EVAL80960VH Evaluation Platform

Board Manual

December 1998

Order Number: 273194-003

Contents

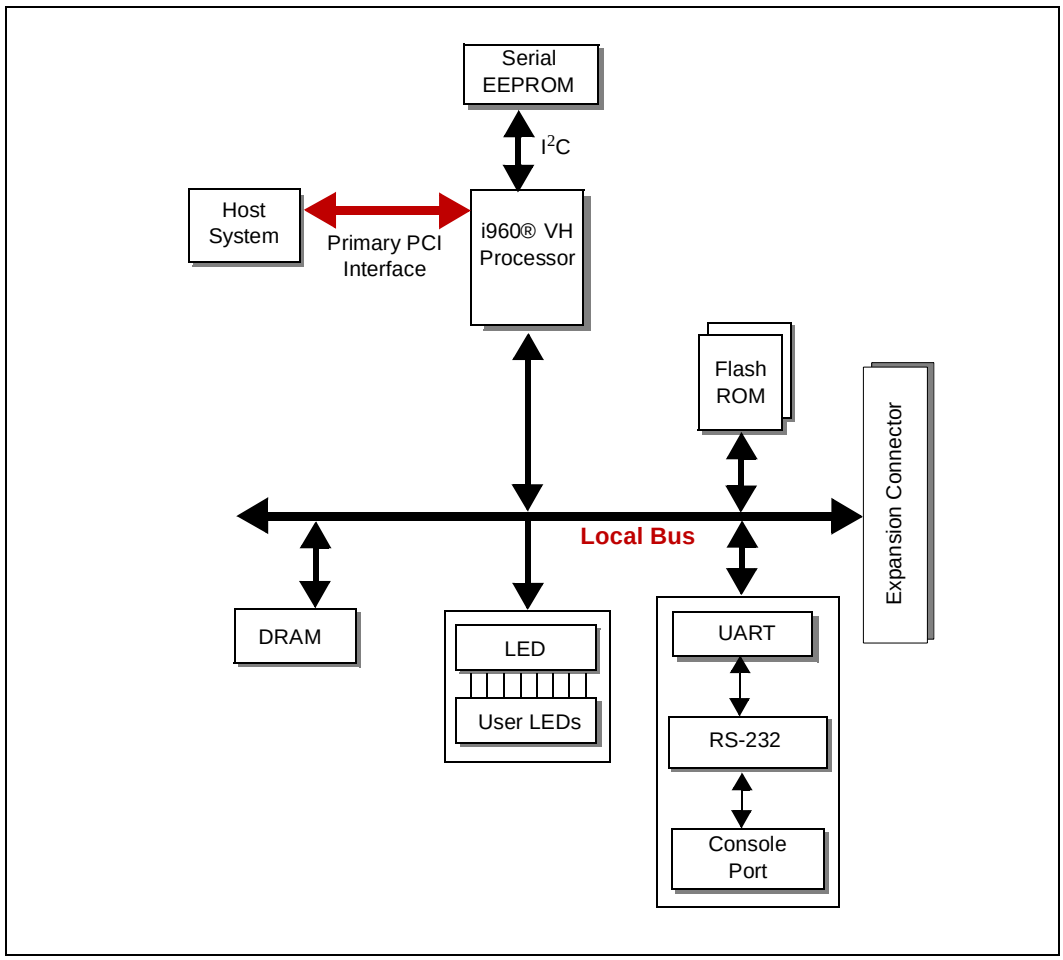
Figures

Tables

Introduction 1

This manual describes the EVAL80960VH evaluation platform for Intel's i960® VH embedded-PCI processor. The i960 VH processor ("80960VH") addresses the needs of I/O processor applications by enabling communications, through message passing and interrupt generation, between it and a host processor or external PCI device. The 80960VH combines an 80960JT core with a PCI bus interface, as well as a memory controller, DMA channels, an interrupt controller interface, and an I^2C Serial Bus. The EVAL80960VH evaluation platform is a half-length PCI adapter board. The board can be installed in any PCI host system that complies with the *PCI Local Bus Specification*, revision 2.1.

Figure 1-1. EVAL80960VH Platform Functional Block Diagram



1.1 EVAL80960VH Features

The i960® VH embedded-PCI processor serves as the main component of a high performance, PCI-based I/O subsystem. The features of the EVAL80960VH platform are enumerated below and shown in Figure 1-1 and Figure 3-1.

- i960® VH embedded-PCI processor
- PCI short-card form factor
- 32-bit primary PCI bus interface
- 4 Mbytes DRAM expandable to 32Mbytes using 72-pin, 5V SIMM socket supporting EDO DRAM organized x32 or x36
- 2 DMA channels on PCI bus
- I^2C Serial Bus
- IK byte serial EEPROM (24C08) (connected to I^2C bus for developer application use)
- Serial console port based on 16C550 UART
- Eight user-programmable LEDs
- 2 Indicator LEDs: processor has failed self-test, and processor is running
- Socketed flash ROM and on-board flash ROM devices
- Logic analyzer connectors for ROM bus and processor interface signals
- Processor local bus expansion header
- SMT dip switch for setting processor configuration signal defaults
- JTAG header (supports JTAG debugger interface)

1.2 CTOOLS Software Development Toolset

Intel's i960 processor software development toolset (CTOOLS) features advanced C/C++-language compilers for the i960 processor family. CTOOLS development toolset is available for Windows* 95/NT-based systems and a variety of UNIX workstation hosts. These products provide execution profiling and instruction scheduling optimizations. The tools also include an assembler, linker, symbolic debugger, and utilities designed for embedded processor software development.

1.2.1 CTOOLS and the MON960 Debug Monitor

The EVAL80960VH platform is equipped with Intel's mon960, an on-board software monitor that allows you to execute and debug programs written for i960 processors. The monitor provides program download, breakpoint, single step, memory display, and other useful functions for running and debugging a program.

The EVAL80960VH platform works with the source-level debuggers provided with CTOOLS, including gdb960 (command line version) and gdb960V (GUI version).

1.2.1.1 MON960 Host Communications

In a host system, mon960 allows you to communicate and download programs developed for the EVAL80960VH platform using the serial port, the PCI interface, or the JTAG interface. The EVAL80960VH platform supports two modes of serial connection: terminal emulation and Host Debugger Interface (HDI). The PCI and JTAG interfaces only support the HDI mode.

1.2.1.2 Terminal Emulation Mode

Terminal emulation software on your host system can communicate to mon960 on the EVAL80960VH platform via an RS-232 serial port. The EVAL80960VH platform supports port speeds from 300 to 115,200 bps. Downloading program code in this mode requires that the terminal emulation software support the XMODEM protocol.

Configure the serial port on the host system for 300-115,200 baud, 8 bits, one stop bit, no parity with XON/XOFF flow control. Six carriage returns must be sent to initiate this mode.

1.2.1.3 Host Debugger Interface Library (HDI) Mode

You may use a source-level debugger, such as Intel's gdb960 and gdb960V to establish serial or PCI, or JTAG communications with the EVAL80960VH platform. The mon960 Host Debugger Interface (HDI) provides a defined messaging layer between mon960 and the debugger. For more information on this interface, see the *MON960 Debug Monitor User's Guide* (Intel order number 484290).

HDI connection requests cannot be detected by mon960 if the user has already initiated a connection using a terminal emulator, or vice versa. Once one mode is initiated, the EVAL80960VH platform must be reset before a different mode can be selected.

1.3 About This Manual

A brief description of the contents of this manual follows.

Chapter 1, "Introduction"	Introduces the EVAL80960VH evaluation board features. This chapter also describes Intel's CTOOLS software development tools, and defines notational conventions and related documentation.
Chapter 2, "Getting Started"	Provides step-by-step instructions for installing the EVAL80960VH platform in a host system and downloading and executing an application program. This chapter also describes Intel's software development tools, the MON960 Debug Monitor, software installation, and hardware configuration.
Chapter 3, "Hardware Reference"	Describes the locations of connectors, switches and LEDs on the EVAL80960VH. Header pinouts and register descriptions are also provided in this chapter.
Chapter 4, "i960® VH Processor Overview"	Presents an overview of the capabilities of the i960® VH embedded-PCI processor and includes the CPU memory map.
Chapter 5, "MON960 Support for EVAL80960VH"	Describes a number of features added to MON960 to support application development on the i960® VH embedded-PCI processor.
Appendix A, "Bill of Materials"	Shows a complete parts list of the EVAL80960VH evaluation board.

Appendix B, “Schematics”	Complete set of schematics for the EVAL80960VH evaluation board.
Appendix C, “PLD Code”	Example PLD code used on EVAL80960VH evaluation board.
Appendix D, “Backplane 0015 Interface”	

1.4 Notational Conventions

The following notation conventions are consistent with other i960 VH processor documentation and general industry standards.

# and $\overline{\text{overbar}}$	In code examples the pound symbol (#) or an overbar is appended to a signal name to indicate that the signal is active at a low voltage.
Bold	Indicates user entry and/or commands. PLD signal names are in bold lowercase letters (for example, **h_off, h_on**).
Italics	Indicates a reference to related documents; also used to show emphasis.
`Courier font`	Indicates `code` examples and file directories and names.
Asterisks (*)	On non-Intel company and product names, a trailing asterisk indicates the item is a trademark or registered trademark. Such brands and names are the property of their respective owners.
UPPERCASE	In text, signal names are shown in uppercase. When several signals share a common name, each signal is represented by the signal name followed by a number; the group is represented by the signal name followed by a variable *(n)*. In code examples, signal names are shown in the case required by the software development tool in use.
Designations for hexadecimal and binary numbers	In text instead of using subscripted “base” designators (for example, FF_{16}) or leading “0x” (for example, 0xFF) hexadecimal numbers are represented by a string of hex digits followed by the letter *H*. A zero prefix is added to numbers that begin with *A* through *F*. (for example, *FF* is shown as *0FFH*.) In examples of actual code, “0x” is used. Decimal and binary numbers are represented by their customary notations. (for example, 255 is a decimal number and 1111 1111 is a binary number. In some cases, the letter *B* is added to binary numbers for clarity.)

1.5 Technical Support

Up-to-date product and technical information is available electronically from:

- Intel’s World-Wide Web (WWW) Location: http://www.intel.com
- EVAL80960VH product information: http://developer.intel.com/design/i960

For technical assistance, electronic mail (e-mail) provides the fastest route to reach engineers specializing in EVAL80960VH issues at the email address: 960tools@intel.com. Posting messages on the Embedded Microprocessor Forum at http://support.intel.com/newsgroups/ is also a direct route for EVAL80960VH technical assistance. See Section 1.5.1.

Within the United States and Canada you may contact the Intel Technical Support Hotline. See Section 1.5.2 for a list of customer support sources for the US and other geographical areas.

1.5.1 Intel Customer Electronic Mail Support

For direct support from engineers specializing in i960® microprocessor issues, send email in english to **960tools@intel.com**.

Questions and other messages may be posted to the Embedded Microprocessor Forum at **http://support.intel.com/newsgroups/**.

1.5.2 Intel Customer Literature and Telephone Support

Contact Intel Corporation for technical assistance for the EVAL80960VH evaluation platform.

Country	Literature	Customer Support Number
United States	800-548-4725	800-628-8686
Canada	800-468-8118 or 303-297-7763	800-628-8686
Europe	Contact local distributor	Contact local distributor
Australia	Contact local distributor	Contact local distributor
Israel	Contact local distributor	Contact local distributor
Japan	Contact local distributor	Contact local distributor

1.5.3 Related Information

To order printed manuals from Intel, contact your local sales representative or Intel Literature Sales in the US at 1-800-548-4725.

Product	Document Name	Company/Order #
All	Developers' Insight CD-ROM	Intel # 273000
80960VH	i960® VH Processor Developer's Manual	Intel # 273173
80960VH	i960® VH Processor Data Sheet	Intel # 273179
MON960	MON960 Debug Monitor User's Guide	Intel #484290
	PCI Local Bus Specification Revision 2.1	PCI Special Interest Group 1-800-433-5177
80960JX	i960® Jx I/O Microprocessor User's Manual	Intel # 272483

Contact Cyclone Microsystems for additional information about their products and literature:

Cyclone Microsystems 25 Science Park New Haven CT 06511	Phone:203-786-5536
	FAX:203-786-5025
	e-mail:info@cyclone.com WWW:http://www.cyclone.com

Getting Started 2

This chapter contains instructions for installing the EVAL80960VH platform in a host system, and downloading and executing an application program using Intel's CTOOLS software development toolsets.

2.1 Pre-installation Considerations

This section provides a general overview of the components required to develop and execute a program on the EVAL80960VH platform. The *MON960 Debug Monitor User's Guide* (Intel order number 484290) describes several of these components, including mon960 commands, the Host Debugger Interface Library (HDIL), and the mondb.exe utility.

2.1.1 Software Development Tools

A number of software development tools are available for the i960® processor family[1]. The installation instructions presented in this chapter were verified using CTOOLS - Intel's i960 processor software development tools. CTOOLS is a complete C/C++ - language software development toolset for developing embedded applications to run on i960 processors. It contains a C/C++ compiler, the gcc960 and ic960 compiler driver programs, an assembler, linker, symbolic debugger, runtime libraries, a collection of software development tools and utilities, and printed and on-line documentation. The MON960 Debug Monitor User's Guide fully describes the components of mon960, including mon960 commands, the Host Debugger Interface Library (HDIL), and the mondb.exe utility. If you are using mon960 and the CTOOLS toolset, then refer to Section 2.2.1, "Installing Software Development Tools" on page 2-1. If you are using other software development tools, then read through this example to gain a general understanding of how to use tools with this board.

2.2 Software Installation

2.2.1 Installing Software Development Tools

Install your development software as described in its manuals. All references in this manual to CTOOLS assume that the default directories were selected during installation. If this is not the case, then substitute the appropriate path for the default path wherever file locations are referenced in this manual.

1. Refer to Intel's http://developer.intel.com web page catalog for a complete list of i960 processor software development and debug tools.

2.3 Hardware installation

Follow these instructions to get your new EVAL80960VH platform running. Be sure all items on the checklist provided in your kit's customer letter were provided with your EVAL80960VH evaluation platform kit.

Warning: Static charges can severely damage the EVAL80960VH platforms. Be sure you are properly grounded before removing the EVAL80960VH evaluation board from the anti-static bag.

2.3.1 Installing the EVAL80960VH Platforms in the Host System

If you are installing the EVAL80960VH platform for the first time, then visually inspect the board for any damage that may have occurred during shipping. If there are visible defects, then return the board for replacement. Follow the host system manufacturer's instructions for installing a PCI adapter. The EVAL80960VH platform is a half-length PCI adapter and requires a PCI slot that is free of obstructions.

2.3.2 Verify EVAL80960VH Platform is Functional

These instructions assume that you have already installed the EVAL80960VH platform in the host system as described in Section 2.3.1.

1. To connect the serial port for communicating with and downloading to the EVAL80960VH platform, connect the RS-232 cable (provided with the EVAL80960VH kit) from a free serial port on a host system to the phone jack-style connector on the EVAL80960VH platform.
2. After power-up, the red FAIL LED should turn off, indicating that the processor has passed its self-test.
3. Press <return> six times on a terminal connected to the EVAL80960VH platform to bring up the mon960 prompt. mon960 automatically adjusts its baud rate to match that of the terminal at start-up.

2.4 Creating and Downloading Executable Files

To download code to the EVAL80960VH platform, your compiler should produce an ELF-format object file. Consult the CTOOLS documentation for information regarding compiling, linking, and downloading applications.

During a download, mon960 checks the link address stored in the ELF file, and stores the file at that location on the EVAL80960VH platform. If the executable file is linked to an invalid address, then mon960 will abort the download.

2.4.1 Sample Download and Execution Using gdb960

This example shows you how to use gdb960 to download and execute a file named myapp via the serial port.

- Invoke gdb960. For example, from a Windows* 95/NT or UNIX command prompt, issue the command:

```
gdb960 -r com2 myapp
```

This command establishes communication and downloads the file myapp.

- To execute the program, enter the command from the gdb960 command prompt:
  ```
  (gdb960) run
  ```

More information on the gdb960 commands mentioned in this section can be found in the *GDB960 User's Manual*.

Hardware Reference 3

This section describes the location and function of physical connectors, switches and LEDs. Refer to Figure 3-1, the physical diagram of the EVAL80960VH platform, for the locations of components discussed in this section.

3.1 Connectors, Switches and LEDs

Figure 3-1 shows the physical locations of the major components on the EVAL80960VH platform. Table 3-1 lists the functions of these components. For a complete list of components on the EVAL80960VH platform, refer to Appendix A, "Bill of Materials".

Figure 3-1. EVAL80960VH Platform Physical Diagram

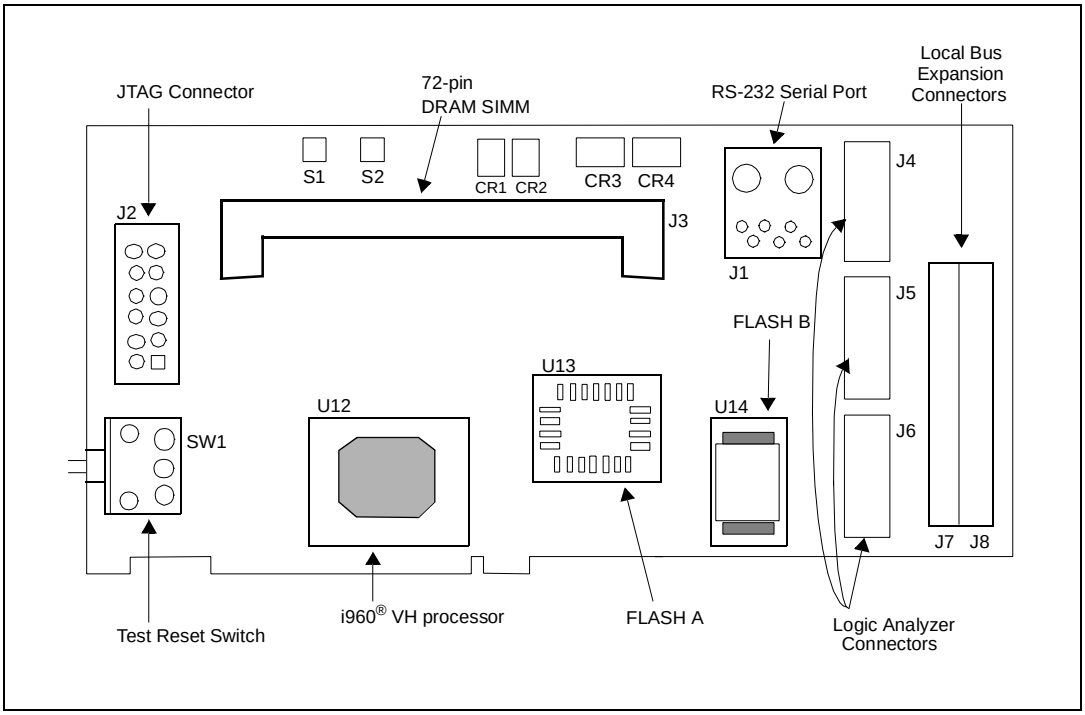


Table 3-1. EVAL80960VH Connectors and LEDS (Sheet 1 of 2)

Item	Description
J9, J10	PCI bus card edge connector
J2	JTAG connector
J1	Serial port connector

Table 3-1. EVAL80960VH Connectors and LEDS (Sheet 2 of 2)

Item	Description
J3	72-pin DRAM SIMM
J7, J8	Expansion headers
J4, J5, J6	Logic Analyzer connectors (Mictor)
CR3, CR4	Eight user LEDs (Figure 3-3)
CR2	Self-test fail LED
CR1	DEN#, indicates data transfer cycles
S1	DIP switch (Table 3-10.)
SW1	Back panel pushbutton reset
S2	DIP switch (Table 3-5.)

3.2 Power Requirements

The EVAL80960VH platform draws power from the PCI bus. The power requirements of the EVAL80960VH platform are shown in Table 3-2.

Table 3-2. EVAL80960VH Platform Power Requirements (Clock Mode DX4)

Voltage	Typical Current	Maximum Current
+3.3 V	0 V*	0 V*
+5 V	.985 A	1.23 A
+12 V	50 mA	50 mA
-12 V	10 mA	10 mA

NOTE: * +3.3V for the 80960VH created on board from +5V

3.3 DRAM

The EVAL80960VH platform is equipped with 4 Mbytes of DRAM. This SIMM module can be upgraded to an 8, 16, or 32 Mbyte module.

3.3.1 DRAM Performance

The EVAL80960VH platform uses extended data out (EDO) DRAM to achieve a zero-wait-state burst at 33 MHz. The memory runs with two wait states in the first cycle of a read burst and one wait state during a write burst. There are no wait states during the burst. No additional recovery cycles are required beyond the intrinsic i960 JT core recovery cycle. Table 3-3 shows the performance numbers for the EVAL80960VH platform.

Note: Bandwidth is sustained bandwidth, not peak.

Table 3-3. DRAM Performance

Cycle Type	Table Clocks	Walt States	Performance Bandwidth
Read Single	4	2	27 Mbytes/sec
Read Burst	4-1-1-1	2-0-0-0	66 Mbytes/sec
Write Single	3	1	33 Mbytes/sec
Write Burst	3-1-1-1	1-0-0-0	76 Mbytes/sec

3.3.2 Upgrading DRAM

The EVAL80960VH is equipped with 4 Mbytes of DRAM in the SIMM socket. It may be expanded by changing the module from 4 up to 32 Mbytes of DRAM. The various memory combinations are shown in Table 3-4. Only 72 pin EDO modules rated at 60 ns should be used on the platform. Either x32 or x36 devices may be used.

Table 3-4. DRAM Configurations

Total Memory	SIMM Module Type
4 Mbyte	1M x 32 or 1M x 36 (4 Mbytes)
8 Mbytes	2M x 32 or 2M x 36
16 Mbytes	4M x 32 or 4M x 36
32 Mbyte	8M x 32 or 8M x 36

3.4 ROM and Flash ROM

A 32-pin PLCC ROM socket is included on the EVAL80960VH platform. The ROM socket at U13 is populated with a 28F020 Flash ROM containing mon960. The ROM at U14 is a 28F016S5 Flash ROM and may be used to store user applications. mon960 includes features to erase and program the 28F016S5 Flash ROM and download code directly into Flash ROM. The Flash ROM at U13 cannot be programmed on-board. It must be removed from the board and programmed using a Flash programmer. Table 3-5 shows the switch settings at switch S2 to select which ROM to boot from.

Table 3-5. ROM Select

ROMSWAP (S2-1)	ROMDISABLE (S2-2)	ACTION
OFF	OFF	Boot from U14 SMT ROM
ON	OFF	Boot from U13 PLCC ROM or ROM emulator inserted into PLCC socket
OFF	ON	Boot from MICTOR connector populated with a ROM emulator
ON	ON	Boot from MICTOR connector populated with a ROM emulator

3.5 Serial Port

The serial port on the EVAL80960VH platform, based on a 16C550 UART, is capable of operation from 300 to 115,200 bps. The port is connected to a phone jack-style plug on the EVAL80960VH platform. The DB25 to RJ-45 cable included with the EVAL80960VH can be used to connect the console port to any standard RS-232 port on the host system.

The UART on the EVAL80960VH platform is clocked with a 1.843 MHz clock and may be programmed to use this clock with its internal baud rate counters. The UART register addresses are shown in Table 3-6. Refer to the 16C550 device data book for a detailed description of the registers and device operation. Note that some UART addresses refer to different registers depending on whether a read or a write is being performed.

Table 3-6. UART Register Addresses

Address	Read Register	Write Register
E000 0000H	Receive Holding Register	Transmit Holding Register
E000 0004H	Unused	Interrupt Enable Register
E000 0008H	Interrupt Status Register	FIFO Control Register
E000 000CH	Unused	Line Control Register
E000 0010H	Unused	Modem Control Register
E000 0014H	Line Status Register	Unused
E000 0018H	Modem Status Register	Unused
E000 001CH	Scratchpad Register	Scratchpad Register

3.6 Logic Analyzer Headers

There are three logic analyzer connectors on the EVAL80960VH platform. The connectors are Mictor type, AMP part # 767054-1. Hewlet-Packard and Tektronix manufacture and sell interfaces to these connectors. The logic analyzer connectors allow for interfacing to the ROM bus along with processor interface signals. Table 3-7 shows the connectors and the pin assignments for each.

Table 3-7. Logic Analyzer Header Definitions (Sheet 1 of 2)

PIN	TEK.	HP	CONN A	CONN B	CONN C
			J6	J4	J5
3	CLK	CLK	CLK_MIC2	CLK_MIC1	
4	3:7	D15	ROMA15	AD15	LRST#
5	3:6	D14	ROMA14	AD14	LOCK#
6	3:5	D13	MA11	AD13	LRDYRCV#
7	3:4	D12	MA10	AD12	RDYRCV#
8	3:3	D11	MA9	AD11	BLAST#
9	3:2	D10	MA8	AD10	DEN#
10	3:1	D9	MA7	AD9	DT/R
11	3:0	D8	MA6	AD8	W_RD#

Table 3-7. Logic Analyzer Header Definitions (Sheet 2 of 2)

PIN	TEK.	HP	CONN A	CONN B	CONN C
			J6	J4	J5
12	2:7	D7	MA5	AD7	D/C
13	2:6	D6	MA4	AD6	WIDTH1
14	2:5	D5	MA3	AD5	WIDTH0
15	2:4	D4	MA2	AD4	ADS#
16	2:3	D3	MA1	AD3	BE3#
17	2:2	D2	MA0	AD2	BE1#
18	2:1	D1	BE1#	AD1	BE1#
19	2:0	D0	BE#2	AD0	BE0#
20	0:0	D0	ROMA16	AD16	XINT0#
21	0:1	D1	ROMA17	AD17	XINT1#
22	0:2	D2	ROMA18	AD18	XINT2#
23	0:3	D3	ROMA19	AD19	XINT3#
24	0:4	D4	ROMA20	AD20	XINT4#
25	0:5	D5	AD0	AD21	XINT5#
26	0:6	D6	AD1	AD22	XINT6#
27	0:7	D7	AD2	AD23	XINT7#
28	1:0	D8	AD3	AD24	NMI#
29	1:1	D9	AD4	AD25	HOLD
30	1:2	D10	AD5	AD26	HOLDA
31	1:3	D11	AD6	AD27	DREQ#
32	1:4	D12	AD7	AD28	DACK#
33	1:5	D13	CE0#	AD29	WAIT#
34	1:6	D14	MWE0#	AD30	
35	1:7	D15	W_RD#	AD31	
36	CLK	CLK	LRST	ALE	

3.7 Expansion Connectors

Table 3-8 details the pinout of the expansion connector. The expansion connectors are located at the right end of the board, away from the end panel. This allows users to use the rest of the slot for custom I/O modules. The expansion connectors are standard.025” socket strips arranged 2 x 25 and numbered as shown Figure 3-2.

Figure 3-2. Expansion Connector Pin Numbering

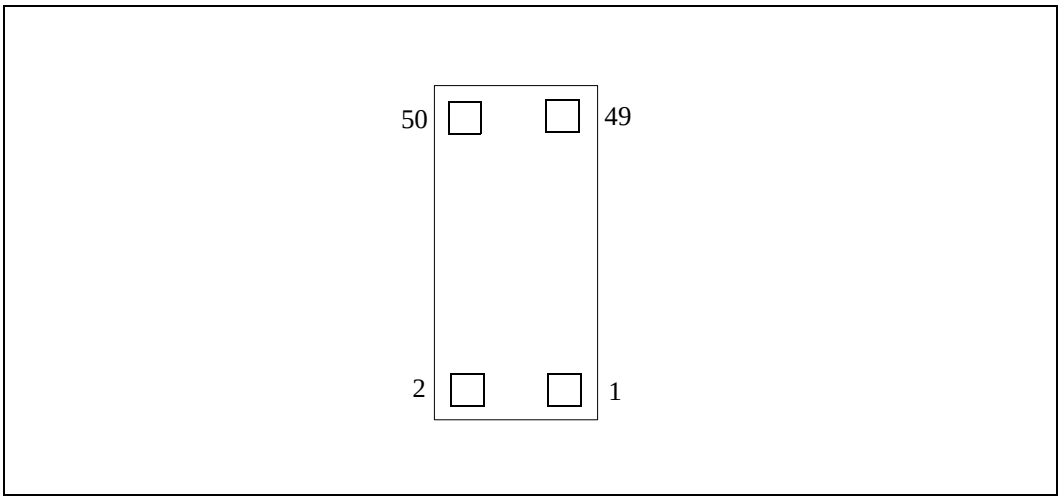


Table 3-8. Expansion Connector Definitions (Sheet 1 of 2)

PIN	CONN A	CONN B
	J7	J8
1	BE0#	AD0
2	BE1#	AD1
3	GND	AD2
4	BE3#	GND
5	BE2#	AD3
6	GND	AD4
7	ADS#	AD5
8	WIDTH0	GND
9	GND	AD6
10	D/C	AD7
11	WIDTH1	AD8
12	GND	GND
13	W_RD#	AD9
14	DT/R	AD10
15	GND	AD11
16	BLAST#	GND

Table 3-8. Expansion Connector Definitions (Sheet 2 of 2)

PIN	CONN A	CONN B
	J7	J8
17	DEN#	AD12
18	GND	AD13
19	RDYRCV#	AD14
20	LRDYRCV#	GND
21	GND	AD15
22	HOLD	AD16
23	LOCK#	AD17
24	GND	GND
25	HOLDA	AD18
26	XINT0#	AD19
27	GND	AD20
28	XINT2#	GND
29	XINT1#	AD21
30	GND	AD22
31	XINT3#	AD23
32	XINT4#	GND
33	GND	AD24
34	XINT6#	AD25
35	XINT5#	AD26
36	GND	GND
37	XINT7#	AD27
38	NMI#	AD28
39	GND	AD29
40	DACK#	GND
41	DREQ3	AD30
42	GND	AD31
43	WAIT#	GND
44	LRST#	GND
45	+5V	ALE
46	+5V	+5V
47	GND	GND
48	GND	GND
49	GND	+5V
50	GND	CLK_EXP

3.8 JTAG Header

The JTAG header allows debugging hardware to be quickly and easily connected to the EVAL80960VH evaluation platform.

The JTAG header is a 16 pin header. A 3M connector (Part Number 2516-6002UG) is required to connect to this header. The pinout for the JTAG header is shown in Table 3-9. The header and connector are keyed using a tab on the connector and a slot on the header to ensure proper installation.

Each signal in the JTAG header is paired with its own ground connection to avoid the noise problems associated with long ribbon cables. Signal descriptions are found in the *i960® VH Processor Developer's Manual*, and *i960® VH Processor Data Sheet.*

Table 3-9. JTAG Header Pinout, Connector J2

PIN	SIGNAL	INPUT/OUTPUT TO 80960VH	PIN	SIGNAL
1	TRST#	IN	2	GND
3	TDI	IN	4	GND
5	TDO	OUT	6	GND
7	TMS	IN	8	GND
9	TCK	IN	10	GND
11	LCDINIT#	IN	12	GND
13	I_RST#	OUT	14	GND
15	PWRVLD	OUT	16	GND

3.9 Switch Settings

Options for switch S1 settings are listed in Table 3-10. Refer to i960® VH Processor Data Sheet (order number 273179) for pin descriptions of these processor signals.

Table 3-10. Switch S1 Settings (Sheet 1 of 2)

Position	Name	Description	Default
S1-1	RETRY (Active only when RST_MODE# is low.)	Determines if the Primary PCI interface will be disabled. ON = allows Primary PCI configuration cycles to occur. OFF = retries all Primary PCI configuration cycles.	OFF
S1-2	RST_MODE#	Determines if the processor is to be held in reset. ON = hold in reset. OFF = allows processor initialization (processor forces RETRY low).	OFF

Table 3-10. Switch S1 Settings (Sheet 2 of 2)

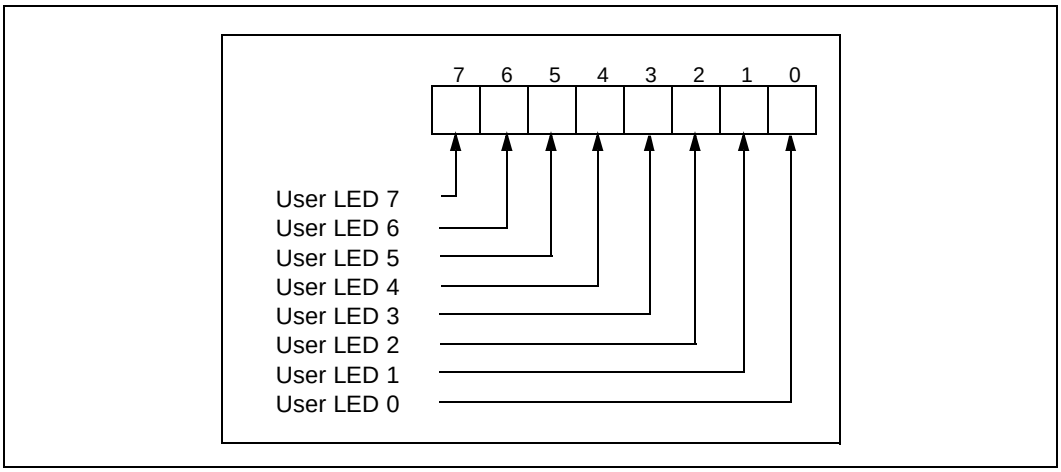
Position	Name	Description	Default
S1-3*	CLKMODE0#	Clock MODE are used to select the mode of operation in terms of 80960 local bus/PCI bus vs. the internal 80960 processor.	OFF
S1-4*	CLKMODE1#	Clock MODE are used to select the mode of operation in terms of 80960 local bus/PCI bus vs. the internal 80960 processor.	OFF
* CLOCK MODES 00 - DX4 mode 01 - DX2 mode 10 - DX mode 11 - select speed using the clock mode bits in the CSR register			

3.10 User LEDs

The EVAL80960VH platform has a bank of eight user-programmable LEDs, located on the upper edge of the adapter board. These LEDs are controlled by a write-only register and used as a debugging aid during development. Software can control the state of the user LEDs by writing to the LED Register, located at E004 0000H. Each of the eight bits of this register corresponds to one of the user LEDs. Clearing a bit in the LED Register by writing a “0” to it turns the corresponding LED on, while setting a bit by writing a “1” to it turns the corresponding LED off. Resetting the EVAL80960VH platform results in clearing the register and turning all the LEDs on. The LED Register bitmap is shown in Figure 3-3.

The user LEDs are numbered in descending order from left to right, with LED7 being on the left when looking at the component side of the adapter.

Figure 3-3. LED Register Bitmap



3.10.1 User LEDs During Initialization

MON960 indicates the progress of its hardware initialization on the user LEDs. In the event that initialization should fail for some reason, the number of the lit LED can be used to determine the cause of the failure. Table 3-11 lists the tests that correspond to each lit LED.

Table 3-11. Startup LEDs MON960

LEDS	Tests
LED 0	UART walking ones test passed.
LED 1	DRAM walking ones test passed.
LED 2	DRAM multiword test passed.
LED 3	Memory controller initialized.
LED 4	Hardware initialization started.
LED 5	Flash ROM initialized.
LED 6	ATU initialized.
LED 7	UART internal loopback test passed.

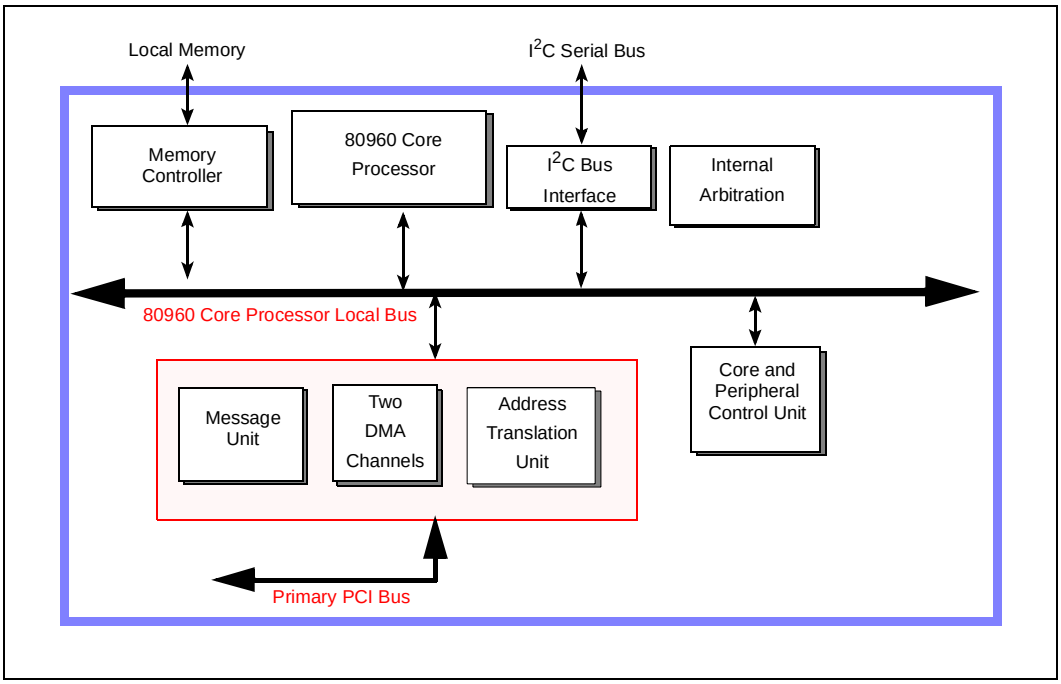
3.11 Serial EEPROM (I^2C)

A 1 Kbyte serial EEPROM is connected to the I^2C bus on the EVAL80960VH platform at address 0. Intel does not define the contents of this device, so it is available for use by the developer. The EEPROM is read and written using the I^2C bus; consult the *i960® VH Processor Developer's Manual* and the *24C08 Serial EEPROM Data Sheet* for more information.

i960® VH Processor Overview 4

This chapter describes the features and operation of the i960 VH embedded-PCI processor on the EVAL80960VH platform. For more detail, refer to the *i960® VH Processor Developer's Manual*.

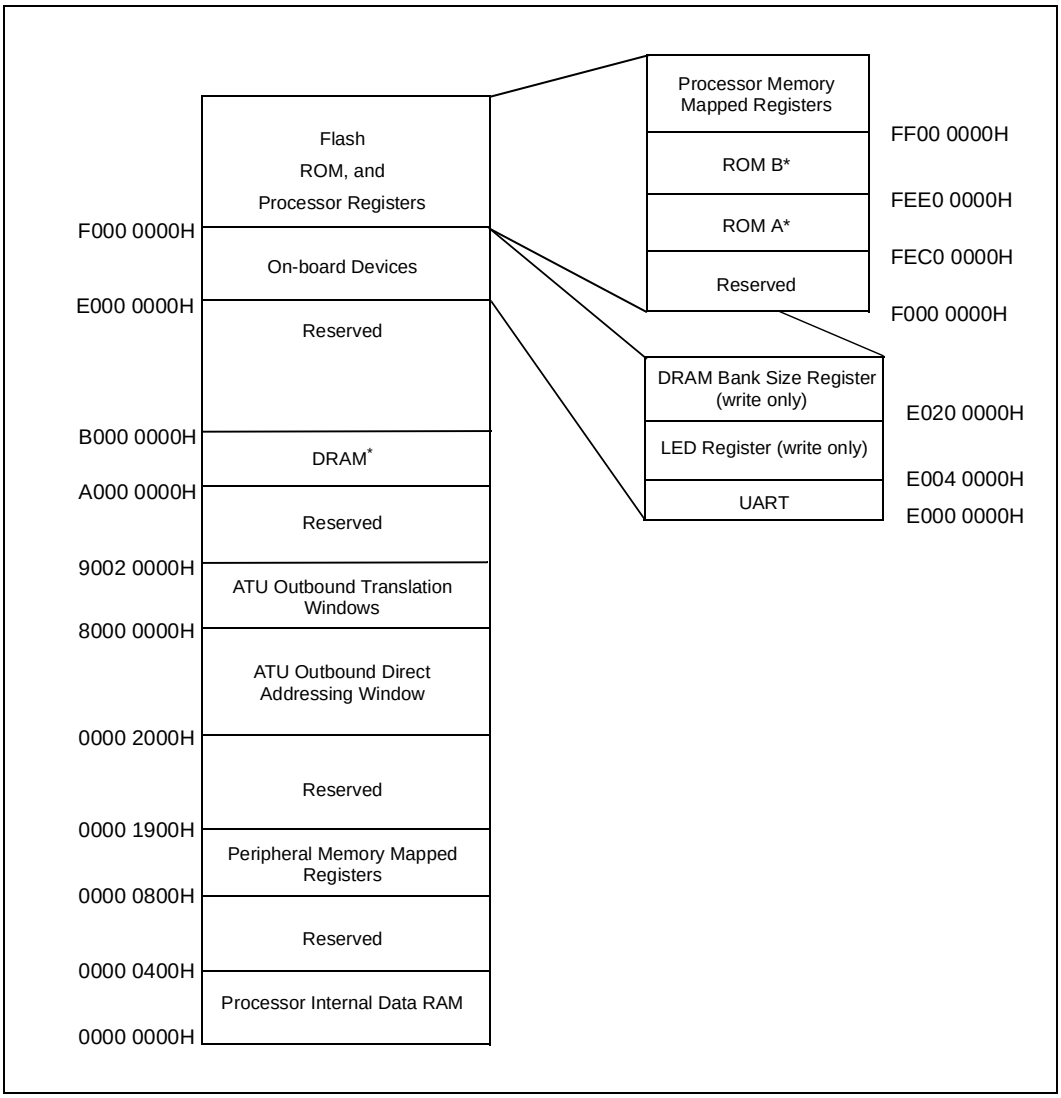
Figure 4-1. i960® VH Processor Block Diagram



4.1 CPU Memory Map

The memory map for the EVAL80960VH platform is shown in Figure 4-2. All addresses below 9002 0000H on the EVAL80960VH platform are reserved for various functions of the i960 VH embedded-PCI processor, as shown on the memory map. Documentation for these areas, as well as the processor memory mapped registers at FF00 0000H and the IBR, can be found in the *i960® VH Processor Developer's Manual*.

Figure 4-2. EVAL80960VH Platform Memory Map



Note: With the DRAM base register set to A000 0000H the mapping for the DRAM, based on DRAM size, is shown in Table 4-1.

Table 4-1. DRAM Mapping

DRAM Size	Address Range
4 Mbytes	A3C0 0000H - A3FF FFFFH
8 Mbytes	A380 0000H - A3FF FFFFH
16 Mbytes	A300 0000H - A3FF FFFFH
32 Mbytes	A200 0000H - A3FF FFFFH

4.1.1 Booting Out of DRAM on the EVAL80960VH Platform

The EVAL80960VH platform provides for the ability to map DRAM to allow booting out of DRAM. This can be accomplished with all DRAM configurations except with a 16 Mbyte module. The IBR is located on the i960 VH processor beginning at location FEFF FF30H. Table 4-2 shows what address the DRAM Base Address Register (DBAR 1520H) should be set to for a given memory module size. Note that addresses above FEFF FF5FH are reserved. If a 16 Mbyte module is used, the base address would be FC00 0000H and the range would be from FF00 0000H to FFFF FFFFH and the IBR would not be contained within the DRAM space. DRAM must be on an address boundary equal to the total size of 4 DRAM banks, in these cases 4M x 4 or 16M x 4.

Table 4-2. DBAR Settings for Booting out of DRAM

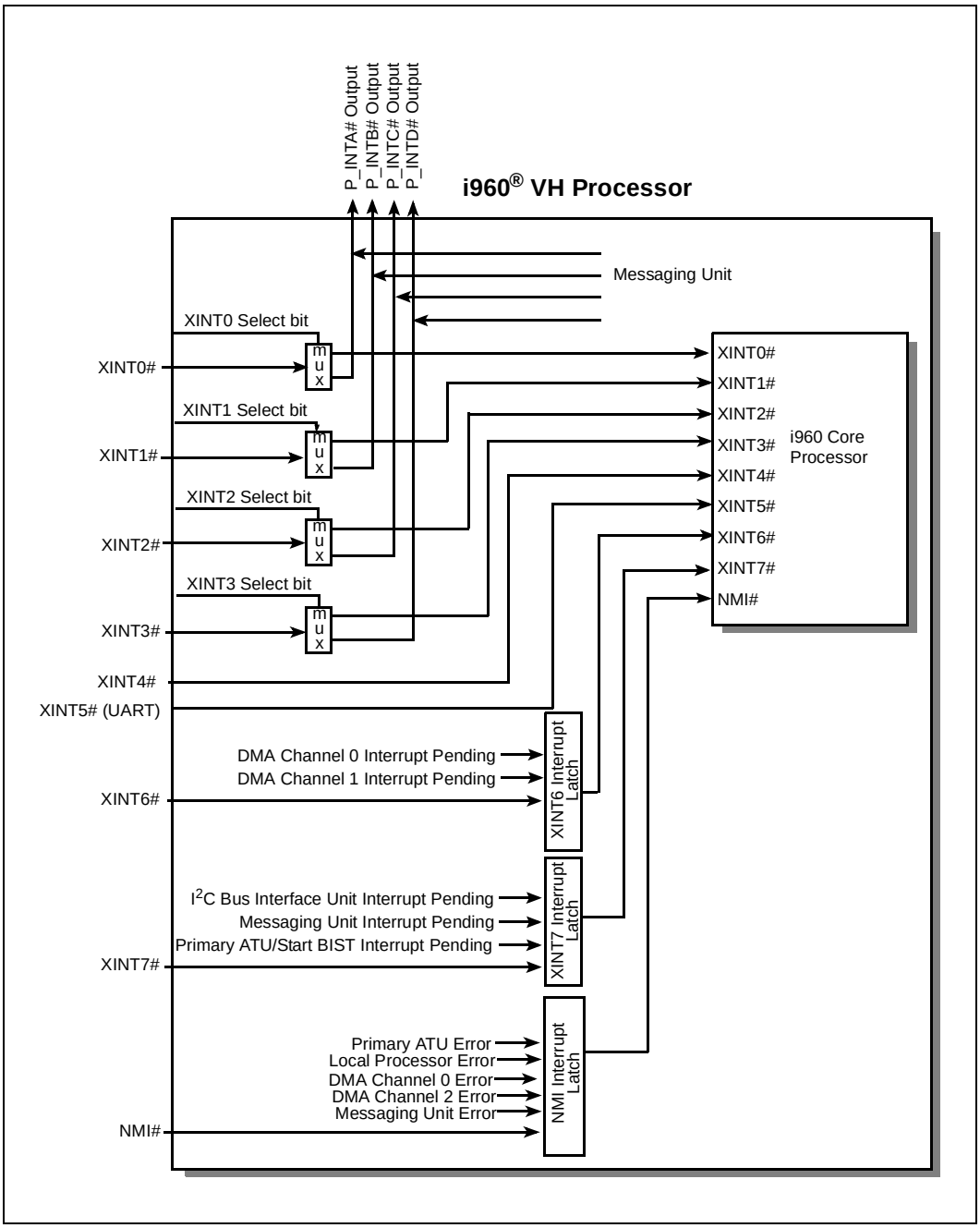
Memory Size	Base Address	Address Range
4 Mbytes	FE00 0000H	FEC0 0000H - FEFF FFFFH
8 Mbytes	FE00 0000H	FE80 0000H - FEFF FFFFH
32 Mbytes	FC00 0000H	FE00 0000H - FFFF FFFFH

4.2 Local Interrupts

The i960 VH embedded-PCI processor is built around an 80960JT core, which has seven external interrupt lines designated XINT0# through XINT7# and NMI#. In the 80960VH processor, these interrupt lines are not directly connected to external interrupts, but pass through a layer of internal interrupt routing logic. Figure 4-3 shows the interrupt connections on the 80960VH processor.

On the EVAL80960VH platform, XINT5# is connected to the 16C550 UART. XINT6#, XINT7#, and NMI# can receive interrupts from internal sources or from external sources. Since all of these interrupts accept signals from multiple sources, a status register is provided for each of them to allow service routines to identify the source of the interrupt. Each of the possible interrupt sources is assigned a bit position in the status register. The interrupt sources for these lines are shown in Figure 4-3. On the EVAL80960VH platform, the NMI# interrupt is not connected to any external interrupt source and receives interrupts only from the internal devices on the i960 VH embedded-PCI processor. Note that error indications are received on NMI#.

Figure 4-3. Interrupt Controller Connections



4.3 CPU Counter/Timers

The i960 VH embedded-PCI processor is equipped with two 32 bit on-chip counter/timers which are clocked with the 80960VH processor clock signal. The 80960VH processor receives its clock from the primary PCI interface clock, generated by the motherboard. Most motherboards generate a 33 MHz clock signal, although the PCI specification only requires a clock frequency between 0 and 33 MHz. The timers can be programmed for single-shot or continuous mode, and can generate interrupts to the processor when the countdown expires.

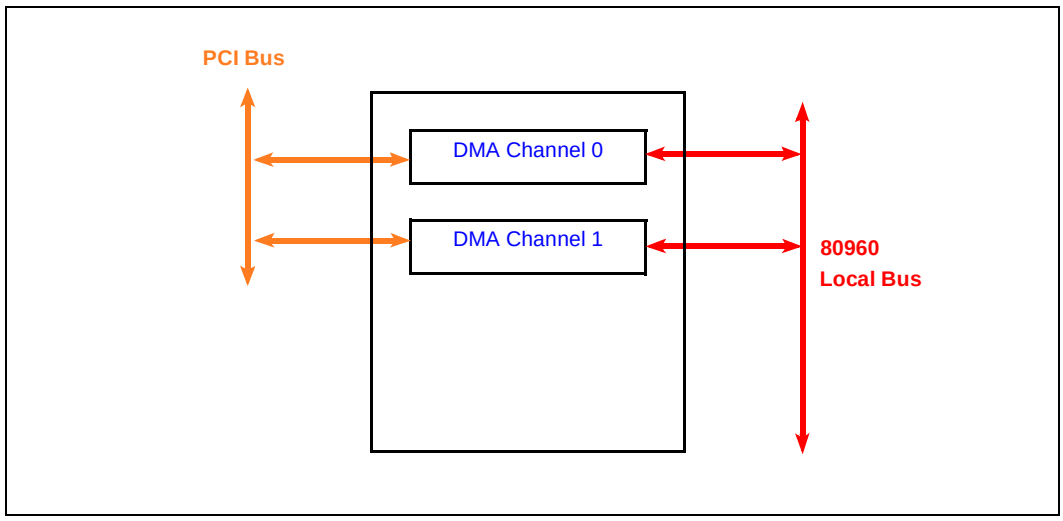
4.4 PCI Interface

The PCI interface on the EVAL80960VH platform provides the i960 VH embedded-PCI processor with a connection to the PCI bus on the host system.

4.5 DMA Channels

The i960 VH embedded-PCI processor features two independent DMA channels, both of which operate on the PCI interface. Both of the DMA channels connect to the 80960VH processor's local bus and can be used to transfer data from PCI devices to memory on the EVAL80960VH platform. Support for chaining, and scatter/gather is built into both channels. The DMA can address the entire 2^{64} bytes of address space on the PCI bus and 2^{32} bytes of address space on the internal bus.

Figure 4-4. i960® VH Processor DMA Controller



MON960 Support for EVAL80960VH 5

This chapter discusses a number of additions that have been made to MON960 to support the EVAL80960VH. For complete documentation on the operation of mon960, see the *MON960 Debug Monitor User's Guide*. The EVAL80960VH evaluation platform ships with mon960 in flash firmware. See Chapter 3, "Hardware Reference" for more information on updating the onboard Flash. See Chapter 1, "Introduction" for a description of CTOOLS.

5.1 MON960 Components

The remaining sections of this chapter assume that mon960 is installed in the onboard Flash. The EVAL80960VH mon960 debug monitor consists of four main components:

- Initialization firmware
- mon960 kernel
- mon960 extensions
- Diagnostics/example code

These four components together are referred to as MON960.

5.1.1 mon960 Initialization

At initialization, mon960 puts the EVAL80960VH platform into a known, functional state that allows the host processor to perform PCI initialization. Once in this state, the mon960 kernel and the mon960 extensions can load and execute correctly. Initialization is performed after a RESET condition. mon960 initialization encompasses all major portions of the i960® VH embedded-PCI processor and EVAL80960VH platform including 80960JT core initialization, Memory Controller initialization, memory initialization, and PCI Address Translation Unit (ATU) initialization.

The EVAL80960VH platform is designed to use the Configuration Mode of the 80960VH processor. Configuration Mode allows the 80960JT core to initialize and control the initialization process before the PCI host configures the 80960VH processor. By utilizing Configuration Mode, the user is given the ability to initialize the PCI configuration registers to values other than the default power-up values. Configuration Mode gives the user maximum flexibility to customize the way in which the 80960VH processor and EVAL80960VH platform appear to the PCI host configuration software.

5.1.2 80960 JT Core Initialization

The 80960JT core begins the initialization process by reading its Initial Memory Image (IMI) from a fixed address in the boot ROM (FEFF FF30H in the i960 core address space). The IMI includes the Initialization Boot Record (IBR), the Process Control Block (PRCB), and several system data structures. The IBR provides initial configuration information for the core and integrated peripherals, pointers to the system data structures and the first instruction to be executed after processor initialization, and checksum words that the processor uses in its self-test routine. In addition to the IBR and PRCB, the required data structures are the:

- System Procedure
- Control Table
- Interrupt Table
- Fault Table
- User Stack (application dependent)
- Supervisor Stack
- Interrupt Stack

5.1.3 Memory Controller Initialization

Since the i960® VH embedded-PCI processor Memory Controller is integral to the design and operation of the EVAL80960VH platform, the operational parameters for Bank 0 and Bank 1 are established immediately after processor core initialization. Memory Bank 0 is associated with the ROM on the EVAL80960VH platform. Memory Bank 1 is associated with the UART and the LED Control Register. Parameters such as Bank Base Address, Read Wait States, and Write Wait States must be established to ensure the proper operation of the EVAL80960VH platform. The Memory Controller is initialized so as to be consistent with the EVAL80960VH platform memory map shown in Figure 4-2.

5.1.4 PCI Interface Initialization

On the PCI bus, the ATU provides access capability between the PCI bus and the local i960 processor bus.

The platform can be initialized into one of three modes. Modes 0, 1, and 2 are described below.

Table 5-1. Initialization Modes

RST_MODE#/ S1-1	RETRY/ S1-1	Initialization Mode	PCI Interface	i960 Core Processor
0/ON	0/ON	Mode 0	Accepts Transactions	Held in Reset
1/OFF	0/ON	Mode 1	Accepts Transactions	Initializes
1/OFF	1/OFF	Mode 2 (Default)	Retries All Configuration Transactions	Initializes

5.1.4.1 Mode 0

Mode 0 allows a host processor to configure the i960® VH embedded-PCI processor while the i960 core processor is held in rest.

The host processor can allocate PCI address space, and assign IRQ numbers. The Memory Controller and ATU can be initialized by the host processor. Program code for the i960 core processor may be downloaded into local memory by the host processor.

The host processor clears the i960 core processor reset by clearing the Core Processor Reset bit in the RRCR. This deasserts the reset signal on the i960 core processor and the processor begins its initialization process.

5.1.4.2 Mode 1

Mode 1 allows each unit of the i960® VH embedded-PCI processor to be initialized in its own manner. All units are reset when the **P_RST#** signal is asserted. Each unit returns to its default state. Be aware that race conditions may exist between i960 core processor operation after reset and PCI configuration.

5.1.4.3 Mode 2

This is the default initialization mode.

Mode 2 allows the i960 core processor to initialize and control the initialization process before the host processor is allowed to configure the 80960VH processor. The *i960® VH Processor Developer's Manual* describes the i960 core processor initialization process. This option is only available if the i960 core processor Initial Memory Image (IMI) is properly loaded into memory at the correct location.

The host processor is prevented from initializing the 80960VH processor. During this time, the ATU will signal a retry on all configuration cycles it receives on the primary PCI bus until the i960 core processor has cleared the Configuration Cycle Disable bit in the RRCR.

By allowing the i960 core processor to control the initialization process, it is possible to initialize the PCI configuration registers to values other than the default power-up values. Certain PCI configuration registers that are read only through PCI configuration cycles are read/write from the i960 core processor. This allows the programmer to customize the way the i960® VH embedded-PCI processor appears to the PCI configuration software.

5.1.5 ATU Initialization

The ATU initialization includes initialization by the i960 JT core and initialization by the PCI host processor. Local initialization occurs first and consists mainly of establishing the operational parameters for access to the EVAL80960VH platform local bus. The Primary Inbound ATU Limit Register (PIALR) is initialized to establish the block size of memory required by the ATU. The PIALR value is based on the installed DRAM configuration. The Primary Inbound ATU Translate Value Register (PIATVR) is initialized to establish the translation value to convert PCI addresses to local addresses. The Primary Outbound Memory Window Value Register (POMWVR) is initialized to establish the translation value for Local-to-PCI accesses. The POMWVR value remains at its default value of “0” to allow the EVAL80960VH platform to access the start of the PCI Memory address map, which is typically occupied by PCI host memory. Likewise, the Primary Outbound I/O Window Value Register (POIOWVR) remains at its default value of “0” to allow the EVAL80960VH platform to access the start of the PCI I/O address map. PCI Doorbell-related parameters are also established to allow for communication between the EVAL80960VH platform and a PCI bus master using the doorbell mechanism.

By default, Primary Outbound Configuration Cycle parameters are not established and Dual Address Cycle (DAC) support is not enabled. The ATU Configuration Register (ATUCR) is initialized to establish the operational parameters for the Doorbell Unit and ATU interrupts and to enable the ATU. The PCI host is responsible for allocating PCI address space (Memory, Memory Mapped I/O, and I/O), and assigning the PCI Base addresses for the EVAL80960VH platform.

5.2 mon960 Kernel

The mon960 Kernel (monitor) provides the EVAL80960VH user with a software platform on which application software can be developed and run. The monitor provides several features that the EVAL80960VH user can use to speed application development. Among the available features are:

- Communication with a terminal or terminal emulation package on a host computer through a serial cable with automatic baud rate detection
- Communication with a symbolic debugger such as gdb960 (available from Intel) using the Host Debugger Interface (HDI) software. gdb960 or mondb can communicate through serial, PCI or JTAG interfaces.
- Communication with the host computer via the PCI bus
- Downloads of object files via the PCI, serial or JTAG port
- Downloads of ELF object files via the PCI bus
- On-board erasure and programming of Intel 28F016S5 Flash ROM at U14
- Memory display and modification capability
- Breakpoint and single-step capability to support debugging of user code
- Disassembly of i960 processor instructions

5.3 Diagnostics

EVAL80960VH platform diagnostic routines serve a two-fold purpose: to verify proper hardware operation and to provide example code for users who need similar functions in their applications.

5.3.1 Board Level Diagnostics

Board level diagnostics exercise all basic areas of the EVAL80960VH platform. Diagnostic routines include DRAM tests, UART tests, LED tests, internal timer tests, I^2C bus tests, and PCI bus tests. PCI bus tests exercise the ATU, the PCI Doorbell unit, and the PCI DMA controller. Interrupts from both local and PCI sources are generated and handled. The PCI bus tests require an external test suite running on a PC to verify complete functionality of the EVAL80960VH platform.

The tests can be invoked using the ‘`po`’ command in either Terminal Emulation Mode or from mondb. This command is explained in *Chapter 4*, *Monitor Commands* of the *MON960 Debug Monitor User’s Guide*.

Bill of Materials A

This appendix identifies all components on the EVAL80960VH Evaluation Platform (Table A-1).

Table A-1. EVAL80960VH Bill of Materials (Sheet 1 of 3)

Location	Cyclone P/N	Part Description	Qty	Mfg	Mfg Part Number
U7	100-1198	IC/SM 74ALS08 (SOIC 14)	1	National Semicon-ductor	DM74ALS08M
U3	100-1199	IC/SM 74ALS04 SOIC	1	National Semicon-ductor	DM74ALS04BM
U2	100-1792	IC/SM 74ABT273 SOIC	1	Texas Instruments	SN74ABT273DW
U15	100-1793	IC/SM 74ABT573 SOIC	1	Texas Instruments	SN74ABT573DW
U4	100-2048	IC / SM 1488A SOIC	1	National Semicon-ductor	DS1488M
U1	100-2049	IC / SM 1489A SOIC	1	National Semicon-ductor	DS1489AM
U10	100-5098	IC/SM MAX767CAP SOIC	1	Maxim	MAX767CAP
U12	102-1055-99	Processor 80960VH from INTEL	1	Intel	102-1055-99
U6	102-1127	VLSI I/O UART 16C550 PLCC	1	Texas Instruments	TL16C550AFN
U14	103-1265	MEM Flash E28F016S5-090 TSOP	1	Intel	E28F0165S5-090
U8	103-1550	MEMORY EEPROM 24C08 SOIC 8	1	Microchip	24LC08SOI
C1-C10,C11, C13,C16-C23, C25,C26,C30, C32,C33, C36-C38, C43-C52	110-3301	CAP CERM SM, 0.01uf (0805)	38	Kemet	C0805C103K5RAC
C24,C31,C41, C42	110-3304	CAP CERM SM, 0.1uf (0805)	4	Kemet	C0805C104K5RAC
R15,R18,R30	126-1001-05	R/SM 1/10W 5% 10 ohm (0805)	3	Dale	CRCW080510R0F
R1,R2	126-1003-05	R/SM 1/10W 5% 1K ohm (0805)	2	Dale	CRCW08051001FRT
R12,R19,R20, R23,R25,R29	126-1004-05	R/SM 1/10W 5% 10K ohm (0805)	6	Dale	CRCW08051002FRT
R4,R5	126-4702-05	R/SM 1/10W 5% 470 ohm (0805)	2	Dale	CRCW 0805 471JT

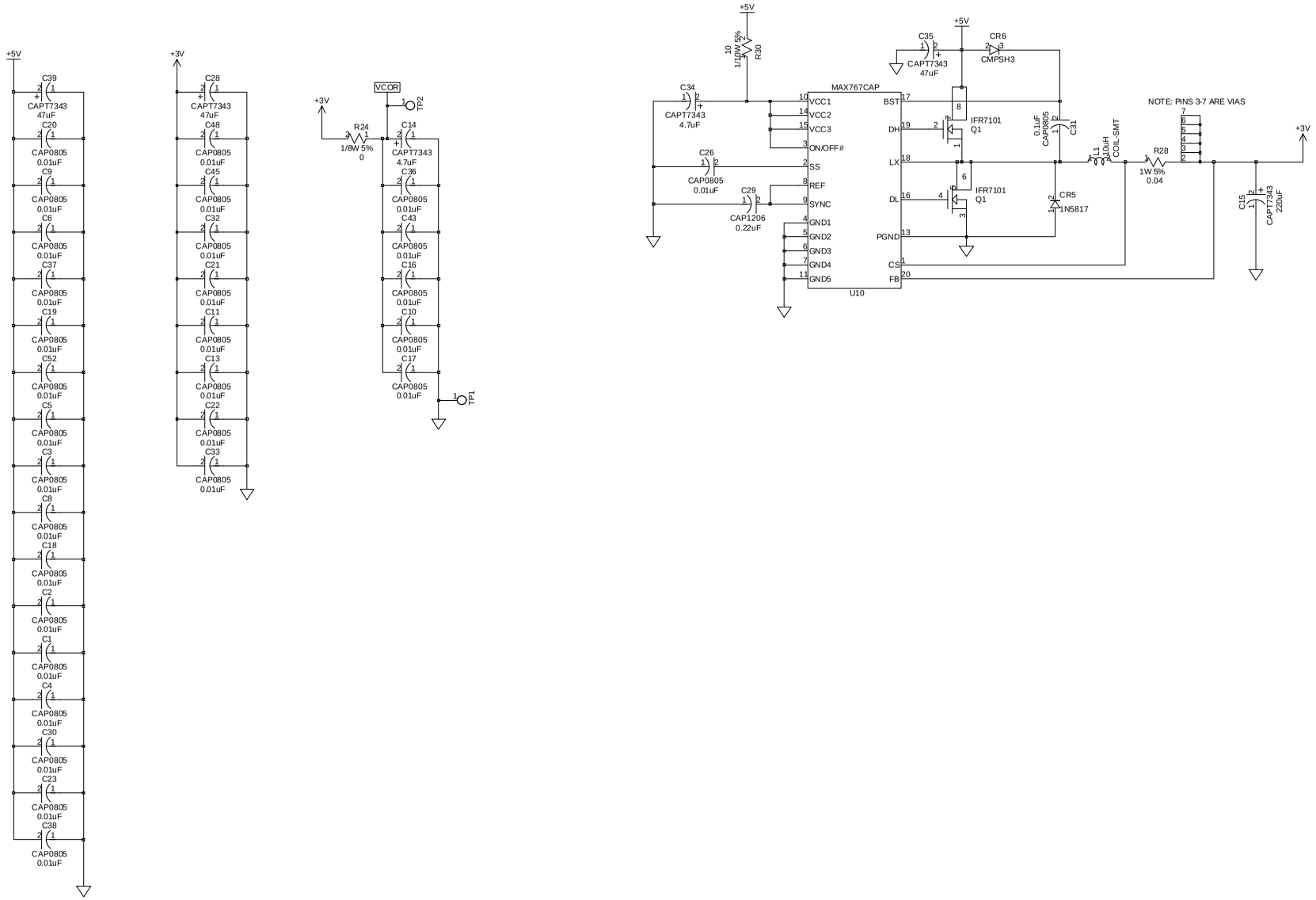
Table A-1. EVAL80960VH Bill of Materials (Sheet 2 of 3)

Location	Cyclone P/N	Part Description	Qty	Mfg	Mfg Part Number
R24	127-0000-05	R/SM 1/8W 5% 000 ohm chip 1206	1	Dale	CRCW1206000Z
J4-J6	130-1438	CONN SM/TH Mictor 38P Recptcl	3	Amp	767054-1
J3	130-1450	CONN SIMM 72P Socket / Angled	1	Amp	822134-3
J1	130-1509	CONN TJ6 PCB 6/6 LP thru hole	1	Kycon	GM-N-66
J2	130-1577	CONN Hdr 16 pin/w shell, pcb	1	Amp	103308-3
Z1	130-1621	JumperJUMP2X1	1	Molex	22-28-4023
J7,J8	130-1717	CONN 2x25 Socket Strip .025”	2	Samtec	BHT-125-04-GF-D
SW1	150-4000	Switch, SW-PUSH 3P	1	Epson	EP12SD1ABE
U5	150-6009	OSC 1.8432MHz 1/2 - Thru hole	1	Kyocera	KH0HC1CSE 1.843
U11	150-8005	Clock Chip CY7B9910-7SC	1	Cypress	CY7B9910-7SC
CR1	150-9100	LED Green	1	Hewlett Packard	HLMP-3507$010
CR2	150-9101	LED-Red	1	Hewlett Packard	HLMP3301$010 HP
CR3,CR4	150-9103	LED-Red-Small Group	2	Dialight	555-4001
U9	193-2001	SOCKET PLCC32 LP Surface Mount	1	Amp	822273-1
U13	193-2002	SOCKET PLCC44 LP Surface Mount	1	Amp	822275-1
	270-0938	PCB 0938A for VH Eval Board	1	N/A	270-0938
N/A	101-0921	PAL MACH111-15JC / PLCC44	1	Vantis	MACH111-15JC
N/A	103-1214-99	MEMORY N28F020-150 / Intel	1	Intel	N28F020-150
N/A	230-1102	SCREW 4.40x3/16 Pan Head Slot	2	McMaster Carr	91792A105
N/A	230-1199	SCREW-NYLON 4/ 40x3/16 Pan Head	2	McMaster Carr	94611A105 (100)
N/A	230-1503	HARDWARE 4-40 x 0.465 Spacer	2	Keystone	1902A at 0.465
N/A	351-0938	Faceplate PCI938	1	Gompf	9334-0406
C29	110-3103	CAP SM, 0.22uf (1206)	1	Philips	12062E224M9BB2
C15	110-3208	CAP TANT SM 220uf, 10V (7343)	1	AVX	TPSE227K010R010
C28,C35,C39, C40	110-3209	CAP TANT SM 47uf, 16V (7343)	4	AVX	TPSD476K016R015

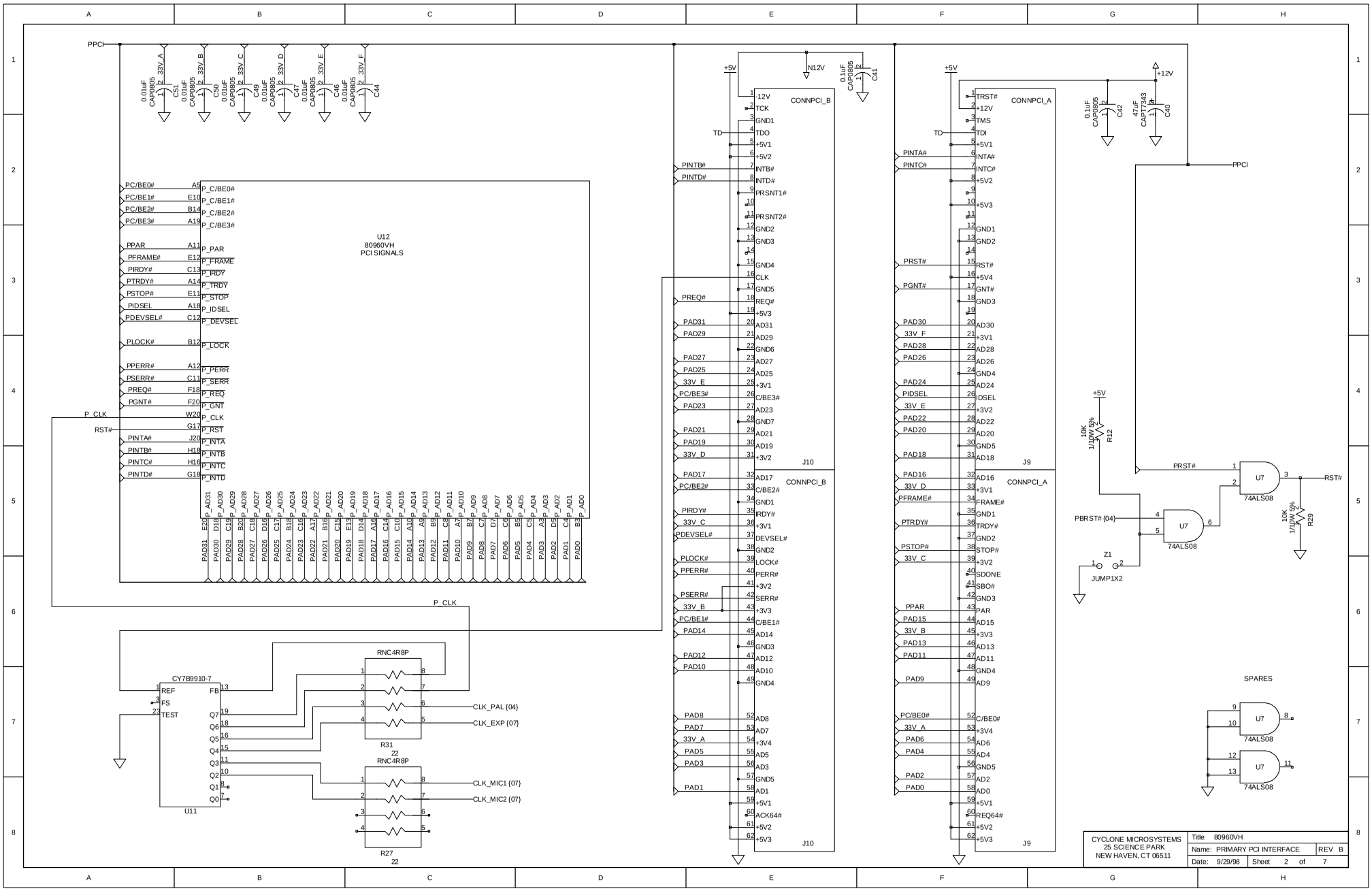
Table A-1. EVAL80960VH Bill of Materials (Sheet 3 of 3)

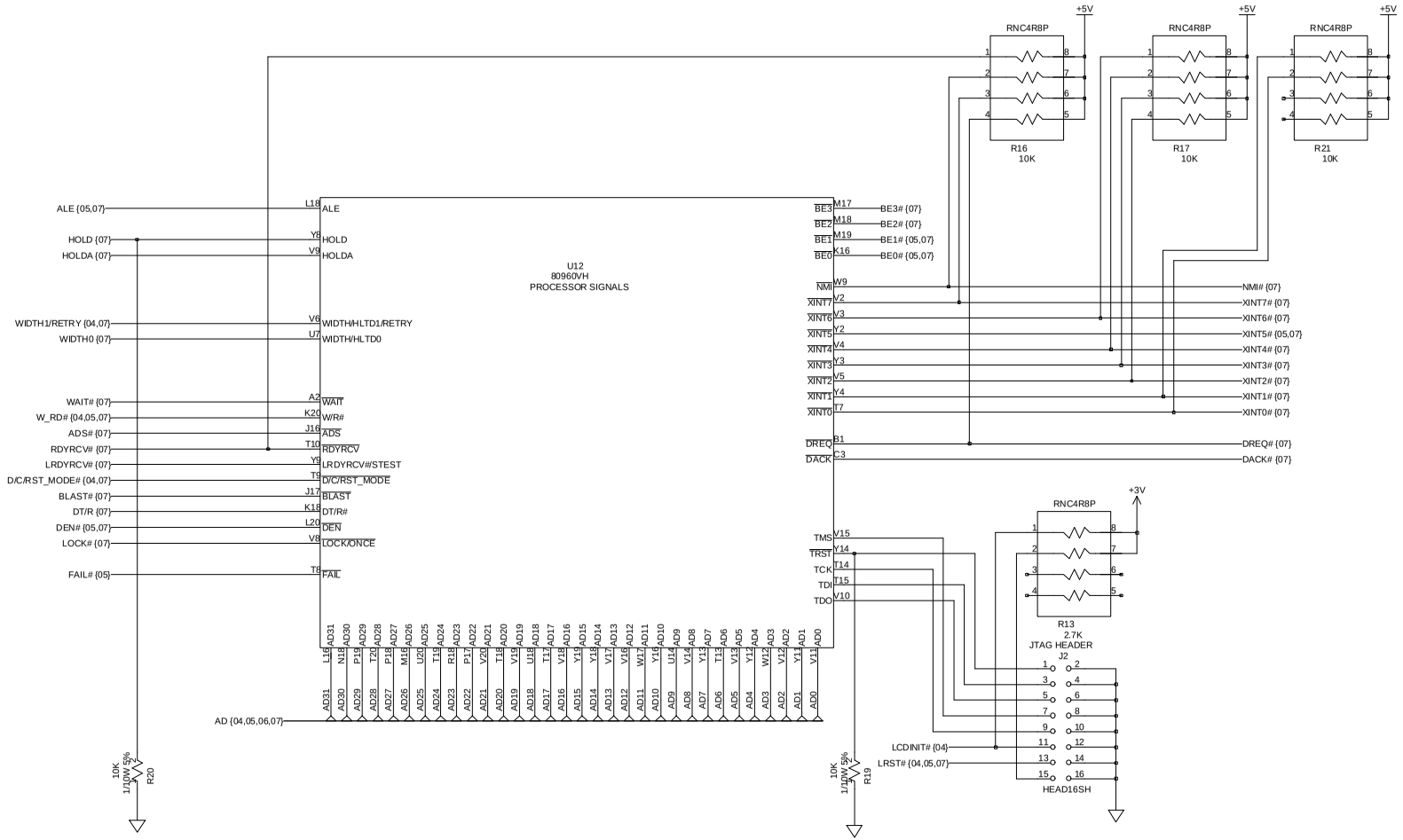
Location	Cyclone P/N	Part Description	Qty	Mfg	Mfg Part Number
C12,C14,C27, C34	110-3211	CAP TANT SM 4.7uf, 35V (7343)	4	AVX	293D475X9035D2T
R28	128-0110	Resistor/SM 1 W 5% 0.04 ohm	1	Dale	WSL-2512R04-1TR
R26	128-0120	Resistor/SM 1/2W 5% 100 ohm	1	Beckman	BCR 1/2 101 JT
R13	129-2101	Resistor Pk SM RNC4R8P 2.7kohm	1	CTS	742083272JTR
R3,R16,R17, R21	129-2102	Resistor Pk SM RNC4R8P 10kohm	4	Philips	2350-034-10103
R8-R11,R22, R27,R31	129-2103	Resistor Pk SM RNC4R8P 22ohm	7	CTS	742083220JTR
R6,R7	129-2104	Resistor Pk SM RNC4R8P 470ohm	2	CTS	742083471JTR
R14	129-2105	Resistor Pk SM RNC4R8P 4.7kohm	1	CTS	742083472JTR
L1	150-1010	Coil 10 uh SM CDR74B-100MC	1	Sumida	CDR74B-100MC
S1,S2	150-4050	Switch/SM DIP4 Mors# DHS-4S	2	Morse	DHS-4S
CR5	160-1101	Diode 1N5817 Surface Mount	1	Central Semicon-ductor	CMSH1-20
CR6	160-1102	Diode CMPSH3 Surface Mount	1	Central Semicon-ductor	CMPSH3
Q1	160-1103	Diode IRF7101 Surface Mount	1	International Rectifier	IRF7101T1

Schematics B

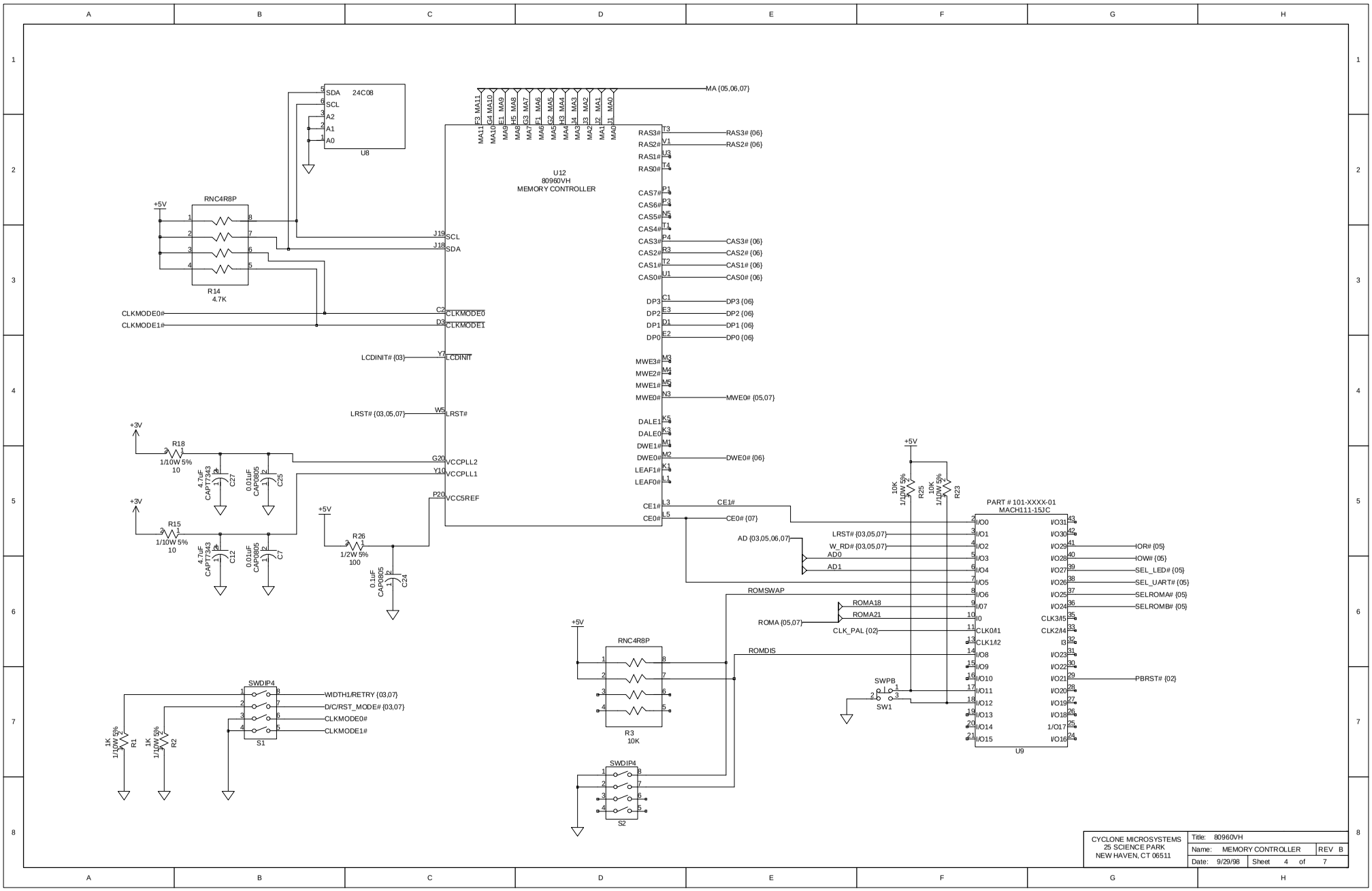


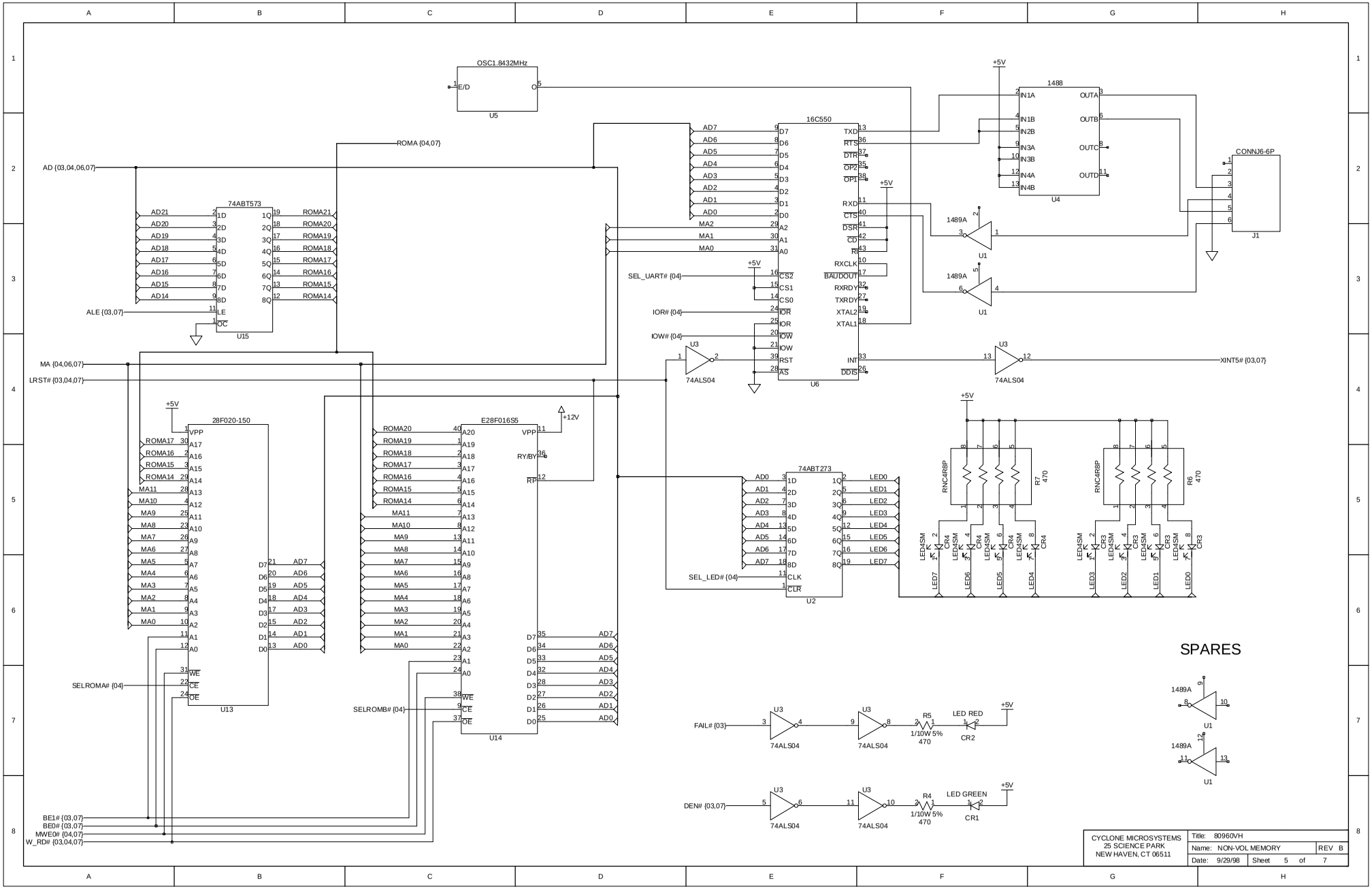
CYCLONE MICROSYSTEMS 25 SCIENCE PARK NEW HAVEN, CT 06511	Title: 80960VH
	Name: 5V to 3.3V/DECOUPLING REV B
	Date: 9/29/98 Sheet 1 of 7

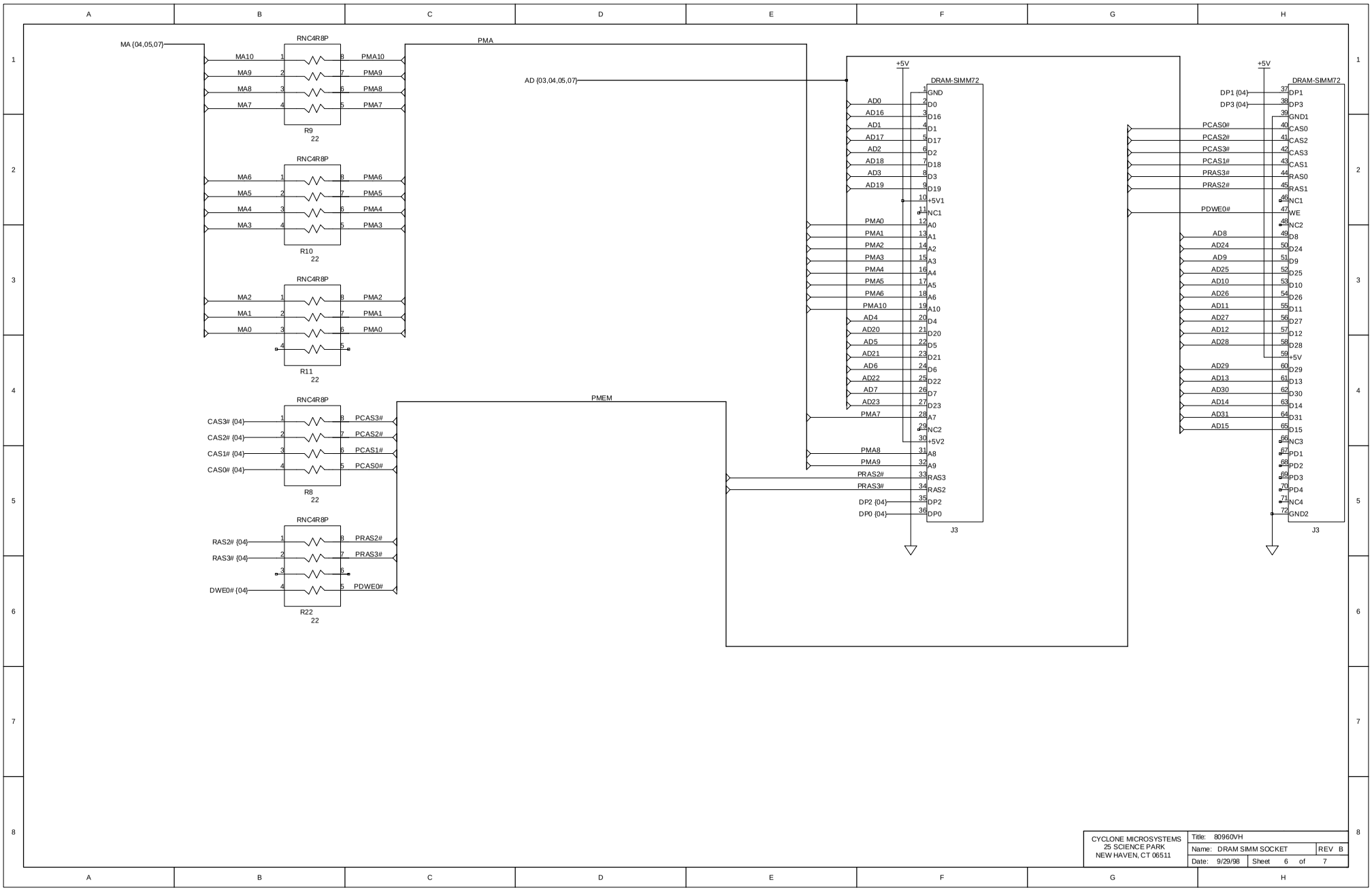

PPCI
U12
80960VH
PCI SIGNALS
CONNPCI_B
CONNPCI_A
J10
J9
P_CLK
RST#
CY7B9910-7
U11
RNC4R8P
R31
R27
22
CLK_PAL {04}
CLK_EXP {07}
CLK_MIC1 {07}
CLK_MIC2 {07}
+5V
+12V
N12V
TD
PBRST# {04}
JUMP1X2
74ALS08
SPARES
CYCLONE MICROSYSTEMS
25 SCIENCE PARK
NEW HAVEN, CT 06511
Title: 80960VH
Name: PRIMARY PCI INTERFACE
REV B
Date: 9/29/98
Sheet 2 of 7

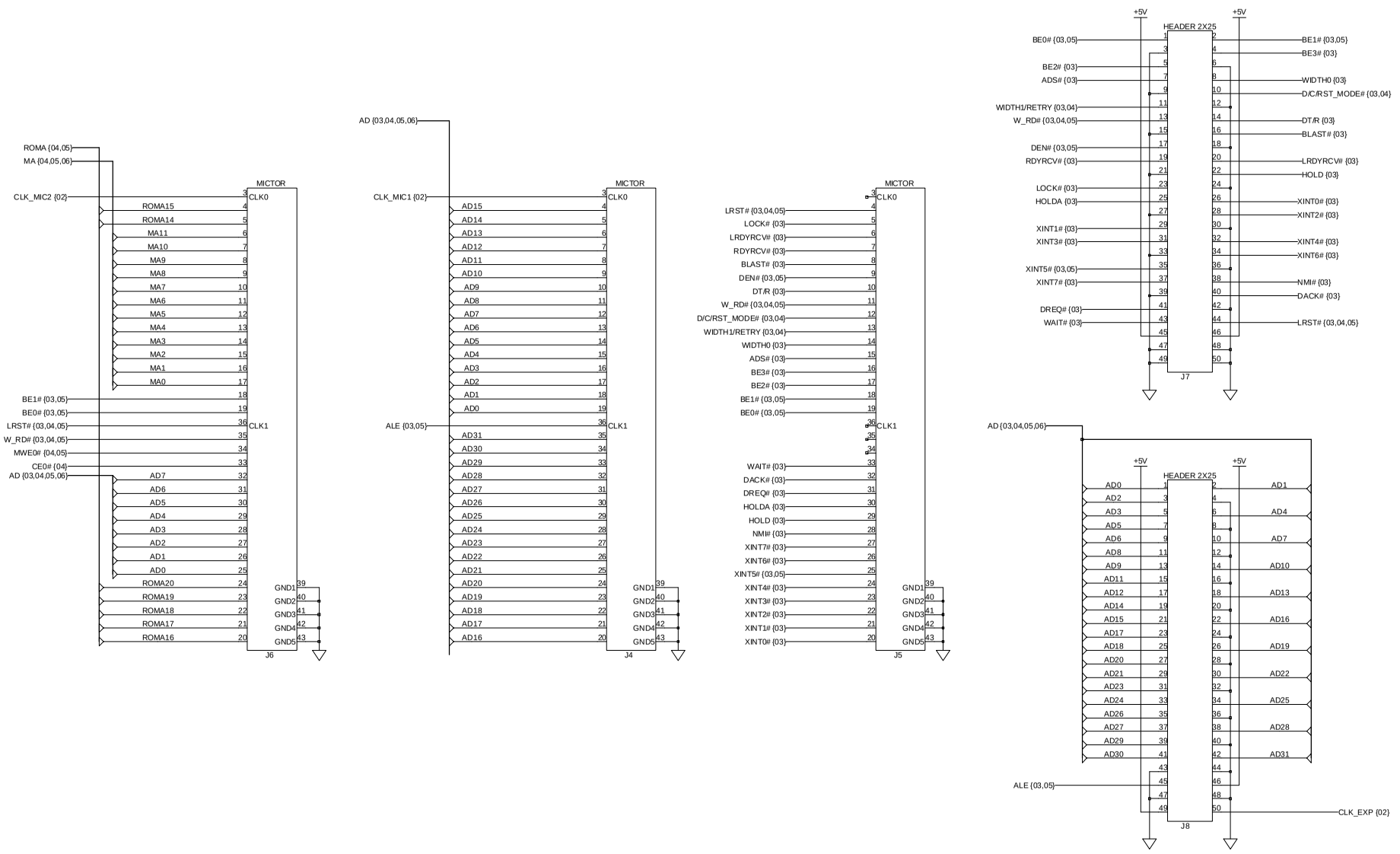


CYCLONE MICROSYSTEMS 25 SCIENCE PARK NEW HAVEN, CT 06511	Title: 80960VH		
	Name: I960 CORE		REV B
	Date: 9/29/98	Sheet 3 of 7	


U8
24C08
SDA
SCL
A2
A1
A0
U12
80960VH
MEMORY CONTROLLER
MA {05,06,07}
RNC4R8P
R14
4.7K
SCL
SDA
CLKMODE0#
CLKMODE1#
CLKMODE0
CLKMODE1
LCDINIT# {03}
LCDINIT
LRST# {03,05,07}
LRST#
RAS3# {06}
RAS2# {06}
CAS3# {06}
CAS2# {06}
CAS1# {06}
CAS0# {06}
DP3 {06}
DP2 {06}
DP1 {06}
DP0 {06}
MWE0# {05,07}
DWE0# {06}
CE1#
CE0# {07}
VCCPLL2
VCCPLL1
VCC5REF
R18
R15
R26
1/10W 5%
10
1/2W 5%
100
4.7uF
CAPT7343
0.01uF
CAP0805
0.1uF
C27
C25
C12
C7
C24
AD {03,05,06,07}
AD0
AD1
W_RD# {03,05,07}
ROMSWAP
ROMA18
ROMA21
ROMA {05,07}
CLK_PAL {02}
ROMDIS
PART # 101-XXXX-01
MACH111-15JC
U9
IOR# {05}
IOW# {05}
SEL_LED# {05}
SEL_UART# {05}
SELROMA# {05}
SELROMB# {05}
PBRST# {02}
10K
R25
R23
SWPB
SW1
R3
10K
SWDIP4
S1
S2
WIDTH1/RETRY {03,07}
D/C/RST_MODE# {03,07}
CLKMODE0#
CLKMODE1#
1K
R1
R2
CYCLONE MICROSYSTEMS
25 SCIENCE PARK
NEW HAVEN, CT 06511
Title: 80960VH
Name: MEMORY CONTROLLER
REV B
Date: 9/29/98
Sheet 4 of 7


OSC1.8432MHz
E/D
O
U5
16C550
74ABT573
74ABT273
28F020-150
E28F016S5
1488
1489A
CONNJ6-6P
74ALS04
RNC4R8P
LED 4SM
LED RED
LED GREEN
SPARES
ROMA {04,07}
AD {03,04,06,07}
ALE {03,07}
MA {04,06,07}
LRST# {03,04,07}
SEL_UART# {04}
IOR# {04}
IOW# {04}
SEL_LED# {04}
SELROMA# {04}
SELROMB# {04}
FAIL# {03}
DEN# {03,07}
XINT5# {03,07}
BE1# {03,07}
BE0# {03,07}
MWE# {04,07}
W_RD# {03,04,07}
+5V
+12V
R5 1/10W 5% 470
R4 1/10W 5% 470
R7 470
R6 470
CR1
CR2
U1
U2
U3
U4
U6
U13
U14
U15
J1
CYCLONE MICROSYSTEMS
25 SCIENCE PARK
NEW HAVEN, CT 06511
Title: 80960VH
Name: NON-VOL MEMORY
REV B
Date: 9/29/98
Sheet 5 of 7


MA {04,05,07}
RNC4R8P
MA10
MA9
MA8
MA7
PMA10
PMA9
PMA8
PMA7
R9
22
RNC4R8P
MA6
MA5
MA4
MA3
PMA6
PMA5
PMA4
PMA3
R10
22
RNC4R8P
MA2
MA1
MA0
PMA2
PMA1
PMA0
R11
22
PMA
AD {03,04,05,07}
+5V
DRAM-SIMM72
GND
D0
D16
D1
D17
D2
D18
D3
D19
+5V1
NC1
A0
A1
A2
A3
A4
A5
A6
A10
D4
D20
D5
D21
D6
D22
D7
D23
A7
NC2
+5V2
A8
A9
RAS3
RAS2
DP2
DP0
J3
AD0
AD16
AD1
AD17
AD2
AD18
AD3
AD19
PMA0
PMA1
PMA2
PMA3
PMA4
PMA5
PMA6
PMA10
AD4
AD20
AD5
AD21
AD6
AD22
AD7
AD23
PMA7
PMA8
PMA9
PRAS2#
PRAS3#
DP2 {04}
DP0 {04}
RNC4R8P
CAS3# {04}
CAS2# {04}
CAS1# {04}
CAS0# {04}
PCAS3#
PCAS2#
PCAS1#
PCAS0#
R8
22
RNC4R8P
RAS2# {04}
RAS3# {04}
DWE0# {04}
PRAS2#
PRAS3#
PDWE0#
R22
22
PMEM
DP1 {04}
DP3 {04}
DP1
DP3
GND1
CAS0
CAS2
CAS3
CAS1
RAS0
RAS1
NC1
WE
NC2
D8
D24
D9
D25
D10
D26
D11
D27
D12
D28
D29
D13
D30
D14
D31
D15
NC3
PD1
PD2
PD3
PD4
NC4
GND2
PCAS0#
PCAS2#
PCAS3#
PCAS1#
PRAS3#
PRAS2#
PDWE0#
AD8
AD24
AD9
AD25
AD10
AD26
AD11
AD27
AD12
AD28
AD29
AD13
AD30
AD14
AD31
AD15
CYCLONE MICROSYSTEMS
25 SCIENCE PARK
NEW HAVEN, CT 06511
Title: 80960VH
Name: DRAM SIMM SOCKET
REV B
Date: 9/29/98
Sheet 6 of 7


ROMA {04,05}
MA {04,05,06}
CLK_MIC2 {02}
MICTOR
CLK0
ROMA15
ROMA14
MA11
MA10
MA9
MA8
MA7
MA6
MA5
MA4
MA3
MA2
MA1
MA0
BE1# {03,05}
BE0# {03,05}
LRST# {03,04,05}
W_RD# {03,04,05}
MWE0# {04,05}
CE0# {04}
AD {03,04,05,06}
AD7
AD6
AD5
AD4
AD3
AD2
AD1
AD0
ROMA20
ROMA19
ROMA18
ROMA17
ROMA16
CLK1
GND1
GND2
GND3
GND4
GND5
J6
AD {03,04,05,06}
CLK_MIC1 {02}
AD15
AD14
AD13
AD12
AD11
AD10
AD9
AD8
AD7
AD6
AD5
AD4
AD3
AD2
AD1
AD0
ALE {03,05}
AD31
AD30
AD29
AD28
AD27
AD26
AD25
AD24
AD23
AD22
AD21
AD20
AD19
AD18
AD17
AD16
J4
LRST# {03,04,05}
LOCK# {03}
LRDYRCV# {03}
RDYRCV# {03}
BLAST# {03}
DEN# {03,05}
DT/R {03}
W_RD# {03,04,05}
D/C/RST_MODE# {03,04}
WIDTH1/RETRY {03,04}
WIDTH0 {03}
ADS# {03}
BE3# {03}
BE2# {03}
BE1# {03,05}
BE0# {03,05}
WAIT# {03}
DACK# {03}
DREQ# {03}
HOLDA {03}
HOLD {03}
NMI# {03}
XINT7# {03}
XINT6# {03}
XINT5# {03,05}
XINT4# {03}
XINT3# {03}
XINT2# {03}
XINT1# {03}
XINT0# {03}
J5
+5V
HEADER 2X25
BE0# {03,05}
BE1# {03,05}
BE2# {03}
BE3# {03}
ADS# {03}
WIDTH0 {03}
WIDTH1/RETRY {03,04}
D/C/RST_MODE# {03,04}
W_RD# {03,04,05}
DT/R {03}
DEN# {03,05}
BLAST# {03}
RDYRCV# {03}
LRDYRCV# {03}
LOCK# {03}
HOLD {03}
HOLDA {03}
XINT0# {03}
XINT1# {03}
XINT2# {03}
XINT3# {03}
XINT4# {03}
XINT5# {03,05}
XINT6# {03}
XINT7# {03}
NMI# {03}
DREQ# {03}
DACK# {03}
WAIT# {03}
LRST# {03,04,05}
J7
AD {03,04,05,06}
AD0
AD1
AD2
AD3
AD4
AD5
AD6
AD7
AD8
AD9
AD10
AD11
AD12
AD13
AD14
AD15
AD16
AD17
AD18
AD19
AD20
AD21
AD22
AD23
AD24
AD25
AD26
AD27
AD28
AD29
AD30
AD31
ALE {03,05}
CLK_EXP {02}
J8
CYCLONE MICROSYSTEMS
25 SCIENCE PARK
NEW HAVEN, CT 06511
Title: 80960VH
Name:LOGIC ANALYZER/EXPANSION
REV B
Date: 9/29/98
Sheet 7 of 7

PLD Code C

```
MODULE VH
TITLE 'VH'
" PATTERN 101-XXXX-01
" REVISION
" AUTHORJohn Neumann
" COMPANYCyclone Microsystems
" DATE   June 10, 1998
" CHIP   MACH111-15JC
    CLK      PIN 11;      "pal clock input
    CE1b     PIN 2;   "ROM space chip enable #1
    RSTb     PIN 3;       "local reset
    WRDb     PIN 4;       "write/read strobe
    AD0      PIN 5;       "address/data bit 0
    AD1      PIN 6;       "address/data bit 1
    CE0b     PIN 7;       "ROM space chip enable #0
    ROMSWAP  PIN 8;       "Select between EEPROM devices
    ROMDIS   PIN 14;      "EEPROM ROM disable
    RA18     PIN 9;       "ROM address bit 18
    RA21     PIN 10;  "ROM address bit 21
    RESET    PIN 17;      "Push Button reset
    SET      PIN 18;      "Push Button set
    PBRSTb   PIN 29;      "Debounce push button reset
    QS0      PIN 43 ISTYPE 'REG_D';"quickswitch mux select 0
    QS1      PIN 42 ISTYPE 'REG_D';"quickswitch mux select 1
    IORb     PIN 41 ISTYPE 'REG_D';  "UART read strobe
    IOWb     PIN 40 ISTYPE 'REG_D';"UART write strobe
    SELLEDb PIN 39 ISTYPE 'REG_D';"LED chip enable
    SELUARTbPIN 38 ISTYPE 'REG_D';"UART chip enable
    SELROMAbPIN 37 ISTYPE 'REG_D';"Select for 28F020
    SELROMBbPIN 36 ISTYPE 'REG_D';"Select for 28F016S5
EQUATIONS
" Quickswitch mux selects for DRAM select, write to address E0200000h with data
    QS0 := WRDb & RSTb & RA21 * !CE1b & AD0
         # QS0 & !(RA21 & !CE1b) & RSTb"hold
         # QS0 & !WRDb & RSTb & RA21 & !CE1b;"hold during a read
    QS0.CLK = CLK;
    QS1 := WRDb & RSTb & RA21 * !CE1b & AD1
         # QS1 & !(RA21 & !CE1b) & RSTb"hold
         # QS1 & !WRDb & RSTb & RA21 & !CE1b;"hold during a read
    QS1.CLK = CLK;
" Select LED is write only to address E0040000h
    !SELLEDb := RA18 & !RA21 & WRDb & !CE1b;
    SELLEDb.CLK = CLK;
" The UART is located at address E0000000h
    !SELUARTb := !RA18 & !RA21 & !CE1b;
    SELUARTb.CLK = CLK;
" UART read and write strobes
    !IORb := !WRDb & RSTb & !CE1b & !SELUARTb;
    IORb.CLK = CLK;
```

```
    !IOWb := WRDb & RSTb & !CE1b & !SELUARTb;
    IOWb.CLK = CLK;
" Flash ROM selects.  Rom A is located at FEC0 0000H to FEC3 FFFFH.
" ROM B is located at FEE0 0000H TO FEFF FFFFH.
" The equations below allow ROM 'B' and ROM 'A' to be 'swapped' thus
" changing the device from which the i960 boots.
    !SELROMAb := ROMSWAP & !CE0b & !RA21 & ROMDIS
          # !ROMSWAP & !CE0b & RA21 & ROMDIS;
    SELROMAb.CLK = CLK;
    !SELROMBb := ROMSWAP & !CE0b & RA21 & ROMDIS
          # !ROMSWAP & !CE0b & !RA21 & ROMDIS;
    SELROMBb.CLK = CLK;
"Push button reset must be debounce before it can be anded with PCI reset
    !PBRSTb= !SET # (!PBRSTb & RESET);
END
```

Backplane 0015 Interface D

D.1 Introduction

The Backplane 0015 board from Cyclone Microsystems allows the EVAL80960VH platform to be installed into a PCI socket on the board and function as a host system board. The Backplane 0015 contains four PCI slots, J1 through J4. Figure D-1 depicts board component locations. The PCI backplane board contains a 33MHz bus clock, arbitration for the four PCI slots, an ATX power supply connector with stand-by voltage power on circuit, a reset circuit, and two LEDs indicating +5V and +3.3V availability.

D.2 Installing the EVAL80960VH into the Backplane 0015

The EVAL80960VH platform must be installed into slot J4 of the backplane board. The enclosed ribbon cable must be connected to allow proper function of the EVAL80960VH as a host board. Please refer to Figure D-2 and Figure D-3 during installation and setup. Install the 26 pin keyed connector into the header position, J5, on the backplane board. The red wire should line up with pin 1 of J5. Next install the other end of the ribbon cable into J7 on the EVAL80960VH. Take care as this connector does not have any keying associated with it. The red wire should line up with pin 50 on J7, towards the top edge of the platform. This cable allows interrupts from the PCI slots on the backplane board to reach the i960VH processor.

D.3 Powering the EVAL80960VH/Backplane System

The PCI backplane board is designed to be powered via an ATX power supply. Before inserting the power cable into the ATX power connector on the backplane board make sure that the +5VSB (stand-by voltage) is off. Inserting the power cable with +5VSB on may cause damage to the start up circuit and the backplane board may not power up. After the connector is installed, depress the momentary power switch to turn on the +5V and +3.3V rails on the ATX power supply.

D.4 Interrupt Routing and IDSELs on the Backplane 0015

The PCI slots J1 through J3 allow up to 3 adapter boards to be installed in the system. These adapter boards are configured by the EVAL80960VH and interrupt the i960VH processor through the ribbon cable. The interrupt routing and IDSELs on the backplane board is shown in Table D-1. Note that the EVAL80960VH has the ability to interrupt itself.

Table D-1. PCI Interrupt Routing and IDSELs

SLOT	INTA	INTB	INTC	INTD	IDSEL
J1	XINT1	XINT2	XINT3	XINT0	AD16
J2	XINT2	XINT3	XINT0	XINT1	AD17
J3	XINT3	XINT0	XINT1	XINT2	AD18
J4	XINT0	XINT1	XINT2	XINT3	AD19

Figure D-1. EVAL80960VH Backplane Board Illustration

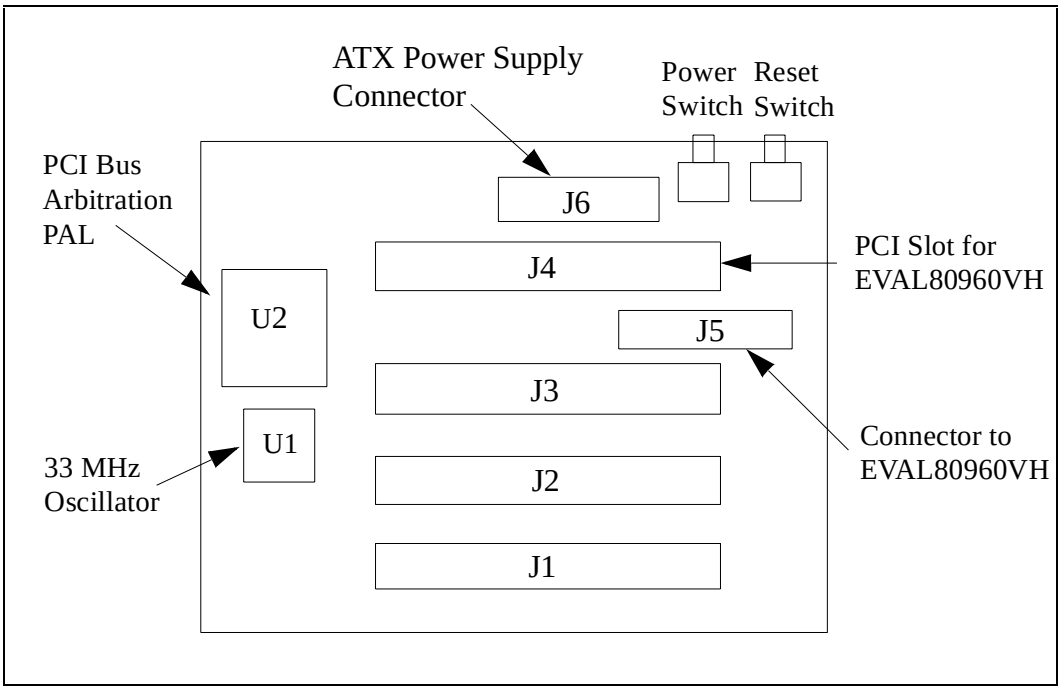


Figure D-2. EVAL80960VH/Backplane 0015 with Eval Board Installed

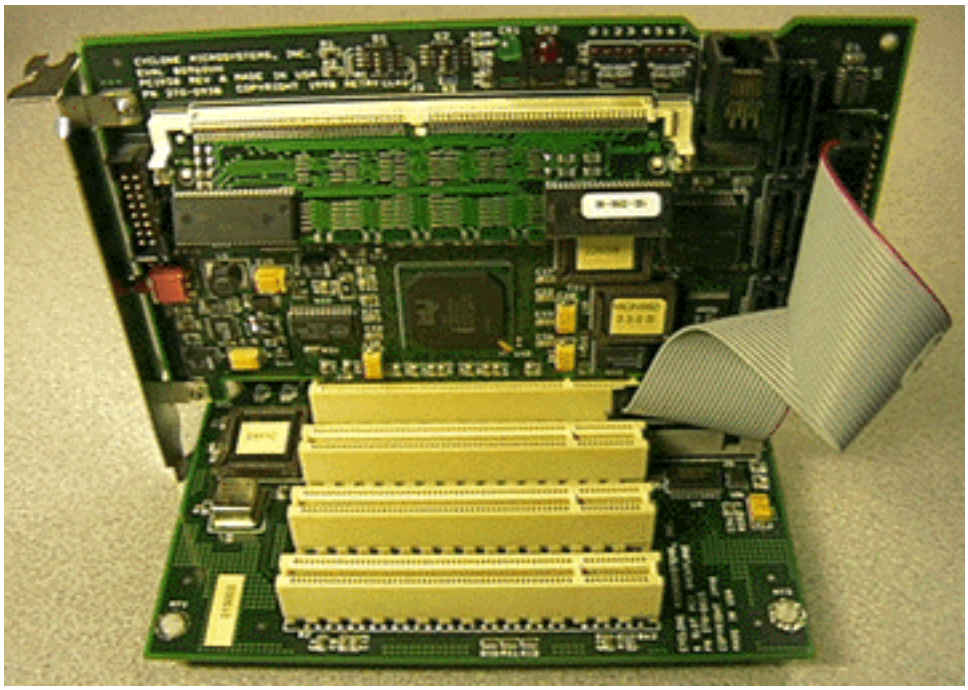
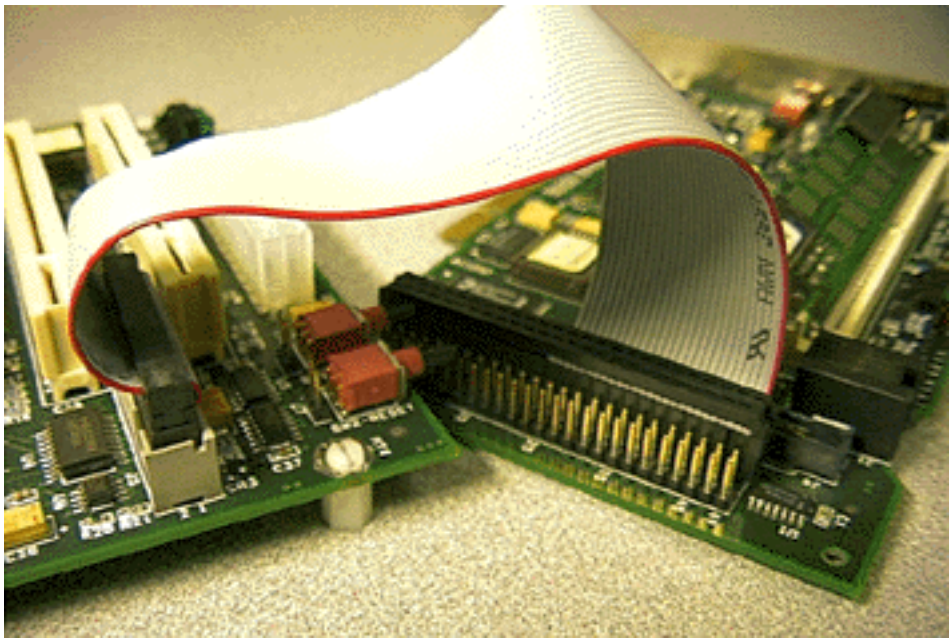


Figure D-3. EVAL80960VH/Backplane 0015 Installation



D.5 PCI Host Detection and Configuration

When installed in the slot J4 of the backplane, the EVAL80960VH acts as the PCI host in the system. The EVAL80960VH determines whether it is in the host state by the XINT4 input. The backplane drives the VH processor's XINT4 low via the ribbon cable. Upon system startup, MON960 uses the presence of an XINT4 interrupt to determine whether it needs to initialize the primary PCI bus. If the EVAL80960VH is the PCI host, the ATU base address (PIABAR) is set to the base of DRAM (A000 0000h), and the master enable bit in the ATU command register (PATUCMD) is set. MON960 then disables XINT4. Users should be aware that XINT4 should not be enabled.

D.6 PCI Initialization

When acting as a PCI host, MON960 extensions are responsible for initializing the devices on the PCI bus of the EVAL80960VH. PCI initialization involves allocating address spaces (Memory, Memory Mapped I/O, and I/O), assigning PCI base addresses, assigning IRQ values (seeTable D-1), and enabling PCI mastership. Devices containing PCI-to-PCI bridges and hierarchical buses are not supported.

D.7 PCI Bios Routines

The PCI BIOS routines are accessible at the MON960 layer and can also be accessed by application level software by using the i960 system call mechanism and the System Procedure Table. The supported BIOS functions are described in the subsections that follow.

pci_bios_present()
find_pci_device()
find_pci_class_code()
generate_special_cycle()
read_config_byte()
read_config_word()
read_config_dword()
write_config_byte()
write_config_word()
write_config_dword()
get_irq_routing_options()
set_pci_irq()

Although the calling interface is different from that used on a DOS-based host, these functions preserve, as closely as possible, the parameters and return values described in *PCI Local Bus Specification*, Revision 2.1. Functions that return multiple values do so by filling in the fields of a structure passed by the calling routine.

pci_bios_present

This function allows the caller to determine whether the PCI BIOS interface function set is present, and the current interface version level. It also provides information about the hardware mechanism used for accessing configuration space and whether or not the hardware supports generation of PCI Special Cycles.

Calling convention:

```
int pci_bios_present (
       PCI_BIOS_INFO *info
       );
```

Return values:

This function always returns SUCCESSFUL.

find_pci_device

This function returns the location of PCI devices that have a specific Device ID and Vendor ID. Given a Vendor ID, a Device ID, and an Index, the function returns the Bus Number, Device Number, and Function Number of the Nth Device/Function whose Vendor ID and Device ID match the input parameters.

Calling software can find all devices having the same Vendor ID and Device ID by making successive calls to this function starting with the index set to “0”, and incrementing the index until the function returns DEVICE_NOT_FOUND. A return value of BAD_VENDOR_ID indicates that the Vendor ID value passed had a value of all “1”s.

Calling convention:

```
int find_pci_device (
        int     device_id,
        int     vendor_id,
        int     index
        );
```

Return values:

This function returns SUCCESSFUL if the indicated device is located, DEVICE_NOT_FOUND if the indicated device cannot be located, or BAD_VENDOR_ID if the vendor_id value is illegal.

find_pci_class_code

This function returns the location of PCI devices that have a specific Class Code. Given a Class Code and an Index, the function returns the Bus Number, Device Number, and Function Number of the Nth Device/Function whose Class Code matches the input parameters.

Calling software can find all devices having the same Class Code by making successive calls to this function starting with the index set to “0”, and incrementing the index until the function returns DEVICE_NOT_FOUND.

Calling convention:

```
int find_pci_class_code (
    int class_code,
    int index
    );
```

Return values:

This function returns SUCCESSFUL if the indicated device is located or DEVICE_NOT_FOUND if the indicated device cannot be located.

generate_special_cycle

This function allows for generation of PCI Special Cycles. The generated special cycle is broadcast on a specific PCI Bus in the system.

PCI Special Cycles are not supported on the 0015 backplane PCI bus.

Calling convention:

```
int generate_special_cycle (
    int bus_number,
    int special_cycle_data
    );
```

Return values:

Since PCI Special Cycles are not supported by the 0015 backplane, this function always returns FUNC_NOT_SUPPORTED.

read_config_byte

This function allows the caller to read individual bytes from the configuration space of a specific device.

Calling convention:

```
int read_config_byte (
    int bus_number,
    int device_number,
    int function_number,
    int register_number,/* 0,1,2,...,255 */
    UINT 8*data
    );
```

Return values:

This function returns SUCCESSFUL if the indicated byte was read correctly or ERROR if there is a problem with the parameters.

read_config_word

This function allows the caller to read individual shorts (16 bits) from the configuration space of a specific device. The Register Number parameter must be a multiple of two (i.e., bit 0 must be set to “0”).

Calling convention:

```
int read_config_word (
    int bus_number,
    int device_number,
    int function_number,
    int register_number,/* 0,2,4,...,254 */
```

```
    UINT 16*data
    );
```

Return values:

This function returns SUCCESSFUL if the indicated word was read correctly or ERROR if there is a problem with the parameters.

read_config_dword

This function allows the caller to read individual longs (32 bits) from the configuration space of a specific device. The Register Number parameter must be a multiple of four (i.e., bits 0 and 1 must be set to “0”).

Calling convention:

```
int read_config_dword (
    int bus_number,
    int device_number,
    int function_number,
    int register_number,/* 0,4,8,...,252 */
    UINT 32*data
     );
```

Return values:

This function returns SUCCESSFUL if the indicated long was read correctly or ERROR if there is a problem with the parameters.

write_config_byte

This function allows the caller to write individual bytes to the configuration space of a specific device.

Calling convention:

```
int write_config_byte (
    int bus_number,
    int device_number,
    int function_number,
    int register_number,/* 0,1,2,...,255 */
    UINT 8*data
    );
```

Return values:

This function returns SUCCESSFUL if the indicated byte was written correctly or ERROR if there is a problem with the parameters.

write_config_word

This function allows the caller to write individual shorts (16 bits) to the configuration space of a specific device. The Register Number parameter must be a multiple of two (i.e., bit 0 must be set to “0”).

Calling convention:

```
int write_config_word (
    int bus_number,
    int device_number,
    int function_number,
    int register_number,/* 0,2,4,...,254 */
    UINT 16*data
    );
```

Return values:

This function returns SUCCESSFUL if the indicated word was written correctly or ERROR if there is a problem with the parameters.

write_config_dword

This function allows the caller to write individual longs (32 bits) to the configuration space of a specific device. The Register Number parameter must be a multiple of four (i.e., bits 0 and 1 must be set to “0”).

Calling convention:

```
int write_config_dword (
    int bus_number,
    int device_number,
    int function_number,
    int register_number,/* 0,4,8,...,252 */
    UINT 32*data
    );
```

Return values:

This function returns SUCCESSFUL if the indicated long was written correctly or ERROR if there is a problem with the parameters.

get_irq_routing_options

This routine returns the PCI interrupt routing options available on the 0015 backplane. Two values are provided for each PCI interrupt pin for each device. One of these values is a bitmap that shows the interrupt to which 80960VH XINT (XINT3:0) is connected. The second value is a link value that provides a way of specifying which PCI interrupt pins are wire-OR’ed together on the motherboard. Interrupt pins with the same link value are wired together.

The PCI Interrupt routing fabric on the 0015 backplane is not reconfigurable (fixed mapping relationships).

Calling convention:

```
int get_irq_routing_options (
    PCI_IRQ_ROUTING_TABLE *table
    );
```

Return values:

This function always returns SUCCESSFUL.

set_pci_irq

The PCI Interrupt routing fabric on the 0015 backplane is not reconfigurable (fixed mapping relationships); therefore, this function is not supported.

Calling convention:

```
int set_pci_irq (
    int int_pin,
    int irq_num,
    int bus_dev
    );
```

Return values:

This function always returns FUNC_NOT_SUPPORTED.

D.8 Additional MON960 Commands

The following commands have been added to the UI interface of MON960 to support the 0015 backplane.

print_pci Utility

A print_pci command to MON960 is accessed through the MON960 command prompt. This command displays the contents of the PCI configuration space on a selected adapter on the primary PCI interface or on the i960 VH processor itself. For more information on the meaning of the fields in PCI configuration space, refer to the *PCI Local Bus Specification* Revision 2.1. The syntax of this command is:

pp <bus number> <device number> <function number>

D.9 Bill of Material

This appendix identifies all components on the Backplane 0015 Interface (Table D-2).

Table D-2. Backplane 0015 Interface Bill of Material (Sheet 1 of 3)

Cyclone P/N	Reference Designator	Description	Qty	Mfg	Mfg Part Number	Note
100-1698	U7	IC/SM 74HC74 SOIC-14	1	Texas Instruments	SN74HC74D	
100-1699	U5	IC/SM 74HC14 SOIC	1	National Semi.	74HC14	
100-1225	U4	IC/SM 74AS760 SOIC	1	Texas Instruments	SN74AS760DW	
100-2200	U6	IC/SM TL7705ACD	1	Texas Instruments	TL7705ACD	
100-3098	U3	IC/SM LVCMOS Fanout Buffr SSOP	1	Motorola	MPC9140	
110-3209	C16,C31	CAP TANT SM 47uf, 16V (7343)	2	AVX	TPSD476K016R0 15	(2)

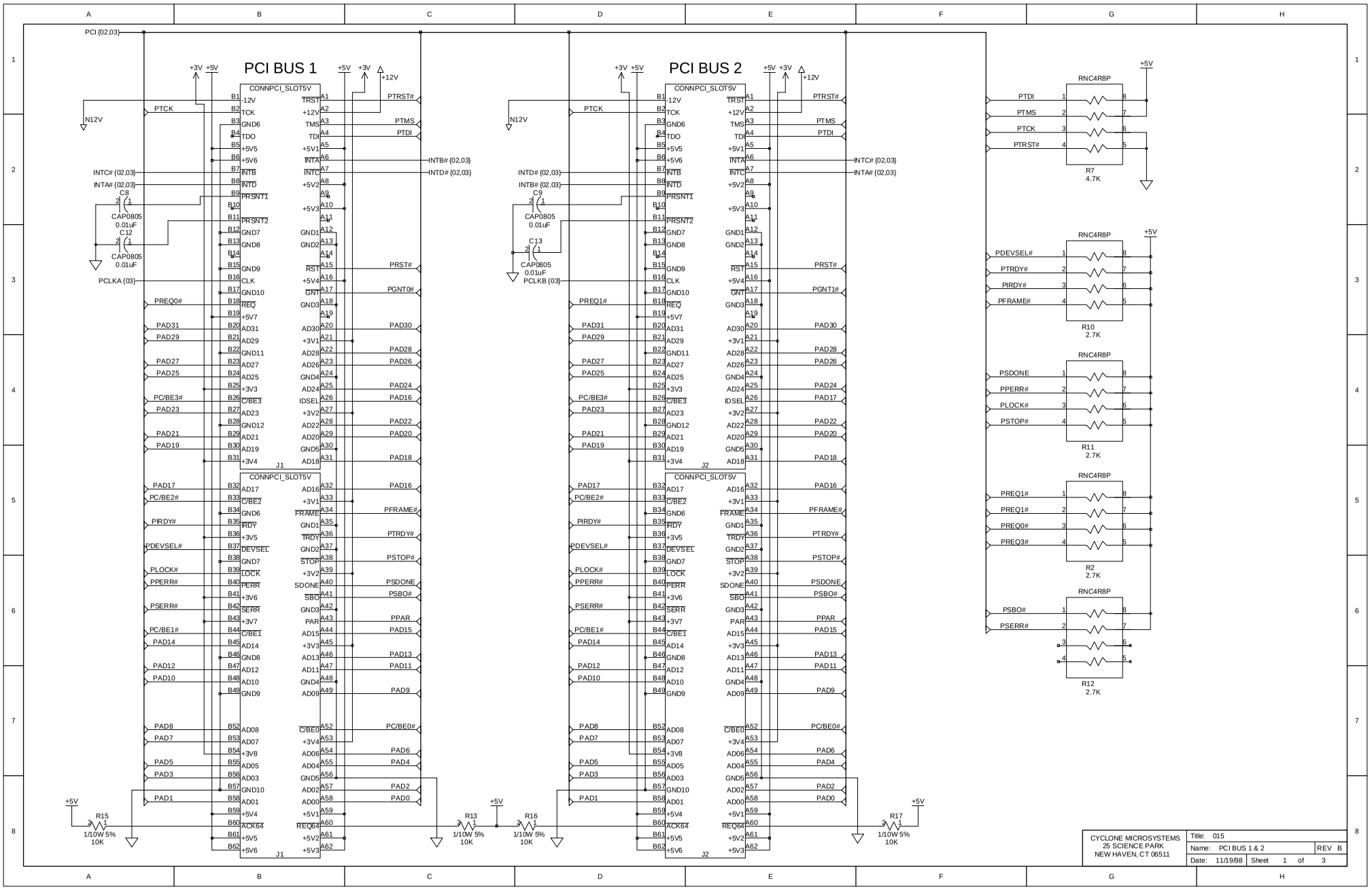
Table D-2. Backplane 0015 Interface Bill of Material (Sheet 2 of 3)

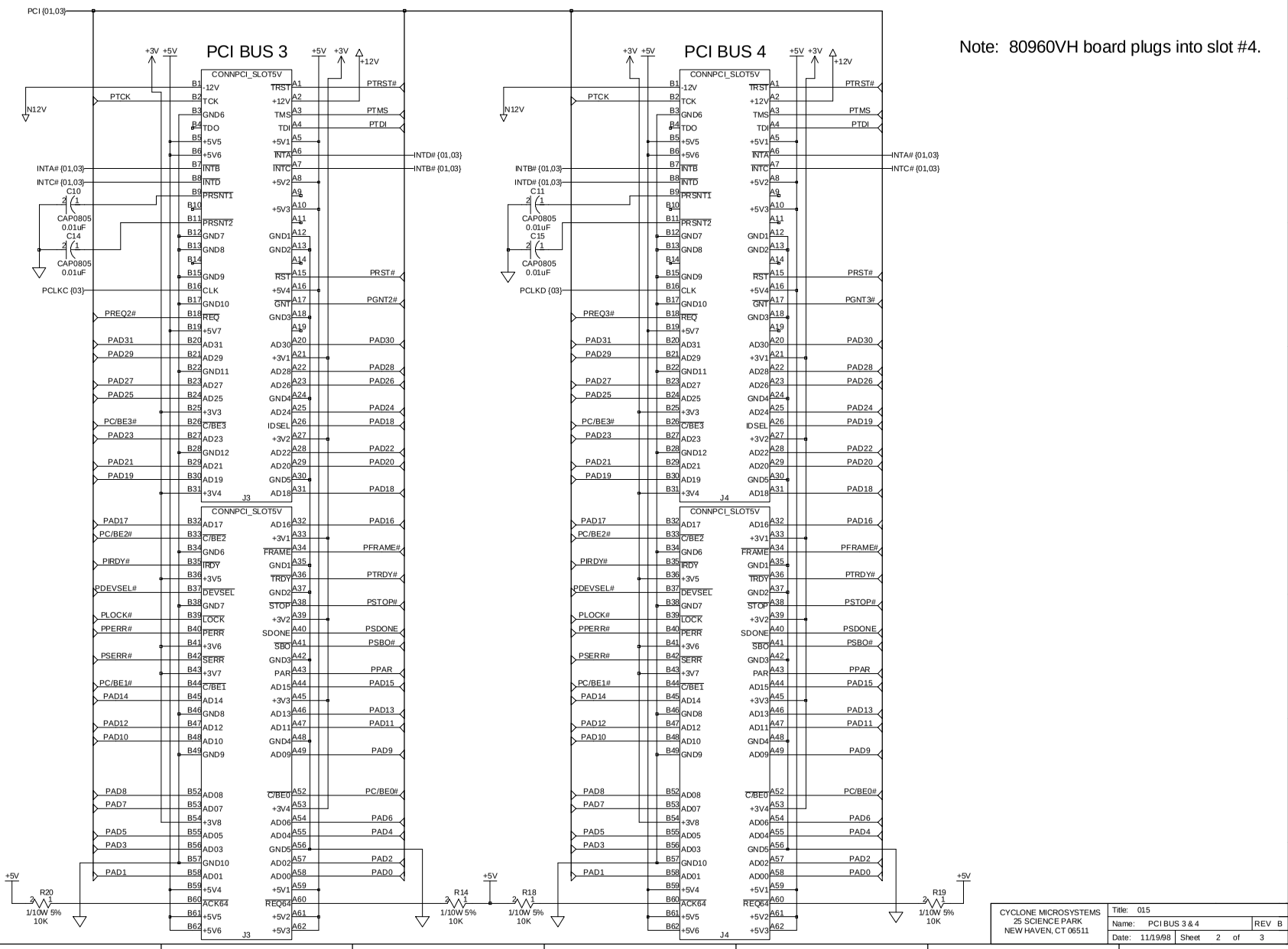
Cyclone P/N	Reference Designator	Description	Qty	Mfg	Mfg Part Number	Note
110-3210	C38	CAP TANT SM 33uf, 10V (7343)	1	Sprague	293D336X9016D 2T	(2)
110-3214	C35	CAP TANT SM 10uf 25/35V (7343)	1	Sprague	293D1060025D2 T	(2)
110-3301	C8-C15,C18-C25	CAP CERM SM, 0.01uf (0805)	16			(1)
110-3304	C1-C7,C17, C26-C30,C32-C34, C36,C37	CAP CERM SM, 0.1uf (0805)	18			(1)
126-0000-05	R21	R/SM 1/10W 5% 000 ohm (0805)	1			(1)
126-1004-05	R5,R13-R20,R22,R24	R/SM 1/10W 5% 10K ohm (0805)	11			(1)
126-2703-05	R6,R23,R25	R/SM 1/10W 5% 2.7k ohm (0805)	3			(1)
126-3301-05	R1,R9	R/SM 1/10W 5% 33 ohm (0805)	2			(1)
126-3302-05	R4	R/SM 1/10W 5% 330 ohm (0805)	1			(1)
126-4702-05	R3	R/SM 1/10W 5% 470 ohm (0805)	1			(1)
129-2101	R2,R10-R12	Resistor Pk SM RNC4R8P 2.7kohm	4	CTS	742083272JTR	(2)
129-2105	R7	Resistor Pk SM RNC4R8P 4.7kohm	1	CTS	742083472JTR	(2)
129-2113	R8	Resistor Pk SM RNC4R8P 33 ohm	1	CTS	742083330JTR	(2)
129-2102	R301	Resistor Pk SM 10K ohm	1	CTS	7420B3103JTR	
130-1460	J1-J4	CONN PCI SLOT 5V/PCB thru hole	4	AMP	145154-4	
130-1519	J6	CONN PCPWR Male /Mini-Fit/20p	1	Molex	39-29-9202	
130-1580	J5	CONN Hdr 26 pin/w shell, PCB	1	AMP	104339-6	
130-1621	Z1,Z2	Jumper JUMP2X1	2	Molex	22-28-4023	
150-4000	SW1,SW2	Switch, SW-PUSH 3P (Right Ang)	2	C&K Switch	EP12SD1ABE	
150-6013	U1	OSC 33.0 MHz Half	1	Ecliptek	ECI900HS33.000 M	
150-9100	CR1,CR2	LED Green	2	Hewlett Packard	HLMP-3507$010	
160-1108	CR3	Diode SM BAS16	1	National Semi.	BAS16	(2)
193-2002	U2	SOCKET PLCC44 LP Surface Mount	1	AMP	822275-1	
230-1197	MT1-MT4	SCREW-NYLON 4/40x3/8" Pan Slot	4	McMaster Carr	95000A108	(3)
230-1507	MT1-MT4	HARDWARE 4-40 x 3/8" Standoff	4	Keystone	1902B (0.375")	(3)
270-0015		PCB 0015A / VH Extender	1			
Note (1) - Provided by assembly house by specified description						

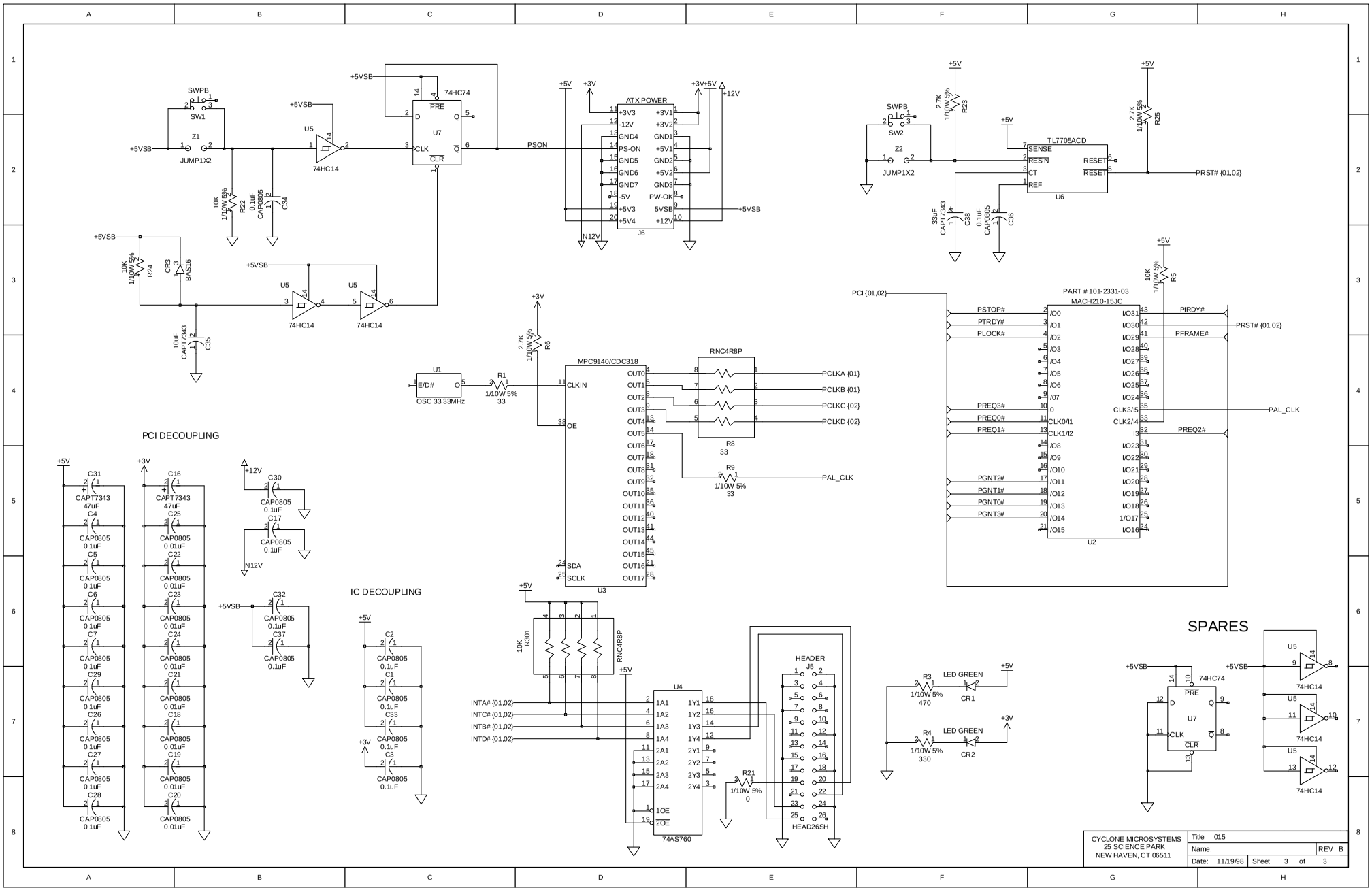
Table D-2. Backplane 0015 Interface Bill of Material (Sheet 3 of 3)

Cyclone P/N	Reference Designator	Description	Qty	Mfg	Mfg Part Number	Note
Note (2) - Provided by Cyclone - stocked at assembly house						
Note (3) - Reference assembly drawing						
Note (4) - Pre-programmed device						

D.10 Schematics


PCI BUS 1
PCI BUS 2
CONNPCI_SLOT5V
J1
J2
PCI {02,03}
N12V
PTCK
PTMS
PTDI
PTRST#
INTA# {02,03}
INTB# {02,03}
INTC# {02,03}
INTD# {02,03}
PRST#
PCLKA {03}
PCLKB {03}
PREQ0#
PREQ1#
PGNT0#
PGNT1#
PAD31
PAD29
PAD27
PAD25
PC/BE3#
PAD23
PAD21
PAD19
PAD17
PC/BE2#
PIRDY#
PDEVSEL#
PLOCK#
PPERR#
PSERR#
PC/BE1#
PAD14
PAD12
PAD10
PAD8
PAD7
PAD5
PAD3
PAD1
PAD30
PAD28
PAD26
PAD24
PAD16
PAD22
PAD20
PAD18
PFRAME#
PTRDY#
PSTOP#
PSDONE
PSBO#
PPAR
PAD15
PAD13
PAD11
PAD9
PC/BE0#
PAD6
PAD4
PAD2
PAD0
C8
C12
C9
C13
CAP0805
0.01uF
R15
R13
R16
R17
1/10W 5%
10K
RNC4R8P
R7
4.7K
R10
2.7K
R11
2.7K
R2
2.7K
R12
2.7K
PDEVSEL#
PSDONE
PPERR#
PREQ1#
PREQ0#
PREQ3#
CYCLONE MICROSYSTEMS
25 SCIENCE PARK
NEW HAVEN, CT 06511
Title: 015
Name: PCI BUS 1 & 2
REV B
Date: 11/19/98
Sheet 1 of 3


PCI BUS 3
PCI BUS 4
Note: 80960VH board plugs into slot #4.
CONNPCI_SLOT5V
J3
J4
PCI {01,03}
PCLKC {03}
PCLKD {03}
PTCK
PTMS
PTDI
PTRST#
PRST#
PGNT2#
PGNT3#
INTA# {01,03}
INTB# {01,03}
INTC# {01,03}
INTD# {01,03}
N12V
+3V
+5V
+12V
C10
C11
C14
C15
CAP0805
0.01uF
R20
R14
R18
R19
1/10W 5%
10K
PREQ2#
PREQ3#
PAD31
PAD29
PAD27
PAD25
PC/BE3#
PAD23
PAD21
PAD19
PAD17
PC/BE2#
PIRDY#
PDEVSEL#
PLOCK#
PPERR#
PSERR#
PC/BE1#
PAD14
PAD12
PAD10
PAD8
PAD7
PAD5
PAD3
PAD1
PAD30
PAD28
PAD26
PAD24
PAD18
PAD22
PAD20
PAD16
PFRAME#
PTRDY#
PSTOP#
PSDONE
PSBO#
PPAR
PAD15
PAD13
PAD11
PAD9
PC/BE0#
PAD6
PAD4
PAD2
PAD0
CYCLONE MICROSYSTEMS
25 SCIENCE PARK
NEW HAVEN, CT 06511
Title: 015
Name: PCI BUS 3 & 4
REV B
Date: 11/19/98
Sheet 2 of 3



CYCLONE MICROSYSTEMS 25 SCIENCE PARK NEW HAVEN, CT 06511	Title: 015 Name: REV B Date: 11/19/98 Sheet 3 of 3

D.11 PAL Code

```
   TITLE   PCI BUS ARBITRATION
 PATTERN   D703
REVISION   C
  AUTHOR   Joe Niedermeyer
 COMPANY   Cyclone Microsystems
    DATE   10-26-94

; D703B4  3-07-95  jpn  Fixing PCI bus St Mach for retry.
; D703B   3-07-95  jpn  Same as D703B4
; D703C1  3-08-95  jpn  Adding TRDY*, LOCK* and STOP*, just in case.

CHIP   D703   MACH210

;** INPUT PINS **;
pin  35 CLK              ; 33MHz system clk

pin  10 HREQn            ; Host (PCI9060) bus request
pin  11 AREQn            ; PCI expansion slot "A" bus request
pin  13 BREQn            ; PCI expansion slot "B" bus request
pin  32 CREQn            ; PCI expansion slot "C" bus request
pin  33 DREQn            ; PCI expansion slot "D" bus request

pin  41 FRAMEn           ; PCI bus FRAME# signal
pin   3 TRDYn            ; PCI bus target ready signal
pin  43 IRDYn            ; PCI bus initiator ready signal
pin   2 STOPn            ; PCI bus STOP# signal
pin   4 LOCKn            ; PCI bus LOCK# signal
pin  42 ASYNCRSTn        ; system reset, asyncronous, active low

;** OUTPUT PINS **;
pin  20 HGNTn      REG   ; Host (PCI9060) bus grant
pin  19 AGNTn      REG   ; PCI expansion slot "A" bus grant
pin  18 BGNTn      REG   ; PCI expansion slot "B" bus grant
pin  17 CGNTn      REG   ; PCI expansion slot "C" bus grant
pin  21 DGNTn      REG   ; PCI expansion slot "D" bus grant

pin  39 DUMMY1     COMB  ; dummy output to use TRDY, LOCK and STOP

;** BURIED SIGNALS **;
NODE  3   P0         REG   ; priority state variable
NODE  7   P1         REG   ; priority state variable
NODE 11   P2         REG   ; priority state variable
NODE 15   P3         REG   ; priority state variable

NODE  6   B0         REG   ; PCI bus state variable
NODE 10   B1         REG   ; PCI bus state variable

NODE 34   T0         REG   ; time-out counter bit
NODE 36   T1         REG   ; time-out counter bit
NODE 38   T2         REG   ; time-out counter bit
```

```
NODE 49   T3         REG   ; time-out counter bit

NODE 37   TIMEOUT    REG   ; time-out node
NODE 53   RST1n      REG   ; reset syncronizing register
NODE 54   RESETn     REG   ; syncronous reset

NODE 14   AGENTQUIT  COMB  ; granted bus agent deasserts bus request

STRING  sH  '( /P3 * /P2 * /P1 * /P0 )'
STRING  sA  '( /P3 * /P2 * /P1 *  P0 )'
STRING  sB  '( /P3 * /P2 *  P1 * /P0 )'
STRING  sC  '( /P3 *  P2 * /P1 * /P0 )'
STRING  sD  '(  P3 * /P2 * /P1 * /P0 )'

STRING  sGRANTED  '( /B1 * /B0 )'
STRING   sPARKED  '(  B1 *  B0 )'
STRING      sOFF  '( /B1 *  B0 )'
STRING   sACTIVE  '(  B1 * /B0 )'


EQUATIONS

HGNTn.CLKF = CLK       HGNTn.SETF = GND
AGNTn.CLKF = CLK       AGNTn.SETF = GND
BGNTn.CLKF = CLK       BGNTn.SETF = GND
CGNTn.CLKF = CLK       CGNTn.SETF = GND
DGNTn.CLKF = CLK       DGNTn.SETF = GND

   P0.CLKF = CLK          P0.RSTF = /ASYNCRSTn
   P1.CLKF = CLK          P1.RSTF = /ASYNCRSTn
   P2.CLKF = CLK          P2.RSTF = /ASYNCRSTn
   P3.CLKF = CLK          P3.RSTF = /ASYNCRSTn

   B0.CLKF = CLK          B0.SETF = GND
   B1.CLKF = CLK          B1.SETF = GND

   T0.CLKF = CLK          T0.SETF = GND
   T1.CLKF = CLK          T1.SETF = GND
   T2.CLKF = CLK          T2.SETF = GND
   T3.CLKF = CLK          T3.SETF = GND

 RST1n.CLKF = CLK       RST1n.SETF = GND
RESETn.CLKF = CLK      RESETn.SETF = GND

TIMEOUT.CLKF = CLK     TIMEOUT.SETF = GND

;*******************************;
;** KEEP UNUSED INPUTS IN FILE **;
;*******************************;

   DUMMY1 = TRDYn * STOPn * LOCKn

;*******************************;
;** SYNCRONOUS RESET GENERATION **;
;*******************************;
```

```
    RST1n :=  ASYNCRSTn
   RESETn :=  RST1n

;*********************************;
;** BUS GRANT TIMEOUT EQUATIONS **;
;*********************************;

     T0 := RESETn *  sGRANTED *  IRDYn * /T0
     T1 := RESETn *  sGRANTED *  IRDYn * /T1 *  T0
         + RESETn *  sGRANTED *  IRDYn *  T1 * /T0
     T2 := RESETn *  sGRANTED *  IRDYn * /T2 *  T1 *  T0
         + RESETn *  sGRANTED *  IRDYn *  T2 * /T1 * /T0
         + RESETn *  sGRANTED *  IRDYn *  T2 * /T1 *  T0
         + RESETn *  sGRANTED *  IRDYn *  T2 *  T1 * /T0
     T3 := RESETn *  sGRANTED *  IRDYn * /T3 *  T2 *  T1 *  T0
         + RESETn *  sGRANTED *  IRDYn *  T3 * /T2 * /T1 * /T0
         + RESETn *  sGRANTED *  IRDYn *  T3 * /T2 * /T1 *  T0
         + RESETn *  sGRANTED *  IRDYn *  T3 * /T2 *  T1 * /T0
         + RESETn *  sGRANTED *  IRDYn *  T3 * /T2 *  T1 *  T0
         + RESETn *  sGRANTED *  IRDYn *  T3 *  T2 * /T1 * /T0
         + RESETn *  sGRANTED *  IRDYn *  T3 *  T2 * /T1 *  T0
         + RESETn *  sGRANTED *  IRDYn *  T3 *  T2 *  T1 * /T0

TIMEOUT := T3 * T2 * T1 * T0

AGENTQUIT  =  /HGNTn *  HREQn
           +  /AGNTn *  AREQn
           +  /BGNTn *  BREQn
           +  /CGNTn *  CREQn
           +  /DGNTn *  DREQn

;************************;
;** BUS GRANT EQUATIONS **;
;************************;

 /HGNTn := sH * sGRANTED * /TIMEOUT
         + sH * sACTIVE
         + sH * sPARKED
 /AGNTn := sA * sGRANTED * /TIMEOUT
         + sA * sACTIVE
         + sA * sPARKED
 /BGNTn := sB * sGRANTED * /TIMEOUT
         + sB * sACTIVE
         + sB * sPARKED
 /CGNTn := sC * sGRANTED * /TIMEOUT
         + sC * sACTIVE
         + sC * sPARKED
 /DGNTn := sD * sGRANTED * /TIMEOUT
         + sD * sACTIVE
         + sD * sPARKED

 ;**************************;
 ;** PCI BUS STATE MACHINE **;
 ;**************************;
```

```
;**
;**         STATE TRANSITIONS
;**
;**          sPARKED :=  (any REQn=0) -> sOFF
;**                   +  (all REQn=1) -> sPARKED
;**
;**              sOFF :=  (FRAMEn=1 & IRDYn=1) -> sGRANTED
;**                   +            (FRAMEn=0) -> sACTIVE
;**                   +  (FRAMEn=1 & IRDYn=0) -> sACTIVE
;**
;**         sGRANTED :=  (FRAMEn=0 & ANY GNT=0) -> sOFF
;**                   +  (FRAMEn=0 & ALL GNT=1) -> sACTIVE
;**                   +  (FRAMEn=1 & TIMEOUT=0) -> sGRANTED
;**                   +  (FRAMEn=1 & TIMEOUT=1) -> sPARKED
;**                   +  (AGENTQUIT=1) & ALL REQ=1) -> sPARKED
;**                   +  (AGENTQUIT=1) & ANY REQ=0) -> sOFF
;**
;**          sACTIVE :=  (any REQn=0 & FRAMEn=1) -> sGRANTED
;**                   +  (all REQn=1 & FRAMEn=1) -> sPARKED
;**                   +               (FRAMEn=0) -> sACTIVE
;**

B0 :=  sPARKED *  AREQn *  BREQn *  CREQn *  DREQn *  HREQn
    +  sPARKED * (/AREQn + /BREQn + /CREQn + /DREQn + /HREQn)
    + sGRANTED * /FRAMEn * (/AGNTn + /BGNTn + /CGNTn + /DGNTn + /HGNTn)
    + sGRANTED *  FRAMEn *  TIMEOUT
    + sGRANTED *  AGENTQUIT
    +  sACTIVE *  FRAMEn *  AREQn *  BREQn *  CREQn *  DREQn *  HREQn
    + /RESETn

B1 :=  sPARKED *  AREQn *  BREQn *  CREQn *  DREQn *  HREQn
    +     sOFF * /FRAMEn
    +     sOFF *  FRAMEn * /IRDYn
    + sGRANTED *  FRAMEn *  TIMEOUT
    + sGRANTED * /FRAMEn *  AGNTn *  BGNTn *  CGNTn *  DGNTn *  HGNTn
    + sGRANTED *  AGENTQUIT *  AREQn *  BREQn *  CREQn *  DREQn *  HREQn
    +  sACTIVE * /FRAMEn
    +  sACTIVE *  FRAMEn *  AREQn *  BREQn *  CREQn *  DREQn *  HREQn
    + /RESETn

;*****************************;
;**  PRIORITY STATE MACHINE  **;
;*****************************;

STATE
MOORE_MACHINE
DEFAULT_BRANCH HOLD_STATE

;**  STATE ASSIGNMENT EQUATIONS **;

 H = /P3 * /P2 * /P1 * /P0       ; This state assignment results
 A = /P3 * /P2 * /P1 *  P0       ; in 11 product terms per variable.
 B = /P3 * /P2 *  P1 * /P0       ; The conventional assignment
 C = /P3 *  P2 * /P1 * /P0       ; results in 14 or more product
```

```
  D =  P3 * /P2 * /P1 * /P0       ; terms for some variables.

;**  STATE TRANSITION EQUATIONS **;

      H :=  H_TO_A -> A
         +  H_TO_B -> B
         +  H_TO_C -> C
         +  H_TO_D -> D

      A :=  A_TO_B -> B
         +  A_TO_C -> C
         +  A_TO_D -> D
         +  A_TO_H -> H

      B :=  B_TO_C -> C
         +  B_TO_D -> D
         +  B_TO_H -> H
         +  B_TO_A -> A

      C :=  C_TO_D -> D
         +  C_TO_H -> H
         +  C_TO_A -> A
         +  C_TO_B -> B

      D :=  D_TO_H -> H
         +  D_TO_A -> A
         +  D_TO_B -> B
         +  D_TO_C -> C

 CONDITIONS

;**  STATE CONDITION EQUATIONS **;

 H_TO_A = /AREQn * sOFF

 H_TO_B =  AREQn * /BREQn * sOFF

 H_TO_C =  AREQn *  BREQn * /CREQn * sOFF

 H_TO_D =  AREQn *  BREQn *  CREQn * /DREQn * sOFF


 A_TO_B = /BREQn * sOFF

 A_TO_C =  BREQn * /CREQn  * sOFF

 A_TO_D =  BREQn *  CREQn * /DREQn * sOFF

 A_TO_H =  BREQn *  CREQn *  DREQn * /HREQn * sOFF


 B_TO_C = /CREQn * sOFF

 B_TO_D =  CREQn * /DREQn  * sOFF
```

```
B_TO_H =  CREQn *  DREQn * /HREQn * sOFF

B_TO_A =  CREQn *  DREQn *  HREQn * /AREQn * sOFF


C_TO_D = /DREQn * sOFF

C_TO_H =  DREQn * /HREQn * sOFF

C_TO_A =  DREQn *  HREQn * /AREQn * sOFF

C_TO_B =  DREQn *  HREQn *  AREQn * /BREQn * sOFF


D_TO_H = /HREQn * sOFF

D_TO_A =  HREQn * /AREQn * sOFF

D_TO_B =  HREQn *  AREQn * /BREQn * sOFF

D_TO_C =  HREQn *  AREQn *  BREQn * /CREQn * sOFF
```