but must be passed along from one module to the next. The jumpering scheme detailed above allows the 4284 to pass along only the Bus Grant signal for the level on which it requests the bus. However, if there will be other potential VMEbus Master modules positioned to the right of the 4284 in your card cage, the Bus Grant In and Out signals on the other levels must be connected to one another, to bypass the 4284.

There are two ways to accomplish this. The first, and likely the easiest if available, is to jumper these signals together on your VME backplane. Consult the documentation provided with your card cage to determine if and how this can be done for your particular card cage.

If backplane jumpering is unavailable or inconvenient, the unused Grant signals can be connected on the 4284 card by wiring the signals from JB9 to JB10. The jumper block pins may be connected using wire–wrap techniques. To bypass level 0, $\overline{\text{BG0In}}$ can be connected to $\overline{\text{BG0Out}}$ by wiring JB9 pin 1 to JB10 pin 1. In the same manner, level 1 can be bypassed by wiring JB9 pin 3 to JB10 pin 3, Level 2 can be bypassed by connecting pin 5 of JB9 to pin 5 of JB10, and wiring pin 7 on the two blocks together bypasses level 3.

2.7 Slot 1 System Controller Jumpers

The 4284 can be configured as a VME slot 1 System Controller by placing jumpers in certain locations. A VMEbus System Controller drives the System Clock and System Reset lines, and performs arbitration when the bus is requested by more than one master at a time. The jumpers discussed below should be installed **ONLY** if the 4284 will reside in Slot 1 of your VME card cage, which is dedicated to the system controller. Table 2–11, below, summarizes the jumper settings. The subsections following the table describe functions enabled by the jumpers.

Table 2–11: Model 4284 – VMEbus System Control Jumpers

Control Function	Jumper Block	Pins
$\overline{\text{SYSRST}}$ Enable	JB8	1 – 2
Level 3 Arbiter	JB9	9 – 10
System Clock	JB12	1 – 2

2.7.1 System Reset Enable – Jumper Block JB8

Connecting pins 1 and 2 of jumper block JB8 allows the 4284 to drive the VME System Reset line ($\overline{\text{SYSRST}}$). The Reset signal is generated by setting bit 0 of the 'C40's VMEbus Modifier Register, at 'C40 address 0x9000 0000, to the logic '1' state. Adding this jumper connects the reset signal to the bus. See Figure 3–1, on page 43 of this manual, for a block diagram of the 4284's reset circuitry.

The other pins in this jumper block are used to select the 4284's VMEbus requester level. See Section 2.6.1 on page 24 for further details.

2.7.2 Bus Arbiter Enable Jumper – Jumper Block JB9

Placing a jumper between pins 9 and 10 of JB9 enables the Model 4284 as a single level VMEbus arbiter on level 3. This allows the 4284 to receive Bus Requests on $\overline{\text{BREQ3}}$ from other VME Master devices and to supply the initial signal on the $\overline{\text{BG3In/Out}}$ daisy chain.

The remaining pins in this block are used to select the Bus Grant In level for the 4284's own VME Master Interface. See Section 2.6.2 on page 24 for further details.

2.7.3 System Clock Jumper – Jumper Block JB12

The Model 4284 can be enabled to drive the VMEbus System Clock line by placing a jumper between pins 1 and 2 of JB12.

2.8 **System Reset Jumper** – Jumper Block JB11

Installing a jumper between pins 1 and 2 of jumper block JB11 causes the VME System Reset signal to reset the 4284's 'C40. This jumper is installed at the factory, and should always be present if the Model 4284 is **not** the slot 1 VME System Controller.

If the Model 4284 is the slot 1 VME System Controller, and you do not want the 4284's internally generated System Reset signal to reset the 'C40, remove the jumper between pins 1 and 2 of jumper block JB11.

2.9 **VMEbus IRQ Jumpers** – Jumper Block JB13

There are seven possible VMEbus interrupts that may be used as interrupt inputs to the 'C40. However, only three of these may be enabled at any given time. Jumpers are installed on jumper block JB13 to assign which three of the seven VME IRQs can be ser–viced by the 'C40. After the proper setting of these jumpers, the three IRQs are referred to as 'C40 interrupt sources VME_IRQ_A, VME_IRQ_B and VME_IRQ_C. Table 2–12, below, describes the possible configurations.

Table 2–12: Model 4284 – VMEbus Interrupt Handler IRQ Configuration Jumpers – Jumper Block JB13

VMEbus IRQ Level	'C40 Interrupt Requests		
	VME_IRQ_A	VME_IRQ_B	VME_IRQ_C
IRQ1	1 – 2	N/A	N/A
IRQ2	3 – 5	11 – 13	13 – 14
IRQ3	2 – 4	11 – 12	12 – 14
IRQ4	5 – 6*	6 – 8	14 – 16
IRQ5	5 – 7	7 – 8	15 – 17
IRQ6	19 – 20	9 – 11*	17 – 19
IRQ7	18 – 20	8 – 10	17 – 18*

N/A – Not Available * – Factory Default Settings

NOTE: Only three of these jumpers should be installed at any given time. Installing more than three jumpers on JB13 shorts IRQ signals together on the VME backplane.

2.10 'C40 Boot Code Address – Jumper Block JB4

The address of the code you want to use to boot the 'C40 processor on your 4284 is partially determined by the setting of this jumper block. When the 'C40 is reset, it can jump to one of four addresses, called Reset Vectors. The Reset Vector for the processor is selected by a pair of bits in the Host Control Register (see Section 3.11.2.3 and Section 4.2 of this manual). The power–up default setting of these bits selects the 'C40's internal ROM. The ROM program instructs the 'C40 to jump to another address and run the boot code found there. That is the address determined by this jumper block. For further details on the reset operation of the 'C40, see Sections 6.7 (Reset Operation) and 12.1 (Processor Initialization) of the Texas Instruments TMS320C4x User's Guide.

Table 2–13, below, describes the jumper settings and the available boot code addresses. While the table lists all eight possible settings of this jumper block, **only two of these settings (Comm Port and EPROM) are valid boot locations.** The factory default settings point to the external EPROM, where the boot program is located that initializes the memory map and makes the front panel LED blink at power–up.

Table 2–13: Model 4284 – 'C40 Boot Code Address Jumper Settings – Jumper Block JB4

Address	Location	Pins 1 & 2	Pins 3 & 4	Pins 4 & 6
COMM PORT	Comm Port	OFF	OFF	OFF
0x0030 0000*	Ext. EPROM	ON	OFF	OFF
0x4000 0000	Do Not Use†	OFF	ON	OFF
0x6000 0000	Not Available	ON	ON	OFF
0x8000 0000	Do Not Use†	OFF	OFF	ON
0xA000 0000	Not Available	ON	OFF	ON
0xC000 0000	Do Not Use†	OFF	ON	ON
undefined	Do Not Use	ON	ON	ON

* Factory Default Setting – Also used for boot from Flash EEPROM or Global DRAM – See Note 4, Next page † – See Note 3, next page·

2.10 'C40 Boot Code Address (continued)

Notes to Experienced 'C40 Users:

(1) The previous discussion assumes that the ROM enable pin (ROMEN) on the 'C40 is held in the high state. There is a 2–pin jumper block associated with this pin on the 'C40. Installing a jumper on this block (JB3) pulls the input low, and leaving the jumper off pulls the input high (the default state). If you wanted to boot the 'C40 from a vector address other than those that can be pointed to by the RESETLOC(0,1) pins (driven by bits in the Host Control Register – see Section 3.8.2.3), you could install this jumper and disable the internal ROM. However, you would then need to plant your own reset vector in a region of memory that is not installed on the 4284. (The essential point of all the above is . . . **don't** install the jumper on JB3 – instead, fetch the reset vector from the 'C40's ROM.)

(2) During the 'C40 reset cycle, jumper block JB4 drives the $\overline{\text{IIOF}}$(1–3) pins on the 'C40, via a multiplexer. Following initialization, the multiplexer switches states, causing the $\overline{\text{IIOF}}$ pins to be driven by the interrupt sources for the 'C40. See Section 3.3, Section 3.4 and Section 4.4 of this manual for further information about interrupt handling on the Model 4284. See the Texas Instruments TMS320C4x User's Guide, Section 3.1.10 ($\overline{\text{IIOF}}$ Flag Register) for further details.

(3) Addresses 0x4000 0000, 0x8000 0000, and 0xC000 0000 point to the Local SRAM, Global DRAM, and SRAM, respectively. While these are valid memory addresses, there is no practical way to load boot code to any of these and retain it while re–booting.

(4) During the boot from standard EPROM, the first address of the Global DRAM is read. Based on the value found there, the remainder of the boot sequence may be executed from DRAM, from the flash EEPROM, or from the standard EPROM.

2.11 'C40 TCLK Connectivity Jumpers – Jumper Block JB5

The TCLK(0,1) I/O signals from the 'C40 on the Model 4284 can be connected to either the front panel's general purpose I/O connector (labeled TCK), or to the Transmit Data input and Receive Data output of the RS–232 level translator, where they can be used to synthesize asynchronous serial communication signals. Jumper block JB5 determines where these signals are connected. Table 2–14, below, describes the settings available on this jumper block.

Table 2–14: Model 4284 – TCLK I/O Connectivity – Jumper Block JB5

TCLK Signal	TCLK Connector	RS–232 Translator
TCLK0	JB5, 1– 2 (Pin 1)	JB5, 5 – 6* (Receive)
TCLK1	JB5, 3 – 4 (Pin 2)	JB5, 7 – 8* (Transmit)
* – Factory Default Settings		

2.12 The Model 4284 Front Panel

Available to the user on the front panel of the Model 4284 are the six buffered Comm Ports from the 'C40, a JTAG connector for the Texas Instruments XDS–510 emulator, an RS–232–level serial communications port, a general–purpose I/O connector (labeled TCK), a software–controllable LED and a Reset button. Each of these resources will be discussed in the sections below, which will give pinouts for the connectors and describe the signals they carry.

2.12.1 Comm Port Connectors

The Model 4284 provides front panel access to the six buffered communications ports, referred to as 'Comm' Ports, from the TMS320C40. There are six Comm Port connectors on the front panel with mark–ings on the panel showing the Comm Port number. For example the designation "COM5" refers to Comm Port 5.

Figure 2–3, at the right, shows the front panel Comm Port layout and the pin assignments. The connectors are keyed to prevent improper insertion.

Comm Port cables can be easily assembled by using a 3M hand press with locator plate #3443–110 and standard 16–conduc–tor, 0.050 pitch flat ribbon cable such as 3M #3365/16. Every Model 4284 ships with six mating connectors (3M #50116–B000) and the optional strain relief (3M #3448–50116). By using these standard components, you can easily configure the Comm Port cables to minimize excess cable length for any given interconnect pattern.

As an aid to consistent assembly tech–niques, make sure that pin 1 of each socket at the end of the cable is connected to pin 1 of the ribbon cable. Pin 1 of the cable is identified by the colored stripe. A typical Comm Port cable is shown in Figure 2–4, on the next page.

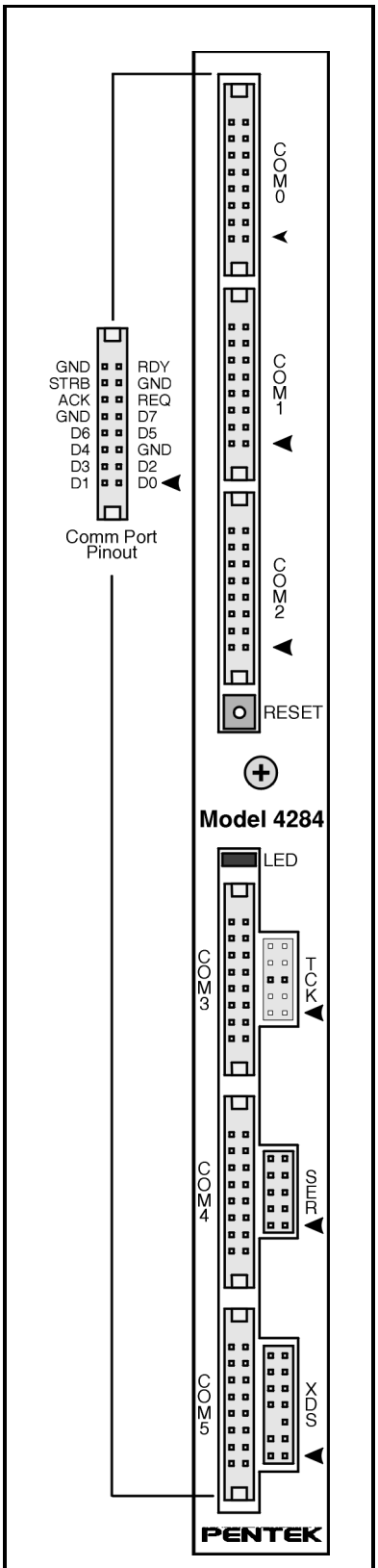

Figure 2–3: Model 4284 Front Panel

2.12 The Model 4284 Front Panel (continued)

2.12.1 Comm Port Connectors (continued)

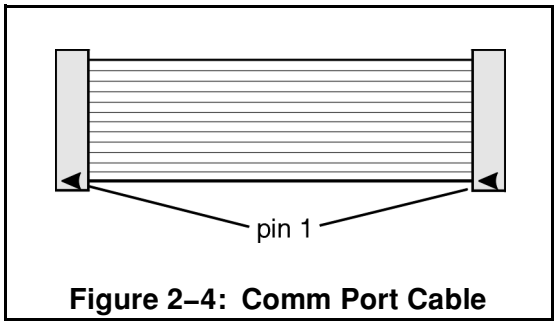

Figure 2–4: Comm Port Cable

All twelve signal lines on the Comm Ports are equipped with a series 24Ω resistor to properly match the characteristic impedance of the flat ribbon wiring used in the Comm Port cables.

Comm Port cables are available preassembled from Pentek as Model 2104. Cable lengths of up to 18 in. can be accommodated.

At Power–up, Comm Ports 0, 1 and 2 are configured as outputs (i. e., to write), and Ports 3, 4 and 5 are configured as inputs (i. e., to read). Changing a Comm Port's I/O configuration is a simple matter of addressing it for the opposite function. In other words, if you wish to use Port 3 as an output, all you need to do is address it to write. If that port were connected to Port 0, then when the data written by Port 3 arrives at Port 0, that port reconfigures itself as an input. See Chapter 8 (Communication Ports) of the Texas Instruments **TMS320C4x User's Guide** for further details.

2.12.2 XDS Emulator Connector

The Model 4284 has provisions for use with the Texas Instruments XDS510 Emulator. The emulator connector is available on the 4284's front panel. This connector is keyed for proper alignment with the socket connector at the end of the emulator cable. Refer to Texas Instruments TMS320C40 Emulator User's Guide or the documentation supplied with the XDS system for operating details. Figure 2–5, at the right, gives the pinout of the emulator connector.

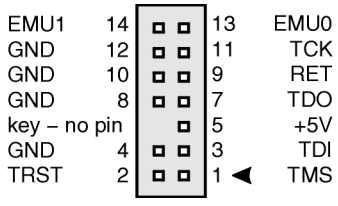

Figure 2–5: XDS Connector Pinout

2.12 The Model 4284 Front Panel (continued)

2.12.3 Serial Port Connector

Although there is no UART on the Model 4284, the 'C40 can synthesize asynchronous serial communications in software using its TCLK0 and TCLK1 I/O pins. These pins may be brought to the front panel directly or through the RS232 driver/receiver by setting the proper jumpers on Jumper block JB5, described in Section 2.10 and Table 2–13 (see page 28). Example 'C' source code for serial communication using the 'C40 TCLKs is included in Appendix A of this manual, and on Pentek's Example Software Diskette.

If a jumper is installed between pins 7 and 8 of jumper block JB5, then pin 3 on the front panel connector labeled SER provides an output signal driven by TCLK1 and shifted to RS–232 levels. If a jumper is installed between pins 5 and 6 of JB5, then pin 2 on this connector can accept an RS–232 level input signal, shift it back to TTL levels and deliver it to the 'C40's TCLK0 pin. Pin 5 of this connector is grounded and the other seven pins are unconnected. Figure 2–6, at the right, shows the pinout of the 4284's Serial Port connector.

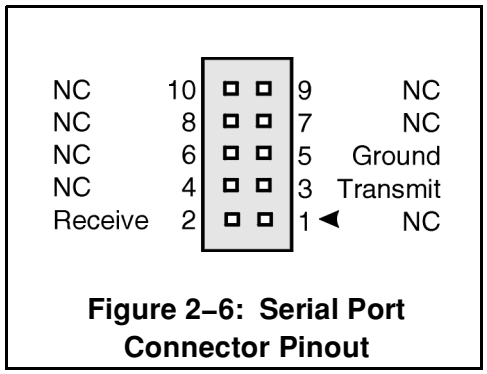

Figure 2–6: Serial Port Connector Pinout

2.12.4 General Purpose I/O (TCK) Connector

Some signals that may be of general use are brought to the front panel connector labeled TCK. These can include the TCLK(0,1) I/O signals if the proper jumpers are placed on jumper block JB5 (see Section 2.10 and Table 2–13, page 28). Figure 2–7, at the right, shows the pinout of the TCK connector.

Please note that while pins 3 – 8 are connected to MIX bus signals, it is possible in some circumstances to use pins 3, 6, 7, and 8 as general purpose TTL outputs. Pins 4 and 5 are connected to MIX bus inputs, and can sometimes be used in as general purpose TTL inputs.

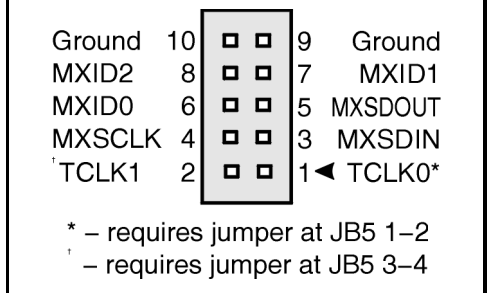

Figure 2–7: General Purpose I/O Connector Pinout

2.12 The Model 4284 Front Panel (continued)

2.12.4 General Purpose I/O (TCK) Connector (continued)

NOTE: The recommended mating socket (i. e., female) connector for both the Serial Port and G. P. I/O connectors is manufactured by Berg. Similar connectors from other manufacturersmay be too wide to fit in the panel cutout if the adjecent Comm Port connector is also installed. Berg's part number for this item is 71600−410, and Pentek's part number for it is 353.01006.

2.12.4.1 Erasing Flash EEPROM from the TCK Connector

In order to write new information into the Model 4284's Flash EEPROM, the EEPROM sector you want to write must first be erased. To erase the entire contents of the Flash EEPROM, connect a shorting jumper between pins 3 and 5 of the front panel TCK connector, then push the Reset button. When the 4284 is reset in this manner, the LED indicator remains lit until the Flash EEPROM is completely erased, and then blinks normally (see Section 2.12.5, below.)

If you want to erase only the specific sector(s) of the Flash EEPROM that you intend to overwrite, please refer to Section 4.3.3.2.

2.12.5 LED Indicator

The LED on the front panel of the Model 4284 is driven by bit 6 of the 'C40 VMEbus Modifier Register at 'C40 address 0x9000 0000 (See Section 3.8 for more details about this register). Writing a '1' to this bit turns the LED on, and writing a '0' turns it off.

2.12.6 Reset Button

Pressing the push−button on the Model 4284's front panel commences a reset cycle that, upon completion, returns the 4284 to its power−up state. See Section 4.2 for more details about the reset operation of the Model 4284.

2.13 Installing MIX Expansion Modules on the Model 4284

For MIX module stack assembly procedures and other important information about Pentek's family of MIX expansion modules, please refer to the manual entitled "MIX Tutorial for VMEbus Systems", Pentek part #800.00001, included with this shipment.

2.14 Installing the Model 4284 in the VME Card Cage

Once the assembly of the Model 4284 and expansion module(s) has been completed, the assembly can be installed in the card cage.

CAUTION!!

TURN OFF ALL POWER TO THE CARD CAGE BEFORE INSERTING OR REMOVING ANY BOARD

Since the MIX baseboard is the only portion of the assembly that actually engages in the back plane, **when inserting and removing the assembly, only use the front panel ejector/handles of the Model 4284** – not those of the expansion modules. After inserting the assembly, push in on the top and bottom ejector/handles of the Model 4284 only to fully seat the connectors in the back plane.

Once the assembly is seated, the captive panel screws at the top and bottom of each front panel should be screwed into the top and bottom rails of the card cage.

When removing the assembly, first loosen all captive screws on the top and bottom of each front panel. Then push the ejector/handles of the Model 4284 *only* away from the center of the panel to eject the assembly from the cage. Once disengaged, pull outward on the Model 4284 ejector/handles to remove the assembly.

2.15 Power–Up Self–Test

When power is applied to the Model 4284, some basic diagnostic routines are run from the EPROM. If no problems are encountered during this procedure, the LED on the 4284's front panel blinks about five times a second (the 'C40's Timer 0 is used as a counter). If the factory boot code was executed and the LED only blinks approximately once per second, then the 'C40 had trouble accessing the shared DRAM. If the factory boot code was executed and the LED does not blink at all, then a more serious problem exists.

Chapter 3: Memory Maps and Register Descriptions

3.1 TMS320C40 Local Bus Memory Map

The Model 4284's TMS320C40 has two 32−bit data buses, called the Local and Global busses. The map for the resources assigned to Local Memory, as seen by the 'C40 processor, is presented below. The following sections give more detailed information about the Local Memory resources and their usage.

Table 3−1: Model 4284 − TMS320C40 Local Memory Map

0x0000 0000− 0x0000 0FFF	Internal ROM (Boot Loader)	**0x1000 0000**	'C40 MIX Bus Control Register
		0x1000 0001	Reserved
0x0000 1000− 0x000F FFFF	Reserved	**0x1000 0002**	'C40 Interrupt Status/Control Register
0x0010 0000− 0x0010 00FF	Internal Peripherals (see Table 3−2, next page)	**0x1000 0003**	'C40 Interrupt Routing Register
		0x1000 0004	'C40 Interrupt to VME Register
0x0010 0100− 0x002F F7FF	Reserved	**0x1000 0005− 0x1FFF FFFF**	Reserved
0x002F F800− 0x002F FBFF	1k Internal RAM (Block 0)	**0x2000 0000− 0x23FF FFFF**	MIX Bus − Module 0
0x002F FC00− 0x002F FFFF	1k Internal RAM (Block 1)	**0x2400 0000− 0x27FF FFFF**	MIX Bus − Module 1
0x0030 0000− 0x0030 7FFF	External 32k or 64k x 8 Boot EPROM (Boot, Look & Set Functions)	**0x2800 0000− 0x2BFF FFFF**	MIX Bus − Module 2
0x0030 8000− 0x00AF FFFF	Reserved	**0x2C00 0000− 0x3FFF FFFF**	Reserved
0x00B0 0000− 0x00B7 FFFF	External 128k or 512k x 8 Flash EEPROM (optional)	**0x4000 0000− 0x4003 FFFF**	External Local SRAM (256k x 32)
0x00B8 0000− 0x0FFF FFFF	Reserved	**0x4004 0000− 0x4007 FFFF**	External Local SRAM − Opt. 005 (256k x32)

3.1 TMS320C40 Local Bus Memory Map (continued)

Table 3–2: TMS320C40 Internal Peripherals			
0x0010 0000–0x0010 000F	Local and Global Control Port	**0x0010 0080–0x0010 008F**	Comm Port 4
0x0010 0010–0x0010 001F	Analysis Module Block Registers	**0x0010 0090–0x0010 009F**	Comm Port 5
0x0010 0020–0x0010 002F	Timer 0 Registers	**0x0010 00A0–0x0010 00AF**	DMA Channel 0
0x0010 0030–0x0010 003F	Timers 1 Registers	**0x0010 00B0–0x0010 00BF**	DMA Channel 1
0x0010 0040–0x0010 004F	Comm Port 0	**0x0010 00C0–0x0010 00CF**	DMA Channel 2
0x0010 0050–0x0010 005F	Comm Port 1	**0x0010 00D0–0x0010 00DF**	DMA Channel 3
0x0010 0060–0x0010 006F	Comm Port 2	**0x0010 00E0–0x0010 00EF**	DMA Channel 4
0x0010 0070–0x0010 007F	Comm Port 3	**0x0010 00F0–0x0010 00FF**	DMA Channel 5

3.2 'C40 MIX Bus Control Register – Read/Write at 0x1000 0000

This 16–bit register is the means by which the 'C40 on the Model 4284 monitors and controls the activities of the expansion modules on the MIX bus. Table 3–3, below, defines the register's bit structure. Brief descriptions of the functions of the bits follow the table. For more detail, refer to the MIX bus specification. All bits in this register are cleared to the '0' state at power–up reset, and they may generally be left in that state.

Table 3–3: Model 4284 – 'C40 MIX Bus Control Register – R/W @ 0x1000 0000								
Bit #	**D15**	**D14**	**D13**	**D12**	**D11**	**D10**	**D9**	**D8**
Bit Name	MXRST	Turbo–MIX	MXDC	MXMIO	$\overline{\text{MXBE3}}$	$\overline{\text{MXBE2}}$	$\overline{\text{MXBE1}}$	$\overline{\text{MXBE}}$
Bit #	**D7**	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
Bit Name	N/U	N/U	$\overline{\text{MXID2}}$	$\overline{\text{MXID1}}$	$\overline{\text{MXID0}}$	SDOUT	SCLK	SDIN

3.2.1 Serial EEPROM Data Transfer – Bits D0 – D2

The three least significant bits in this register are concerned with the transfer of serial data to the EEPROM that may be contained on a MIX module. At the time of this writing, none of Pentek's MIX modules for VMEbus systems contain such an EEPROM.

3.2 'C40 MIX Bus Control Register (continued)

3.2.1 Serial EEPROM Data Transfer (continued)

However, the D0 bit of this register would be used as a serial data stream to be written to the module's EEPROM. D2 contains a serial stream read from the module's EEPROM, and D1 has the clock signal which controls the transfer.

These bits are also brought to the front panel's General Purpose I/O connector (TCK), where the SDOUT and SCLK lines may be used as TTL inputs, and the SDIN line may be used as a TTL output.

3.2.2 MIX Module ID – Bits D3 – D5

During a MIX bus reset (when the MXRST line, D15 of this register, is high), a MIX baseboard (e. g., the Model 4284) determines how many expansion modules are present in its MIX stack by reading the states of these three bits. Any installed module will drive the bit associated with its stack position to the low state when the Reset bit is high. If no module is installed in a given stack position, then the bit associated with that position will be high during Reset.

These bits also serve as chip select lines for the modules' Serial EEPROMs (if they exist), when the reset signal is inactive. Additionally, they are brought to the front panel's General Purpose I/O connector (TCK), where they may be used as TTL outputs.

3.2.3 MIX Byte Enables – Bits D8 – D11

These four bits allow a byte sent over the MIX bus to be justified in one of four positions on the 32 bit data bus, or a word to be justified in either of two positions. Only one of these four bits should be in the low state for byte transfers, two should be low for word transfers, and all four are low for Longword transactions.

When the 4284 is booted from the factory–installed EPROM, these bits are all set low, which enables Longword exchanges. For use with Pentek MIX modules, this default configuration is the setting you will want for almost all MIX bus transactions. Table 3–4, on the next page, describes the relationship between which byte enable bits are activated (i. e., low) and which bits of the MIX data bus will be used in a given data exchange.

3.2 'C40 MIX Bus Control Register (continued)

3.2.3 Mix Byte Enables (continued)

Table 3–4: Model 4284 – MIX Bus Byte Enables and Data Bus Usage

Transfer Type	Cycle # (Note1)	Byte Enable Signals				Data Bus Utilization				Physical Address		Comment
		$\overline{MXBE3}$	$\overline{MXBE2}$	$\overline{MXBE1}$	$\overline{MXBE0}$	D31-D24	D23-D16	D15-D8	D7-D0	A1	A0	
32–bit	**–**	L	L	L	L	m	n	k	l	0	0	Single Cycle
32–bit	**1**	H	H	L	L	X	X	k	l	0	0	1 word +
2 cycle	**2**	L	L	H	H	X	X	m	n	1	0	1 word
24–bit	**–**	H	L	L	L	X	m	n	l	0	0	Single Cycle
24–bit	**–**	L	L	L	H	m	n	l	X	0	1	Single Cycle
24–bit	**1**	H	L	L	L	X	X	n	l	0	0	1 word +
2 cycle	**2**	H	L	H	H	X	X	X	m	1	0	1 byte
24–bit	**1**	L	L	L	H	X	X	l	X	0	1	1 byte +
2 cycle	**2**	L	L	H	H	X	X	m	n	1	0	1 word
16–bit	**–**	H	H	L	L	X	X	m	l	0	0	Single Cycle
16–bit	**–**	L	L	H	H	X	X	m	l	1	0	Single Cycle
16–bit	**1**	H	L	L	H	X	X	l	X	0	1	2 bytes =
2 cycle	**2**	H	L	H	H	X	X	X	m	1	0	1 word
8–bit	**–**	H	H	H	L	X	X	X	l	0	0	Single Cycle
8–bit	**–**	H	H	L	H	X	X	l	X	0	1	Single Cycle
8–bit	**–**	H	L	H	H	X	X	X	l	1	0	Single Cycle
8–bit	**–**	L	H	H	H	X	X	l	X	1	1	Single Cycle

Notes: (1) Cycle # indicates the order in which multiple MIX cycles MUST be run.
(2) X = no byte, m = most significant byte, n = next more significant byte, k = next less significant byte, l = least significant byte
(3) For Pentek MIX modules, use the 32–bit single cycle setting, which is the default.

3.2.4 MIX Bus Status Bits – Bits D12 and D13

The MXMIO (Memory/IO select) and MXDC (decode) bits are used by MIX bus slave modules to determine the type of access requested. For use with Pentek MIX modules for VMEbus systems, these two bits should ALWAYS be held in the low state.

3.2 'C40 MIX Bus Control Register (continued)

3.2.5 Turbo–MIX – D14

This feature is supported ONLY on 4284's equipped with 50 MHz TMS320C40's! This bit is unused on 4284's with 40 MHz 'C40's.

Turbo–MIX mode allows faster transfers to occur over the MIX bus. These accel–erated transfers are effectively a violation of the MIX bus timing specification, as all external wait states are ignored (i. e., the RDY signal is generated internally).

Another important operating characteristic of Turbo–MIX mode is that the MIX bus transceivers are always enabled, which means that the 'C40 is unable to properly access it's Local SRAM, EPROM or the registers at 0x1000 0000 – 0x1000 0004. Several of Pentek's MIX modules have been tested and found to operate properly in Turbo–MIX mode, and changes may be made to some other modules to support this feature. Contact Pentek at (201) 818–5900 for the latest list of MIX modules supporting Turbo–MIX.

Turbo–MIX mode is enabled on 50 MHz 4284 boards by setting bit D14 to the '1' state. When this bit is cleared to the '0' state, Turbo–MIX mode is disabled, and normal MIX bus timing specifications are met.

3.2.6 MIX Bus Reset – D15

All expansion modules on the MIX bus are held in their power–up reset state while this bit is held high. For normal operation of the modules, this bit should be low.

3.3 'C40 Interrupt Status and Control Register – Read/Write at 0x1000 0002

The architecture of the Model 4284 allows several interrupt sources to share a single interrupt input on the 'C40. Therefore, when the 'C40 receives an interrupt, this regis–ter should be read to determine where the interrupt came from.

The eight LSBs of this register are accessible by the 'C40 for read operations only. These bits are initially cleared to the '0' state by the power–up reset (a $\overline{\text{C40RST}}$ also clears this register). They are set to the '1' state by the individual interrupt sources when they issue their interrupts. The VME interrupt status bits are cleared back to '0' at the end of the VMEbus IACK cycle, when the 'C40 reads the IACK Vector from the interrupter. The MIX interrupt status bits are cleared when the 'C40 accesses the inter–rupting module. The VMEbus interrupt status bit is cleared by a 'C40 IACK instruc–tion. The Timeout interrupt status bit is cleared by any write operation to this register.

3.3 'C40 Interrupt Status and Control Register (continued)

The three remaining bits in this register (D8 – D10) are accessible for 'C40 read and write operations. These bits contain the binary–coded level of the IRQ signal that is asserted when the 'C40 interrupts the VMEbus. The level on which you wish to interrupt the VMEbus is written to these three bits. When serviced by a VME Interrupt Handler, this is the IACK level to which the 4284 responds with the vector previously written to the IACK Vector Register (see Section 3.11.1).

Table 3–5, below, summarizes the bit structure of the 'C40 Interrupt Status and Control Register. The actual sources of the VMEbus IRQ's are set by jumpers placed on JB13. Refer to Section 2.9 of this manual for further details. The MIX bus interrupts are generated by the device in the stack position with the corresponding number (i. e., the module in MIX slot 1 generates MXINT1, etc.).

Table 3–5: Model 4284 – 'C40 Interrupt Status and Control Register – R/W @ 0x1000 0002

Bit #	D15	D14	D13	D12	D11	D10	D9	D8
Bit Name	N/U	N/U	N/U	N/U	N/U	IRQlev2	IRQlev1	IRQlev0
Bit #	D7	D6	D5	D4	D3	D2	D1	D0
Bit Name	TimeOutInt	HosInt	MXINT2	MXINT1	MXINT0	VME_IRQ_C	VME_IRQ_B	VME_IRQ_A

3.4 'C40 Interrupt Routing Register – Read/Write at 0x1000 0003

The TMS320C40 on the Model 4284 has four interrupt inputs, labeled herein as INT_0 through INT_3. The various sources of interrupts such as VME, MIX modules, host register access and bus timeout circuitry can be flexibly routed by setting certain bits in this register. These interrupts are logically OR'ed together, so it is possible to have multiple events driving a single interrupt input. It is also possible (though not very useful) to have a single event driving multiple interrupt inputs.

Table 3–6, on the next page, describes the bit structure of this register. As shown in the table, the least significant byte of the register (D0 – D7) connects interrupt sources to the 'C40's INT_0 input, the next byte (D8 – D15) connects the sources to INT_1, the third byte (D23 – D16) routes interrupts to INT_2, and the most significant byte (D31 – D24) connects with INT_3. All bits are high true, i. e., a '1' in a given bit connects the indicated source to that byte's interrupt input.

3.4 'C40 Interrupt Status and Control Register (continued)

Table 3–6: Model 4284 – 'C40 Interrupt Routing Register – R/W @ 0x1000 0003								
Bit #	**D31**	**D30**	**D29**	**D28**	**D27**	**D26**	**D25**	**D24**
Bit Name	TimeOutInt	HostInt	MXINT2	MXINT1	MXINT0	VME_IRQ_C	VME_IRQ_B	VME_IRQ_A
Bit Function	1 = Interrupt connected to 'C40 INT_3 0 = Interrupt not connected to 'C40 INT_3							
Bit #	**D23**	**D22**	**D21**	**D20**	**D19**	**D18**	**D17**	**D16**
Bit Name	TimeOutInt	HostInt	MXINT2	MXINT1	MXINT0	VME_IRQ_C	VME_IRQ_B	VME_IRQ_A
Bit Function	1 = Interrupt connected to 'C40 INT_2 0 = Interrupt not connected to 'C40 INT_2							
Bit #	**D15**	**D14**	**D13**	**D12**	**D11**	**D10**	**D9**	**D8**
Bit Name	TimeOutInt	HostInt	MXINT2	MXINT1	MXINT0	VME_IRQ_C	VME_IRQ_B	VME_IRQ_A
Bit Function	1 = Interrupt connected to 'C40 INT_1 0 = Interrupt not connected to 'C40 INT_1							
Bit #	**D7**	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
Bit Name	TimeOutInt	HostInt	MXINT2	MXINT1	MXINT0	VME_IRQ_C	VME_IRQ_B	VME_IRQ_A
Bit Function	1 = Interrupt connected to 'C40 INT_0 0 = Interrupt not connected to 'C40 INT_0							

3.5 'C40 Interrupt to VMEbus Register – Write Only at 0x1000 0004

Writing any data to this register causes the VMEbus IRQ assigned in the 'C40 Interrupt Status and Control Register (see Section 3.3) to be asserted. The interrupt is cleared by a 'C40 reset, or when a VMEbus master writes to the VMEBus IACK register. This interrupt's status can be read in the 'C40 VMEbus Modifier Register at 0x9000 0000, bit D7.

3.6 Other Local Memory Resources

Other resources that may be found in the 'C40's Local Bus memory map are the 'C40's internal Boot Loader ROM and two internal 1 kByte RAM blocks. Also accessed on the Local Bus are the External Boot EPROM (32k x 8, beginning at address 0x0030 0000), the optional Flash EEPROM (128k x 8 for Option 002 or 512k x 8 for Option 003, beginning at address 0x00B0 0000) and the Local 256k x 32 SRAM (512k x 32 in units with the extended Local SRAM, Option 005), beginning at address 0x4000 0000.

The Flash EEPROM supports non–volatile storage of programs or other data. For more information about how data is written into this memory region, and other details about its use, please refer to Section 4.3 of this manual.

The MIX bus is effectively seen as extension of the 'C40's Local Bus. Each MIX module can be accessed in its own 64 M–Longword region on the Local Bus. Module 0's region begins at 0x2000 0000, Module 1's at 0x2400 0000, and Module 2's at 0x2800 0000.

3.7 TMS320C40 Global Memory Map

The 'C40's Global bus provides the processor with access to the Global SRAM, in addition to those resources that are either shared with, or involved with, the VMEbus. The memory map for these resources is presented in Table 3–7, below.

Table 3–7: Model 4284 – 'C40 Global Bus Memory Map

0x8000 0000–0x800F FFFF	1M x 32 Dual Port DRAM*	**0xA000 0001**	VME IACK Level 1 Code (001)
		0xA000 0002	VME IACK Level 2 Code (010)
0x8010 0000–0x801F FFFF	1M x 32 Dual Port DRAM (Options 007, 008, and 009)*	**0xA000 0003**	VME IACK Level 3 Code (011)
		0xA000 0004	VME IACK Level 4 Code (100)
0x8020 0000–0x802F FFFF	1M x 32 Dual Port DRAM (Options 008 and 009)*	**0xA000 0005**	VME IACK Level 5 Code (101)
		0xA000 0006	VME IACK Level 6 Code (110)
0x8030 0000–0x803F FFFF	1M x 32 Dual Port DRAM (Option 009 only)*	**0xA000 0007**	VME IACK Level 7 Code (111)
0x8040 0000–0x87FF FFFF	Reserved	**0xA000 0008–0xAFFF FFFF**	Reserved
0x8800 0000–0x8FFF FFFF	VSBus Master Access Region (Option 012 only)	**0xB000 0000–0xBFFF FFFF**	VMEbus Master Access
0x9000 0000	'C40 VMEbus Modifier Register	**0xC000 0000–0xC003 FFFF**	External Global SRAM (256k x 32)
0x9000 0001	VSB Control Register (Option 012 only)	**0xC004 0000–0xC007 FFFF**	Ext. Global SRAM – Opt. 006 (256k x 32)
0x9000 0002–0xA000 0000	Reserved		

* – The DRAM is also accessible to VME, beginning at the VME_base address

3.8 The 'C40 VMEbus Modifier Register – Read/Write at 0x9000 0000

The TMS320C40 on the Model 4284 may become a VMEbus Master by accessing the 256 M–Longword portion of its memory map beginning at 'C40 address 0xB000 0000. The VMEbus address space that it will access and the desired data width is designated when the 'C40 writes to a register mapped into its Global Bus.

This VMEbus Modifier Register contains Data Transfer Select (TRSEL) bits, Address Modifier bits and two Page Address bits. Note that this register need only be written once unless a change in address space or data width is needed for subsequent transfers.

Besides the Address Modifier and Transfer Select bits, the 'C40 VMEbus Modifier Register contains bits to fire the VMEbus System Reset (if the 4284 is configured as a slot 1 System Controller), to lock the VMEbus and to control the front panel LED Indicator. There are also two VMEbus Page Address bits, and a status bit indicating that the 'C40 has issued an interrupt to the VMEbus. Table 3–8, at the top of the next page, gives this register's bit definition. The sections beginning below the table give detailed descriptions of the functions of each bit or group of bits.

3.8 The 'C40 VMEbus Modifier Register – (continued)

Table 3–8: Model 4284 – 'C40 VMEbus Modifier Register – R/W @ 0x9000 0000								
Bit #	**D15**	**D14**	**D13**	**D12**	**D11**	**D10**	**D9**	**D8**
Bit Name	VME_PA31	VME_PA30	VME_AM5	VME_AM4	VME_AM3	VME_AM2	VME_AM1	VME_AM0
Bit #	**D7**	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
Bit Name	HstIntStat	LED	VME_Lock	TRSEL3	TRSEL2	TRSEL1	TRSEL0	Sys_Rst

3.8.1 VMEbus System Reset – Bit D0

This bit is used to fire a VME System Reset pulse, if the 4284 is configured as the slot 1 System Controller. While reading the description of this reset, on the following page, it may help to refer to Figure 3–1, below.

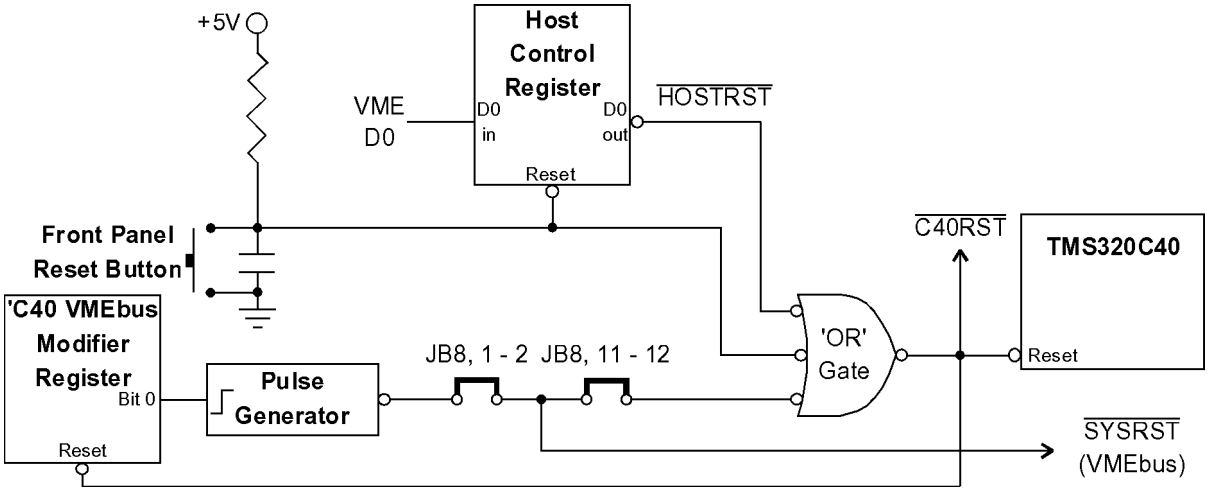

Figure 3–1: Model 4284 Reset Circuit Diagram

Writing a '1' to this bit triggers a 1–shot type pulse generator. If a jumper is installed between pins 1 and 2 of JB8, this pulse is delivered to the VMEbus System Reset line. If another jumper is installed between pins 1 and 2 of JB11, then this pulse also resets the 'C40 processor on the 4284, by means of the internal $\overline{\text{C40RST}}$ signal. An auxiliary function of the $\overline{\text{C40RST}}$ signal is to clear this register, thus resetting this bit to the '0' state.

3.8 The 'C40 VMEbus Modifier Register (continued)

3.8.2 VMEbus Transfer Mode Select – Bits D1 – D4

VMEbus address bits A1 through A31 are implemented as 31 separate bus lines. These lines are used to explicitly address 2^{31} 16–bit words.

VME address bit A0 does not actually correspond to an address line on the bus, but is used in the control of two data strobe lines: DS0 and DS1. DS0 becomes active for the lower byte of the 16–bit word, and DS1 for the upper byte. For single byte transfers, only one DS strobe is active. For 16–bit transfers, both data strobes become active at the same time. For 32–bit transfers, besides activating both strobes, the LWORD (longword) bus line is also asserted.

The TRSEL(3–0) bits determine the memory cycle operation of the Bus Master Interface of the Model 4284. This interface allows the unit to access the VMEbus for read, write, and interrupt acknowledge (IACK) cycles. Data transfers for reads and writes may be 8, 16, or 32–bits wide, thus supporting many different types of VMEbus devices. The TRSEL(3–0) bits should be set up prior to conducting bus cycles and may be changed at any time by writing a new pattern. The bit definitions for this register are shown in Table 3–9, at the top of the following page. Details for each type of data cycle can be found in the sections beginning below.

3.8.2.1 32–bit Longword Memory Cycle

Data is transferred in a single 32–bit cycle with $\overline{DS0}$, $\overline{DS1}$, and $\overline{LWORD}$ (longword) lines asserted on the VMEbus. The 32–bit word of the TMS320C40 is aligned on a 32–bit word boundary in VMEbus address space. This implies that VME address A01 = L. All 32 bits of the TMS320C40 and VMEbus data busses are used.

3.8.2.2 16–bit Word Memory Cycle – Double Cycle

Data is transferred in two consecutive 16–bit word cycles with both $\overline{DS0}$ and $\overline{DS1}$ asserted. The first word is sent with VME address A01 = L, and the second with A01 = H. Both transfers use data lines D0 through D15 of the VMEbus.

The TMS320C40 uses its own data lines D0 through D15 for the first transfer and lines D16 through D31 for the second.

3.8 The 'C40 VMEbus Modifier Register (continued)

3.8.2 VMEbus Transfer Mode Select (continued)

Table 3–9: Model 4284 – VMEbus Transfer Select (TRSEL) Bit Functions

TRSEL BITS 3 (D4)	2 (D3)	1 (D2)	0 (D1)	Description	VMEbus Strobes	VMEbus Addr	VMEbus Data	'C40 Data	Comments
x	x	0	0	32–bit Long Word	$\overline{DS0}$, $\overline{DS1}$	A01 = L $\overline{LW}$ = L	D0 – 31	D0 – 31	Single 32–bit Transfer Cycle
x	0	0	1	16–bit Word Double Cycle (Long Word Transfer as two 16–bit words)	$\overline{DS0}$, $\overline{DS1}$	A01 = L $\overline{LW}$ = H	D0 – 15	D0 – 15	First of Two 16–bit Cycles
					$\overline{DS0}$, $\overline{DS1}$	A01 = H $\overline{LW}$ = H	D0 – 15	D16 – 31	Second of Two 16–bit Cycles
0	1	0	1	16–bit Word Single Word Transfer	$\overline{DS0}$, $\overline{DS1}$	A01 = L $\overline{LW}$ = H	D0 – 15	D0 – 15	Low Order 16–bit Word Cycle
1	1	0	1	16–bit Word Single Word Transfer	$\overline{DS0}$, $\overline{DS1}$	A01 = H $\overline{LW}$ = H	D0 – 15	D16 – 31	High Order 16–bit Word Cycle
0	0	1	0	8–bit Byte Single Byte Transfer	$\overline{DS0}$	A01 = L $\overline{LW}$ = H	D0 – 7	D0 – 7	Lowest Order Byte Cycle
0	1	1	0	8–bit Byte Single Byte Transfer	$\overline{DS1}$	A01 = L $\overline{LW}$ = H	D8 – 15	D8 – 15	Next to Lowest Order Byte Cycle
1	0	1	0	8–bit Byte Single Byte Transfer	$\overline{DS0}$	A01 = H $\overline{LW}$ = H	D0 – 7	D16 – 23	Next to Highest Order Byte Cycle
1	1	1	0	8–bit Byte Single Byte Transfer	$\overline{DS1}$	A01 = H $\overline{LW}$ = H	D8 – 15	D24 – 31	Highest Order Byte Cycle
X	X	1	1	IACK Cycle	$\overline{DS0}$	A1 – A3	D0 – D7	D0 – D7	IACK Vector Read

LW = VMEbus line LWORD, x = don't care

3.8.2.3 16–bit Word Memory Cycle – Single Cycle

In this case, only a single 16–bit word is transferred. Selection of the word is accomplished by setting TRSEL3 = VME address A01. Both DS0 and DS1 are asserted and VMEbus data lines D0 through D15 are used. If TRSEL3 = L, TMS320C40 data lines D0 through D15 are used, and if TRSEL3 = H, 'C40 data lines D16 through D31 are used.

3.8.2.4 8–bit Byte Memory Cycle – Single Cycle

In this case, a single byte is transferred. Selection of the byte is accomplished by setting TRSEL3 = VME address A01 and TRSEL2= VME address A00. Each of the four possible cycles is described in Table 3–9, above. Note that only one data strobe line is asserted, depending on TRSEL2 (A00). Also note the data line correspondence between the data busses for the TMS320C40 and the VMEbus.

3.8 The 'C40 VMEbus Modifer Register (continued)

3.8.2 VMEbus Transfer Mode Select (continued)

3.8.2.5 IACK Cycle

The IACK cycle setting is used by the TMS320C40 when acknowledging that it has initiated an interrupt on the VMEbus. See Section 4.5.1.1 for a complete description of the IACK cycle.

3.8.3 VMEbus Lock – Bit D5

When the 'C40 tries to obtain mastership of the VMEbus, it may set this bit to the '1' state to retain exclusive control of the bus. This bit should be set before requesting the bus from the arbiter. It is used to enable DMA transactions over VME, and for other kinds of inseparable multi–cycle bus operations (e. g., Read–Modify–Write). The system controller may not grant the bus to another device until this bit is cleared to '0'.

3.8.4 LED Indicator Driver – Bit D6

The LED Indicator on the Model 4284's front panel is turned on when this bit is in the '1' state. To turn the LED off, clear this bit to the '0' state.

3.8.5 'C40 Interrupt to VMEbus Status – Bit D7

This bit is accessible to the 'C40 for read operations only. It is a status bit indicating that the Model 4284's 'C40 has sent a pending interrupt to the VMEbus System. The bit is set to a '1' by the 'C40 immediately after it sends the interrupt to VME, and is cleared to '0' by the VMEbus Interrupt Handler during the IACK cycle.

3.8.6 VMEbus Address Modifier – Bits D8 – D13

There are three addressing modes defined for VMEbus transfers: 32–bit, 24–bit, and 16–bit modes. These modes are provided to support a variety of bus devices ranging from simple I/O functions with only a few memory locations to large RAM arrays with hundreds of megabytes of storage.

The addressing mode is determined by a special 6–bit code that is placed on six dedicated VMEbus lines by the current VMEbus Master and is known as the Address Modifier Code, or AM Code.

3.8 The 'C40 VMEbus Modifier Register (continued)

3.8.6 VMEbus Address Modifier (continued)

The VMEbus specification defines the usage of certain AM codes as shown in , above. As a VMEbus Master, the Model 4284 can output all the AM codes assigned by VME specification. Note that while the 4284 can generate block transfer AM codes (3F, 3B, 0F and 0B), its VMEbus Master interface does not provide support for block transfers. As a VME Slave, it responds to the codes indicated in the table. User–defined codes may be included by modifying the AM Code PAL.

Table 3–10: Model 4284 – Address Modifier Codes

AM Code	Function
***0x3F**	A24 – Standard Supervisory Block Transfer†
0x3E	A24 – Standard Supervisory Program Access
0x3D	A24 – Standard Supervisory Data Access
***0x3B**	A24 – Standard Nonprivileged Block Transfer
0x3A	A24 – Standard Nonprivileged Program Access
0x39	A24 – Standard Nonprivileged Data Access
0x2D	A16 – Short Supervisory Access
0x29	A16 – Short Nonprivileged Access
***0x0F**	A32 – Extended Supervisory Block Transfer†
***0x0E**	A32 – Extended Supervisory Program Access
0x0D	A32 – Extended Supervisory Data Access
***0x0B**	A32 – Extended Nonprivileged Block Transfer†
***0x0A**	A32 – Extended Nonprivileged Program Access
0x09	A32 – Extended Nonprivileged Data Access
0x10 – 0x1f	User Defined
All Others	Reserved

* - The Model 4284 can generate this AM code as a VMEbus Master, but does not respond to it as a Slave.

† - The Model 4284 cannot master VMEbus block transfers.

3.8 The 'C40 VMEbus Modifier Register (continued)

3.8.7 VMEbus Page Address – Bits D14 and D15

The Global Memory map for the Model 4284 (see Section 3.7), allocates a region containing 256 M–addresses to VMEbus Master Access from the 'C40. Because all 'C40 data transactions are longwords, this 256 M–long–word address region is equivalent to 1 GByte of VME address space. The A32 VME–bus access region, however, is 4 GBytes deep. The two most sig–nificant bits in the 'C40 VMEbus Modifier Register allow the 4 GByte VME A32 region to be divided into four 1 GByte pages for access by the 'C40. D15 in this register is therefore equivalent to the A31 VME address bit, and D14 is equivalent to A30.

3.9 VMEbus IACK Level Codes – Read Only at 0xA000 0001 – 0xA000 0007

When an Interrupt Handler in a VMEbus system receives an interrupt on a level within its range, it first seeks control of the data transfer bus. After it has been granted bus control, it drives the $\overline{\text{IACK}}$ line low and places a 3–bit code on the LSBs of the VME address bus (A01 – A03), corresponding to the level on which it was interrupted. In the Model 4284, these seven locations contain the IACK codes. These codes are actually the lower 3 bits of their addresses, i. e., the code at location 0xA000 0005 is 101_2.

For example, if the 4284 is interrupted on level 5, it (after gaining bus control) asserts $\overline{\text{IACK}}$ and places 101_2 on the 3 lowest VME address bus lines. The interrupting devices monitor these 3 LSBs, so the interrupter on level 5 can recognize that this cycle acknowledges its interrupt.

When reading the odd–numbered registers in this group (i. e., codes 1, 3, 5 and 7), the code resides in the upper 16 bits of the 32–bit longword. In these cases, the result of the read operation needs to be "word–swapped" (or shifted right 16 bits) before placing the code on the address bus.

3.10 Other Global Memory Resources

Also mapped into the 'C40's Global Memory space are the 256k x 32 Global SRAM (512k x 32 in units with the extended Global SRAM, Option 006), beginning at address 0xC000 0000, and the Dual Ported DRAM, shared with the VMEbus.

The DRAM may be 1, 2 or 4 M–longwords deep, depending upon the options pur–chased. Regardless of its depth, the DRAM is mapped at the beginning of the 'C40's Global Bus memory region, i. e., it begins at address 0x8000 0000. For VMEbus mas–terw, the DRAM region begins at the VME_base address.

Remember that the 'C40 always accesses this memory on longword address bound–aries, while external VMEbus Masters address it on byte boundaries. When writing this memory from the VME side, be certain that the data is properly aligned, so the 'C40 can interpret it properly.

3.11 VMEbus Slave Memory Resources

The Model 4284 responds as a VMEbus slave device for accesses to its Dual–Ported DRAM and to the IACK Vector and Host Control Registers.

The IACK Vector that the 4284 responds with when participating in an IACK cycle may be programmed by the host computer. The vector is contained in the IACK Vector Register (at the A16_base address), which is mapped into A16 space. Another A16 register (the Host Control Register, at address A16_base+0x04) is provided so that the Host computer may reset or interrupt the TMS320C40. Other desired 'C40 interrupt sources are selected at jumper block JB13 (see Section 2.8).

The A16_base address is selected by setting jumpers (see Section 2.4.1). These two registers are enabled for D16 accesses only.

3.11.1 VMEbus IACK Vector Register – Read/Write at A16_base + 0x00

This 8–bit register is used to store the IACK Vector Address (VA in the table below), which are placed on the VME Data bus when a resource on the Model 4284 issues an interrupt to VME. Table 3–11, below, shows the correspondence between the bits in this register and the VME Data Bus lines that are their destinations.

NOTE: This register must be re–written to clear a VMEbus interrupt generated by the 4284. The interrupt is not automatically cleared by the hardware operations of the IACK cycle.

Table 3–11: Model 4284 – VMEbus IACK Vector Register – R/W @ A16_base

Bit #	D7	D6	D5	D4	D3	D2	D1	D0
Bit Name	VA7	VA6	VA5	VA4	VA3	VA2	VA1	VA0

3.11.2 VMEbus Host Control Register – Read/Write at A16_base + 0x04

This register provides a means for the Host computer to issue interrupts and resets to the 'C40. Two of the bits in this register drive the RESETLOC pins on the 'C40, which determine where the 'C40 finds its boot code after a reset. Another bit can be used to prohibit VME access to the 4284's DRAM. Table 3–12, below, describes the Host Control Register's bit structure. The subsections beginning at the top of the next page describe the functions of this register's bits.

Table 3–12: Model 4284 – VMEbus Host Control Register – R/W @ A16_base+0x04

Bit #	D7	D6	D5	D4	D3	D2	D1	D0
Bit Name	N/U	N/U	N/U	DRAM_INH	VME_RST1	VME_RST0	HOST_INT	HOST_RST

3.11 VMEbus Slave Memory Resources (continued)

3.11.2 VMEbus Host Control Register (continued)

3.11.2.1 Host Reset – Bit D0

This bit can be used to implement a VMEbus software reset function. It is active when set to logic '1', and reset to '0' by the power on reset ($\overline{\text{PONRS}}$). It is 'or–ed' with other 'C40 reset sources, as shown in Figure 3–1, in Section 3.8.1. See Section 4.1 for more details about the reset operation the 4284.

3.11.2.2 Host Interrupt – Bit D1

This bit can be used as a VMEbus software interrupt to the 'C40. This interrupt can be routed to any of the 'C40's four interrupt inputs (referred to herein as $\overline{\text{INT0}}$ – $\overline{\text{INT3}}$), by setting bits in the 'C40 Interrupt Routing Register at 'C40 address 0x1000 0003. See Section 3.4 for more details.

The bit is active when set to logic '1'. It is reset to '0' by the general purpose reset signal C40RST (see Figure 3–1 on page 43), or by the 'C40's IACK signal. See Section 4.1 for more details about reset operation. See Section 4.4 for more details about interrupt handling.

3.11.2.3 VMEbus Reset Vector Select – Bits D2 and D3

Bits D3 and D2 of the Model 4284's VMEbus Host Control Register drive the RESETLOC(1,0) pins on the 'C40 Processor. These pins tell the processor where to look to find the address of the initialization code to be run when the RESET input is driven low. See Section 2.10 of this manual for more details. For a complete description of 'C40 reset operation, See the Texas Instruments' **TMS320C4x User's Guide**, Sections 6.7 (Reset Operation) and 12.1 (Processor Initialization.)

Table 3–13, below, shows which location the 'C40 boots from, as a function of the settings of the VME_RST(0,1) bits in the VMEbus Host Control Register.

Table 3–13: Model 4284 – Reset Vector Select Bits

VME_RST1	VME_RST0	Memory Block	Boot Address
0	0	Local	0x0000 0000
0	1	Local	0x7FFF FFFF
1	0	Global	0x8000 0000
1	1	Global	0xFFFF FFFF

3.11 VMEbus Slave Memory Resources (continued)

3.11.2 VMEbus Host Control Register (continued)

3.11.2.4 DRAM Inhibit – Bit D4

When bit 4 in the Model 4284's VMEbus Host Control Register is set to a '1', access to the 4284's DRAM as a VMEbus slave resource is prevented. The Model 4284 effectively disappears from the VMEbus A24 and A32 Slave memory map when this bit is set. This bit must be cleared to the '0' state if VME access to the 4284's DRAM is necessary. This feature is only implemented on 4284's with PC board revisions D and higher. On earlier boards, VME slave access to the 4284's DRAM is always enabled.

3.12 Model 4284 VSB Interface (Option 012)

The VME Subsystem Bus (defined in IEEE Specification 1096–1988), or VSB, allows processor boards to access additional memory and/or I/O over a local bus, to remove some data traffic from the VME bus and improve total system throughput. The local bus utilizes the outer rows of pins (rows A and C) on the VME P2 connector, which are unused in the standard VMEbus implementation. The subsystem bus is physically implemented by connecting two or more adjacent VMEbus P2 connectors by means of an overlay board, placed over the pins that protrude from the rear of the card cage's backplane. Overlay boards are available from many sources, in lengths to cover from two to six P2 connectors. For example, Overlay boards may be ordered from ELMA Electronic, (44350 Grimmer Blvd., Fremont, CA 94538, Tel: (510) 656–3400 Fax: (510) 656–3783), as Model #69–B0**x**VSB, where **x** is the number of slots that the VSB back–plane will cover.

The Model 4284 operates as a VSB master and interrupt handler using the VSB1400A/B chip set from PLX Technologies. The 4284 cannot be accessed as a VSB slave, nor can it issue VSB interrupts. Mastership of the VSBus by an external VME device accessing the 4284 as a slave is not supported.

There are two resources on the Model 4284 that are used during VSB transactions. The 'C40 memory region between 0x8800 0000 and 0x8FFF FFFF is used to conduct VSBus master accesses, and a VSB Control Register is provided at 'C40 address 0x9000 0001 (see Table 3–7, on page 42). This register is described in detail in Section 3.12.1, below.

3.12.1 The VSB Control Register (Option 012 only) – R/W @ 'C40 Global Address 0x9000 0001

Table 3–14 at the top of the next page, shows the bit arrangement of the VSB Control Register on the Model 4284, Option 012. Finctional descriptions of the bits and fields are given in the subsections that follow the table. All bits in this register are cleared to the logic '0' state at power–up.

3.12 Model 4284 VSB Interface (Option 012) (continued)

3.12.1 The VSB Control Register (continued)

Table 3–14: Model 4284, Option 012 – VSB Control Register – R/W @ 'C40 Global Address 0x9000 0001

Bit #	D31	D30	D29	D28	D27	D26	D25	D24
Bit Name	Reserved							
Bit Function	Write as '0' – Mask when Reading							
Bit #	**D23**	**D22**	**D21**	**D20**	**D19**	**D18**	**D17**	**D16**
Bit Name	Reserved						VSA_1	VSA_0
Bit Function	Write as '0' – Mask when Reading						1 = 2 (2^1) 0 = 0	1 = 1 (2^0) 0 = 0
Bit #	**D15**	**D14***	**D13***	**D12**	**D11**	**D10**	**D9**	**D8**
Bit Name	BLT_ Enable	VSB_ $\overline{\text{BERR}}$	VSB_ $\overline{\text{IRQ}}$	VSB_$\overline{\text{LOCK}}$ $\overline{\text{Enable}}$	Space_1	Space_0	Size_1	Size_0
Bit Function	1 = Enable 0 = Disable	1 = False 0 = True		1 = Disable 0 = Enable	See Table 3–15, page 54		See Table 3–16, page 55	
Bit #	**D7**	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
Bit Name	VSA_31	VSA_30	VSA_29	Reserved				
Bit Function	1 = 2G (2^{31}) 0 = 0	1 = 1G (2^{30}) 0 = 0	1 = 512M (2^{29}) 0 = 0	Write as '0' – Mask when Reading				

* – These bits are Read Only. All bits are cleared to tle logic '0' state at power–up.

3.12.1.1 Low–Order VSB Address Bits – D17 (VSA_1) and D16 (VSA_0)

The VSA_0 (D16) and VSA_1 (D17) bits drive the two least significant VSB Address lines (AD00 and AD01) during the address broadcast phase, respectively on the VSBus during the address phase of VSB transactions. These are the low–order address bits; AD00 and AD01, respectively. They are used to allow byte and word transactions to be conducted on the 32–bit VSBus. Table 3–17, on page 55, shows how these bits, the high–order address bits in this register (D7 (VSA_31), D6 (VSA_30) and D5 (VSA_29)) and the 'C40's Global Address Bus are mapped into VSB addresses.

3.12.1.2 Block Transfer Enable Bit – D15 (BLT_Enable)

Set this bit to the logic '1' state to allow block transfers over the VSBus. For single byte, word or longword transfers, clear this bot to the logic '0' state.

3.12 Model 4284 VSB Interface (Option 012) (continued)

3.12.1 The VSB Control Register (continued)

3.12.1.3 VSB Bus Error Bit – D14 (VSB_$\overline{\text{BERR}}$), Read Only

If this bit reads back in the logic '0' state, a VSBus error occurred during the previous data cycle. This bit will read back in the logic '1' state if the previous data cycle completed without error.

3.12.1.4 VSB Interrupt Request Bit – D13 (VSB_$\overline{\text{IRQ}}$), Read Only

The VSB_$\overline{\text{IRQ}}$ line is used by slaves to request an interrupt from a master. When this bit reads back inthe logic '0' state, an interrupt is being requested. A logic '1' in this bit indicates that no interrupt is requested. The master responds to an interrupt request in one of two ways:

1) It might initiate a VSB IACK (Interrupt Acknowledge) cycle. Slaves that have an interrupt request pending participate in the IACK cycle to determine the one slave whose interrupt requests will be serviced. The active master then reads status and ID information from this slave.

2) It might initiate a VSB read cycle, polling the various slave devices for status and ID information until a device that has an interrupt request pending is found.

3.12.1.5 VSB_$\overline{\text{LOCK_Enable}}$ Bit – D12

The VSBus supports Indivisible–Access data cycles (e. g., Read–Modify–Write). The processor conducting such cycles needs the ability to retain mastership of the bus until all phases of the access are complete. This bit supports such activity by locking out other processors that may be requesting the bus during this time. Clear this bit to the logic '0' state to enable the bus lock. When the Indivisible–Access cycle is complete, this bit should be set to logic '1' state to allow other processors access to the VSBus. If the Model 4284 is the only VSBus master in your system, this bit may be left in the default logic '0' state.

3.12 Model 4284 VSB Interface (Option 012) (continued)

3.12.1 The VSB Control Register (continued)

3.12.1.6 VSB Address Space Select Bits – D11 (Space_1) & D10 (Space 0)

These two bits drive the Space1 and Space 0 Address Lines on the VSB Backplane, through inverting buffers. These lines are analogous to the VME Address Modifier lines, AM0 – AM5, in that they select an address space. The VSB specification defines three address spaces, System, I/O and Alternate. Table 3–15, below, gives the settings of this bit field to select each available address space. Both of these bits default to the logic '0' state at power–up, which selects System address space.

Table 3–15: Model 4284, Option 012 – Address Space Selection

Address Space	Space_1 (D11)	Space_0 (D10)
System Address Space	0	0
I/O Address Space	0	1
Alternate Address Space	1	0
Reserved – Do Not Use	1	1

NOTE: These bits are inverted before delivery to the VSB Space1 and Space0 address lines.

3.12.1.7 VSB Data Size Bits – D9 (Size_1) & D8 (Size 0)

These bits drive the $\overline{\text{SIZE0}}$ & $\overline{\text{SIZE1}}$ address lines on the VSBus. During the address broadcast phase, the active master drives $\overline{\text{SIZE0}}$ – $\overline{\text{SIZE1}}$ with a size code. The size code indicates the number of byte location(s) that the master wishes to access during the data transfer phase. The VSB specification defines four sizes for data transfer: Single–Byte, Double–Byte, Triple–Byte, and Quad–Byte. During these byte transfers, the master requests access to byte locations. During Single–Byte transfers, access to one byte is requested, and during Quad–Byte transfers, four consecutive bytes are requested. During Double– and Triple–byte transfers, the master requests two and three consecutive bytes respectively. Table 3–16, at the top of the next page, shows the bit settings for the four data sizes.

3.12 Model 4284 VSB Interface (Option 012) (continued)

3.12.1 The VSB Control Register (continued)

3.12.1.7 VSB Data Size Bits (continued)

Table 3–16: Model 4284, Option 012 – Data Size Selection

Data Size	Size_1 (D9)	Size_0 (D8)
Longword (Quad–byte)	0	0
Single–byte	0	1
Word (Double–byte)	1	0
Triple–byte	1	1

3.12.1.8 High–Order VSB Address (Page) Bits – VSA_31 (D7), VSA_30 (D6) & VSA_29 (D5)

The VSA_31 (D7), VSA_30 (D6) and VSA_29 (D5) bits drive the three most significant VSB address lines (AD31, AD30 and AD29, respectively) during the address broadcast phase. They essentially behave as page bits, allowing access to the entire 4 GByte VSB address region using only a 128 M–Longword (1 Gbyte) region of 'C40 memory. Table 3–17, below, shows how these three bits, the two low–order address bits (D17 (VSA_1) and D16 (VSA_0)) in this register, and the 'C40 Global Address bus map into VSB Address bits.

Table 3–17: Model 4284, Option 012 – VSB Address Bit Mapping

VSB Address/Data Bits																																
AD 31	AD 30	AD 29	AD 28	AD 27	AD 26	AD 25	AD 24	AD 23	AD 22	AD 21	AD 20	AD 19	AD 18	AD 17	AD 16	AD 15	AD 14	AD 13	AD 12	AD 11	A3 10	AD 9	AD 8	AD 7	AD 6	AD 5	AD 4	AD 3	AD 2	AD 1	AD 0
D 7	D 6	D 5	GA 26	GA 25	GA 24	GA 23	GA 22	GA 21	GA 20	GA 19	GA 18	GA 17	GA 16	GA 15	GA 14	GA 13	GA 12	GA 11	GA 10	GA 9	GA 8	GA 7	GA 6	GA 5	GA 4	GA 3	GA 2	GA 1	GA 0	D 17	D 16
VSBCR			'C40 Global Address Bits																											VSBCR	

NOTE: VSBCR = VSB Control Register

This page is intentionally blank

Chapter 4: Working with the Model 4284

4.1 Introduction

The discussions in the sections that follow will make reference to items contained in Figure 4–1, on the following page. This is a more detailed block diagram of the Model 4284 than the one presented in Figure 1–1, which only showed which resources were accessible from which bus. By contrast, Figure 4–1 shows what other resources may be affected when a given resource is accessed.

4.2 Resetting and Booting the Model 4284

There are three distinct paths by which the 'C40 processor on the Model 4284 may be reset. These are the Power–up/Push–button Reset, the VME System Reset, and the Host Control Reset. There are subtle differences in the results of each of these resets, which will be described in the sections below.

4.2.1 The Power–Up or Push–Button Reset

When power is initially applied to the Model 4284, the RESET input on the TMS320C40 is held low for a brief period of time by an RC network. This activation of the RESET input places the 'C40 processor in the state described in Table 6–4 (Pin States After System Reset), in Section 6.7 (Reset Operation) of the Texas Instruments **TMS320C4x User's Guide**. Assuming that no jumper is installed on JB3 (this enables the Internal Boot ROM, and is the factory default condition), when the Reset is released control of the 'C40 passes to the code found at the Reset Vector location. This location is determined by the contents of the VME_RST(1,0) bits in the Host Control Register (see Table 3–13, in Section 3.11.2.3 of this manual). The power–up reset also clears these two bits, thus pointing to the 'C40's Internal ROM at location 0x0000 0000. The instructions found there direct the 'C40 to read its IIOF(1–3) pins and, based on the result of that read and an internally stored table (see Table 4–1, below), jump to an address where it will find the code to boot load.

Table 4–1: Model 4284 – 'C40 Boot Code Address Jumper Settings – Jumper Block JB4

Address	Location	Pins 1 & 2	Pins 3 & 4	Pins 5 & 6
COMM PORT	Comm port	OFF	OFF	OFF
0x0030 0000*	Ext. EPROM	ON	OFF	OFF
0x4000 0000	Do Not Use	OFF	ON	OFF
0x6000 0000	Not Available	ON	ON	OFF
0x8000 0000	Do Not Use	OFF	OFF	ON
0xA000 0000	Not Available	ON	OFF	ON
0xC000 0000	Do Not Use	OFF	ON	ON
Undefined	Do Not Use	ON	ON	ON

* – Factory Default Setting – Also used for boot from Flash EEPROM or Global DRAM

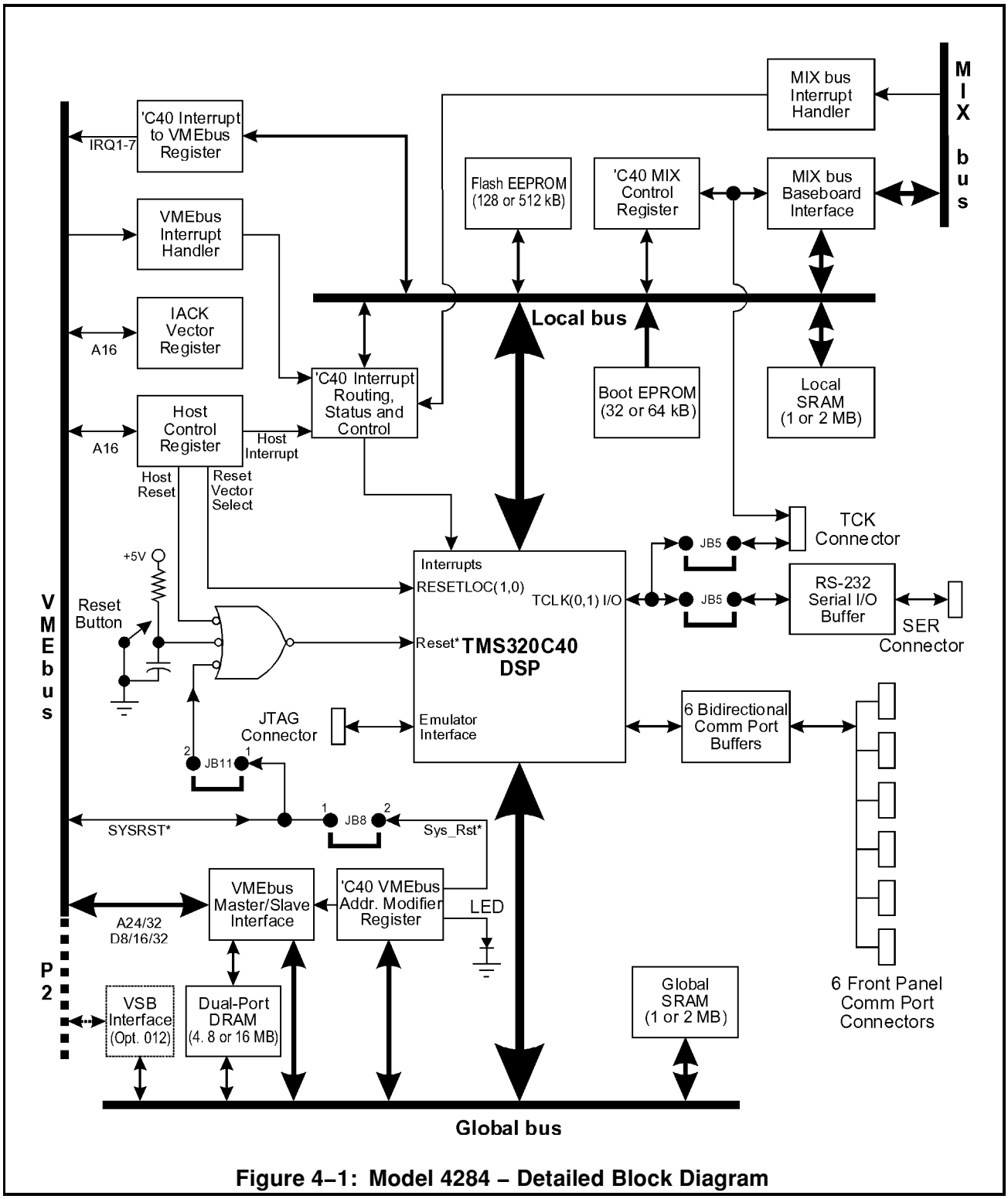

Figure 4–1: Model 4284 – Detailed Block Diagram

4.2 Resetting and Booting the Model 4284 (continued)

4.2.1 The Power–Up or Push–Button Reset (continued)

The 'C40's IIOF(1–3) pins are driven by the output of a multiplexer. During resets, the multiplexer transfers the logic states found at Jumper Block JB4 (installed jumper = 0, absent jumper = 1) to the IIOF pins. The factory setting of JB4 sets the jump location to 0x0030 0000, which is the beginning of the 4284's EPROM. The code provided in the EPROM, after it is boot loaded, initializes some of the 'C40's internal registers, among them the Local and Global Bus Interface Control Registers. Pointers to the Look, Set and other useful functions contained in the EPROM (including the Flash EEPROM support functions) are planted in DRAM. These are described in Table 4–2, below. The EPROM code also initializes the registers on the 4284 that are external to the 'C40, as described in Table 4–3, at the top of the next page.

Table 4–2: Model 4284 – DRAM Locations of Pointers to EPROM Functions

VME Slave Offset (DRAM_Base+...)	'C40 Address	Contents
0x0000 0800	0x8000 0200	Function (command) address from Host
0x0000 0804	0x8000 0201	First Command Argument - Address for Look/Set Function or Source Adress for block move of Flash Boot Address.
0x0000 0080	0x8000 0202	Data from Host for set function or destination address for block move
0x0000 080C	0x8000 0203	Second Command Argument - Data to Host for Look or Word cound -1 for block move
0x0000 0810	0x8000 0204	Ready to Host (0=not ready / 1=ready)
0x0000 0814	0x8000 0205	Pointer to LOOK function
0x0000 0818	0x8000 0206	Pointer to SET function
0x0000 081C	0x8000 0207	Pointer to BLOCK MOVE function
0x0000 0820	0x8000 0208	Aux ready flag for factory use
0x0000 0824	0x8000 0209	Test pattern 0xA5A5 A5A5
0x0000 0828	0x8000 020A	Test pattern 0x5A5A 5A5A
0x0000 082C	0x8000 020B	Test pattern 0x1234 5678
0x0000 0830	0x8000 020C	Test pattern 0x8765 4321
0x0000 0834	0x8000 020D	Test pattern 0xFFFF FFFF
0x0000 0838	0x8000 020E	Firmware Version 0x10C=1.0c
0x0000 083C	0x8000 020F	Test pattern 0x8000 0200
0x0000 0840	0x8000 0210	Pointer to FLASH BOOT function
0x0000 0844	0x8000 0211	Pointer to FLASH LOOK function
0x0000 0848	0x8000 0212	Pointer to FLASH SET function
0x0000 084C	0x8000 0213	Pointer to FLASH ERASE function
0x0000 0850	0x8000 0214	Pointer to FLASH ERASE SECTOR function
0x0000 0854	0x8000 0215	Pointer to FLASH BYTE LOAD function
0x0000 0858	0x8000 0216	Pointer to FLASH BYTE LOAD function
0x0000 085C	0x8000 0217	Pointer to FLASH BYTE UNLOAD
0x0000 0860	0x8000 0218	Pointer to FLASH WORD LOAD function
0x0000 0864	0x8000 0219	Pointer to FLASH WORD UNLOAD function

4.2 Resetting and Booting the Model 4284 (continued)

4.2.1 The Power–Up or Push–Button Reset (continued)

Table 4–3: Model 4284 – Register States After Boot from Factory EPROM

Register	Address	Condition	Comments
IACK Vector Register	A16_base+0x00	Indeterminate	Must be initialized by user code before interrupting VME
HOST Control Register	A16_base+0x04	All bits cleared (0)	Host Reset & Interrupt Inactive, Reset Vector is start of internal ROM, A24 access enabled
'C40 VME Modifier Register	0x9000 0000	0x3D40*	Page=0, AM=3D, Host Int cleared, LED on*, bus lock off, 32-bit VME transfers
'C40 MIX Bus Control Register	0x1000 0000	All bits cleared (0)	MIX bus will be ready for use after MXRST (D15) is toggled, byte enables set for 32-bit, single-cycle transfers
'C40 Interrupt Status and Control Register	0x1000 0002	D0-D7=0, D8-D10 are indeterminate	No pending interrupts, Interrupter Level must be written to D8-D10 before interrupting VME
'C40 Interrupt Routing Register	0x1000 0003	0x0000 4000	Host Interrupt ONLY enabled, connected to INT_1
'C40 Interrupt to VME Register	0x1000 0004	Cleared (0)	Initialize IACK Vector Reg and IRQ_lev (0-2) bits(D8-D10) in 'C40 Interrupt S&C Reg before issuing this interrupt
* - The state of the LED bit (and the LED) should toggle at ≅5 Hz following standard EPROM initalization			

After register initialization, the EPROM code directs the 'C40 to read the first location in the Global DRAM ('C40 address 0x8000 0000). Based on the contents of that memory location, the 'C40 will perform one of the following actions:

1) If the data at 0x8000 0000 is 0xA55A 5AA5, then the 'C40 uses the location pointed to by the next Global DRAM address (0x8000 0001) as the boot load address.

2) If the data at 0x8000 0000 is 0xC33C 3CC3, then the 'C40 will return to EPROM and continue to execute the factory default boot code found there. This is used to block the boot loading of valid code stored in the Flash EEPROM (see below), if desired. The processor is left running a program that toggles the front panel LED and awaiting an interrupt to direct it elsewhere.

3) If neither of the codes listed above are found at 0x8000 0000, the 'C40 then reads the first address of the Flash EEPROM (0x00B0 0000.) Based on the contents of this address, one of the following will occur:

   a) If the data at 0x00B0 0000 is 0xA55A 5AA5, the 'C40 will boot load and execute the code contained in the Flash EEPROM.

4.2 Resetting and Booting the Model 4284 (continued)

4.2.1 The Power–Up or Push–Button Reset (continued)

3) DRAM Boot (continued)

b) If the data at 0x00B0 0000 is anything different from 0xA55A 5AA5, the 'C40 will resume executing the code from the standard EPROM. The processor is left running a program that toggles the front panel LED and awaiting an interrupt to direct it elsewhere.

NOTE: The boot sequence described above is followed only if your Model 4284 contains Boot EPROM Revision E or higher. If your 4284 is unable to boot from DRAM in the manner described above, contact Pentek at (201) 818–5900, and we will arrange for an EPROM upgrade.

Table 4–4, below, summarizes the addresses read during the boot procedure, the significant data codes that may found at those addresses, and the actions taken if those codes are detected.

Table 4–4: Model 4284 – Addresses Read During EPROM Boot

Address	Resource	Data Code	Action
0x8000 0000	Global DRAM	A55A 5AA5	Boot Load from address pointed to by data at 0x8000 0001
		C33C 3CC3	Resume Boot of EPROM Code (Blocks Flash EEPROM Boot)
		Any other data	Action based on contents of 0x00B0 0000
0x00B0 0000	Flash EEPROM	A55A 5AA5	Boot Load from Flash EEPROM (see Section 4.3.5)
		Any other data	Resume Boot of EPROM code

If no problems are encountered in the boot procedure, and the processor followed one of the two possible paths that lead to continuing with the standard EPROM boot, the LED on the 4284's front panel will blink about five times a second (the blink rate is dependent upon the speed of the processor – the 'C40's Timer 0 is used as a counter). If the factory boot code was executed and the LED only blinks around once per second, then the 'C40 has had trouble accessing the shared DRAM. If the factory boot code was executed and the LED does not blink at all, then a more serious problem exists.

Pressing the Reset button on the Model 4284's front panel will initiate a reset cycle identical to the power–up reset described above.

4.2 Resetting and Booting the Model 4284 (continued)

4.2.2 The VME System Reset

In order for the VME System Reset signal to reach the 'C40 processor on the Model 4284, a jumper must be installed between pins 1 and 2 of jumper block JB11 (see Section 2.8). The reset cycle that results from a VME System Reset is nearly identical to the Power−Up Reset described above, except that the Host Control Register is not cleared, so its contents will be unchanged by this reset.

If the 4284 is to be your Slot 1 VME System Controller, it must be configured to drive the VME System Reset line by installing a jumper between pins 1 and 2 of Jumper Block JB8. In this case, you may or may not want the System Reset signal to reset the 'C40 (i. e., the presence or absence of the jumper at JB11 1−2 is dictated by the requirements of your application).

The System Reset pulse delivered by the 4284 comes from a one−shot generator (74HCT4538), and is triggered by writing a '1' to bit D0 of the VMEbus Modifier Register at 'C40 address 0x9000 0000.

4.2.3 The Host Control Reset

This reset is triggered when a VMEbus Master writes a '1' to bit D0 of the VME Host Control Register at address A16_base+0x04. The 'C40 processor will be held in the reset state until either (a) a VMEbus Master writes a '0' to D0 of the Host Control Register, or (b) a Power−Up or Push−Button Reset occurs.

The effect of this Reset on the 4284 is nearly identical to that of a Power−Up Reset. The differences lies in the fact that, like the VME System Reset, this does not clear the Host Control Register (i. e., this bit does not reset itself). Additionally, the Host Control Reset **DOES NOT RESET the MIX or VSB interfaces**, which are reset by the Power−up, Push−button and VME Syatem Resets.

4.2.4 Booting From User Code

By rearranging the jumpers placed on JB4, the 'C40 on the Model 4284 can be made to look at the Comm Ports, rather than the Pentek−supplied EPROM, for its boot code. Table 4−1, in Section 4.2.1 of this manual (see page 57), describes the jumper setting needed to boot from the Comm Port Interface. Jumper settings are also physically available that point to the Local and Global SRAMs, but these are not very useful because the SRAMs can only be written by the 'C40.

4.2 Resetting and Booting the Model 4284 (continued)

4.2.4 Booting from User Code (continued)

The boot may also be run from a program residing in the 4284's Global DRAM or in the Flash EEPROM. To boot from either of the above, the jumpers on JB4 should remain at the factory settings (i. e., to boot from EPROM), shown in Table 4–1. The EPROM boot code directs the 'C40 to read the first location in the DRAM (0x8000 0000). Based on the contents of that address, the 'C40 may boot from DRAM, Flash EEPROM or resume the standard EPROM boot. See Section 4.2.1, above, for further details

To boot the 4284 from a Comm Port, set the jumpers on JB4 as shown in the top line of Table 4–1, install the 4284 in the VME card cage, and apply power. The loader waits for the first input data from any of the six Comm Ports, then loads the code from that channel.

NOTE: When the Model 4284 is booted from the code provided in the factory–supplied Boot EPROM, the value stored in the 'C40's internal Local Memory Interface Control Register (address 0x0010 0004) is manipulated. If the EPROM boot is allowed to complete, this register will contain the value 0x3DEF 0000, which allows the 'C40 to properly access the MIX bus. If, however, the boot is re–directed to another source (e. g., Flash EEPROM or Comm Port), the instruction that sets this value will never be reached. In such cases, the "new" boot code must also write the value 0x3DEF 0000 to 'C40 address 0x0010 0004, to enable access to the MIX bus.

A more detailed description of the boot load procedure and the structure of the boot source program's data stream can be found in Section 13.7.2 (Boot Loading Sequence) of the Texas Instruments **TMS320C4x User's Guide**. An example of 'C40 Assembly Language source code for a boot loader program is provided in Section 13.7.4 (Example of Communication Port Boot Loading) of that manual.

4.3 Flash EEPROM Operations

The Flash EEPROM Options (Options 002 (128k x 8) and 003 (512k x 8)) support non–volatile storage of programs for boot loading. Other data, such as short tables, may also be stored here. This resource is mapped into the 'C40's Local Bus memory map, beginning at address 0x00B0 0000.

Any data to be stored in this memory region must be in Intel HEX format. A program for converting COFF program files into this format, **`hex30`**, is provided by Texas Instrtuments as a part of their Optimizing 'C' Compiler package.

4.3 Flash EEPROM Operations (continued)

In most cases, when this option is included on the Model 4284, the user's goal is to have the processor boot load the code in the Flash EEPROM immediately upon power−up or warm reset. The procedures involved in accomplishing that goal will involve all of the operations one might want to perform using the Flash memory. We will therefore describe the procedures for loading bootable HEX code into the Flash EEPROM, and boot loading into that code, in the sections below.

4.3.1 Converting COFF Files into Intel HEX Format

Program files to be written into the 4284's Flash EEPROM must be in Intel HEX format. COFF (Common Object File Format) files, of the type normally written into the 4284's DRAM or SRAM regions, can be converted into Intel HEX format by creating an appropriately−named linker command file for the program, and passing the linker command file's name to the **hex30** program, supplied by Texas Instruments. The COFF file to be converted should have the extension **.out** or **.x40**. The linker command file's name should have the extension **.cmd**. An example of such a linker command file, **boot84.cmd**, is presented in Figure 4−2, below, and is also included in Section A.6 of this manual. The command line syntax to create the Intel HEX file **demo84.hex** from the COFF file **led84.out**, using the linker command file shown in Figure 4−2 would be:

```
hex30 boot84.cmd<enter>
```

```
/************* FILE - boot84.cmd **************/
/**********************************************/
/*        Boot build for C40 EEPROM           */
/**********************************************/
led84.out                /* Input file         */
-o demo84.hex            /* HEX output file    */
-i                       /* intel format       */
-memwidth 8              /* 8-bit memory       */
-boot                    /* Convert all .sect  */
-bootorg     0h          /* Boot source addr   */
-cg   3D9EC000h          /* Global mem conf.   */
-cl   3DEF4110h          /* Local mem conf.    */
-e    c0000c53h          /* PC after load addr */
-iack 4003fc00h          /* IACK dummy read    */
-ivtp 4003fc00h          /* IV pointer         */
-tvtp 4003fc00h          /* TV pointer         */
/**********************************************/
```

Figure 4−2: Model 4284 – Linker Command File for COFF – HEX File Conversion

4.3 Flash EEPROM Operations (continued)

4.3.1 Converting COFF Files into Intel HEX Format (continued)

To modify the linker command file shown at the bottom of the previous page to work with COFF files of your own, substitute the name of your COFF file on the **/*Input file*/** line, and the name you wish the new HEX file to be given after the **-o** on the **/*HEX output file*/** line. Examine the map file created when your COFF file was compiled to find the entry point, and substitute that address after the **-e** on the **/*PC after load addr*/** line.

Before compiling the file that will be passed to the conversion routine shown above, it may be necessary to add the line shown below to the **.main** section of your source code. This line is needed if the code involves access to the MIX bus. It establishes a pointer to the 'C40's Local Control Port from Flash EEPROM space.

```
*(unsigned int*)0x100004=0x3DEF4000
```

4.3.2 Loading the HEX File into the Shared Global DRAM

The procedure above created a HEX file, residing in your workstation, that is suitable for writing into the 4284's Flash EEPROM. The Flash EEPROM on the Model 4284 resides on the 'C40's Local bus, and can be accessesed **ONLY** by the 'C40. Therefore, the HEX file must be written into the 4284's shared memory area (i. e., the Global DRAM) before it can be written into the Flash EEPROM.

The HEX file should be written into the 4284's Global DRAM in the format shown in Table 13–4 (Byte–Wide Configured Memory) , in Section 13.7.3 (External Memory Boot Loading) of the Texas Instruments **TMS320C4x User's Guide**. The contents of the HEX file must be offset from the beginning of the block that will be transferred into the Flash EEPROM by 8 bytes (2 longwords). This is necessary so that the Flash Boot Flag (0xA55A 5AA5) and Flash Boot Code Start Address (0x00B0 0008) may be inserted at the first two locations in the block.

For example, if one wished to load a HEX file at the Global DRAM block beginning at VMEbus address DRAM_base+0x0000 1000, the first byte of the HEX file should be placed at VMEbus address DRAM_base+0x0000 1008.

4.3.3 Erasing the Flash EEPROM

Before new data can be loaded into the Flash EEPROM, the existing data must first be erased. Two methods are available for erasing the Flash memory. The first method is the simpler of the two, but it can only be used erase the entire content of the EEPROM. The second method can be used to selectively erase 64 kByte sectors of the EEPROM.

4.3 Flash EEPROM Operations (continued)

4.3.3 Erasing the Flash EEPROM (continued)

4.3.3.1 Erasing All of Flash EEPROM from the TCK Connector

To erase the entire contents of the Flash EEPROM, connect a shorting jumper between pins 3 and 5 of the front panel TCK connector (see Figure 4–3, below), and then push the Reset button. When the 4284 is reset in this manner, the LED indicator will remain lit until the Flash EEPROM is completely erased, and then blink normally.

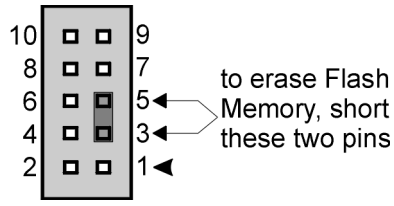

Figure 4–3: TCLK I/O Connector

4.3.3.2 Erasing Selected Flash EEPROM Sectors

The Flash EEPROM used on the Model 4284 is partitioned into 64 kByte sectors. Option 002 (128 kBytes) has two sectors, and Option 003 (512 kBytes) has eight sectors. Table 4–5, below, lists the starting addresses of each Flash EEPROM sector.

Table 4–5: Model 4284 – Flash EEPROM Sector Start Addresses

Sector	Start Address
0	0x00B0 0000
1	0x00B1 0000
2	0x00B2 0000
3	0x00B3 0000
4	0x00B4 0000
5	0x00B5 0000
6	0x00B6 0000
7	0x00B7 0000

NOTE: Option 002 contains only sectors 0 and 1, Option 003 contains all 8 sectors

4.3 Flash EEPROM Operations (continued)

4.3.3 Erasing the Flash EEPROM (continued)

4.3.3.2 Erasing Selected Flash EEPROM Sectors (continued)

Individual sectors of the Flash memory can be erased by calling some of the functions described in Table 4–2, on page 59 of this manual. The procedure is described below:

1) Read the pointer to the Flash Erase Sector function from DRAM_base+0x0854. Write the result of that read into the Host Function Command Pointer location, DRAM_base+0x0000 0800.

2) Write the address of the sector to be erased (from Table 4–5, previous page) to the Flash Boot Address location, DRAM_base+0x0000 0804.

3) Write a '0' to the Host Ready Flag Location, DRAM_base+0x0000 0810.

4) Set the Host Interrupt Bit in the 4284's Host Control Register to the '1' state, by writing a '2' to A16_base+0x0004.

5) The firmware will now begin to erase the Flash EEPROM sector at the address given in step (2), on the previous page.

6) Poll the Host Ready Flag (DRAM_base+0x0000 0810). This will be set to the '1' state by the firmware upon completion of the sector erase operation.

If necessary, the procedure described above can be repeated to erase additional sectors. Note that the entire Flash EEPROM can also be erased using the procedure above, if you begin by copying the Flash Erase function pointer (at DRAM_base+0x0000 0850) to the Host Command location, instead of the Flash Erase Sector pointer.

4.3.4 Moving the Code from Global DRAM to Flash EEPROM

After the Flash EEPROM sectors you wish to move your code into have been erased, you are ready to copy your code from the DRAM to the Flash EEPROM. This will also involve the use of some of the firmware functions described in Table 4–2 (see page 59). A description of the procedure begins at the top of the next page.

4.3 Flash EEPROM Operations (continued)

4.3.4 Moving the Code from Global DRAM to Flash EEPROM (continued)

1) Read the pointer to the Flash Word Load function from DRAM_base+0x0000 0860. Write the result of that read into the Host Function Command Pointer location, DRAM_base+0x0000 0800.

2) Write the address of the source data in DRAM to the Flash Boot Address location, DRAM_base+0x0000 0804. If, for example, the HEX file was written to a Global DRAM block beginning at VMEbus address DRAM_base+0x0000 1000 (as described in Section 4.3.2), the equivalent 'C40 address (0x8000 0400) would be written to DRAM_base+0x0000 0804).

3) Write the starting address of the Flash EEPROM sector that you wish to move the code into (from Table 4–5, page 66) to the Block Move Destination location (DRAM_base+0x0000 0808). **If you wish to be able to boot this code by any of the conventional methods (i. e., power–up, reset button, VME System Reset or Host Control Reset), move the code to EEPROM sector 0.**

4) Subtract 1 from the number of longwords to be moved into the Flash EEPROM (i. e., (number of bytes / 4) – 1). Write the result into the Block Length location, DRAM_base+0x0000 080C.

5) Write a '0' to the Host Ready Flag Location, DRAM_base+0x0810.

6) Set the Host Interrupt Bit in the 4284's Host Control Register to the '1' state, by writing a '2' to A16_base+0x0004.

7) The firmware will now begin to move 32–bit longwords from the DRAM address given in step (2) of this procedure to the Flash EEPROM sector at the address given in step (3) (see previous page).

8) Poll the Host Ready Flag (DRAM_base+0x0000 0810). This will be set to the '1' state by the firmware upon completion of the block move operation.

4.3.5 Booting the Flash EEPROM Program

After following the procedure above, you have a choice of methods that you may use to execute the code that you have stored in the Flash EEPROM. If you loaded the code into sector 0 of the Flash EEPROM, then it can be exe–cuted at power–up (see Section 4.2.1), by pushing the reset button, or by any of the other reset methods described in Section 4.2.

4.3 Flash EEPROM Operations (continued)

4.3.5 Booting the Flash EEPROM Program (continued)

If the code was not loaded into sector 0, then the code cannot be booted by conventional methods. It can be booted, however, by the firmware functions in EPROM. The procedure for this is described below.

1) Read the pointer to the Flash Boot function from DRAM_base+0x0860. Write the result of that read into the Host Function Command Pointer location, DRAM_base+0x0800.

2) Write the Flash EEPROM address of the code you wish to boot to the Flash Boot Address location, DRAM_base+0x0804.

3) Set the Host Interrupt Bit in the 4284's Host Control Register to the '1' state, by writing a '2' to A16_base+0x04.

4) Firmware will now boot the code that it finds at the Flash EEPROM at the sector address given in step (2).

NOTE: When the Model 4284 is booted from the code provided in the factory−supplied Boot EPROM, the value stored in the 'C40's internal Local Memory Interface Control Register (address 0x0010 0004) is manipulated. If the EPROM boot is allowed to complete, this register will contain the value 0x3DEF 0000, which allows the 'C40 to properly access the MIX bus. If, however, the boot is re−directed to another source (e. g., Flash EEPROM), the instruction that sets this value will never be reached. In such cases, the "new" boot code must also write the value 0x3DEF 0000 to 'C40 address 0x0010 0004, to enable access to the MIX bus.

4.4 Downloading Programs to the Model 4284

'C40 programs in Common Object File Format (COFF) can be transferred into the 4284's shared DRAM over the VMEbus. One recommended method for achieving this file transfer is via Pentek's SwiftNet Communications Protocols. Another is via the Texas Instruments XDS−510 Emulator.

4.4.1 Downloading Programs with SwiftNet

Pentek's SwiftNet Communications Protocols are used to provide data transfer links over a distributed DSP network. Included with SwiftNet is a utility program, **PNCFG**, used to identify and provide essential information about the processors in the network. Another Program, **PNLOAD**, is used to transfer COFF files from the host system on which they were developed to target DSPs defined in **PNCFG**. COFF files may be developed in Pentek's SwiftTools Environment, which can call **PNLOAD** to accomplish the file transfer, or with any 'C40 compatible compiler/assembler/linker package. See the SwiftNet and SwiftTools Operating Manuals for further details.

4.4.2 Downloading Programs with the XDS−510 Emulator

Users of the Texas Instruments XDS−510 Emulator can transfer 'C40 COFF programs into the 4284 via the front panel JTAG connector. For that code to be properly executed, however, the 4284's register set must be put into its boot state. This must be done by the user, because operating environment of the emulator prevents the 4284 from executing its boot code at power−up.

There are also two registers internal to the 'C40 which must be initialized before you try to access the external registers. The Global Memory Interface Control Register, at address 0x0010 0000, must be set to 0x3D9E C000, and the Local Memory Interface Control Register, at address 0x0010 0004, must be set to 0x3DEF 4710. You may either load these values from the emula−tor's command interface, or place commands to initialize the registers at the beginning of any program you may load. For more information about the use of these registers, see Sections 7.3 (Memory Interface Control Registers) and 7.4 (Programmable Wait States) of the the Texas Instruments **TMS320C4x User's Guide**.

One method of initializing the external register set is to issue the **RUN** com−mand from the emulator, and then push the reset button, before loading your code. Another method is to initialize the registers explicitly at the beginning of your code, after setting up the internal registers (see the last paragraph on the previous page). See Table 4−3, in Section 4.2.1 of this manual (see page 60), for the external register addresses and states to which they should be initialized.

4.5 Handling Interrupts on the Model 4284

Interrupts targeted for the four interrupt inputs of the 'C40 processor on the Model 4284 can come from any three of the seven VMEbus IRQs , from the three MIX module interrupt lines, from the Host Control Register, or may be caused by a VMEbus time−out. The sections below describe and contrast the handling of interrupts from all these sources.

4.5.1 Handling VMEbus Interrupts

Up to three of the seven VMEbus IRQs may be connected to the 'C40 processor on the Model 4284. These are selected by placing jumpers on jumper block JB13. Table 4−6, below, describes the settings available on this jumper block.

Table 4−6: Model 4284 − VMEbus IRQ Configuration Jumpers Jumper Block JB13

VMEbus IRQLevel	'C40 Interrupt Requests		
	VME_IRQ_A	VME_IRQ_B	VME_IRQ_C
IRQ1	1 – 2	N/A	N/A
IRQ2	3 – 5	11 – 13	13 – 14
IRQ3	2 – 4	11 – 12	12 – 14
IRQ4	5 – 6*	6 – 8	14 – 16
IRQ5	5 – 7	7 – 8	15 – 17
IRQ6	19 – 20	9 – 11*	17 – 19
IRQ7	18 – 20	8 – 10	17 – 18*

N/A = Not Available * – Factory Default Settings

Selecting the VMEbus IRQs that can interrupt the processor does not quite finish the job of interrupt enabling, however. The 'C40 Interrupt Routing Register, at 'C40 address 0x1000 0003, must be programmed to a value that will select the interrupts you need to respond to (from among the eight possible interrupt sources on the 4284), and connects one or more of them to one or more of the four interrupt inputs on the 'C40.

4.5 Handling Interrupts on the Model 4284 (continued)

4.5.1 Handling VMEbus Interrupts (continued)

Table 4–7, below, describes the connections that can be made by this register. A '1' in any bit connects the source to the indicated interrupt. As shown in the table, the low order byte of this 32–bit register (D0 – D7) connects interrupt sources to INT_0, the next higher order byte (D8 – D15) connects them to INT_1, the second highest order byte (D16 – D23) connects with INT_2, and the most significant byte delivers interrupts to INT_3. As far as the VME IRQs are concerned, the least significant bit in each byte (D0, D8, D16 and D24) connects VME_IRQ_A to that byte's 'C40 interrupt input, the next higher order bit in each byte (D1, D9, D17 and D25) connects VME_IRQ_B, and VME_IRQ_C is routed to the 'C40 by means of the third bit in each byte of the register (D2, D10, D18, and D26). By default, the only interrupt that this reg–ister enables (when the 4284 is booted with the code Pentek supplies in the EPROM), is the Host Control Interrupt, which gets connected to INT_1.

Table 4–7: Model 4284 – 'C40 Interrupt Routing Register – R/W @ 0x1000 0003

Bit #	D31	D30	D29	D28	D27	D26	D25	D24
Bit Name	TimeOutInt	HostInt	MXINT2	MXINT1	MXINT0	VME_IRQ_C	VME_IRQ_B	VME_IRQ_A
Bit Function	1 = Interrupt connected to 'C40 INT_3 0 = Interrupt not connected to 'C40 INT_3							
Bit #	D23	D22	D21	D20	D19	D18	D17	D16
Bit Name	TimeOutInt	HostInt	MXINT2	MXINT1	MXINT0	VME_IRQ_C	VME_IRQ_B	VME_IRQ_A
Bit Function	1 = Interrupt connected to 'C40 INT_2 0 = Interrupt not connected to 'C40 INT_2							
Bit #	D15	D14	D13	D12	D11	D10	D9	D8
Bit Name	TimeOutInt	HostInt	MXINT2	MXINT1	MXINT0	VME_IRQ_C	VME_IRQ_B	VME_IRQ_A
Bit Function	1 = Interrupt connected to 'C40 INT_1 0 = Interrupt not connected to 'C40 INT_1							
Bit #	D7	D6	D5	D4	D3	D2	D1	D0
Bit Name	TimeOutInt	HostInt	MXINT2	MXINT1	MXINT0	VME_IRQ_C	VME_IRQ_B	VME_IRQ_A
Bit Function	1 = Interrupt connected to 'C40 INT_0 0 = Interrupt not connected to 'C40 INT_0							

Because this architecture allows multiple interrupt sources to share a single interrupt input on the 'C40, another register is provided to determine the actual source of the interrupt. One of the eight LSBs of the 'C40 Interrupt Status and Control Register, at address 0x1000 0002, is set to the '1' state by the source of an incoming interrupt. Table 4–8, on the next page, describes the significance of these bits. The 'C40 must read this register to determine where the interrupt came from. If the source of the interrupt turns out to be one of the VMEbus interrupts, the 'C40 initiates an Interrupt Acknowledge (IACK) cycle, described in the following section. Section 3.3 provides fur–ther detail about the 'C40 Interrupt Status and Control Register.

4.5 Handling Interrupts on the Model 4284 (continued)

4.5.1 Handling VMEbus Interrupts (continued)

Table 4–8: Model 4284 – 'C40 Interrupt Status and Control Register – R/W @ 0x1000 0002								
Bit #	**D7**	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
Bit Name	TimeOutInt	HosInt	MXINT2	MXINT1	MXINT0	VME_IRQ_C	VME_IRQ_B	VME_IRQ_A

4.5.1.1 The VMEbus IACK Cycle

When a VMEbus Interrupt Handler detects an interrupt on a level that it is assigned to monitor, it must first arbitrate for control of the data transfer bus. When the bus is granted to the Handler, it places a 3–bit code on the three LSBs of the VME address bus that is equivalent to the level it was interrupted on (e. g., if the interrupt occurred in IRQ7, the Handler would put three 1's on the least significant address lines, A01 to A03), and asserts the $\overline{\text{IACK}}$ and $\overline{\text{AS}}$ signals. The 4284 fetches these address codes from the IACK level code registers at addresses 0xA000 0001 – 0xA000 0007. If the interrupt was on an odd IRQ level (1, 3, 5 or 7), the code read from these registers is in the upper 16 bits of the 32–bit longword, and must be shifted into the lower word before placing the code on the address bus. See Section 3.9 for more detail about these registers.

When the $\overline{\text{IACK}}$ signal goes low, all Interrupters that have a pending interrupt monitor those three address lines and their $\overline{\text{IACKIN}}$ line. The $\overline{\text{IACK}}$ signal is directly connected to the $\overline{\text{IACKIN}}$ line for slot 1, the system controller. If the system controller has no pending interrupt, or if the interrupt it has pending is not on the level indicated by the address lines, then it passes the signal to its $\overline{\text{IACKOUT}}$ line, where it connects to the $\overline{\text{IACKIN}}$ of the module in slot 2. The $\overline{\text{IACK}}$ signal is passed down the daisy chain in this manner until it reaches a device that has a pending interrupt on the indicated level. That device does not pass the signal on to the next slot, but instead places its IACK Vector Code on the data bus and asserts the $\overline{\text{DTACK}}$ signal.

When the Handler detects the low on $\overline{\text{DTACK}}$, it reads the Interrupter's IACK Vector Code from the Data bus, and de–asserts the $\overline{\text{AS}}$, the address bits and the $\overline{\text{IACK}}$ signal. This completes the IACK cycle and the 'C40 can commence execution of an appropriate Interrupt Service Routine. At some point, either during the IACK cycle or the ISR (depending upon the characteristics of the Interrupter), the interrupting device removes its IRQ, which clears the corresponding bit in the 4284's Interrupt Status Register.

4.5 Handling Interrupts on the Model 4284 (continued)

4.5.2 Handling MIX Bus Interrupts

Interrupts issued by expansion modules on the MIX bus are connected directly to the 'C40 Interrupt Routing Register at address 0x1000 0003, so there is no need to set any jumpers to pass these interrupts. The Routing register, described by Table 4–7 on page 72, may be programmed to deliver interrupt signals from the three module positions to any of the 'C40's four interrupt inputs.

As stated above, the low order byte of this 32–bit register (D0 – D7) connects interrupt sources to INT_0, the next higher order byte (D8 – D15) connects them to INT_1, the second highest order byte (D16 – D23) connects with INT_2, and the most significant byte delivers interrupts to INT_3. The interrupt from MIX module 0 is connected to the indicated 'C40 input by writing a '1' to the 4th bit of that interrupt's byte (i. e., D3, D11, D19 or D27). Module 1's interrupt is connected by the 5th bit in each byte (D4, D12, D20 or D28), and Module 2's by the 6th bit (D5, D13, D21 or D29).

When an interrupt is detected on any of the four 'C40 inputs, the 'C40 first reads the Interrupt Status and Control Register to determine where the inter–rupt came from (see Table 4–8, at the top of the previous page). Although the MIX specification includes provisions for bus–vectored interrupts (similar to the VMEbus implementation of the IACK cycle), none of Pentek's MIX mod–ules for VMEbus systems implement such interrupts. Thus, no IACK cycle occurs in response to an interrupt from a MIX module, and the ISR can begin as soon as the Handler is ready. When the Interrupter is first addressed by the Handler, it will de–assert its interrupt signal, which will clear the corre–sponding bit in the 4284's Interrupt Status Register.

4.5.3 Handling the Host Control and Timeout Interrupts

The Host Control Interrupt is similar to the "mailbox" type of interrupt function implemented on many VMEbus boards. On the Model 4284, this interrupt occurs when the "host controller" (actually, any VMEbus Master) writes a '1' to the D1 bit of the Host Control Register, at VME address A16_base+0x04. Many experienced VMEbus users prefer this implementa–tion to interrupts via IRQ, because no VME IACK cycle is needed in response. To clear the Host Interrupt, a 'C40 IACK instruction must be exe–cuted.

The Host Interrupt can be routed to any of the four 'C40 interrupt inputs by setting the appropriate bits in the 'C40 Interrupt Routing Register at address 0x1000 0003 (see Table 4–7, on page 72). Setting the D6 bit to the '1' state will connect this interrupt to 'C40 INT_0. D14 routes this interrupt to INT_1, D22 sends it to INT_2, and a '1' in D30 will pass this interrupt to INT_3. The Pentek boot code in the EPROM supplied with the 4284 connects the Host interrupt to INT_1.

4.5 Handling Interrupts on the Model 4284 (continued)

4.5.3 Handling the Host Control and Timeout Interrupts (continued)

The Timeout Interrupt occurs when the 'C40 is acting as a VMEbus Master, and the module it is trying to access does not respond within 70 msec. This interrupt is also connected to the 'C40 via the Interrupt Routing Register (again, see Table 4–7, on page 72). Setting the D7 bit in this register to the '1' state will connect this interrupt to INT_0. D15 routes this interrupt to INT_1, D23 sends it to INT_2, and a '1' in D31 will pass this interrupt to INT_3. The timeout interrupt is cleared by any write operation to the 'C40 Interrupt Status and Control Register, at address 0x1000 0002.

As with the interrupts discussed in the preceding sections, the 'C40 should respond to stimulus at any interrupt input by reading the Interrupt Status Register, at address 0x1000 0002, to identify the interrupter. The processor may then service the interrupt appropriately.

4.6 Interrupting with the Model 4284

The Model 4284 can issue interrupts over the VMEbus, and participate as an interrupter in the VME IACK cycle. As a MIX bus Baseboard, the 4284 cannot use the MIX bus interrupt lines to interrupt co–processor modules which may be connected to its MIX bus, but can interrupt these devices using the "mailbox" method. The sections below describe the Interrupter operation of the Model 4284.

4.6.1 Interrupting the VMEbus

The VMEbus Interrupter level of the Model 4284 is determined by the set–tings of the IRQlev(0–2) bits (D8 – D10) in the 'C40 Interrupt Status and Control Register, at address 0x1000 0002. D10 is the most significant bit, and D8 is the least significant. These bits should be set to the binary equivalent of the level you wish to interrupt the VMEbus on. For example, if you want to interrupt VME on IRQ level 3, you would set these bits to 011_2.

A VMEbus Interrupt is asserted by the 4284 when any data is written to the 'C40 Interrupt to VMEbus Register, at address 0x1000 0004. When the 4284's interrupt is detected by a VMEbus Interrupt Handler, it will begin an IACK cycle, as described in Section 4.5.1.1. When the $\overline{\text{IACKIN}}$ signal arrives at the 4284, with the 3–bit code on the address bus that matches its Interrupter level setting, the 4284 places the contents of its IACK Vector Register, which resides at the A16_base address, on the lower 8 bits of the VME data bus, and asserts $\overline{\text{DTACK}}$. The Handler can then read the Vector and service the 4284's interrupt. To clear the interrupt and complete the IACK cycle, the Interrupt Handler must re–write the IACK Vector Register.

4.6 Handling Interrupts on the Model 4284 (continued)

4.6.2 Interrupting MIX Modules

All Pentek MIX modules that can respond to interrupts have a mailbox register that is used as an interrupt from the Baseboard. Consult the MIX module's Operating Manual for the address of this register on any particular module. No IACK cycle results from this type of interrupt. The module simply reads the register and services the interrupt.

4.7 Mastering the VMEbus

To perform bus master access in VMEbus memory space, the 'C40 accesses the region of its memory map between 0xB000 0000 and 0xBFFF FFFF. This 256M x 32 region corresponds to a page of 1 GByte of VME address space. Prior to accessing VMEbus memory space the following board resources must be set up:

1) Set the Address Modifier bits (D8 – D13), in the 'C40 VMEbus Modifier Register at address 0x9000 0000, for the desired address mode: A16, A24, or A32. This is described in Section 3.8.6. The Address Modifier table from that section is repeated below as Table 4–9.

Table 4–9: Model 4284 – Address Modifier Codes

AM Code	Function
***0x3F**	A24 – Standard Supervisory Block Transfer†
0x3E	A24 – Standard Supervisory Program Access
0x3D	A24 – Standard Supervisory Data Access
***0x3B**	A24 – Standard Nonprivileged Block Transfer
0x3A	A24 – Standard Nonprivileged Program Access
0x39	A24 – Standard Nonprivileged Data Access
0x2D	A16 – Short Supervisory Access
0x29	A16 – Short Nonprivileged Access
***0x0F**	A32 – Extended Supervisory Block Transfer†
***0x0E**	A32 – Extended Supervisory Program Access
0x0D	A32 – Extended Supervisory Data Access
***0x0B**	A32 – Extended Nonprivileged Block Transfer†
***0x0A**	A32 – Extended Nonprivileged Program Access
0x09	A32 – Extended Nonprivileged Data Access
0x10 – 0x1f	User Defined
All Others	Reserved

* - The Model 4284 can generate this AM code as a VMEbus Master, but does not respond to it as a Slave.

† - The Model 4284 cannot master VMEbus block transfers.

4.7 Mastering the VMEbus (continued)

2) The 'C40 address is computed by dividing the 30 least significant bits of the VMEbus address by four and adding this to 0xB000 0000. This is necessary because the 'C40 addresses 32–bit words and the VMEbus addresses 8–bit bytes. The VMEbus "window" in the 'C40 memory map begins at 0xB000 0000.

3) Check that the address modifier codes (AM codes, in the 'C40 VMEbus Modifier Register) are being issued correctly as described in Table 4–9, on page 76 (see also Section 3.8.6). Verify which AM code(s) the slave device will respond to. If the VMEbus device being accessed does not respond because the wrong address modifier is used, the bus cycle will fail to complete (i. e., the slave device will not issue $\overline{\text{DTACK}}$).

4) Make sure the transfer select (TRSEL) bits in the 'C40 VMEbus Modifier Register are set correctly to match the data width of slave device. For example, if the slave is not capable of conducting a long–word cycle (D32), it will not properly complete the cycle and cause an error. Proper use of these bits is described in Section 3.8.2. The table from that section, which summarizes their settings, is reproduced below as Table 4–10.

For A32 address mode operation, the 2 most significant VMEbus address bits (without scaling by four) must be loaded into the Page Address Bits (D14 and D15) of the 'C40 VMEbus Modifier Register, as described in Section 3.8.7.

Table 4–10: Model 4284 – VMEbus Transfer Select (TRSEL) Bit Functions

TRSEL BITS 3 (D4)	2 (D3)	1 (D2)	0 (D1)	Description	VMEbus Strobes	VMEbus Addr	VMEbus Data	'C40 Data	Comments
x	x	0	0	32–bit Long Word	$\overline{\text{DS0}}$, $\overline{\text{DS1}}$	A01 = L $\overline{\text{LW}}$ = L	D0 – 31	D0 – 31	Single 32–bit Transfer Cycle
x	0	0	1	16–bit Word Double Cycle (Long Word Transfer as two 16–bit words)	$\overline{\text{DS0}}$, $\overline{\text{DS1}}$	A01 = L $\overline{\text{LW}}$ = H	D0 – 15	D0 – 15	First of Two 16–bit Cycles
					$\overline{\text{DS0}}$, $\overline{\text{DS1}}$	A01 = H $\overline{\text{LW}}$ = H	D0 – 15	D16 – 31	Second of Two 16–bit Cycles
0	1	0	1	16–bit Word Single Word Transfer	$\overline{\text{DS0}}$, $\overline{\text{DS1}}$	A01 = L $\overline{\text{LW}}$ = H	D0 – 15	D0 – 15	Low Order 16–bit Word Cycle
1	1	0	1	16–bit Word Single Word Transfer	$\overline{\text{DS0}}$, $\overline{\text{DS1}}$	A01 = H $\overline{\text{LW}}$ = H	D0 – 15	D16 – 31	High Order 16–bit Word Cycle
0	0	1	0	8–bit Byte Single Byte Transfer	$\overline{\text{DS0}}$	A01 = L $\overline{\text{LW}}$ = H	D0 – 7	D0 – 7	Lowest Order Byte Cycle
0	1	1	0	8–bit Byte Single Byte Transfer	$\overline{\text{DS1}}$	A01 = L $\overline{\text{LW}}$ = H	D8 – 15	D8 – 15	Next to Lowest Order Byte Cycle
1	0	1	0	8–bit Byte Single Byte Transfer	$\overline{\text{DS0}}$	A01 = H $\overline{\text{LW}}$ = H	D0 – 7	D16 – 23	Next to Highest Order Byte Cycle
1	1	1	0	8–bit Byte Single Byte Transfer	$\overline{\text{DS1}}$	A01 = H $\overline{\text{LW}}$ = H	D8 – 15	D24 – 31	Highest Order Byte Cycle
X	X	1	1	IACK Cycle	$\overline{\text{DS0}}$	A1 – A3	D0 – D7	D0 – D7	IACK Vector Read

LW = VMEbus line LWORD, x = don't care

4.7 Mastering the VMEbus (continued)

5) The Transfer Select bits (TRSEL0 through TRSEL3) must be set to determine the type of VMEbus transfer. These bits are also contained in the 'C40 VMEbus Modifier Register (bits D1 – D4, see Section 3.8.2).

6) The LOCK bit, also in the 'C40 VMEbus Modifier Register (D5), should be set for the desired mode of operation. See Section 3.8.3 for more discussion.

Once these facilities have been established, the 'C40 may perform read and write cycles to the VMEbus memory page. When a memory cycle is initiated by the 'C40, the Bus Master Interface on the Model 4284 requests bus ownership by asserting one of the Bus Request lines ($\overline{\text{BREQ0}}$ through $\overline{\text{BREQ3}}$). The bus request level is determined by the setting of jumper block JB8, as described in Section 2.6.1.

The Bus Arbiter acknowledges the request for bus service and, if it is available, grants the bus to the Model 4284 by sending the appropriate Bus Grant line true ($\overline{\text{BG0In}}$ through $\overline{\text{BG3In}}$). In response to this grant, the Model 4284 asserts the Bus Busy line ($\overline{\text{BBSY}}$) during the memory cycle.

Once granted access to the bus, the Model 4284 retains bus ownership until another master asserts a Bus Request. This scheme is called Release On Request. Alternatively, the 'C40 may lock out other access to the bus using the LOCK bit. In general, this bit should be kept low, especially during program development. After programs are successfully debugged, the LOCK bit can be used to boost transfer rates. This bit is also used to create indivisible bus cycles (e. g., Read–Modify–Write), and in DMA transactions over the VMEbus.

In troubleshooting bus requester operation, the following items should be checked:

1) Check that the Bus Grant jumpers are correct on the board, on the backplane, and on other boards. Each of the 4 bus request levels uses a daisy chain bus grant line originating from the slot 1 arbiter and passing through each board. The daisy chain input signal to each board is $\overline{\text{BG}n\text{In}}$ and the output signal is $\overline{\text{BG}n\text{Out}}$.

2) If the board does use bus request level *n*, and it is not requesting service, then it must pass the bus grant signal through from $\overline{\text{BG}n\text{In}}$ to $\overline{\text{BG}n\text{Out}}$. This is performed by circuitry on the board which is part of the bus requester function.

3) If the board **does not** use bus request level *n*, then the bus grant signal must be bypassed around the board at all times. This can be achieved using backplane jumpers or on–board jumpers. See Section 2.6.3 for more details.

   In order for any board to conduct a bus cycle it must receive a low (true) bus grant signal at its $\overline{\text{BG}n\text{In}}$ input pin.

4.7 Mastering the VMEbus (continued)

4) Make sure the correct VMEbus address is used. Remember, a VMEbus address is accessed by the 'C40 as follows:

$$'C40 Address = \frac{VMEbus\ Address}{4} + 0xB000\ 0000$$

If an A32 address is accessed, the 2 most significant bits must be set in the Page Address Bits (D14 and D15) of the 'C40 VMEbus Modifier Register, as described in Section 3.8.6.

4.8 Using the Model 4284's VSB Interface (Option 012 ONLY!)

4.8.1 The 4284 as a VSBus System Arbiter

The PLX VSB1400A/B chip set provides single–level arbitration only on the VSBus. If a bus request is received from one master while another is in control of the bus, the PLX chips hold the request and the assert $\overline{BUSY}$ until the conclusion of the transaction in progress.

4.8.2 Model 4284 VSB Block Transfers

Model 4284 allows for VSB block transfers, which provide for transfers greater than one data unit (byte, word, or longword) at a time. In block transfers, a single $\overline{DS}$ signal is asserted, and there is only a one address broadcast phase, followed by several data transfers. The master transfers data, while the slave increments an address counter for each transfer until all the data has been sent or received. VSB block transfers on the Model 4284 are enabled by bit D15 in the VSB Control Register ('C40 Global Address 0x9000 0001). Block transfers are enabled when that bit is set to the logic '1' state.

4.9 Mastering the MIX Bus

The 'C40 on the Model 4284 accesses MIX modules from its Local bus. Each module position on the MIX stack is assigned a 64 M–Longword region of 'C40 memory space. Module 0''s region extends from 0x2000 0000 to 0x23FF FFFF, Module 1's from 0x2400 0000 to 0x27FF FFFF, and the Module 2 region is from 0x2800 0000 to 0x2BFF FFFF. We refer to the region between 0x2000 0000 and 0x2BFF FFFF as the 'C40's MIX bus "window".

4.9 Mastering the MIX Bus (continued)

Mastering MIX transactions from the 4284 is simply a matter of reading from or writing to the proper area of the 'C40 MIX bus window. The 4284 also supports mastering of the MIX bus by processor modules in the MIX stack, called Upper MIX Bus Masters, or UMBMs. For details about how one of these modules can access others, refer to the MIX Module's Operating Manual. Note that unlike the Models 4200 and 4201 MIX baseboards, the 4284 baseboard does not add any offset to the UMBM's window on the other module positions. Therefore, the 4284's 'C40 MIX bus window is not involved in the UMBM's MIX address calculations.

4.10 Using the Model 4284's Memories

The Dual−Port DRAM in the Model 4284 is accessible by any VMEbus Master and by the 'C40 processor. This makes this memory area ideal for exchange of data and pro−gram information between processors in a multi−processor system. 'C40 programs in Common Object File Format (COFF) can be deposited here by the VME System Execu−tive, and can either be executed directly from the DRAM or moved by the 'C40 into its Global or Local SRAM prior to execution.

The 4284's DRAM is seen as a contiguous memory block of 1M, 2M or 4M longwords, depending upon the memory option purchased. The starting address of this memory block, from the 'C40's point of view, is 0x8000 0000, on the Global bus. From the point of view of a VMEbus Master, the DRAM begins the base address, set by jumper block SW2 (see Section 2.4.3 for details). Remember that while the 'C40 addresses the DRAM as longwords, VMEbus masters address all resources as bytes. As a result, each 'C40 address in DRAM is equivalent to four VMEbus addresses. For example, if a VMEbus master wrote a block of 256 bytes to addresses beginning at VME_base+0x0000 1000, the 'C40 would read that data as a block of 64 longwords beginning at address 0x8000 0400 in its own memory map.

The Local and Global SRAMs on the Model 4284 are directly accessible ONLY by the 'C40. The Local SRAM appears as a contiguous block of 256k or 512k longwords, depending upon the installed options, and begins at 'C40 address 0x4000 0000, on the Local bus. The Global SRAM also appears as a contiguous block of 256k or 512k long−words, again depending upon options. This memory block begins at 'C40 address 0xC000 0000, on the Global bus.

For applications involving real−time signal processing, it is important to know how quickly data can be moved from one memory area to another. Table 4−11 (on page 82) through Table 4−15 (on page 83), give the times needed to transfer a single longword from each resource available to the 'C40 processor on the Model 4284 to any another available resource, or to another region within that resources address range.

The data in Table 4−11, Table 4−12 and Table 4−13 represent typical performance for a Model 4284 with a 50 MHz 'C40, in block repeat transfer mode, 'C40 DMA transfer mode, and Turbo−MIX mode, respectively. Table 4−14 and Table 4−15 represent the typical performance of a Model 4284 equipped with a 40 MHz 'C40, in block repeat and 'C40 DMA modes (Turbo−MIX is not supported on 40 MHz 4284's).

4.10 Using the Model 4284's Memories (continued)

The 'C40 processor on the 4284 was the master device for all data tranactions. In block repeat mode, the data transfers are mastered by the 'C40's CPU, in a loop using the RPTB instruction. See Example 12–10 (Use of Block Repeat to Find a Maximum or a Minimum), in Section 12.2.5 (Repeat Modes) of the Texas Instruments **TMS320C4x User's Guide** for an example of the use of this instruction. Transfers performed in this manner are documented in Table 4–11 and Table 4–14, for 50 MHz and 40 MHz devices, respectively. In applications where more processing bandwidth is required, it may be advantageous to allow the 'C40's DMA controller to handle the data transfer tasks. In most cases, however, actual transfer times will be slower in 'C40 DMA mode than in block repeat mode. Section 12.6.3 (DMA Coprocessor) of the **TMS320C4x User's Guide** presents several DMA examples. Table 4–12 and Table 4–15 document the 4284's performance in 'C40 DMA transfer mode, for 50 MHz and 40 MHz processors, respectively.

To acquire the MIX bus transfer rate data presented in these tables, two different MIX co–processor modules were used, to illustrate the effect of MIX bus wait states. Pentek's Model 4247 'C30 processor card is an example of a device that does not impose wait states on MIX bus transactions. Tranactions with that board are documented in the rows and columns of the tables marked MIX (no wait). The Model 4257 'C40 co–processor, on the other hand, does impose wait states when it transacts with the MIX bus. Results of the tests performed with that board are listed in the rows and columns marked MIX (wait).

For all transfers listed in the tables involving the VMEbus, the data's actual source or destination was the 4284's Global DRAM. All DRAM (and, by extension, VME) transfer times include the DRAM refresh cycle of 175 nsec every 6.4 msec. Finally, all transfer times listed in the tables are the average time for one 32–bit longword, based on a transfer of 1000 longwords.

4.10 Using the Model 4284's Memories (continued)

Table 4–11: Model 4284 – Transfer Times for One Longword (Block Repeat Mode) – 50 MHz 'C40

From:↓ To: →	'C40 Int. RAM	Local SRAM	Global SRAM	Global DRAM*	MIX (no wait)	MIX (wait)	VMEbus*
'C40 Int. RAM	40 nsec	80 nsec	80 nsec	248 nsec	280 nsec	440 nsec	406 nsec
Local SRAM	40 nsec	160 nsec	80 nsec	248 nsec	360 nsec	520 nsec	406 nsec
Global SRAM	40 nsec	80 nsec	160 nsec	326 nsec	280 nsec	440 nsec	489 nsec
Global DRAM*	208 nsec	208 nsec	327 nsec	495 nsec	286 nsec	442 nsec	655 nsec
MIX (no wait)	280 nsec	360 nsec	280 nsec	286 nsec	560 nsec	720 nsec	406 nsec
MIX (wait)	440 nsec	520 nsec	440 nsec	442 nsec	720 nsec	880 nsec	444 nsec
VMEbus*	367 nsec	367 nsec	489 nsec	655 nsec	367 nsec	443 nsec	810 nsec

* – DRAM timing includes refresh time of 175 nsec every 6.4 msec – VMEbus transfers are all to or from DRAM
All timing results are given as average for 1 longword transfer based on 1000 longword transfers

VME Timing

DS↓ to DTACK↓	DTACK↓ to DS↑	DS↑ to DTACK↑
(assertion)	(response)	(de–assertion)
250 nsec	≅60 nsec	90 nsec (max)

Table 4–12: Model 4284 – Transfer Times for One Longword ('C40 DMA Mode) – 50 MHz 'C40

From:↓ To: →	'C40 Int. RAM	Local SRAM	Global SRAM	Global DRAM*	MIX (no wait)	MIX (wait)	VMEbus*
'C40 Int. RAM	80 nsec	80 nsec	80 nsec	250 nsec	280 nsec	440 nsec	407 nsec
Local SRAM	80 nsec	160 nsec	80 nsec	250 nsec	360 nsec	520 nsec	407 nsec
Global SRAM	80 nsec	80 nsec	160 nsec	328 nsec	280 nsec	440 nsec	490 nsec
Global DRAM*	250 nsec	250 nsec	328 nsec	496 nsec	287 nsec	443 nsec	657 nsec
MIX (no wait)	280 nsec	360 nsec	280 nsec	287 nsec	560 nsec	720 nsec	407 nsec
MIX (wait)	440 nsec	520 nsec	440 nsec	443 nsec	720 nsec	880 nsec	448 nsec
VMEbus*	407 nsec	407 nsec	490 nsec	656 nsec	407 nsec	448 nsec	811 nsec

* – DRAM timing includes refresh time of 175 nsec every 6.4 msec – VMEbus transfers are all to or from DRAM
All timing results are given as average for 1 longword transfer based on 1000 longword transfers

VME Timing

DS↓ to DTACK↓	DTACK↓ to DS↑	DS↑ to DTACK↑
(assertion)	(response)	(de–assertion)
250 nsec	≅60 nsec	90 nsec (max)

Table 4–13: Model 4284 – Transfer Times for One Longword (Turbo–MIX Mode) – 50 MHz 'C40

Transfer Type		Transfers between no–wait MIX Module and				
		'C40 Int. RAM	Global SRAM	Global DRAM*	MIX (wait)	VMEbus*
Block Repeat	**Write to MIX Module**	120 nsec	120 nsec	208 nsec	240 nsec	367 nsec
	Read from MIX Module	80 nsec	80 nsec	248 nsec	240 nsec	406 nsec
'C40 DMA	**Write to MIX Module**	120 nsec	120 nsec	250 nsec	240 nsec	407 nsec
	Read from MIX Module	120 nsec	120 nsec	250 nsec	240 nsec	407 nsec

* – DRAM timing includes refresh time of 175 nsec every 6.4 msec – VMEbus transfers are all to or from DRAM
All timing results are given as average for 1 longword transfer based on 1000 longword transfers

VME Timing

DS↓ to DTACK↓	DTACK↓ to DS↑	DS↑ to DTACK↑
(assertion)	(response)	(de–assertion)
250 nsec	≅60 nsec	90 nsec (max)

4.10 Using the Model 4284's Memories (continued)

Table 4–14: Model 4284 – Transfer Times for One Longword (Block Repeat Mode) – 40 MHz 'C40

From:↓ To: →	'C40 Int. RAM	Local SRAM	Global SRAM	Global DRAM*	MIX (no wait)	MIX (wait)	VMEbus*
'C40 Int. RAM	50 nsec	100 nsec	100 nsec	260 nsec	400 nsec	550 nsec	411 nsec
Local SRAM	50 nsec	200 nsec	100 nsec	260 nsec	500 nsec	650 nsec	411 nsec
Global SRAM	50 nsec	100 nsec	200 nsec	357 nsec	400 nsec	550 nsec	507 nsec
Global DRAM*	210 nsec	210 nsec	358 nsec	518 nsec	402 nsec	552 nsec	669 nsec
MIX (no wait)	300 nsec	500 nsec	300 nsec	310 nsec	800 nsec	950 nsec	411 nsec
MIX (wait)	500 nsec	650 nsec	450 nsec	495 nsec	950 nsec	1100 nsec	460 nsec
VMEbus*	386 nsec	368 nsec	507 nsec	669 nsec	406 nsec	552 nsec	819 nsec

* – DRAM timing includes refresh time of 175 nsec every 6.4 msec – VMEbus transfers are all to or from DRAM
All timing results are given as average for 1 longword transfer based on 1000 longword transfers

VME Timing		
DS↓ to DTACK↓	DTACK↓ to DS↑	DS↑ to DTACK↑
(assertion)	(response)	(de–assertion)
250 nsec	≅60 nsec	90 nsec (max)

Table 4–15: Model 4284 – Transfer Times for One Longword ('C40 DMA Mode) – 40 MHz 'C40

From:↓ To: →	'C40 Int. RAM	Local SRAM	Global SRAM	Global DRAM*	MIX (no wait)	MIX (wait)	VMEbus*
'C40 Int. RAM	100 nsec	100 nsec	100 nsec	262 nsec	400 nsec	550 nsec	412 nsec
Local SRAM	100 nsec	200 nsec	100 nsec	262 nsec	500 nsec	650 nsec	412 nsec
Global SRAM	100 nsec	100 nsec	200 nsec	359 nsec	400 nsec	550 nsec	509 nsec
Global DRAM*	262 nsec	262 nsec	359 nsec	520 nsec	404 nsec	554 nsec	670 nsec
MIX (no wait)	400 nsec	500 nsec	400 nsec	404 nsec	800 nsec	950 nsec	412 nsec
MIX (wait)	550 nsec	650 nsec	550 nsec	554 nsec	950 nsec	1100 nsec	554 nsec
VMEbus*	413 nsec	412 nsec	509 nsec	670 nsec	412 nsec	554 nsec	821 nsec

* – DRAM timing includes refresh time of 175 nsec every 6.4 msec – VMEbus transfers are all to or from DRAM
All timing results are given as average for 1 longword transfer based on 1000 longword transfers

VME Timing		
DS↓ to DTACK↓	DTACK↓ to DS↑	DS↑ to DTACK↑
(assertion)	(response)	(de–assertion)
250 nsec	≅60 nsec	90 nsec (max)

This page is intentionally blank

Appendix A: Programming Examples

A.1 Setting up the Model 4284

The following C program uses a Bit3 Model 603 PC/AT-VMEbus adaptor, in the Slot 1 System Controller position, as the VMEbus Master to control the initial setup of a Model 4284. This code also exercises the firmware Look and Set functions.

```
#include "io84.h"
#include "def84v.h"
#include "iocom.h"

extern unsigned int a16_addr;
extern int a24_addr;
extern int page;
extern int access_mode;
extern int processor;
extern int diag_mode;
extern int reset_flag;
extern unsigned long module;
extern int tout;

unsigned long look84_func, set84_func, ready;

void setup84 (void)
{
   int stat;
   page = 0;
   inportb (0x202);
   inportb (COMMAND_REG);
   if ((inportb(0x202) & 1)!=0)
   {
      printf ("\npower cable off\n");
      exit(0);
   }
   outportb (0x200,0x90);
   stat=inportb (0x202) & 0xc5;

   if (stat!=0)
      printf ("\nsetup84 status error\n");
   if ((stat & 0x80)!=0)
      printf ("\nparity error\n");
   if ((stat & 0x40)!=0)
      printf ("\nbus error\n");
   if ((stat & 0x1)!=0)
      printf ("\npower off error\n");
   if (stat==0)
      printf ("\nsetup84 ready\n");
```

A.1 Setting up the Model 4284 (continued)

```
   if (reset_flag)
      reset_vme ();
   outportb (COMMAND_REG,0x8);          /* use address page  register */
   outportb (ADDMOD_REG, A16P);
   outportb (ADDMOD_REG,A24);
   set_processor_page (processor);
   if (access_mode == 0)
      outportb (ADDMOD_REG,A16NP);
   if (access_mode == 1)
      outportb (ADDMOD_REG,A16P);
   if (access_mode == 2)
      outportb (ADDMOD_REG,A24);
}

void init84 (void)
{
   int image, i;                            /* Reset and Release C40a */
   if (reset_flag)
   {
      outportb(0x20d,A16);
      image = peek (SEG,CONTROL_REG + a16_addr) & (~RESET84_A);
      poke (SEG,CONTROL_REG + a16_addr,(RESET84_A | image));
      for (i=0;i<5000;i++);                                  /* Delay */
      poke (SEG,CONTROL_REG + a16_addr,((~RESET84_A) & image));
      for (i=0;i<5000;i++);
   }
   set_processor_page (0);
   outportb (0x20d,A24);
   tout = 1024;
   while (peek_long(RDY) != 1)                /* wait for c40 ready */
   {
      timer ();
      if ( tout == 0 )
         break;
      if (kbhit()!=0)
      {
         tout=0;
         break;
      }
   }
   if (tout != 0)
   {
      look84_func = peek_long (LOOK);
      set84_func = peek_long (SET);
      ready = peek_long (RDY);
      printf ("4284 look = %lx\n",look84_func);
      printf ("4284 set = %lx\n",set84_func);
      printf ("4284 rdy = %lx\n",ready);
      printf ("4284/C40 ready\n");
   }
```

A.1 Setting up the Model 4284 (continued)

```
   else
   {
      ready = peek_long (RDY);
      printf ("4284 rdy = %lx\n",ready);
      printf ("4284/C40 not here !!!!!!!!!!\n");
      getch ();
   }
}

run84 (entry_point)
unsigned long entry_point;
{
   unsigned hi,low;
   unsigned long command;

   outportb (0x20d,A24);
   poke_long (CMD,entry_point);
   poke_long (RDY,0x0);                          /* Zero Ready Flag */
   outportb (0x20d,A16);                    /* Short Address AM Code */
   poke (SEG,CONTROL_REG + a16_addr,INT84_A);     /* Interrupt C40a */
   outportb (0x20d,A24);                     /* Standard Addressing */
}

void set84 (unsigned long addr,unsigned long data)
{
   outportb (0x20d,A24);
   poke_long (CMD,set84_func);
   poke_long (ARG2,data);
   poke_long (ARG1,addr);
   poke_long (RDY,0);                            /* Zero Ready Flag */
   outportb (0x20d,A16);                    /* Short Address AM Code */
   poke (SEG,CONTROL_REG + a16_addr,INT84_A);     /* Interrupt C40a */
   outportb (0x20d,A24);                     /* Standard Addressing */
   tout=8196;

   while (peek_long(RDY) != 1)
   {
      if (intchk() == 0)
         break;
      if (kbhit()!=0)
         break;
   }
}
```

A.1 Setting up the Model 4284 (continued)

```
unsigned long look84 (unsigned long addr)
{
   unsigned long data;
   outportb (0x20d,A24);
   poke_long (CMD,look84_func);
   poke_long (ARG1,addr);
   poke_long (RDY,0);                                /* Zero Ready Flag */
   outportb (0x20d,A16);                          /* Short Address AM Code */
   poke (SEG,CONTROL_REG + a16_addr,INT84_A);        /* Interrupt C40a */
   outportb (0x20d,A24);                           /* Standard Addressing */
   tout=8196;

   while (peek_long(RDY) != 1)
   {
      if (intchk() == 0)
         break;
      if (kbhit()!=0)
         break;
   }
   data = peek_long (ARG3);
   return (data);
}

void set_page84 (int proc)
{
   outportb (0x20d,A24);
   outportb (0x20a,0x0 | (a24_addr<<4));              /* Set to proc A */
}

void send84 (int comm_port,unsigned long data)
{
   static int counter = 0;
   int comm_offset;

   /* Sends to CommPort */
   tout=1024;
   comm_offset = comm_port * 0x10;
   while ((look84(0x100040 + comm_offset) & 0x1e0L) == 0x1e0L )
   {
      timer ();
      if (tout == 0 )
      {
         printf ("\nTimeout CommPort\n");
         break;
      }
```

A.1 Setting up the Model 4284 (continued)

```
      if (kbhit()!=0)
      {
         tout=0;
         break;
      }
   }
   set84 (0x100042 + comm_offset,data);

   printf ("%08lx ",data);
   ++counter;
   if (counter == 8)
   {
      counter = 0;
      printf ("\n");
   }
}

/***************************** io84.h ******************************/
void init84 (void);
void set84 (unsigned long addr,unsigned long data);
unsigned long look84 (unsigned long addr);
unsigned long get_offset (int processor, int card_type);

/***************************** def84v.h ****************************/
#define NORMAL     1

/* Constants */
#define BLANK        32
#define MAXPOS        1                  /* Number of items in MENU -1 */
#define BLOCK       223
#define TOP           4
#define LEFTCOLUMN   25
#define WIDTH        30
#define HEIGHT       15
#define BORDERCOLOR  14
#define TEXTCOLOR    11
#define PASSCOLOR    10
#define ERRORCOLOR   12
#define UP         0x48
#define DOWN       0x50
#define LEFT       0x4b
#define RIGHT      0x4d
#define SEG      0xd000              /* segment correponding to VMEbus */
#define LOOP      65536
```

A.1 Setting up the Model 4284 (continued)

```
/* C40 Handshake LOOK / SET Pointers */
#define CMD       0x800                    /* Command - Jump Address */
#define ARG1      0x804                   /* Argument #1 L/S Address */
#define ARG2      0x808                      /* Argument #2 Set Data */
#define ARG3      0x80c                    /* Argument #3 Look Data */
#define RDY       0x810                       /* C40 Rdy Indication */
#define LOOK      0x814                       /* Fetch Data Address */
#define SET       0x818                         /* Set Data Address */
#define BLKMOVE   0x81c                       /* Block Move Address */
#define REVISION  0x83c                       /* Boot Code revision */
#define FBOOT     0x844                          /* C40 Boot Loader */
#define FLOOK     0x848                     /* Look at Flash Memory */
#define FSET      0x84c                          /* Set Flash Memory */
#define FERASE    0x850                   /* Erase Flash Memory Chip */
#define SERASE    0x854                 /* Erase Flash Memory Sector */
#define FLDB      0x858                           /* Flash Load Byte */
#define FUNLDB    0x85c                        /* Flash Unload Byte */
#define FLDW      0x860                         /* Flash Load Word */
#define FUNLDW    0x864                        /* Flash Unload Word */

#define COMMAND_REG  0x208                        /* VME command reg */
#define ADDPAGE_REG  0x20A                       /* Address page reg */
#define ADDMOD_REG   0x20D                  /* Address modifier reg */
#define A16NP        0x29                   /* A16 address modifier */
#define A16P         0x2d                   /* A16 address modifier */
#define A16          0x2d                   /* A16 address modifier */
#define A24          0x3d                   /* A24 address modifier */
#define A16_Base     0x0                 /* Control Reg Base Address */
#define A24_Base     0x0                  /* Global Mem Base Address */
#define VME_IACK_REG 0                          /* VME I/O Register */
#define CONTROL_REG  4                    /* Reset control Register */

/* I/O Register bit definitions */
#define RESET84_A    1
#define RELEASE      0
#define INT84_A      2

#define ESC          27
#define SP           ' '
#define BS           '\b'
#define CR           '\r'
#define EQUAL        '='
#define PLUS         '+'
#define MINUS        '-'

void set_processor_page (unsigned int proc);
void bist84 ();
```

A.1 Setting up the Model 4284 (continued)

```
/***************************** iocom.h ******************************/
#define SEG 0xd000                  /* segment correponding to VMEbus */
void set_processor_page (unsigned int proc);
void set_page84 (int proc);
void set_page70 (int proc);
void set_page80 (int proc);
void poke_long (unsigned long addr,unsigned long val);
unsigned long peek_long (unsigned long addr);
void setbit3 (unsigned long addr,unsigned long data);
unsigned long lookbit3 (unsigned long addr);
void timer (void);
int intchk (void);
void send (int comm_port,unsigned long data);
```

A.2 Issuing and Handling VMEbus Interrupts

The C program segments below are used in Pentek's testing of the Model 4284 to verify proper operation of the Model 4284 as a VMEbus interruptor and Interrupt Handler. Another function included here tests the 4284's response to the VMEbus Timeout Interrupt. Again, the interface to the VMEbus is the Bit3 Model 403 PC/AT to VME Adaptor.

```
test_vme_iack84 (int irq_level)
{
   int i, p, psave, stat;
   int istat, stat_mask, err_flag=0;
   unsigned long lstat;

   system ("cls");

   printf ("\n4284 VME IRQ test\n" );
   outportb (0x208,(irq_level | 0x8));              /* Set IRQ level */
   poke (SEG,CONTROL_REG + a16_addr,1);  /* Clr interrupt thru C40 reset */
   poke (SEG,CONTROL_REG + a16_addr,0);
   for (i=0; i<100; i++);                                  /* Delay */
   set84 (0x100020, 0x306);          /* LED and timer interrupts off */
   for (i=0; i<100; i++);                                  /* Delay */
   printf ("\n\nTesting IRQ Interrupter - %x\n",irq_level);
   outportb (0x208,(irq_level | 0x8));              /* Set IRQ level */
   outportb (0x20d,A16);                    /* Short Address AM Code */
   set84 (0x10000002, (unsigned long)(irq_level << 8));
   lstat = look84 (0x10000002);                 /* Read C40 irq level */
   if ((lstat & 0x700L) != (unsigned long)(irq_level << 8))
   {
      printf ("\nIRQ level not set properly- %lx\n",lstat);
      ++err_flag;
   }
```

A.2 Issuing and Handling VMEbus Interrupts (continued)

```
for(i=0; i<0x100; i++)
{
   outportb (0x20d,A16);                    /* Short Address AM Code */
   poke (SEG,VME_IACK_REG + a16_addr,i);
                                    /* Set IACK vector and Clr int */
   stat = peek (SEG,VME_IACK_REG + a16_addr);  /* Read IACK vector */
   if ((stat & 0xff) != i)
   {
      printf ("\nReadback vector register: is %0x should be %0x\n",stat,i);
      ++err_flag;
   }
   lstat = look84 (0x90000000);                  /* C40 status read */
   if ((lstat & 0x80) != 0x0)
   {
      printf ("\nC40 interrupt to VME did not clear !\n");
      ++err_flag;
   }
   set84(0x90000000,0x06);                          /* IACK cycle */
   set84(0x10000004,0x00);                         /* C40 VME INT */
   lstat = look84(0x90000000);                  /* C40 status read */
   if ((lstat & 0x80) != 0x80)
   {
      printf ("\nC40 interrupt to VME not set !\n");
      ++err_flag;
   }
   stat=inportb (0x20e);                       /* Do VME IACK cycle */
   if ((stat & 0xff) != i)
   {
      set84(0x100030, 0x306);                       /* Trigger out */
      set84(0x100030, 0x302);                       /* Trigger out */
      printf("\nBad vector address is %0x should be %0x\n",stat,i);
      stat = peek (SEG,VME_IACK_REG + a16_addr);‘ /* Read IACK vector */
      if ((stat & 0xff) != i)
      printf("\nReadback vector register: is %0x should be %0x\n",stat,i);
      ++err_flag;
      if (err_flag > 5)
      break;
   }
   else
      printf ("  Read vector address > %0x\r",stat);
}
if (err_flag != 0)
{
   printf ("\nErrors detected during testing - %d\n",err_flag);
   getch ();
}
```

A.2 Issuing and Handling VMEbus Interrupts (continued)

```
   printf ("\n\nTesting IRQ Handler - %x\n",irq_level);
   err_flag = 0;
   outportb (0x20d,A16);                    /* Short Address AM Code */
   set84 (0x10000002, (unsigned long)(irq_level << 8));
   lstat = look84(0x10000002);                 /* Read C40 irq level */
   if ((lstat & 0x700L) != (unsigned long)(irq_level << 8))
      printf ("\nIRQ level not set properly- %lx\n",lstat);
   for(i=0; i<0x100; i++)
   {
      outportb (0x20d,A16);                 /* Short Address AM Code */
      poke(SEG,VME_IACK_REG + a16_addr,i);  /* Set IACK vector and Clr int */
      stat = peek (SEG,VME_IACK_REG + a16_addr);  /* Read IACK vector */
      if ((stat & 0xff) != i)
      printf ("\nReadback vector register: is %0x should be %0x\n",stat,i);
      lstat = look84 (0x90000000);               /* C40 status read */
      if ((lstat & 0x80) != 0x0)
         printf ("\nC40 interrupt to VME did not clear !\n");

      set84 (0x90000000,0x06);                      /* IACK cycle */
      set84 (0x10000004,0x00);                     /* C40 VME INT */
      lstat = look84 (0x90000000);               /* C40 status read */
      if ((lstat & 0x80) != 0x80)
         printf ("\nC40 interrupt to VME not set !\n");

      /* VME IACK cycle */
      lstat=look84 (0xa0000000 + irq_level);
      if ((irq_level & 1) == 1)        /* Shift upper to lower if odd */
         lstat = (lstat >> 16);
      if ((lstat & 0xff) != i)
      {
         set84 (0x100030, 0x306);                   /* Trigger out */
         set84 (0x100030, 0x302);                   /* Trigger out */
         printf ("\nBad vector address is %0lx should be %0x\n",lstat,i);
         ++err_flag;
         if (err_flag > 5)
            break;
      }

      else
         printf ("  Read vector address > %0x\r",stat);
   }

if (err_flag != 0)
   {
      printf ("\nErrors detected during testing - %d\n",err_flag);
      getch ();
   }
```

A.2 Issuing and Handling VMEbus Interrupts (continued)

```
   poke (SEG,CONTROL_REG + a16_addr,1);                    /* C40 reset*/
   poke (SEG,CONTROL_REG + a16_addr,0);
}

test_timeout84 ()
{
   int r0, r1, count=100, i, err_flag=0;
   long lstat;
   system ("cls");
   printf ("\n4284 Bus timeout test");
   set84 (0x90000000,0x2d00);                          /* VME A16 mode */
   set84 (0x10000002,0x0);         /* Reset timeout int status if set */
   lstat = look84 (0x10000002);                     /* C40 status read */
   if ((lstat & 0x80) != 0x0)
   {
      printf ("\nBus timeout status not cleared !\n");
      ++err_flag;
   }

   while (count)
   {
      lstat = look84 (0xb0000100);           /* Read non-existent I/O */
      for (i=0; i<10; i++);             /* Delay for timeout to occur */
      lstat = look84 (0x10000002);                  /* C40 status read */

      if ((lstat & 0x80) != 0x80)
      {
         printf ("\nBus timeout status did not set !\n");
         ++err_flag;
      }
      lstat = look84 (0x10000002);                  /* C40 status read */

      if ((lstat & 0x80) != 0x80)
      {
         printf ("\nBus timeout cleared when reading status!\n");
         ++err_flag;
      }
      set84 (0x10000002,0x0);             /* Reset timeout int status */
      lstat = look84 (0x10000002);                  /* C40 status read */
      if ((lstat & 0x80) != 0x0)
      {
         printf ("\nBus timeout status will not clear!\n");
         ++err_flag;
      }
      count--;
   }
```

A.2 Issuing and Handling VMEbus Interrupts (continued)

```
set84 (0x90000000,0x3d00);                          /* VME A24 mode */
printf ("\nTest complete !");
if (err_flag != 0)
{
   printf ("\nErrors detected during testing - %d\n",err_flag);
   getch ();
}
```

A.3 Using the VMEbus Master Interface and Comm Ports

The following C program excerpts come from a program written to control a Pentek VMEbus A/D and D/A Converter board using a Model 4284. The VMEboard-specific code has been removed. These code fragments illustrate the use of both the VMEbus master interface and the comm port interfaces of the Model 4284. The Model 4265 A/D converter is the device that the 4284 communicates with over the Comm Port.

```
typedef volatile unsigned int vuint;

/* VMEbus address modifiers */
#define A16    0x2D
#define A24    0x3D
#define A32    0x0D

/* 4265 VME base address and 4284 VME access constants  */
#define A24BASE      0x800000/4                  /* 4265 base address */
#define A24ACCESS    0xB0000000       /*4284 VME Master Access region */
#define VME_AM_CODE  0x90000000         /* 4284 VME Modifier Register */

/* define functions and pointers specific to your VMEbus board here */

/* functions and pointers for the 4284 */
void delay (vuint cnt);
void trigger (void);
void led_on (void);
void led_off (void);
void toggle_led (void);
vuint in_comm_port (vuint port_no, uint *data);
vuint out_comm_port (vuint port_no, uint data);

vuint *datain, *dataout, *out_in, *cntrl, *channel;
vuint *vme_reg, *timer1_address, *timer0_address;
```

A.3 Using the VMEbus Master Interface and Comm Ports (continued)

```
main()
{
   void init_ptrs (void);
   void init_84 (void);
   vuint i, o, port, control;

   /* Initialize 4284 */

   init_ptrs ();
   init_84 ();

/* call initialization functions specific to your VMEbus board here */
}

void init_ptrs(void)             /* Function to create 4284 pointers */
{
   /* 4284 pointers */
   datain  = (vuint *)0x80000300;
   dataout = (vuint *)0x80000301;
   out_in  = (vuint *)0x80000302;
   cntrl   = (vuint *)0x80000303;
   channel = (vuint *)0x80000304;

   /* Buffers in Global DRAM for storing comm port data */
   /* one is used for input data to the 'C40 */
   /* while the other is used for output data */

   OutBufA = (vuint *)0x80001000u;
   InBufA  = (vuint *)0x80002000u;

   OutBufB = (vuint *)0x80003000u;
   InBufB  = (vuint *)0x80004000u;

   OutBufC = (vuint *)0x80005000u;
   InBufC  = (vuint *)0x80006000u;

   OutBufD = (vuint *)0x80007000u;
   InBufD  = (vuint *)0x80008000u;

   timer0_address = (vuint *)0x100020u;
   timer1_address = (vuint *)0x100030u;
   vme_reg        = (vuint *)0x90000000u;
}
```

A.3 Using the VMEbus Master Interface and Comm Ports (continued)

```
void init_84(void)  /* Function to initialize 4284 pointers and buffers */
{
   uint i, j;
   *datain  = 0;
   *dataout = 0;
   *out_in = 0x7055u;    /* the control for in and out c40 comm ports */
                        /* 0x70 = input control, 0x55 = output control */
   *cntrl = 0xffff0000u;      /* dual port DRAM - user control - flag */
   *vme_reg  = A24 << 8;

   /* set up ramp test function in output buffers */
   for(i = 0; i < 0x2000; i = i + 4)
   {
      for (j = 0; j < 4; j++)
      {
         *(OutBufA + (i+j)) = (i << 18) | (j << 8);
         *(OutBufB + (i+j)) = (i << 18) | (j << 8);
         *(OutBufC + (i+j)) = (i << 18) | (j << 8);
         *(OutBufD + (i+j)) = (i << 18) | (j << 8);
      }
   }
}

void led_on(void)   /* Function to turn on Model 4284 front panel LED */
{
   *vme_reg = (*vme_reg) | 0x40;
}

void led_off(void)   /* Function to turn off Model 4284 front panel LED */
{
   *vme_reg = (*vme_reg) & (~0x40);
}

void toggle_led(void) /* Function to toggle (on/off) Model 4284 front panel LED */
{
   *vme_reg = (*vme_reg) ^ 0x40;
}

vuint in_comm_port(vuint port_no, vuint *data)
                                        /* C40 input comm port function */
{
   vuint *comm_port_address, i;
   comm_port_address =(vuint *)((port_no * 16) + 0x100040u);
```

A.3 Using the VMEbus Master Interface and Comm Ports (continued)

```
   if (((*comm_port_address) & 0x1e00u) != 0)
   {
      *data = *(comm_port_address + 1);
      if (((*data) & 0xff) != 0x60)
      {
         trigger ();
      }
      else
      {
         trigger ();
      }
      return (1);
   }
   else return (0);
}

vuint out_comm_port(vuint port_no, vuint data)
                                        /* C40 output comm port function */
{
   vuint *comm_port_address;
   comm_port_address =(vuint *)((port_no * 16) + 0x100040u);
   if (((*comm_port_address) & 0x1e0u) != 0x1e0u)
   {
      *(comm_port_address + 2) = (data & 0xffff0f00u) | (*out_in);
      led_on ();
      trigger ();
      return (1);
   }
   else
   {
      led_off ();
      return (0);
   }
}

void delay (vuint cnt)
{
   int i;
   for (i=0; i < cnt; i++);
}

void trigger (void)
      /* create trigger pulse with timer1 for software/hardware debug */
{
      *timer1_address = 0x302u;
      *timer1_address = 0x306u;
      *timer1_address = 0x302u;
}
```

A.4 Working with MIX Modules and MIX Interrupts

The following C program uses the 4284 to control the operation of a Model 4243 dual channel A/D-D/A Converter MIX module. A switch at the top of the program allows the data interface between the two devices to occur via either polling or interrupt.

```
#define POLLING    0       /* Set these variables to opposite states */
#define INTERRUPTS 1
#define MOD0    0x20000000        /* Address to access MIX Position 0 */
#define MOD1    0x24000000        /* Address to access MIX Position 1 */
#define MOD2    0x28000000        /* Address to access MIX Position 2 */
#define MOD     MOD0
unsigned int *StatusControl, *ExpBusControlReg;
unsigned int *CtcMode, *Ch0, *Ch1, *Ch2;
unsigned int sig[2048],*sigP,i,temp, *DataBuffer;
unsigned int *int01_vector;
unsigned int *int02_vector;

#if INTERRUPTS
   void c_int02 ()
   {
      unsigned int temp;
      unsigned int i;
      sigP = sig;                          /* Read Data from Input FIFO */
      for (i=0;i<1024;i++)
         *sigP++=*DataBuffer;
      sigP = sig;                          /* Write Data to Output FIFO */
      for (i=0;i<1024;i++)
         *DataBuffer=*sigP++;
      temp = *StatusControl;  /* Read Status Register to clear Interrupt */
      asm("   iack @0");              /* Clear Any Pending Interrupts */
   }
#endif

main ()

   *(unsigned long *) 0x100020 = 0;                /* Turn Off Timer */

   #if INTERRUPTS
      /* This example maps MIXINT0 to IIOF3 */
      *(unsigned long *)0x10000000 = 0;     /* Clear Mix Bus Ctrl Reg */
      /* Map MXINT0 to IIOF3  */
      *(unsigned long *)0x10000003 = *(unsigned long *)0x10000003 | 0x08000000;
      /* Store Interrupt handler for IIOF3 */
      int02_vector = (unsigned int *)c_int02;
      asm("   ldep ivtp,ar0");
      asm("   ldi @_int02_vector,r0");
      asm("   sti r0,*+ar0(6)");
      asm("   iack @0");              /* Clear Any Pending Interrupts */
   #endif
```

A.4 Working with MIX Modules and MIX Interrupts (continued)

```
   /* Initialize Pointers to 4243 registers */
   StatusControl = (unsigned int *) (MOD | 0x0);
   CtcMode    = (unsigned int *) (MOD | 0x80);
   Ch0        = (unsigned int *) (MOD | 0x83);
   Ch1        = (unsigned int *) (MOD | 0x82);
   Ch2        = (unsigned int *) (MOD | 0x81);
   DataBuffer = (unsigned int *) (MOD | 0x01);

   i = *StatusControl;                    /* Clear Pending Interrupts */
   asm("   ldi 09000h,iif");                     /* Enable interrupts */
   asm("   or 2000h,st");                     /* Initialize the 4243 */
   *StatusControl=0x0u;                              /* Reset FIFO's */
   *CtcMode=0x34000000u;                                 /* Ch0 mode */
   *Ch0=0x64000000u;                  /* Input Sample rate divisor LSB */
   *Ch0=0x00000000u;                  /* Input Sample rate divisor MSB */
   *CtcMode=0x74000000u;                                 /* Ch1 mode */
   *Ch1=0x64000000u;                 /* Output Sample rate divisor LSB */
   *Ch1=0x00000000u;                 /* Output Sample rate divisor MSB */
   *CtcMode=0x96000000u;                                 /* Ch2 mode */
   *Ch2=0x06000000u;                    /* Filter Cutoff Freq = 20 kHz */

   #if INTERRUPTS
      *StatusControl=0x93000000u;              /* Release and acquire in
          single-channel mode, and enable Input FIFO half full interrupt */
   #else
      *StatusControl=0x83000000u;
                        /* Release and acquire in single-channel mode */
   #endif

   while(1)
   {
      ;
      #if POLLING
         while ((*StatusControl & 0x00020000u) != 0);
                                     /* wait for input FIFO half-full */
         for (i=0;i<1024;i++)
            sig[i]=*DataBuffer;          /* Read Data from Input FIFO */
         while ((*StatusControl & 0x00100000u) == 0);
                               /* wait for output FIFO not half-full */
         for (i=0;i<1024;i++)
            *DataBuffer=sig[i];          /* Write Data to Output FIFO */
      #endif
   }
}
```

A.5 Using Turbo-Mix

The following C program fragment demonstrates how one would set up a 50-MHz Model 4284 Baseboard to conduct Turbo-MIX transactions with a MIX Module (Turbo-MIX mode is not supported on 4284's with 40 MHz processors.) The Turbo-MIX bit is set in the 'C40's MIX Bus Control Register, and a word is written to the Local Memory Interface Control Register (internal to the 'C40) that determines the rate at which MIX accesses can occur.

It is important to be aware that for Turbo-MIX mode to be enabled, the program code **MUST** reside in Global Memory or in Internal RAM. This is because the MIX bus tranceivers are always enabled in this mode, making the 'C40 unable to access its Local SRAM, EPROM, or the registers at addresses 0x1000 0000 - 0x1000 0004.

```
unsigned int c40_reg_save
c40_control = (unsigned int *) 0x100000u;
c40_reg     = (unsigned int *) 0x10000000u;

/* NOTE: When Turbo-MIX mode is set, program code MUST reside in Global
         SRAM or 'C40 Internal RAM.  In this mode, the 'C40 is unable
         to access EPROM, Local SRAM, and registers from 0x1000 0000 -
         0x1000 0004 */

/* Set Turbo-MIX bit in 'C40 MIX Bus Control Reg */
c40_reg_save = *(c40_reg)
*(c40_reg) = *(c40_reg) | 0x4000;

/*Set MIX access (internal RDY) time - choose from list below */

/* *(c40_control+4) = 0x3def4030u;                    50 nsec access */
   *(c40_control+4) = 0x3def4130u;       /* 100 nsec access - typical */
/* *(c40_control+4) = 0x3def4230u;                   150 nsec access */
/* *(c40_control+4) = 0x3def4330u;                   200 nsec access */
/* *(c40_control+4) = 0x3def4430u;                   250 nsec access */
/* *(c40_control+4) = 0x3def4530u;                   300 nsec access */
/* *(c40_control+4) = 0x3def4630u;                   350 nsec access */
/* *(c40_control+4) = 0x3def4730u;                   400 nsec access */

/********************** put MIX access code here ********************/

*(c40_control+4) = 0x3def0000f;
                             /* Enable External RDY - disable internal */
*(c40_reg) = c40_reg_save;                    /* Clear Turbo-MIX bit */
```

A.6 Linker Command File for COFF to Intel HEX File Conversion

The Linker Command file presented below, **boot84.cmd**, can be used to convert the COFF file **led84.out** into an Intel HEX file called **demo84.hex**. To accomplish this, the Linker Command file's name is passed into the **hex30** conversion utility (supplied with the Texas Instruments Optimizing 'C' Compiler), with the following syntax:

```
hex30 boot84.cmd<enter>
```

```
/************* FILE - boot84.cmd **************/
/**********************************************/
/*        Boot build for C40 EEPROM           */
/**********************************************/
led84.out               /* Input file         */
-o demo84.hex           /* HEX output file    */
-i                      /* intel format       */
-memwidth 8             /* 8-bit memory       */
-boot                   /* Convert all .sect  */
-bootorg     0h         /* Boot source addr   */
-cg   3D9EC000h         /* Global mem conf.   */
-cl   3DEF4110h         /* Local mem conf.    */
-e    c0000c53h         /* PC after load addr */
-iack 4003fc00h         /* IACK dummy read    */
-ivtp 4003fc00h         /* IV pointer         */
-tvtp 4003fc00h         /* TV pointer         */
/**********************************************/
```

For further information about the use of this linker command file, see Section 4.3.1 of this manual.

A.7 Hex File Loader and Starter

```
/*********************************************************************/
/*           4284 Standalone HEX loader and Program Starter          */
/*********************************************************************/
/* General:                                                          */
/*   This program loads and executes a TMS320C40 Hex output file on  */
/*   a Pentek Model 4284 DSP Board.  It also has the capability of   */
/*   programming a program in flash memory if this option is present */
/*                                                                   */
/*  Usage:     loadhx84 <-options> filename                          */
/*  Options:      -l       load only (don't run)                     */
/*                -r       Run Program in Flash Memory               */
/*                -e       Erase Flash Memory                        */
/*                -f       Load Flash Memory                         */
/*                -s       Sector in Flash Memory (0-7)              */
/*                -a       A32 Base Address                          */
/*                -b       A16 Base Address                          */
/*                -o       Offset in memory to load image from start */
/*                         of global memory                          */
/*********************************************************************/
```

A.7 Hex File Loader and Starter (continued)

```
#include <ctype.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "def84.h"

/*-----------------------------------------------------------------------*/
/* LOADER DEFINITIONS                                                    */
/*-----------------------------------------------------------------------*/
long load_hex_file();
void boot_global();
void boot_flash();
void load_flash_memory();
void boot_processor();
void erase_sector();
void boot_override();
extern unsigned int peek_word();
extern void poke_word();
extern unsigned long peek_long();
extern void poke_long();

/************************************************************************/
/*                                                                      */
/* MAIN()                                                               */
/*                                                                      */
/************************************************************************/
void main (argc, argv)
int argc;
char *argv[];
{
      int           i;
      int           files=0;
      long          word_count;
      char          file_name[80];
      unsigned int  processor = 0;          /* Default To Processor 0 */
      unsigned long offset = 0x4008;
                       /* Default Offset from start of Global Memory  */
   unsigned char run_only = 0;                    /* Run program only */
   unsigned char load_only = 0;                  /* Load program only */
   unsigned char load_flash = 0;                 /* Load Flash Memory */
   unsigned char erase_flash = 0;               /* Erase Flash Memory */
   unsigned int  a16_base = 0;            /* Default A16 Base Address */
   unsigned long dpr_base = 0;                /* Default Base Address */
   unsigned int  sector = 0;          /* Sector In Flash Memory (0-7) */
```

A.7 Hex File Loader and Starter (continued)

```
   /*-------------------------------------------------------------------*/
   /*                PROCESS COMMAND LINE ARGUMENTS                     */
   /*-------------------------------------------------------------------*/
   for (i = 1; i < argc; ++i)
{
   char *argp = argv[i];
   if (*argp == '-')
   {
      while (*++argp) switch(*argp)
      {
         case 'L':
         case 'l': load_only = 1;
            printf ("Program Load Only\n");
            break;
         case 'R':
         case 'r': run_only = 1;
            printf ("Run Program From Flash\n");
            break;
         case 'E':
         case 'e': erase_flash = 1;
            printf ("Erase Flash Contents\n");
            break;
         case 'P':
         case 'p': sscanf(++argp,"%lx",&processor);
            if (processor > 0x3L )
            {
               printf ("Invalid processor selected!");
               printf ("   %d\n",processor);
               exit (0);
            }
            printf ("processor = %d\n", processor);
            break;
         case 'A':
         case 'a': sscanf(++argp,"%x",&dpr_base);
            if (dpr_base > 0xffff)
            {
               printf ("Invalid dpr address selected!");
               printf ("  %lx\n",((long)(dpr_base) << 16));
               exit (0);
            }
            printf ("dpr Base = %lx\n",((long)(dpr_base) << 16));
            break;
```

A.7 Hex File Loader and Starter (continued)

```
            case 'B':
            case 'b': sscanf(++argp,"%x",&a16_base);
               if (a16_base > 0xffff)
               {
                  printf ("\nInvalid a16 address selected!");
                  printf ("   %x\n",a16_base);
                  exit (0);
               }
               printf ("a16 Base = %x\n",a16_base);
               break;
            case 'O':
            case 'o': sscanf(++argp,"%x", &offset);
               printf("Offset = %x\n", offset);
               break;
            case 'F':
            case 'f': load_flash = 1;
               printf ("Load Program into Flash Memory\n");
               break;
            case 'S':
            case 's': sscanf(++argp,"%x", &sector);
               if (sector > 0x7)
               {
                  printf ("\nInvalid sector selected!");
                  printf ("   %x\n",sector);
                  exit(0);
               }
               printf ("Sector = %x\n", sector);
               break;
               default : continue;
            }
         }
         else
         {
            strcpy (file_name, argp);
            files = 1;
            break;
         }
      }

      setup(a16_base, dpr_base);                                    /* Setup
      boot_override(a16_base);   /* Override any code stored in Boot ROM */
      if (peek_long(REVISION) < 0x10e)
                     /* Check to see if the revision of firmware supports
                                    loading hex files and flash memory */
      {
         printf ("EEPROM revision does not support this feature\n");
         exit (0);
      }
```

A.7 Hex File Loader and Starter (continued)

```
   if (!run_only && !erase_flash)  /* Load program into Global Memory */
   {
      if (files == 0)
      {
         printf ("\nEnter HEX file name -> ");
         scanf ("%s", file_name);
      }
      strcat (file_name,".hex");
      word_count = load_hex_file (file_name, offset);

   /* If program is being executed from global memory, execute it */
   /* If program is being loaded into flash, load flash */
   /* If booting from flash memory, reset processor */
      if (!load_only && !load_flash)
         boot_global (a16_base, offset);
      else if (load_flash)
      {
         /* Erase Sector in flash memory to store Program */
         erase_sector (a16_base, sector);
         /* Load Flash Memory */
         load_flash_memory (a16_base, sector, offset, word_count);
         /* If booting from flash memory, Reset Processor */
         if (!load_only)
               boot_flash (a16_base, sector);
      }
   }
   else if (erase_flash)                        /* Erase Flash Memory */
      erase_sector (a16_base, sector);
   else                                  /* Run Program in Flash Memory */
      boot_flash (a16_base, sector);
   printf ("Boot Complete\n");
}

long load_hex_file (file_name, offset)
char *file_name;
unsigned long offset;
{
   FILE   *fp;
   int done=0;
   unsigned long i;
   unsigned char ch;
   char string[80];
   unsigned long data;
   char dl, dh;
   unsigned long addr;
   unsigned int length;
   long word_count = 0;
```

A.7 Hex File Loader and Starter (continued)

```
if (( fp = fopen(file_name, "rb" )) == NULL )
{
   printf ( "Could not open file - %s ! ",file_name );
   exit (0);
}
printf ("\nLoading %s to %lx \n",file_name,offset);
while (done == 0)
{
   /* See if this is a hex file */
   if (feof(fp))
   {
      printf ("\nEOF1 was detected !");
      exit (0);
   }
   ch = fgetc(fp);
   if (ch != ':')
   {
      printf ("\nNot a HEX file !");
      exit (0);
   }

   /* Get data length */
   string[0] = fgetc (fp);
   string[1] = fgetc (fp);
   string[2] = 0;
   sscanf (string,"%x",&length);
   length = length/4;

   if (length == 0)
   {
      printf ("\nLoad complete\n");
      done=1;
      break;
   }

   /* Skip over the rest of header */
   for (i=0; i<6; i++)
   {
      ch = fgetc (fp);
   }
```

A.7 Hex File Loader and Starter (continued)

```
      /* Get data and store it into buffer */
      for (i=0; i<length; i++)
      {
         string[6] = fgetc (fp);
         string[7] = fgetc (fp);
         string[4] = fgetc (fp);
         string[5] = fgetc (fp);
         string[2] = fgetc (fp);
         string[3] = fgetc (fp);
         string[0] = fgetc (fp);
         string[1] = fgetc (fp);
         string[8] = 0;
         sscanf (string,"%lx",&data);
         addr = offset + i * 4;
         poke_long (addr, data);
         ++word_count;
      }
      printf (".");
      offset = offset + length * 4;
      /* Skip over cksum and CR/LF */
      fgets (string,80,fp);
   }
   return (word_count);
}

void boot_processor (a16_base)
unsigned int a16_base;
{
   poke_word(CONTROL_REG + a16_base, RESET84_A);        /* Reset 4284 */
   poke_word(CONTROL_REG + a16_base, 0);              /* Release 4284 */
}

void boot_global (a16_base, offset)
unsigned int a16_base;
unsigned long offset;
{
   unsigned long boot84_func;
   unsigned long i;
   unsigned long temp;
   /* Set Pattern in first location of global memory which indicates */
   /* to the processor to boot up the application specified in the   */
   /* next memory location                                           */
   poke_long(0, 0xa55a5aa5L);
   poke_long(4, 0x80000000L + offset/4);
   /* Change 8 bit mem width to 32 */
   poke_long(offset, 0x20);
   /* Reset Processor to start program */
    boot_processor(a16_base);
}
```

A.7 Hex File Loader and Starter (continued)

```
void boot_flash (a16_base, sector)
unsigned int a16_base;
unsigned int sector;
{
   unsigned long boot84_func;
   unsigned long i;
   unsigned long temp;
   /* Read function table within 4284 memory to
                                locate pointer to c40 loader function */
   boot84_func = peek_long (FBOOT);
   printf ("boot 84 func = %lx\n", boot84_func);

   /* Load Boot function address as the command */
   poke_long (CMD, boot84_func);
   printf ("%lx\n", peek_long(CMD));

   /* Arg1 will contain the C40 address to start of data */
   poke_long(ARG1,0xb00000 + (sector * 0x10000) + 8);

   /* Clear out RDY flag */
   poke_long (RDY,0);

   poke_word (CONTROL_REG + a16_base,INT84_A);      /* Interrupt C40a */
}

void load_flash_memory (a16_base, sector, offset, word_count)
unsigned int a16_base;
unsigned int sector;
unsigned long offset;
long word_count;
{
   unsigned long func;
   unsigned long i;
   unsigned long temp;

   /* Make sure mem width is set to 8 - Flash Memory is an 8 bit device */
   poke_long (offset, 0x8);

   /* Proceeding the program block, add 2 additional parameters which */
   /* will indicate that the flash contains code to be booted and the */
   /* Start of the program in flash memory. */
   poke_long (offset - 8, 0xa55a5aa5);
   poke_long (offset - 4, 0xb00000 + (sector * 0x10000) + 8);

   /* Adjust offset and word count to include 2 new parameters */
   word_count += 2;
   offset -= 8;
   printf (" offset = %lx  word count = %d\n", offset, word_count);
```

A.7 Hex File Loader and Starter (continued)

```
   /* Read function table within 4284 memory to locate pointer to
                                        'C40 load flash memory function */
   /* Load Program into Flash Memory */
   func = peek_long (FLDW);
   printf ("load flash func = %lx\n", func);

   /* Load Boot function address as the command */
   poke_long (CMD,func);
   printf ("%lx\n", peek_long(CMD));

   /* Arg1 will contain the ['C40 address to start of data */
   poke_long (ARG1,0x80000000L + offset/4);

   /* Arg2 will contain the 'C40 destination Address of flash memory */
   poke_long (ARG2,0xb00000L + (sector * 0x10000));

   /* Arg3 will contain the Word count - 1 */
   poke_long (ARG3,word_count - 1);

   /* Clear out RDY flag */
   poke_long (RDY,0);
   poke_word (CONTROL_REG + a16_base,INT84_A);      /* Interrupt C40a */
   /* Wait for Command to Complete */
   while (peek_long(RDY) != 1);
}

void erase_sector (a16_base, sector)
unsigned int a16_base;
unsigned int sector;
{
   unsigned long func;
   /* Read function table within 4284 memory to locate pointer to
                               'c40 erase flash memory sector function */
   func = peek_long (SERASE);
   printf ("Erase Function = %lx\n", func);

   /* Load func as Command and Address of Sector */
   poke_long (CMD,func);

   poke_long (ARG1,0xb00000L + (sector * 0x10000));

   poke_long (RDY,0);         /* Clear out Command Complete Indicator */

   poke_word (CONTROL_REG + a16_base,INT84_A);     /* Interrupt 'C40a */

   while (peek_long(RDY) != 1);       /* Wait for Command to Complete */
}
```

A.7 Hex File Loader and Starter (continued)

```
/* This function overrides a program previously loaded from Boot ROM */
void boot_override (a16_base)
unsigned int a16_base;
{
   int  i;
   /* Load first location in dual port memory with a pattern which */
   /* indicates to the Boot Code to ignore flash memory */
   poke_long (0, 0xc33c3cc3L);
   poke_long (RDY,0);         /* Clear out Command Complete Indicator */

   poke_word (CONTROL_REG + a16_base, RESET84_A);       /* Reset 4284 */
   poke_word(CONTROL_REG+a16_base,0);                                    /

   /* Wait for Command to Complete */
   while (peek_long(RDY) != 1);

   /* Reset the boot override command */
   poke_long (0, 0x0L);
}
```

A.8 Unix Mapping File

```
/* FILE - MAPUNX.C - These modules map VME into virtual memory on a UNIX
system and handle the subsequent definitions for low level memory access. */

#include            <sys/ioctl.h>
#include            <sys/param.h>
#include            <sys/types.h>
#include            <sys/mman.h>
#include            <sys/file.h>
#include            <fcntl.h>
#include            <unistd.h>
#include            <sgtty.h>
#include            <stdio.h>
#include            <ctype.h>
#undef              PAGESIZE
#define BT_RRAM     0x1000000
#define BT_RBIO     0x0400000
#define BT_LBIO     0x0200000
#define PAGESIZE24  0x01000000
#define PAGESIZE16  65536
#define A16         0x2d
#define A32         0x0d
char                *Address;       /* VMEbus A24/A32 memory pointer */
char                *Ctladdr;           /* VMEbus A16 memory pointer */
char  *Status;
```

A.8 Unix Mapping File (continued)

```
/* map physical memory to virtual memory -  NOTE: parameters in
   Init_Memory are NOT used in a virtual memory system. */

void setup(a16_base, dpr_base)
unsigned int a16_base;
unsigned long dpr_base;
{
      intfd;
      longloopcount;
      if (( fd = open("/dev/sbus1",O_RDWR)) == -1 )
   {
      perror("OPEN FAILED: ");
      return;
   }
      if((Address = mmap((caddr_t)0, PAGESIZE24, PROT_READ | PROT_WRITE,
          MAP_SHARED, fd, (off_t)BT_RRAM) ) == 0xff)
   {
      perror ("MMAP 1 FAILED: ");
      return;
   }

      if((Ctladdr = mmap((caddr_t)0, PAGESIZE16, PROT_READ | PROT_WRITE,
                        MAP_SHARED, fd, (off_t)BT_RBIO) ) == 0xff)
   {
      perror ("MMAP 2 FAILED: ");
      return;
   }

      if((Status = mmap((caddr_t)0, PAGESIZE16, PROT_READ | PROT_WRITE,
                         MAP_SHARED, fd, (off_t)BT_LBIO) ) == 0xff)
   {
      perror ("MMAP 3 FAILED: ");
      return;
   }

   /* Clear Local Node Error Latches */
   Status[0] |= 0x80;

   Status[9] = 0x8;                  /* Set use address page register */
   Status[8] = 0x4f;  /* Set page size and use address modifier register
*/
   Status[10] = dpr_base >> 24;
   Status[11] = (dpr_base >> 16) & 0xff;
   Status[12] = A32;      /* Set A32 Extended Supervisory Data Access */
}
```

A.8 **Unix Mapping File** (continued)

```
/* low level memory access routines */
void poke_long (Addr, Data)
unsigned long Addr;
unsigned long Data;
{
   unsigned int   B1,B2,B3,B4;
   unsigned short *addrP, data;

   B2 = (unsigned int) (Data & 0xff);
   B1 = (unsigned int) ((Data >> 8) & 0xff);
   B4 = (unsigned int) ((Data >> 16) & 0xff);
   B3 = (unsigned int) ((Data >> 24) & 0xff);

   addrP = (unsigned short *)(Address + Addr);
   *addrP++ = (B1 << 8) | B2;
   *addrP = (B3 << 8) | B4;

   #ifdef DEBUG
      printf ("poke_long  Addr = %lx  Data = %lx \n", Addr, data);
   #endif
}

unsigned long peek_long (Addr)
unsigned long Addr;
{
   unsigned int   Lo, Hi;
   unsigned long  data;
   unsigned short *addrP;

   addrP = (unsigned short *)(Address + Addr);
   Lo = *addrP++;
   Hi = *addrP;

   data = (long) (((long)Hi << 16) | (Lo & 0xffff));
   #ifdef DEBUG1
      printf ("peek_long  Addr = %lx  Data = %lx \n", Addr, data);
   #endif
   return (data);
}

void poke_word (Addr, Data)
unsigned int Addr;
unsigned int Data;
{
      #ifdef DEBUG
         printf("poke_word  Addr = %lx  Data = %lx \n", Addr, Data);
      #endif
```

A.8 Unix Mapping File (continued)

```
      Status[12] = A16;
      *(unsigned short *) (Ctladdr + Addr ) = Data;
      Status[12] = A32;
   }

   unsigned int peek_word (Addr)
   unsigned int Addr;
   {
      unsigned int data;
      Status[12] = A16;
      data = *(unsigned short *) (Ctladdr + Addr );
      Status[12] = A32;
      #ifdef DEBUG
         printf ("peek_word  Addr = %lx  Data = %lx \n", Addr, data);
      #endif
      return (data);
}
/*********************** end of file MAPUNX.C ***********************/
```

A.9 Timer Routines

The two 'C' routines listed below can be used to start and stop Timer 1 on the Model 4284's 'C40 processor.

```
/* timer1.c */
/* timer1 routines */

#ifdef   MACH_C40
#define  TGCR1 0x100030
#define  TC1   0x100034
#define  TP1   0x100038
#else
#define  TGCR1 0x808030
#define  TC1   0x808034
#define  TP1   0x808038
#endif

long far *addr;
```

A.9 Timer Routines (continued)

```
/* ------------------------------------ */
void start_timer1 ()
{
   /* start timer1 counter */
   addr  = (long far*)TC1;
   *addr = 0;
   addr  = (long far*)TP1;
   *addr = 0xffffffff;
   addr  = (long far*)TGCR1;
   *addr = 0x03c0;
}
/* ------------------------------------ */
long stop_timer1()
{
   /* stop and read timer1 counter */
   addr  = (long far*)TGCR1;
   *addr = 0x0300;
   addr  = (long far*)TC1;
   return (*addr);
}
/* ------------------------------------ */
```

A.10 Using the TCLK Signals for Serial I/O

The 'C' program listed below simulates a UART. It synthesizes asynchronous communications signals using the 'C40's TCLK0 and TCLK1 I/O signals. These signals can be jumpered such that they are brought to the 4284's front panel Serial Port connector , or an RS-232 driver/receiver. See Table 2-14, in Section 2.11 for jumper information, and Section 2.12.3 for a description of the Serial Port connector.

```
#define FLOAT 1
#define EOF -1
#define NULL 0

volatile unsigned int *timer0_address, *timer1_address, *c40_reg;
volatile unsigned int clock, clockd2;
unsigned char string[100];
static char header_a[] = "Pentek 4284/C40 Processor Ready";
static char tmsg[] =  "Enter string to echo -->";
static char crlf[] = { 0xd,0xa,0x0 };
```

A.10 Using the TCLK Signals for Serial I/O (continued)

```
main()
{
   unsigned int i, op=1, ip=1, data;
   c40_reg  = (unsigned int *) 0x340000u;
   init_uart ();
   prints (crlf);
   prints (header_a);
   prints (crlf);
   prints (crlf);

   while (1)
   {
      prints (crlf);
      prints (tmsg);
      gets (string);
      prints (crlf);
      prints (string);
   }
}

init_uart ()
{
   unsigned int i, op=1, ip=1, tmp;

   timer0_address = (unsigned int *)0x100020u;
   timer1_address = (unsigned int *)0x100030u;

   /* Setup as a RS232 transmitter */
   *timer1_address = 0x306u;                          /* Mark line */
   /* Setup as a RS232 reciever */
   *(timer0_address+8) = 0x7fffffffu;                    /* period */
   *timer0_address = 0x300u;                            /* control */

   /* Look for marking */
   do
   {
      tmp = *timer0_address & 0x8u;
   } while (tmp == 0);

   /* Look for start bit */
   do
   {
      tmp = *timer0_address & 0x8u;
   } while (tmp != 0);

   /* Start bit detected - start timer */
   *timer0_address = 0x3c0u;                            /* control */
```

A.10 Using the TCLK Signals for Serial I/O (continued)

```
   /* Look for end of start bit */
   do
   {
      tmp = *timer0_address & 0x8u;
   } while (tmp == 0);

   /* Get timer value for use as clock */
   clock = *(timer0_address+4)/2;
   clockd2 = clock/2;                    /* 9600 baud - clockd2 = 521; */

   *(timer0_address+8) = clockd2;              /* Start receive clock */
   *(timer1_address+8) = clock;
   *timer1_address = 0x3c6u;                  /* Start transmit clock */
   delay(100);
}

#undef putchar
putchar (unsigned int data)
{
   int i, tmp;

   /* Set start bit */
   *timer1_address = 0x3c2u;

   /* Look for next low clock transition */
   do
   {
      tmp = *timer1_address & 0x800u;
   } while (tmp == 0);

   /* Look for next high clock transition */
   do
   {
      tmp = *timer1_address & 0x800u;
   } while (tmp != 0);

   /* Send 8 data bits to line */
   for ( i=0; i<8; i++ )
   {
      *timer1_address = 0x382u | ((data&1)<<2);
      data = data >> 1;
      do
      {
         tmp = *timer1_address & 0x800u;
      } while (tmp == 0);
```

A.10 Using the TCLK Signals for Serial I/O (continued)

```
      do
      {
         tmp = *timer1_address & 0x800u;
      } while (tmp != 0);
   }

   /* Set stop bit */
   *timer1_address = 0x3c6u;
   do
   {
      tmp = *timer1_address & 0x800u;
   } while (tmp == 0);
   do
   {
      tmp = *timer1_address & 0x800u;
   } while (tmp != 0);
}

printf(fmt,args)
char *fmt; unsigned args;
{
   format(putchar,fmt,&args);
}

#undef getchar
getchar ()
{
   unsigned int data=0;
   unsigned int i, tmp;

   /*Look for marking */
   do
   {
   tmp = *timer0_address & 0x8u;
   } while (tmp == 0);

   /* Look for start bit */
   do
   {
      tmp = *timer0_address & 0x8u;
   } while (tmp != 0);

   /* Start bit detected - start half bit timer */
   *(timer0_address+8) = clockd2;              /* Start receive clock */
   *timer0_address = 0x3c0u;                                 /* control */
```

A.10 Using the TCLK Signals for Serial I/O (continued)

```
   /* Look for next low clock transition */
   do
   {
      tmp = *timer0_address & 0x800u;
   } while (tmp != 0);

   /* Look for next high clock transition */
   do
   {
      tmp = *timer0_address & 0x800u;
   } while (tmp == 0);

   /* Start full bit timer */
   *(timer0_address+8) = clock;                /* Start receive clock */
   *timer0_address = 0x3c0u;                                 /* control */

   /* Get 8 data bits from line */
   for ( i=0; i<8; i++ )
   {
      do
      {
         tmp = *timer0_address & 0x800u;
      } while (tmp != 0);
      do
      {
         tmp = *timer0_address & 0x800u;
      } while (tmp == 0);
      data = data | ((*timer0_address & 0x8u) << 5);
      data = data >> 1;
   }

   /* Look at stop bit */
   do
   {
      tmp = *timer0_address & 0x800u;
   } while (tmp != 0);
   do
   {
      tmp = *timer0_address & 0x800u;
   } while (tmp == 0);
   data = data >> 1;
   if ((data & 0x7f) == 0x0d)
      data=0x0a;
   putchar (data);
   return (data);
}
```

A.10 Using the TCLK Signals for Serial I/O (continued)

```
#undef gets
char *gets (line)
char *line;
{
   char *cp;
   unsigned int i;

   cp = line;
   while ((i = getchar()) != EOF && i != 0x0a)
      *cp++ = i;
   *cp = 0;
   if (i == EOF && cp == line)
      return NULL;
   return (line);
}

static char *
_fmtcvt (ap, base, cp, len)
int *ap; register char *cp;
{
   register unsigned long val;
   static char digits[]="0123456789abcdef";
   if (len == sizeof(long))
      val = *(long *)ap;
   else if (base > 0)
      val = *(unsigned *)ap;
   else
      val = *ap;
   len = 0;
   if (base < 0)
   {
      base = -base;
      if ((long)val < 0)
      {
         val = -val;
         len = 1;
      }
   }
   do
   {
      *--cp = digits[(int)(val%base)];
   } while ((val /= base) != 0);
   if (len)
      *--cp = '-';
   return cp;
}
```

A.10 Using the TCLK Signals for Serial I/O (continued)

```
prints(s)
unsigned int *s;
{
   int i;
   i=0;
   while (*(s+i) != 0)
   {
      putchar(*(s+i));
      i++;
   }
}
format (putsub, fmt, argp)
register int (*putsub)(); register char *fmt; char *argp;
{
   register int c;
   union
   {
      int *ip;
      char *cp;
      char **cpp;
      #ifdef FLOAT
         double *dp;
      #endif
   } args;
   int charcount;
   int rj, fillc;
   int maxwidth, width;
   int i, k;
   char *cp;
   auto char s[200];
   charcount = 0;
   args.cp = argp;
   while (c = *fmt++)
   {
      if (c == '%')
      {
         s[14] = 0;
         rj = 1;
         fillc = ' ';
         maxwidth = 10000;
         if ((c = *fmt++) == '-')
            {
               rj = 0;
               c = *fmt++;
         }
         if (c == '0')
         {
            fillc = '0';
            c = *fmt++;
         }
```

A.10 Using the TCLK Signals for Serial I/O (continued)

```
if (c == '*')
{
   width = *args.ip++;
   c = *fmt++;
}
else
{
   for (width = 0 ; isdigit(c); c = *fmt++)
      width = width*10 + c - '0';
}
if ( c == '.' )
{
   if ((c = *fmt++) == '*')
   {
      maxwidth = *args.ip++;
      c = *fmt++;
   }
   else
   {
      for (maxwidth = 0 ; isdigit(c) ; c = *fmt++)
         maxwidth = maxwidth*10 + c - '0';
   }
}
i = sizeof(int);
if (c == 'l')
{
   c = *fmt++;
   i = sizeof(long);
}
else if (c == 'h')
   c = *fmt++;
switch (c)
{
   case 'o':
      k = 8;
      goto do_conversion;
   case 'u':
      k = 10;
      goto do_conversion;
   case 'x':
      k = 16;
      goto do_conversion;
   case 'd':
      k = -10;
do_conversion:
   cp = _fmtcvt(args.cp, k, s+14, i);
   args.cp += i;
   break;
```

A.10 Using the TCLK Signals for Serial I/O (continued)

```
            case 's':
               i = strlen(cp = *args.cpp++);
               goto havelen;
         #ifdef FLOAT
            case 'e':
            case 'f':
            case 'g':
               ftoa (*args.dp++, s, maxwidth==10000?6:maxwidth, c-'e');
               i = strlen (cp = s);
               maxwidth = 200;
               goto havelen;
         #endif
            case 'c':
               c = *args.ip++;
            default:
               *(cp = s+13) = c;
               break;
         }

         i = (s+14) - cp;
         havelen:
         if ( i > maxwidth )
            i = maxwidth;
         if (rj)
         {
            if ((*cp == '-' || *cp == '+') && fillc == '0')
            {
               --width;
               if ((*putsub)(*cp++) == -1)
                  return -1;
            }
            for (; width-- > i ; ++charcount)
               if ((*putsub)(fillc) == -1)
                  return -1;
         }
         for (k = 0 ; *cp && k < maxwidth ; ++k)
            if ((*putsub)(*cp++) == -1)
               return -1;
         charcount += k;
         if (!rj)
         {
            for (; width-- > i ; ++charcount)
               if ((*putsub)(' ') == -1)
                  return -1;
         }
      }
```

A.10 Using the TCLK Signals for Serial I/O (continued)

```
      else
      {
         if ((*putsub)(c) == -1)
            return -1;
         ++charcount;
      }
   }
   return charcount;
}

static char *scnstr;
static char quit;

sscanf (string, fmt, arg)
char   *string, *fmt;
int    *arg;
{
   int     sgetc ();
   scnstr = string;
   quit = 0;
   return scanfmt (sgetc, fmt, &arg);
}

static sgetc (what)
{
   if (what == 0)
   {
      if (*scnstr)
         return * scnstr++ & 255;
      quit = 1;
   }
   else
   {
      if (!quit)
         return * --scnstr & 255;
   }
   return - 1;
}

static int  maxwidth;
static int  (*gsub) ();
char   *strchr ();

scanfmt (getsub, fmt, args)
```

A.10 Using the TCLK Signals for Serial I/O (continued)

```
int (*getsub) ();
register char  *fmt;
register int  **args;
{
#ifdef FLOAT
   double  atof ();
#endif
   long    lv;
   register int c, count, base, cc;
   char suppress, lflag, widflg;
   char *cp;
   auto char tlist[130];
   static char list[] = "ABCDEFabcdef9876543210";
   static char vals[] =
   {
      10, 11, 12, 13, 14, 15, 10, 11, 12, 13, 14, 15, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0
   };

   count = 0;
   gsub = getsub;
   while (c = *fmt++)
   {
      if (c == '%')
         {
            widflg = lflag = suppress = 0;
            maxwidth = 127;
            if (*fmt == '*')
            {
               ++fmt;
               suppress = 1;
            }
         if (isdigit (*fmt))
         {
            maxwidth = 0;
            do
            {
               maxwidth = maxwidth * 10 + *fmt - '0';
            } while (isdigit (*++fmt));
             widflg = 1;
         }
         if (*fmt == 'l')
         {
            lflag = 1;
            ++fmt;
         }
```

A.10 Using the TCLK Signals for Serial I/O (continued)

```
switch (cc = *fmt++)
{
   case '%':
      c = '%';
      goto matchit;
   case 'h':            /* specify short (for compatibility) */
      lflag = 0;
      goto decimal;
   case 'D':
      lflag = 1;
   case 'd':
   decimal:
   c = 12;
   base = 10;
   goto getval;
   case 'X':
      lflag = 1;
   case 'x':
      c = 0;
      base = 16;
      goto getval;
   case 'O':
      lflag = 1;
   case 'o':
      c = 14;
      base = 8;

   getval:
   if (skipblank ())
      goto stopscan;
   if (getnum (&list[c], &vals[c], base, &lv) == 0)
      goto stopscan;
   if (!suppress)
   {
      if (lflag)
         *(long *) (*args++) = lv;
      else
         **args++ = lv;
      ++count;
   }
   break;
   #ifdef FLOAT
      case 'E':
      case 'F':
         lflag = 1;
```

A.10 Using the TCLK Signals for Serial I/O (continued)

```
      case 'e':
      case 'f':
         if (skipblank ())
         goto stopscan;
      if (getflt (tlist))
         goto stopscan;
      if (!suppress)
      {
         if (lflag)
            *(double *) (*args++) = atof (tlist);
         else
            *(float *) (*args++) = atof (tlist);
         ++count;
      }
      break;
#endif
case '[':
   lflag = 0;
   if (*fmt == '^' || *fmt == '~')
   {
      ++fmt;
      lflag = 1;
   }
   for (cp = tlist; (c = *fmt++) != ']';)
      *cp++ = c;
   *cp = 0;
   goto string;
case 's':
   lflag = 1;
   tlist[0] = ' ';
   tlist[1] = '\t';
   tlist[2] = '\n';
   tlist[3] = 0;

string:
   if (skipblank ())
      goto stopscan;
charstring:
   if (!suppress)
      cp = (char *) * args++;
   widflg = 0;
   while (maxwidth--)
   {
      if ((c = (*gsub) (0)) == EOF)
         break;
```

A.10 Using the TCLK Signals for Serial I/O (continued)

```
            if (lflag ? (strchr (tlist, c) != 0) : (strchr (tlist, c) == 0))
            {
               (*gsub) (1);                /* unget last character */
               break;
            }
            if (!suppress)
               *cp++ = c;
            widflg = 1;
         }
         if (!widflg)
            goto stopscan;
         if (!suppress)
         {
            if (cc != 'c')
               *cp = 0;
            ++count;
         }
         break;

         case 'c':
            if (!widflg)
               maxwidth = 1;
            tlist[0] = 0;
            lflag = 1;
            goto charstring;
      }
   }
   else
      if (isspace (c))
      {
         if (skipblank ())
            goto stopscan;
      }
      else
      {
      matchit:
         if ((*gsub) (0) != c)
         {
            (*gsub) (1);
            goto stopscan;
         }
      }
   }
```

A.10 Using the TCLK Signals for Serial I/O (continued)

```
      stopscan:
      if (count == 0
      {
         if ((*gsub) (0) == EOF)
         return EOF;
      (*gsub) (1);
   }
   return count;
}

skipblank ()
{
   while (isspace ((*gsub) (0)));
   if ((*gsub) (1) == EOF)
      return EOF;
   return 0;
}

#ifdef FLOAT
   getflt (buffer)
   char *buffer;
   {
      register char  *cp;
      register int    c;
      char    decpt, sign, exp;
      sign = exp = decpt = 0;
      for (cp = buffer; maxwidth--; *cp++ = c)
      {
         c = (*gsub) (0);
         if (!isdigit (c))
         {
            if (!decpt && c == '.')
               decpt = 1;
            else
            {
               if (!exp && (c == 'e' || c == 'E') && cp != buffer)
               {
                  sign = 0;
                  exp = decpt = 1;
                  continue;
               }
               else
               {
               if (sign || (c != '-' && c != '+'))
               {
                  (*gsub) (1);
                  break;
               }
            }
```

A.10 Using the TCLK Signals for Serial I/O (continued)

```
            sign = 1;
         }
         *cp = 0;
         return cp == buffer;
      }
#endif

getnum (list, values, base, valp)
char   *list;
char   *values;
long   *valp;
{
   register char  *cp;
   register int    c, cnt;
   long    val;
   int     sign;

   if (maxwidth <= 0)
      return 0L;
   val = cnt = sign = 0;
   if ((c = (*gsub) (0)) == '-')
   {
      sign = 1;
      ++cnt;
   }
   else
      if (c == '+')
         ++cnt;
   else
      (*gsub) (1);
   for (; cnt < maxwidth; ++cnt)
   {
      if ((cp = strchr (list, c = (*gsub) (0))) == 0)
      {
         if (base == 16 && val == 0 && (c == 'x' || c == 'X'))
            continue;
         (*gsub) (1);
      break;
      }
      val *= base;
      val += values[cp - list];
   }
   if (sign)
      *valp = -val;
   else
      *valp = val;
   return cnt;
}
```

A.10 Using the TCLK Signals for Serial I/O (continued)

```
ftoa(number, buffer, maxwidth, flag)
double number; register char *buffer;
{
   register int i;
   int exp, digit, decpos, ndig;

   static double round[] =
   {
      1.0e+0,
      0.5e+0,
      0.5e-1,
      0.5e-2,
      0.5e-3,
      0.5e-4,
      0.5e-5,
      0.5e-6,
      0.5e-7,
      0.5e-8,
      0.5e-9,
      0.5e-10,
      0.5e-11,
      0.5e-12,
      0.5e-13,
      0.5e-14,
      0.5e-15,
      0.5e-16,
      0.5e-17,
      0.5e-18,
   };

   ndig = maxwidth+1;
   exp = 0;
   if (number < 0.0)
   {
      number = -number;
      *buffer++ = '-';
   }
   if (number > 0.0)
   {
      while (number < 1.0)
      {
         number *= 10.0;
         --exp;
      }
      while (number >= 10.0)
      {
         number /= 10.0;
         ++exp;
      }
   }
```

A.10 Using the TCLK Signals for Serial I/O (continued)

```
if (flag == 2)                                          /* 'g' format */
{
   ndig = maxwidth;
   if (exp < -4 || exp >= maxwidth)
      flag = 0;                                /* switch to 'e' format */
}
else if (flag == 1)                                     /* 'f' format */
   ndig += exp;

if (ndig >= 0)
{
   if ((number += round[ndig>16?16:ndig]) >= 10.0)
   {
      number = 1.0;
      ++exp;
      if (flag)
         ++ndig;
   }
}

if (flag)
{
   if (exp < 0)
   {
      *buffer++ = '0';
      *buffer++ = '.';
      i = -exp - 1;
      if (ndig <= 0)
         i = maxwidth;
      while (i--)
         *buffer++ = '0';
      decpos = 0;
   }
   else
   {
      decpos = exp+1;
   }
}
else
{
   decpos = 1;
}
```

A.10 Using the TCLK Signals for Serial I/O (continued)

```
   if (ndig > 0)
   {
      for (i = 0 ; ; ++i)
      {
         if (i < 16)
         {
            digit = (int)number;
            *buffer++ = digit+'0';
            number = (number - digit) * 10.0;
         }
         else
            *buffer++ = '0';
         if (--ndig == 0)
            break;
         if (decpos && --decpos == 0)
            *buffer++ = '.';
      }
   }

   if (!flag)
   {
      *buffer++ = 'e';
      if (exp < 0)
      {
         exp = -exp;
         *buffer++ = '-';
      }
      else
         *buffer++ = '+';
      if (exp >= 100)
      {
         *buffer++ = exp/100 + '0';
         exp %= 100;
      }
      *buffer++ = exp/10 + '0';
      *buffer++ = exp%10 + '0';
   }
   *buffer = 0;
}
```

A.10 Using the TCLK Signals for Serial I/O (continued)

```
delay(unsigned int val)
{
   int tmp;

   /* Delay for 100 bit times before starting */
   for (val=0; val<100; val++ )
   {
      /* Look for low clock transition */
      do
      {
         tmp = *timer1_address & 0x800u;
      } while (tmp != 0);
      /* Look for next high clock transition */
      do
      {
         tmp = *timer1_address & 0x800u;
      } while (tmp == 0);
   }
}
```

Appendix B: Boot EPROM Source Code Listing

B.1 Boot EPROM Source Code Listing

Below is a 'C40 Assembly Language listing of the boot code and look/set functions contained in the factory-installed EPROM provided by Pentek with Model 4284.

```
       .list
*
*       TITLE 'PROCESSOR INITIALIZATION'
*
        .global RESET,INIT,BEGIN
        .global NMI,INT0,INT1,INT2,INT3,TINT0
        .global NON_MASK,ISR0,ISR1,ISR2,ISR3,TIME0
        .global LOOK,SET
*
*************************************************************************
*                                                                       *
*                      LOOK/SET memory usage                            *
*       HOST      C40                                                   *
*       --------  ----------                                            *
*      0x000200  0x80000200 = Function address from Host                *
*      0x000201  0x80000201 = Address for look/set function             *
*                           = Source address for block move             *
*                            = Flash boot address                       *
*      0x000202  0x80000202 = Data from Host for set function           *
*                           = Destination address for block move        *
*      0x000203  0x80000203 = Data to Host from look function           *
*                           = Word count-1  for block move              *
*      0x000204  0x80000204 = Ready to Host  (0=not ready / 1=ready)    *
*      0x000205  0x80000205 = Pointer to LOOK function                  *
*      0x000206  0x80000206 = Pointer to SET function                   *
*      0x000207  0x80000207 = Pointer to  BLOCK MOVE function           *
*      0x000208  0x80000208 = Aux ready flag for factory use            *
*      0x000209  0x80000209 = Test pattern 0xa5a5a5a5                   *
*      0x00020a  0x8000020a = Test pattern 0x5a5a5a5a                   *
*      0x00020b  0x8000020b = Test pattern 0x12345678                   *
*      0x00020c  0x8000020c = Test pattern 0x87654321                   *
*      0x00020d  0x8000020d = Test pattern 0xffffffff                   *
*      0x00020e  0x8000020e = Test pattern 0x00000000                   *
*      0x00020f  0x8000020f = Firmware Version 0x10a = 1.0a             *
*      0x000210  0x80000210 = Test pattern 0x80000200                   *
*                                                                       *
*       FLASH support                                                   *
*                                                                       *
*      0x000211  0x80000211 = Pointer to FLASH BOOT function            *
*      0x000212  0x80000212 = Pointer to FLASH LOOK function            *
*      0x000213  0x80000213 = Pointer to FLASH SET function             *
*      0x000214  0x80000214 = Pointer to FLASH ERASE function           *
*      0x000215  0x80000215 = Pointer to FLASH ERASE SECTOR function    *
*      0x000216  0x80000216 = Pointer to FLASH BYTE LOAD function       *
*      0x000217  0x80000217 = Pointer to FLASH BYTE UNLOAD function     *
*      0x000218  0x80000218 = Pointer to FLASH WORD LOAD function       *
*      0x000219  0x80000219 = Pointer to FLASH WORD UNLOAD function     *
*                                                                       *
*************************************************************************
```

B.1 Boot EPROM Source Code Listing (continued)

```
*
*       PROCESSOR INITIALIZATION FOR THE TMS320C40.
*
*
*       IN THIS SECTION, CONSTANTS THAT CANNOT BE REPRESENTED
*       IN THE SHORT FORMAT ARE INITIALIZED.
        .text
*
RESET   .word   INIT            ; RS-load address INIT to HOST
NMI     .word   NON_MASK        ; Non maskable interrupt NMI
TINT0   .word   TIME0           ; Timer 0 interrupt processing
INT0    .word   ISR0            ; INT0-
INT1    .word   ISR1            ; INT1-
INT2    .word   ISR2            ; INT2-
INT3    .word   ISR3            ; INT3-
LOOKADR .word   LOOK            ; LOOK function
SETADR  .word   SET             ; SET function
MOVEADR .word   BLKMOV          ; Block move function
NOPFUNC .word   NOTHIN          ; Do nothing function
PATRNS  .word   PATRN           ; Test patterns
FBOOTA  .word   FBOOT           ; FLASH BOOT function
FLOOKA  .word   FLOOK           ; FLASH LOOK function
FSETA   .word   FSET            ; FLASH SET function
FLDBA   .word   FLDB            ; FLASH LOAD BYTE function
FUNLDBA .word   FUNLDB          ; FLASH UNLOAD BYTE function
FLDWA   .word   FLDW            ; FLASH LOAD WORD function
FUNLDWA .word   FUNLDW          ; FLASH UNLOAD WORD function
ERASEA  .word   ERASE           ; FLASH ERASE function
SERASEA .word   SERASE          ; FLASH SECTOR ERASE function
BWIDE   .word   B_WIDE          ;
WWIDE   .word   W_WIDE          ;
*
MASK    .word   0ffffffffH      ;
BOOTF   .word   0003fffffH      ; Boot failed blink value
BOOTP   .word   00003ffffH      ; Boot passed blink value
IIEMASK .word   000000001H      ; Interrupt enable mask
IIFMASK .word   000000090H      ; Interrupt enable mask external
*
IVTADR  .word   0002FFC00H      ;*Beginning address of interrupt vectors
CTRL    .word   000100000H      ; Pointer for peripheral-bus memory map
GLOINT  .word   03D9EC000H     ; Init of global memory interface control
(0)
LOCALINT.word   03DEF4000H     ; Init of local memory interface control
(4)
FLASHINT.word   03DEF4110H      ; Init of local during flash access (4)
C40REG  .word   010000000H      ; Processor CONTROL registers
VMEREG  .word   090000000H      ; Processor VME register
STCK    .word   0002FFF00H      ; Beginning of stack
PATRN   .word   0A5A5A5A5H      ;
        .word   05A5A5A5AH      ;
        .word   012345678H      ;
        .word   087654321H      ;
        .word   0ffffffffH      ;
        .word   000000000H      ;
        .word   04284010fH      ; Firmware Version 1.0f
```

B.1 Boot EPROM Source Code Listing (continued)

```
*
PROC_ADDR     .word 80000200H    ;
PROC_TEST     .word 80000209H    ; test address
FBASE         .word 00b00000H    ;
FBOOTL        .word 00b00000H    ; Points to boot flash flag
GBOOTL        .word 80000000H    ; Points to boot global flag
FBOOTJ        .word 00b00004H    ; Contains flash address to boot from
FSEQ55        .word 00b05555H    ;
FSEQAA        .word 00b02AAAH    ;
FSEQRST       .word 0f0H         ; Read/Reset
FSEQWRT       .word 0a0H         ; Write
FSEQER1       .word 080H         ; Erase seq start
FSEQER2       .word 010H         ; Chip erase
FSEQER3       .word 030H         ; Sector erase
*
FLASHFG       .word 0A55A5AA5H   ; Boot flash flag
FLASHNB       .word 0C33C3CC3H   ; Dont Boot flash flag
FBOOTT        .word 0            ; Holds boot address
*
INIT                                    ;
        LDPK    02Fh                  ; Point the DP register to page 2Fh
         LDA     @CTRL,AR0              ; Point to control register
        LDI     @GLOINT,R0             ; Init global memory interface control
         STI     R0,*AR0                ;
        LDI     @LOCALINT,R0           ; Init local memory interface control
         STI     R0,*+AR0(4)            ;
*
         LDI     @STCK,SP               ; Init stack pointer to 2fff00h
         LDI     1800H,ST               ; Clear and enable cache and
                                        ; disable OVM
         LDI     @IVTADR,R0             ; Setup interrupt vectors
         LDPE    R0,IVTP                ; Set expansion register
*
* Turn on front panel LED  and assert System Reset
         LDA     @VMEREG,AR3            ;
        LDI     03D41H,R0              ; LED on / A24 address mode /sysreset
         STI     R0,*AR3                ; Set register
         NOP                            ;
         NOP                            ;
         NOP                            ;
         NOP                            ;
         NOP                            ;
         LDI     03D40H,R0              ; LED on / A24 address mode
         STI     R0,*AR3                ; Set register
*
* Setup interrupt controller
         LDA     @C40REG,AR3            ;
         LDI     04000H,R0              ; Enable Host int. to INT1
         STI     R0,*+AR3(3)            ; Set register
* Setup Mix control register
*        CALL    CKERASE                ; See if flash should be erased
         LDI     0,R0                   ; Init Mix control register
         STI     R0,*AR3                ; Set register
*
```

B.1 Boot EPROM Source Code Listing (continued)

```
*
* Setup pointers for HOST to use
        LDA     @PROC_ADDR,AR2          ; Processor base address
        LDI     @NOPFUNC,R0             ; Init GO address
        STI     R0,*+AR2(0)             ;
        LDI     @LOOKADR,R0             ; Init LOOK address
        STI     R0,*+AR2(5)             ;
        LDI     @SETADR,R0              ; Init SET address
        STI     R0,*+AR2(6)             ;
        LDI     @MOVEADR,R0             ; Init BLOCK MOVE address
        STI     R0,*+AR2(7)             ;
        LDI     @FBOOTA,R0              ; Init BOOT address
        STI     R0,*+AR2(17)            ;
        LDI     @FLOOKA,R0              ; Init LOOK address
        STI     R0,*+AR2(18)            ;
        LDI     @FSETA,R0               ; Init SET address
        STI     R0,*+AR2(19)            ;
        LDI     @ERASEA,R0              ; Init ERASE address
        STI     R0,*+AR2(20)            ;
        LDI     @SERASEA,R0             ; Init SECTOR ERASE address
        STI     R0,*+AR2(21)            ;
        LDI     @FLDBA,R0               ; Init LOAD BLOCK BYTE address
        STI     R0,*+AR2(22)            ;
        LDI     @FUNLDBA,R0             ; Init UNLOAD BLOCK BYTE address
        STI     R0,*+AR2(23)            ;
        LDI     @FLDWA,R0               ; Init LOAD BLOCK WORD address
        STI     R0,*+AR2(24)            ;
        LDI     @FUNLDWA,R0             ; Init UNLOAD BLOCK WORD address
        STI     R0,*+AR2(25)            ;
*
* Store data patterns in memory
        LDA     @PATRNS,AR1             ; Processor test patterns
        LDA     @PROC_TEST,AR4          ; Processor test address
        LDI     *AR1++,R0               ;
        RPTS    6                       ; Move seven patterns
        STI     R0,*AR4++               ;
||      LDI     *AR1++,R0               ;
        STI     R0,*AR4++               ;
        NOP                             ;
        NOP                             ;
        NOP                             ;
*
* Check test patterns
        LDA     @PATRNS,AR1             ; Processor test patterns
        LDA     @PROC_TEST,AR4          ; Processor test address
        LDA     6,AR5                   ;
CKDATA
        LDI     *AR1++,R0               ;
||      LDI     *AR4++,R1               ;
        CMPI    R0,R1                   ;
        BNZ     READBD                  ;
        DB      AR5,CKDATA              ;
```

B.1 Boot EPROM Source Code Listing (continued)

```
CKEND
        NOP                             ;
        NOP                             ;
        NOP                             ;
*
READGD
        LDI     @BOOTP,R0               ; Timer period Pass
        B       SETTMR                  ;
READBD
        LDI     @BOOTF,R0               ; Timer period Fail
SETTMR
        STI     R0,*+AR0(028H)          ;
        LDI     03C0H,R0                ; Start timer
        STI     R0,*+AR0(020H)          ;
*
* Set proccessor ready
        LDI     1,R0                    ;
        STI     R0,*+AR2(4)             ; Set ready flag
*
* Check ready flag   -  Not used
*       LDI     *+AR2(4),R1             ; Check ready flag
*       CMPI    1,R1                    ; Did it write in ????
*       BNZ     READBD                  ; No so blink LED slowly
*
*
        CALL    CKERASE                 ; See if flash should be erased
        CALL    CKBOOTG                 ; See if boot from global set
*
       LDA     @GBOOTL,AR1             ; See if block boot from flash set
        LDI     *AR1,R0                   ;
        CMPI    @FLASHNB,R0             ;
        BEQ     NOFSHBOT                ; Dont boot flash
        CALL    CKBOOTF                 ; See if boot from flash set
NOFSHBOT
       LDI     @IIEMASK,IIE            ; Enable interrupts - Timer 0 only
        LDI     @IIFMASK,IIF              ; Enable interrupts - IIF1 only
        IACK    *+AR2(0)                  ; Clear VME interrupt
        OR      2000H,ST                  ; Global interrupt enable
*
*
WAIT
        IDLE                            ;
        B       WAIT                    ;
*
*
NON_MASK        ; Non maskable interrupt NMI-loads
                ; address NMI to PC
        RETI                    ;
```

B.1 Boot EPROM Source Code Listing (continued)

```
TIME0           ; Timer 0 interrupt processing
        PUSH    ST                      ;
        PUSH    R0                      ;
        PUSHF   R0                      ;
        PUSH    R1                      ;
        PUSHF   R1                      ;
        PUSH    AR0                     ;
        PUSH    DP                      ;
       LDPK    02Fh                   ; Point the DP register to page 2Fh
* Toggle front panel LED
        LDA     @VMEREG,AR0             ;
        LDI     040H,R0                 ; LED bit
        LDI     *+AR0(0),R1             ; Get register setting
        XOR     R1,R0                   ; Flip LED bit
        STI     R0,*+AR0(0)             ; Set register
        POP     DP                      ;
        POP     AR0                     ;
        POPF    R1                      ;
        POP     R1                      ;
        POPF    R0                      ;
        POP     R0                      ;
        POP     ST                      ;
        RETI                    ;
*
ISR0            ; INT0-
        RETI                    ;
ISR1            ; INT1- Mix interrupt processing
        PUSH    ST                      ;
        PUSH    R0                      ;
        PUSHF   R0                      ;
        PUSH    R1                      ;
        PUSHF   R1                      ;
        PUSH    AR0                     ;
        PUSH    AR1                     ;
        PUSH    AR2                     ;
        PUSH    AR3                     ;
        PUSH    DP                      ;
       LDPK    02Fh                   ; Point the DP register to page 2Fh
        LDA     @PROC_ADDR,AR2          ; Processor base address
        LDI     2,R0                    ;
        STI     R0,*+AR2(8)             ; Set Aux ready flag
        LDI     *+AR2(0),R0             ; Get address from HOST
        CALLU   R0                      ;
```

B.1 Boot EPROM Source Code Listing (continued)

```
RETURN
        LDI     1,R0                    ;
        STI     R0,*+AR2(4)             ; Set ready flag to Host
        IACK    *+AR2(0)                ; Clear VME interrupt
        POP     DP                      ;
        POP     AR3                     ;
        POP     AR2                     ;
        POP     AR1                     ;
        POP     AR0                     ;
        POPF    R1                      ;
        POP     R1                      ;
        POPF    R0                      ;
        POP     R0                      ;
        POP     ST                      ;
        RETI                            ;
LOOK
        LDI     *+AR2(1),AR0            ; Get address from 0x201
        LDI     *AR0,R0                 ; Read word
        STI     R0,*+AR2(3)             ; Store into 0x203
        RETSU                           ;
SET
        LDI     *+AR2(1),AR0            ; Get address from 0x201
        LDI     *+AR2(2),R0             ; Get word to write
        STI     R0,*AR0                 ; Store at pointer
        RETSU                           ;
*
BLKMOV
        LDI     *+AR2(1),AR0            ; Get src address from 0x201
        LDI     *+AR2(2),AR1            ; Get destination address
        LDI     *+AR2(3),RC             ; Get count
        RPTB    BLKMVL                  ;
        LDI     *AR0++,R0               ;
BLKMVL
        STI     R0,*AR1++               ;
        RETSU                           ;
*
NOTHIN
        RETSU                           ;
FBOOT           ; FLASH BOOT function
        CALL    FLASHI                  ;
        LDI     *+AR2(1),AR1            ; Get BOOT address from 0x201
FBOOTE
        LDHI    010h,AR0                ;
        LDI     0,R0                    ; Set start address flag off
        LDI     *AR1++(1),R1            ; Get word width
        LDI     @WWIDE,R10              ;
        LSH     26,R1                   ;
        BN      LOAD0                   ;
        NOP     *AR1++(1)               ;
        LDI     @BWIDE,R10              ;
        ADDI    2,AR1                   ;
```

B.1 Boot EPROM Source Code Listing (continued)

```
LOAD0
        CALLU   R10                     ;
        STI     AR2,*AR0                ; Store gbus control reg
        CALLU   R10                     ;
        STI     AR2,*+AR0(4)            ; Store lbus control reg
LOAD2
        CALLU   R10                     ;
        SUBI3   1,AR2,RC                ; Set block size
        CMPI    -1,RC                   ; If zero byte size start program
        BEQ     IVTP_LOAD               ;
        CALLU   R10                     ;
        LDI     AR2,AR0                 ; Set destination address
        LDI     R0,R0                   ; Test start address flag
        LDIZ    AR2,R9                  ; Load start address if flag is off
        LDI     -1,R0                   ; Set start & dest flag on
        SUBI    1,R10                   ; Sub address with loop
        CALLU   R10                     ;
        LDI     1,R0                    ; Set dest flag off
        ADDI    1,R10                   ;
        B       LOAD2                   ; Jump to load new block when looped
IVTP_LOAD
        CALLU   R10                     ;
        LDPE    AR2,IVTP                ; Load IVTP pointer
TVTP_LOAD
        CALLU   R10                     ;
        LDPE    AR2,TVTP                ; Load TVTP pointer
        CALLU   R10                     ;
        IACK    *AR2                    ; Send IACK signal out
        BU      R9                      ; Branch to start of program
LOOP_B
        RPTB    LOAD_B                  ; PGM load loop
B_WIDE
        LWL0    *AR1++(1),AR2           ;
        NOP                             ;
        LWL1    *AR1++(1),AR2           ;
        NOP                             ;
        LWL2    *AR1++(1),AR2           ;
        NOP                             ;
        LWL3    *AR1++(1),AR2           ;
        NOP                             ;
        LDI     R0,R0                   ; Test load address flag
        BNN     B_END                   ;
LOAD_B
        STI     AR2,*AR0++(1)           ; Store word to destination
B_END
        RETSU                           ;
LOOP_W
        RPTB    LOAD_W                  ; PGM load loop
W_WIDE
        LDI     *AR1++(1),AR2           ;
        LDI     R0,R0                   ; Test load address flag
        BNN     W_END                   ;
```

B.1 Boot EPROM Source Code Listing (continued)

```
LOAD_W
        STI     AR2,*AR0++(1)           ; Store word to destination
W_END
        RETSU                           ;
        CALL    FLASHU                  ;
        RETSU
FLOOK           ; FLASH LOOK function
        CALL    FLASHI                  ;
        LDI     *+AR2(1),AR0            ; Get address from 0x201
        LDI     *AR0,R0                 ; Read word
        STI     R0,*+AR2(3)             ; Store into 0x203
        CALL    FLASHU                  ;
        RETSU                           ;
FSET            ; FLASH SET function
        CALL    FLASHI                  ;
        LDA     @FSEQ55,AR3             ; Write command
        LDI     0aaH,R0                 ;
        STI     R0,*AR3                 ;
        LDA     @FSEQAA,AR3             ;
        LDI     055H,R0                 ;
        STI     R0,*AR3                 ;
        LDA     @FSEQ55,AR3             ;
        LDI     @FSEQWRT,R0             ;
        STI     R0,*AR3                 ;
        LDI     *+AR2(1),AR3            ; Get address from 0x201
        LDI     *+AR2(2),R0             ; Get word to write
        STI     R0,*AR3                 ; Store at pointer
        AND     080H,R0                 ;
FSETL
        LDI     *AR3,R1                 ; Wait till write complete
        AND     080H,R1                 ;
        CMPI    R0,R1                   ;
        BNE     FSETL                   ;
        CALL    FLASHU                  ;
        RETSU                           ;
FLDB            ; FLASH LOAD BYTE function
        CALL    FLASHI                  ;
        LDI     *+AR2(1),AR0            ; Get src address from 0x201
        LDI     *+AR2(2),AR1            ; Get destination address
        LDI     *+AR2(3),RC             ; Get count
        RPTB    FLDBL                   ;
        LDI     *AR0++,R0               ;
        CALL    WRITEFLASH              ;
FLDBL
        NOP     *AR1++                  ;
        CALL    FLASHU                  ;
        RETSU                           ;
```

B.1 Boot EPROM Source Code Listing (continued)

```
FUNLDB          ; FLASH UNLOAD BYTE function
        CALL    FLASHI                  ;
        LDI     *+AR2(1),AR0            ; Get src address from 0x201
        LDI     *+AR2(2),AR1            ; Get destination address
        LDI     *+AR2(3),RC             ; Get count
        RPTB    FUNLDBL                 ;
        LDI     *AR0++,R0               ;
FUNLDBL
        STI     R0,*AR1++               ;
        CALL    FLASHU                  ;
        RETSU                           ;
FLDW            ; FLASH LOAD WORD function
        CALL    FLASHI                  ;
        LDI     *+AR2(1),AR0            ; Get src address from 0x201
        LDI     *+AR2(2),AR1            ; Get destination address
        LDI     *+AR2(3),RC             ; Get count
        RPTB    FLDWL                   ;
        LDI     *AR0++,R0               ;
        CALL    WRITEFLASH              ;
        NOP     *AR1++                  ;
        LSH     -8,R0                   ;
        CALL    WRITEFLASH              ;
        NOP     *AR1++                  ;
        LSH     -8,R0                   ;
        CALL    WRITEFLASH              ;
        NOP     *AR1++                  ;
        LSH     -8,R0                   ;
        CALL    WRITEFLASH              ;
FLDWL
        NOP     *AR1++                  ;
        CALL    FLASHU                  ;
        RETSU                           ;
FUNLDW          ; FLASH UNLOAD WORD function
        CALL    FLASHI                  ;
        LDI     *+AR2(1),AR0            ; Get src address from 0x201
        LDI     *+AR2(2),AR1            ; Get destination address
        LDI     *+AR2(3),RC             ; Get count
        RPTB    FUNLDWL                 ;
        LDI     0,R0                    ;
        LDI     *AR0++,R1               ;
        AND     0FFh,R1                 ;
        OR      R1,R0                   ;
        LDI     *AR0++,R1               ;
        AND     0FFh,R1                 ;
        LSH     8,R1                    ;
        OR      R1,R0                   ;
        LDI     *AR0++,R1               ;
        AND     0FFh,R1                 ;
        LSH     16,R1                   ;
        OR      R1,R0                   ;
        LDI     *AR0++,R1               ;
        AND     0FFh,R1                 ;
        LSH     24,R1                   ;
        OR      R1,R0                   ;
```

B.1 Boot EPROM Source Code Listing (continued)

```
FUNLDWL
        STI     R0,*AR1++               ;
        CALL    FLASHU                  ;
        RETSU                           ;
ERASE           ; FLASH ERASE function
        CALL    FLASHI                  ;
        LDA     @FSEQ55,AR3             ; Erase Chip command
        LDI     0aaH,R0                 ;
        STI     R0,*AR3                 ;
        LDA     @FSEQAA,AR3             ;
        LDI     055H,R0                 ;
        STI     R0,*AR3                 ;
        LDA     @FSEQ55,AR3             ;
        LDI     @FSEQER1,R0             ;
        STI     R0,*AR3                 ;
        LDA     @FSEQ55,AR3             ;
        LDI     0aaH,R0                 ;
        STI     R0,*AR3                 ;
        LDA     @FSEQAA,AR3             ;
        LDI     055H,R0                 ;
        STI     R0,*AR3                 ;
        LDA     @FSEQ55,AR3             ;
        LDI     @FSEQER2,R0             ;
        STI     R0,*AR3                 ;
ERASEL
        LDI     *AR3,R1                 ; Wait till ERASE complete
        TSTB    080H,R1                 ;
        BZ      ERASEL                  ;
        CALL    FLASHU                  ;
        RETSU                           ;
SERASE          ; FLASH SECTOR ERASE function
        CALL    FLASHI                  ;
        LDA     @FSEQ55,AR3             ; Erase Chip command
        LDI     0aaH,R0                 ;
        STI     R0,*AR3                 ;
        LDA     @FSEQAA,AR3             ;
        LDI     055H,R0                 ;
        STI     R0,*AR3                 ;
        LDA     @FSEQ55,AR3             ;
        LDI     @FSEQER1,R0             ;
        STI     R0,*AR3                 ;
        LDA     @FSEQ55,AR3             ;
        LDI     0aaH,R0                 ;
        STI     R0,*AR3                 ;
        LDA     @FSEQAA,AR3             ;
        LDI     055H,R0                 ;
        STI     R0,*AR3                 ;
        LDI     *+AR2(1),AR3            ; Get address from 0x201
        LDI     @FSEQER3,R0             ;
        STI     R0,*AR3                 ;
```

B.1 Boot EPROM Source Code Listing (continued)

```
SERASEL
        LDI     *AR3,R1                 ; Wait till ERASE complete
        TSTB    080H,R1                 ;
        BZ      SERASEL                 ;
        CALL    FLASHU                  ;
        RETSU                           ;
FLASHI
        LDA     @CTRL,AR0               ; Point to control register
        LDI     @FLASHINT,R0            ; Init local memory interface control
        STI     R0,*+AR0(4)             ;
        CALL    RFLASH                  ; Reset FLASH
        RETSU                           ;
FLASHU
        LDA     @CTRL,AR0               ; Point to control register
        LDI     @LOCALINT,R0            ; Init local memory interface control
        STI     R0,*+AR0(4)             ;
        CALL    RFLASH                  ; Reset FLASH
        RETSU                           ;
RFLASH
        LDA     @FSEQ55,AR3             ; Read/Reset command
        LDI     0aaH,R0                 ;
        STI     R0,*AR3                 ;
        LDA     @FSEQAA,AR3             ;
        LDI     055H,R0                 ;
        STI     R0,*AR3                 ;
        LDA     @FSEQ55,AR3             ;
        LDI     @FSEQRST,R0             ;
        STI     R0,*AR3                 ;
        LDA     @FBASE,AR3              ;
        LDI     *AR3,R0                 ;
        RETSU                           ;
READFLASH
; Enter with address in AR1 and data returned in R0
        CALL    FLASHI                  ;
        LDI     0,R0                    ;
        LDI     *AR1++,R1               ;
        AND     0FFh,R1                 ;
        OR      R1,R0                   ;
        LDI     *AR1++,R1               ;
        AND     0FFh,R1                 ;
        LSH     8,R1                    ;
        OR      R1,R0                   ;
        LDI     *AR1++,R1               ;
        AND     0FFh,R1                 ;
        LSH     16,R1                   ;
        OR      R1,R0                   ;
        LDI     *AR1++,R1               ;
        AND     0FFh,R1                 ;
        LSH     24,R1                   ;
        OR      R1,R0                   ;
        PUSH    R0                      ;
        CALL    FLASHU                  ;
        POP     R0                      ;
        RETSU                           ;
```

B.1 Boot EPROM Source Code Listing (continued)

```
WRITEFLASH
; Enter with address in AR1 and data in R0
        PUSH    R0                      ;
        LDA     @FSEQ55,AR3             ; Write command
        LDI     0aaH,R1                 ;
        STI     R1,*AR3                 ;
        LDA     @FSEQAA,AR3             ;
        LDI     055H,R1                 ;
        STI     R1,*AR3                 ;
        LDA     @FSEQ55,AR3             ;
        LDI     @FSEQWRT,R1             ;
        STI     R1,*AR3                 ;
        STI     R0,*AR1                 ; Store data at destination
        AND     080H,R0                 ;
FWRTL
        LDI     *AR1,R1                 ; Wait till write complete
        AND     080H,R1                 ;
        CMPI    R0,R1                   ;
        BNE     FWRTL                   ;
        POP     R0                      ;
        RETSU                           ;
CKBOOTF
        LDA     @FBOOTL,AR1             ;
        CALL    READFLASH               ;
        CMPI    @FLASHFG,R0             ;
        BNE     CKBOOTFX                ;
        LDA     @FBOOTJ,AR1             ;
        CALL    READFLASH               ;
        STI     R0,@FBOOTT              ;
        CALL    FLASHI                  ;
        LDA     @FBOOTT,AR1             ;
        B       FBOOTE                  ; Do boot
CKBOOTFX
        RETSU
CKBOOTG
        LDA     @GBOOTL,AR1             ;
        LDI     *AR1,R0                 ; Get boot flag
        CMPI    @FLASHFG,R0             ;
        BNE     CKBOOTGX                ; Dont boot from global
        LDI     *+AR1(1),R0             ; Get boot address
        STI     R0,@FBOOTT              ;
        LDA     @FBOOTT,AR1             ;
        B       FBOOTE                  ; Do boot
CKBOOTGX
        RETSU
```

B.1 Boot EPROM Source Code Listing (continued)

```
CKERASE
        LDI     0,R0                    ;
        STI     R0,*AR3                 ;
        LDI     *AR3,R1                 ;
        BNE     CKERASX                 ;
        LDI     5,R0                    ;
        STI     R0,*AR3                 ;
        LDI     *AR3,R1                 ;
        CMPI    R0,R1                   ;
        BNE     CKERASX                 ;
        LDI     0,R0                    ;
        STI     R0,*AR3                 ;
        LDI     *AR3,R1                 ;
        BNE     CKERASX                 ;
        CALL    ERASE                   ;
CKERASX
        RETSU                           ;
ISR2                                    ; INT2-
        RETI                            ;
ISR3                                    ; INT3-
        RETI                            ;
        .end
```