SCRAMNet® SC150 Network

Rehostable Adapter
Hardware Reference

Document No. D-T-MR-REHOST##-A-0-A3

FOREWORD

SCRAMNet® is a registered trademark of Systran Corporation. US Patent # 4,928,289.
Systran™ is a trademark of Systran Corporation.
TRI-STATE® is a registered trademark of National Semiconductor Corporation.

Revised: April 18, 2000

Systran Corporation
4126 Linden Avenue
Dayton, OH 45432-3068 USA
(800) 252-5601

FCC

This product is intended for use in industrial, laboratory or military environments. This product uses and emits electromagnetic radiation, which may interfere with other radio and communication devices. The user may be in violation of FCC regulations if this device is used in other than the intended market environments.

CE

As a component part of another system, this product has no intrinsic function and is therefore not subject to the European Union CE EMC directive 89/336/EEC.

TABLE OF CONTENTS

APPENDICES

FIGURES

TABLES

This page intentionally left blank

1. INTRODUCTION

1.1 How to Use This Manual

1.1.1 Purpose

The purpose of this document is to aid the engineer integrating the SCRAMNet SC150 Rehostable Adapter to a specific design. This will be accomplished by defining the signal interface, defining the interface requirements, and by showing an implementation example. Information is also provided on memory subsystems and hardware assembly. A section on LEDS Support Circuitry is also included.

☞ **NOTE**: SCRAMNet SC150 (previously SCRAMNet+) is referred to in this manual as "SCRAMNet."

1.1.2 Scope

This document is intended for SCRAMNet network system engineers. To use this manual effectively you need an understanding of the SCRAMNet network hardware.

1.1.3 Style Conventions

The following conventions are used in this document:

- The CTRL C symbol indicates that you must press the CTRL key while you simultaneously press another key (in this case, C).
- Names of files, parameters and commands are in bold type; for example, **sysconfigtab** file.
- Names of called routines are followed by open and closed parentheses and are printed in italics; for example, *dma_read()*.
- Directory path names are in italics; for example, *mkdir /usr/*SCRAMNet.
- VHDL is boxed, Courier, 9 pt font.
- Hexadecimal values within paragraph text are written with the word hex italicized and one point smaller than the context font following the value; for example, 03FF *hex*. Hexadecimal values in tables or in code blocks may be written using abbreviated notation; for example 0x03FF.
- CSR and ACR Register bits and bit ranges are specified by the register identification followed by the bit or range of bits in brackets [ ]; for example, CSR6[4], CSR3[15:0], ACR[2]
- In a prompt, square brackets indicate that the enclosed item is the default response. For example, [y] means the default response is Yes.

1.2 Related Information

- *SCRAMNet Network Media User's Guide*
  (Document No. D-T-MU-MEDIA)
- *SCRAMNet Utilities User Manual*
  (Document No. C-T-MU-UTIL)

1.3 Quality Assurance

Systran Corporate policy is to provide our customers with the highest quality products and services. In addition to the physical product, the company provides documentation, sales and marketing support, hardware and software technical support, and timely product delivery. Our quality commitment begins with product concept, and continues after receipt of the purchased product.

Systran's Quality System conforms to the ISO 9001 international standard for quality systems. ISO 9001 is the model for quality assurance in design, development, production, installation and servicing. The ISO 9001 standard addresses all 20 clauses of the ISO quality system and is the most comprehensive of the conformance standards.

Our Quality System addresses the following basic objectives:

- Achieve, maintain and continually improve the quality of our products through established design, test, and production procedures.
- Improve the quality of our operations to meet the needs of our customers, suppliers, and other stakeholders.
- Provide our employees with the tools and overall work environment to fulfill, maintain, and improve product and service quality.
- Ensure our customer and other stakeholders that only the highest quality product or service will be delivered.

The British Standards Institution (BSI), the world's largest and most respected standardization authority, assessed Systran's Quality System. BSI's Quality Assurance division certified we meet or exceed all applicable international standards, and issued Certificate of Registration, number FM 31468, on May 16, 1995. The scope of Systran's registration is: “Design, manufacture and service of high technology hardware and software computer communications products.” The registration is maintained under BSI QA's bi-annual quality audit program.

Customer feedback is integral to our quality and reliability program. We encourage customers to contact us with questions, suggestions, or comments regarding any of our products or services. We guarantee professional and quick responses to your questions, comments, or problems.

1.4 Technical Support

Technical documentation is provided with all of our products. This documentation describes the technology, its performance characteristics, and includes some typical applications. It also includes comprehensive support information, designed to answer any technical questions that might arise concerning the use of this product. We also publish and distribute technical briefs and application notes that cover a wide assortment of topics. Although we try to tailor the applications to real scenarios, not all possible circumstances are covered.

Although we have attempted to make this document comprehensive, you may have specific problems or issues this document does not satisfactorily cover. Our goal is to offer a combination of products and services that provide complete, easy-to-use solutions for your application.

If you have any technical or non-technical questions or comments (including software), contact us. Hours of operation are from 8:00 a.m. to 5:00 p.m. Eastern Standard/Daylight Time.

- Phone: **(937) 252-5601** or **(800) 252-5601**
- E-mail: **support@systran.com**
- Fax: **(937) 252-1349**

1.5 Ordering Process

To learn more about Systran products or to place an order, please use the following contact information. Hours of operation are from 8:00 a.m. to 5:00 p.m. Eastern Standard/Daylight Time.

- Phone: **(937) 252-5601** or **(800) 252-5601**
- E-mail: i**nfo@systran.com**
- World Wide Web address: **www.systran.com**

This page intentionally left blank

2. PRODUCT OVERVIEW

2.1 Overview

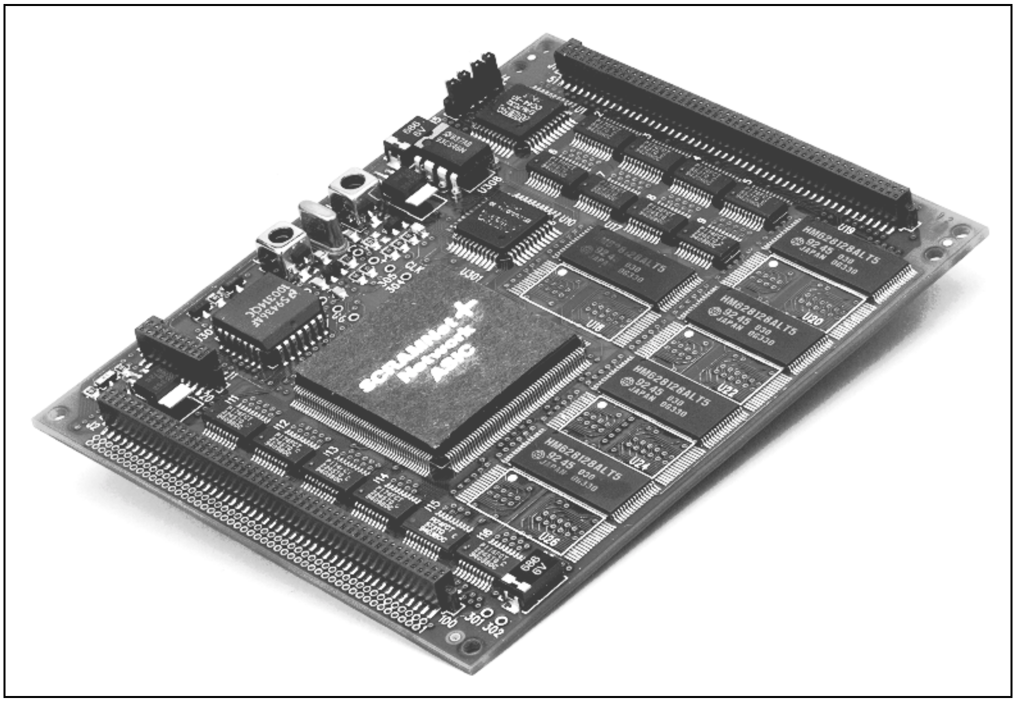


Figure 2-1 Rehostable Adapter

2.2 Description

The SCRAMNet Rehostable Adapter is a 3.3" x 4.0" circuit card holding the SCRAMNet "core" circuitry. The following is required to obtain a working SCRAMNet node:

- +5 V @ 0.7 Amps
- A connection to a Systran Media card (the physical media interface)
- User-provided circuitry to interface the Rehostable Adapter to the users design

The goal of the SCRAMNet Rehostable Adapter is to provide an affordable "core" piece of a SCRAMNet node for use in embedded applications.

2.3 Specifications

The Rehostable Adapter is the "core" of a SCRAMNet node and therefore provides the following specifications:

- Connection to any Systran supported physical media
- 16.7 MB/s throughput (MAX)

- Maximum 2 MB on-card SCRAMNet memory (optional 4 MB or 8 MB memory using an expansion card)
- Shared memory and Error based interrupt support
- Shared memory and CSR address-hit comparators (requires additional external circuitry)
- Full SCRAMNet hardware and software compatibility

3. SIGNAL DEFINITION

3.1 Overview

This section contains a complete pinout description, definition of the Control/Status Registers (CSRs), and a detailed description of the interface signals.

3.2 Pinout Description

This is a listing of the I/O pin definitions for the Rehostable Adapter. All active low signals have '~' as their first character. The bold, italicized signals must be driven by host-specific Hardware.

Rehostable Adapter

HREQ	Host Request
R_W	Read / Write
TS1	Transaction Select 1
TS2	Transaction Select 2
INT_PEND	Host Interrupt Pending line
***HA*[31:2]**	Host Address Port
***HD*[31:0]**	Host Data bus
***TSP*[2:0]**	Transaction Size and Position
HACK	Host Cycle Acknowledge
A_D_LTCH	Address Data Latch (Latch Data signal during a write)
HREQ_PEND	Host Request Pending (Handshake line showing a pending host request)
OE_A2B[3:0]	Output Enable
CNTRL_STATUS[20,18,17,16, 7]	Control Status bits
CS_MODE	Control Status Mode (1 = Non-Network)
STS1	Status of Transaction Select 1
STS2	Status of Transaction Select 2

Memory Control Port

MA[22:2]	Memory Address bus
MD[31:0]	Memory Data bus
MBCS	External Memory Board Chip Select
~MWE[3:0]	Memory Write Enables
ACR[7:0]	ACR bus
MEM_DET	Memory Detect for External RAM
~M_DOE	Memory Data Output Enable
~ACR_DOE	ACR Data Output Enable
~ACR_WE	ACR Write Enable
MCI[2:0]	Memory Size Configuration Bus

Address Comparator

MEM_HIT	Memory Address Hit

CSR_ME[1:0]	CSR Map Code

Misc & Control

~RST_IN	Rest Input (External power monitor pin - must be (implemented by host)
BUSY	EEPROM Read is active - ***HREQ***'s will not be acknowledged
CLK	37.5 MHz clock (26.66 ns) Output
~SW_OE0	Switch Read Output Enable

Media Card Interface

LINK_A_B	Redundant link A or B selection
L_INSERT_LED	Insert Status
L_CD_LED	Carrier Detect Status
F_RELAY	Fiber Optic Relay Sense and Control
L_MECHSW	Mechanical Switch Output
S_CLK	Serial Output Clock
S_DATA	Bi-directional Serial Data
S_DIR	Serial Data Direction Indicator
Trigger	Single multiplexed line containing external trigger information
EXT_PWR	External power output +5 Vdc @ 500 mA minimum guaranteed 2 A maximum current limited
~TX0\TX0	Differential 100 K ECL pair containing channel 0 transmit data
~TX1\TX1	Differential 100 K ECL pair containing channel 1 transmit data
~RX0\RX0	Differential 100 K ECL pair containing channel 0 receive data
~RX1\RX1	Differential 100 K ECL pair containing channel 1 receive data

Testing

TEST	(Not used in normal operation) = GND
TEST_WR	(Not used in normal operation) = GND

3.3 CSR Registers

The following list describes the internal registers of the Rehostable Adapter.

CSRs specific to the Rehostable Adapter have a description that starts with “Rehostable Adapter”.

CSR0	General SCRAMNet Control
CSR1	SCRAMNet Errors
CSR2	General SCRAMNet Control
CSR3	Number of Nodes and Tx Node ID (Rx ID and Transmitted AGE)
CSR4	Interrupt FIFO Address (Lower 16 bits)
CSR5	Interrupt FIFO Address (Upper 7 bits)
CSR6	Not implemented in the Rehostable Adapter design
CSR7	Not implemented in the Rehostable Adapter design
CSR8	Rehostable Adapter ASIC extended control and status
CSR9	Rehostable Adapter Error Interrupt MASK Register
CSR10	Rehostable Adapter Shared Memory Address and ENABLE (LSW), replaces physical switches
CSR11	Rehostable Adapter Shared Memory Address (MSW), replaces physical switches
CSR12	Rehostable Adapter Virtual Paging with Enable
CSR13	Rehostable Adapter General Purpose Counter/Timer
CSR14	Not implemented in the Rehostable Adapter design
CSR15	Not implemented in the Rehostable Adapter design
CSR16	Not implemented in the Rehostable Adapter design

See Appendix A for a detailed description of each CSR.

3.4 Detailed Description of Signals

This subsection provides a more detailed description of the Rehostable Adapter interface signals. The signals are arranged in alphabetical order. The capacitance loading values for each of the signals is contained in Table 3-8 Rehostable Adapter Signal Capacitance at the end of the section.

ACR[7:0]

The ACR bus is a TTL level (asserted HI) set of signals provided from the memory module during a memory access. These control various functions/features of the Rehostable Adapter product. Table 3-1 shows the definitions of the use of the ACR bus.

Table 3-1 ACR Bus Definitions

Bit #	Function
0	Receive Interrupt Enable
1	Transmit Interrupt Enable
2	External Trigger 1
3	External Trigger 2
4	HIPRO
5	External Trigger 3 (Internal Rehostable Adapter ASIC 4 K only)
6	Undefined

~ACR_DOE

The ACR Data Output Enable signal is an active low TTL level signal that is sourced from the Rehostable Adapter ASIC to the memory module. This signal instructs the memory module to place ACR information onto the MD bus.

~ACR_WE

The ACR Write Enable signal is an active low TTL level signal that is sourced from the Rehostable Adapter ASIC to the memory module. This signal instructs the memory module to write the present data on the MD bus to the ACR location specified by the MA bus.

A_D_LTCH

The A_D_LATCH signal latches several host-specific signals during a host-specific transaction with the Rehostable Adapter ASIC. These signals are ***HD***[31:0], ***HA***[22:2], ***R_W***, TS[2:1] and ***TSP***[2:0]. When A_D_LATCH occurs, these signals must be valid so they may be latched. All latching occurs inside the Rehostable Adapter ASIC except for latching ***HD***[31:0] into the external pipeline registers.

BUSY

The BUSY signal is an active high TTL level signal sourced from the Rehostable Adapter ASIC. This signal becomes active whenever reset activities start (that is, when ~RST_IN goes low). This signal will remain active until reset processing is finished (that is, when the configuration EEPROM is read).

CLK

The Clock signal is an active high TTL level signal sourced from the Rehostable Adapter ASIC. This clock is used for the entire Rehostable Adapter bus and user interface timing. It is a 37.5 MHz, 50% duty cycle-clock source. All Rehostable Adapter ASIC transactions are synchronous to this clock.

CNTRL_STATUS[20,18,17,16,7]

These bus signals are TTL logic level HI sourced from the Rehostable Adapter ASIC. It is a multiplexed bus that is based on the CS_MODE indicator. When CS_MODE is asserted (TTL logic level HI), then the signals are defined as “non-network”. When CS-MODE de-asserts (TTL logic level LOW), the CNTRL_STATUS will be in transition within two host clocks.

Non-network required (user determined) CNTRL_STATUS signals should be “Flow Through” latched on the CS_MODE signal. The CNTRL_STATUS bus may be latched in the network mode with the signal NACK. Figure 3-1 shows the timing relationship between CS_MODE and the CNTRL_STATUS bus. CS_MODE as shown only goes low during a network cycle (shown by BGN [Bus Grant Network]). BGN, Label 1, causes CS_MODE to go low. Only after the CS_MODE is low for at least one host Clock does the CNTRL_STATUS bus change. Also, note from the timing that the CNTRL_STATUS bus has setup and hold times defined from the NACK (Network Cycle Acknowledge) signal; this is indicated by labels 5a and 5b. The rising edge of NACK can be used to latch the CNTRL_STATUS bus, assuring there will be no setup or hold violations. The falling edge of NACK causes CS-MODE to rise.

There are two fundamental definitions of each CNTRL_STATUS line. These definitions correspond to network and non-network transactions occurring on the Rehostable Adapter bus. Table 3-2 defines the CNTRL_STATUS bus in each of the modes.

Table 3-2 Control Status Bus Definition

CNTRL_STATUS	NETWORK (CS_MODE = 0)	NON-NETWORK (CS_MODE = 1)
20	RX_F_WR (Page 3-12)	TRIG3 (Page 3-13)
18	OID (Page 3-11)	INT_RXFIFO (Page 3-9)
17	RX_INT (Page 3-12)	INT_ERROR (Page 3-8)
16	RX_RETRY (Page 3-12)	RD_CSR5 (Page 3-11)
7	Not Used	INT_ARMED (Page 3-8)

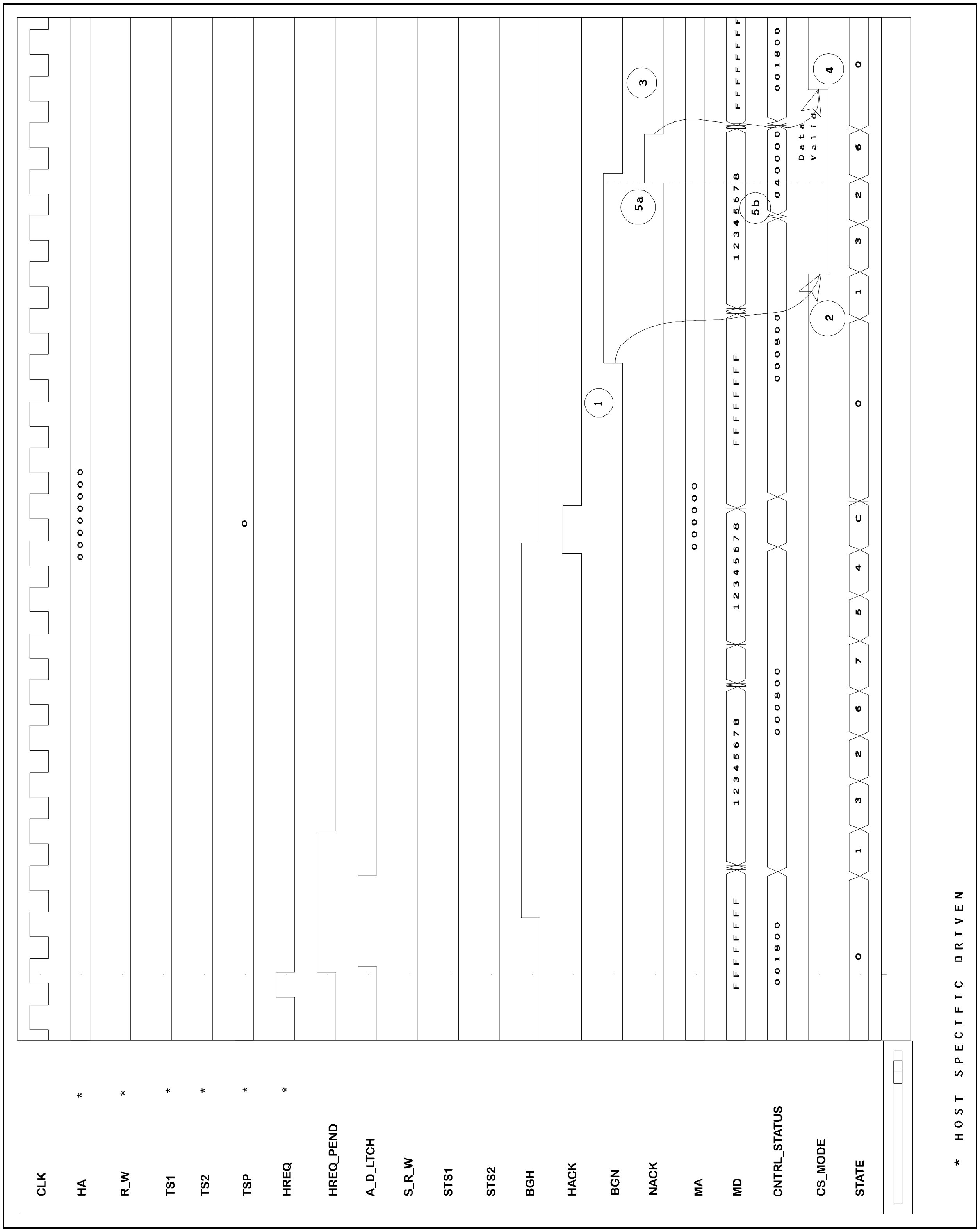


Figure 3-1 CS-Bus and CS Mode Timing

CSR_ME[1:0]

The CSR_ME bus is an active high TTL level bus sourced from the Rehostable Adapter ASIC. This bus is the output of the asynchronous CSR decode comparator logic built into the Rehostable Adapter ASIC. Specifically, this logic compares the ***HA***[31:2] bus to the local CSR address (set up by the host logic), and outputs a code on this line if the address is in the specified CSR size range. Table 3-3 contains the possible output codes found on the CSR_ME bus.

Table 3-3 CSR_ME Bus Output Codes

CSR_ME1	CSR_ME0	Definition
0	0	- No Access -
0	1	Internal Rehostable Adapter ASIC CSR's
1	0	- Illegal Code -
1	1	External Host Interface CSR’s

CS_MODE

The CS_MODE signal shows the state of the CNTRL_STATUS bus. When the CS_MODE is asserted (TTL logic level HI) then the CNTRL_STATUS bus contains signals primarily concerned with the Rehostable Adapter ASIC status (non-network mode). When CS_MODE is de–asserted (logic level LOW), the CNTRL_STATUS bus takes on definitions corresponding to network traffic. It is suggested that important non–network mode signals should be “Flow Through” latched on the CS_MODE signal. Ample set up and hold time are provided for this.

EXT_PWR

External power output +5 Vdc @ 500 mA minimum guaranteed 2 A maximum current limited.

F_RELAY

Fiber-optic relay drive and sense. Current limited and current sensing. It is CMOS driven - TTL sourced from the Rehostable Adapter ASIC.

HA[31:2]

The Host Address bus has a dual function. In its first capacity, it can be used to help decode the Rehostable Adapter devices physical address on the host computer system. In this function, it uses ***HA***[31:6] and compares these lines to both a physical memory address and a physical CSR address. These comparison bits are available to the host-specific logic for use. Transactions smaller than 32 bits are decoded on the ***TSP*** lines. In its second function, these lines provide the offset address for both CSR based and shared-memory based transactions. In these modes only lower address lines are used:

Memory.......................... ***HA***[22 (or lower):2]
CSR .... ***HA***[5:2]
NMT's .... ***HA***[22:2]

HACK

The Host Acknowledge signal comes from the Rehostable Adapter ASIC to the host-specific logic notifying the host-specific logic that the requested cycle has been completed. On a read cycle from the Rehostable Adapter ASIC, the data is available on the MD[31:0] on the rising edge of ***HACK***. This signal is only one clock wide.

HD[31:0]

The HD bus is a TTL level bi-directional bus of signals that connects the host interface to the data pipe.

HREQ

This line, when asserted (TTL level HI), notifies the Rehostable Adapter ASIC that the host-specific hardware has a transaction for the Rehostable Adapter ASIC to handle. At this time, the following input lines are required to be valid and stable:

R_W
TS1
TS2
HA[22.2]
TSP[2:0]
(***HD***[31:0] This is on the “Host” side of the data pipe.)

HREQ_PEND

The Host Request Pending signal is asserted (TTL logic level HI) from the Rehostable Adapter ASIC when the Rehostable Adapter ASIC has recognized the Host Request signal (***HREQ***). It will remain active through the start of the actual ASIC transaction that the host-specific logic has requested. The HREQ_PEND signal forms the handshake for the control lines to the host-specific interface. The host-specific logic is never allowed to assert ***HREQ*** while HREQ_PEND is asserted. Likewise, the host-specific logic must remove the ***HREQ*** when the HREQ_PEND is asserted or risk a potential double cycle.

INT_ARMED (See CNTRL_STATUS, page 3-5)

The Interrupt System Armed signal is asserted (TTL logic level HI) from the Rehostable Adapter ASIC during a host transaction or null time. This signal is found on CNTRL_STATUS[7] only during non-network mode (CS_MODE = HI). Specifically, this signal is status of the Rehostable Adapter interrupt engine. If this signal is asserted the Rehostable Adapter interrupt system has been programmed (software) for operation and has the potential to generate an interrupt. When an interrupt is generated, the host-specific logic is expected to return a “HI” on the ***INT_PEND*** signal to “stop” asserting the interrupt line.

INT_ARMED defines the “stopping” signal. No interrupt will ever be generated when INT_ARMED is DE_ASSERTED (LOW). Interrupt Armed can be re-asserted by a software write to CSR1 following selection of all preconditions.

INT_ERROR (See CNTRL_STATUS, page 3-5)

This signal is asserted (TTL logic level HI) from the Rehostable Adapter ASIC during a non-network transaction or null time (no Rehostable Adapter bus activity). This signal is found on CNTRL_STATUS[17] only during non-network mode (CS_MODE = HI). This signal is one of two signals used to generate interrupts from the Rehostable Adapter ASIC. This signal becomes active if the control status registers are set up for receiving interrupt messages from errors and the mask register is set to “watch” for specific errors. Specifically, this implies that an error condition has occurred and that interrupts are armed. For further information see 4.5.1 The SCRAMNet ASIC Interrupt System, page 4-11.

INT_PEND

The Host Interrupt Pending signal is used to prevent Rehostable Adapter from generating a second interrupt while the Host is handling a previous interrupt. The host-specific logic will generate an interrupt to the Host if INT_ARMED and either INT_FIFO or INT_ERROR is true. These signals can only be armed if CSRs are configured correctly and no previous interrupt is pending. If CSR1 is written to, it will re-arm interrupts. If INT_FIFO_EMPTY and/or INT_ERROR are still asserted when CSR1 is written to, Rehostable Adapter will generate another interrupt.

INT_RXFIFO

This signal is asserted (TTL logic level HI) from the Rehostable Adapter ASIC during a non-network transaction or null time (no Rehostable Adapter bus activity). This signal is found on CNTRL_STATUS[18] only during non-network mode (CS_MODE = HI). This signal is one of two signals used to generate interrupts from the Rehostable Adapter ASIC. INT_RXFIFO becomes active if the control status registers are set up for receiving interrupt messages from the network, and a qualifying message has come in. Specifically, this implies that the Interrupt FIFO contains interrupt information and that interrupts are armed. For further information see 4.5.1 The SCRAMNet ASIC Interrupt System.

LINK_A_B

When this signal is high, the Redundant Media Cards will send the “A” inputs to the Rehostable Adapter. When low, the “B” inputs will be sent to the Rehostable Adapter. This output is CMOS driven TTL levels to and from the host.

L_CD_LED

LED drive from host, +5 V when active (Carrier Detect), 7 mA current limit.

L_INSERT_LED

LED drive from host, +5 V when active (Insert), 7 mA current limit.

L_MECHSW

When this signal is HI, the Media Cards so equipped will switch to the insert mode. This output is CMOS driven TTL levels to and from the host.

MA[22:2]

The MA bus is a TTL level bus of signals that is primarily used to drive the external memory board. The address on these lines is a physical Rehostable Adapter address. Specifically, the address of an incoming network message will be reflected on these lines. This is true regardless of the memory size present or virtual paging (the MBCS line takes care of these issues for the memory card). The MA bus also reflects the ***HA*** bus for all host transactions.

NOTE: For CSR transactions, the memory size and virtual paging have no effect on the upper MA bus.

MBCS

The Memory Board Chip Select is a TTL-level signal that is a chip select (asserted HI) for the memory module. The memory modules are NOT responsible for decoding of the upper MA bus lines to determine if a memory hit has occurred. Instead the MBCS line is asserted to provide this function. This line is generated with the memory size and virtual paging parameters taken into account. It is used for off-card RAM expansion.

MCI[2:0]

The MCI bus is an active high TTL level bus sourced from the external memory system (that is, the host memory or the add-on memory module) to the Rehostable Adapter ASIC. This bus codes the amount of external memory present. Table 3-4 decodes this bus. (Note that this bus is available for the user to read in a control/status register).

Table 3-4 MCI Bus Codes

MCI[2:0]		Memory
Bin	**Dec**	**Size**
111	7	4 KB
110	6	128 KB
101	5	512 KB
100	4	1 MB
011	3	2 MB
010	2	4 MB
001	1	8 MB
000	0	- Invalid -

MD[31:0]

The MD bus is a TTL level bi-directional bus of signals used to move data in and out of the ASIC. The ASIC, the host-specific data pipeline (single level), and the memory module all share the MD bus.

NOTE: The Rehostable Adapter ASIC exclusively controls this bus.

MEM_DET

Memory detect for external RAM.

MEM_HIT

The Memory Hit signal is an active high TTL level signal sourced from the Rehostable Adapter ASIC. This signal is the output of the asynchronous memory decode comparator logic on the Rehostable Adapter ASIC. Specifically, this logic compares the ***HA***[31:2] bus to the internal memory address (specified in a CSR) and outputs a HI on this line if the address is in the specified memory size range. See the paragraph on the SCRAMNet ASIC Address Decoder and Its Application To the Rehostable Adapter, page 4-13, for more information.

~MWE[3:0]

The ~MWE bus is a TTL level (asserted LOW) set of signals which are write-enables corresponding to individual byte-lanes on the memory module. The most significant ~MWE line controls the most significant byte-lane of the MD bus. Table 3-5 shows the ~MWE bus lines and their corresponding byte-lanes.

Table 3-5 ~MWE Bus Lines and Byte-lanes

~MWE	MD
3	[31:24]
2	[23:16]
1	[15:8]
0	[7:0]

~M_DOE

The Memory Data Output Enable signal is a active low TTL level signal which is sourced from the Rehostable Adapter ASIC to the memory module. This signal instructs the memory module to drive the MD bus with data.

OE_A2B[3:0]

Enables data onto the HD bus during a write.

OID

The Our ID signal is asserted (TTL logic level HI) from the Rehostable Adapter ASIC during a network transaction. This signal is found on CNTRL_STATUS[18] only during network mode (CS_MODE = LOW). This signal is a status signal meaning that the ID field on the received network message matches this nodes receiver ID (RXID). Under “normal” operation of the Rehostable Adapter ASIC the transmitter ID (TXID) and the receiver ID (RXID) are equal, and therefore this signal would mean the message received belonged to this node.

RD_CSR5 (See CNTRL_STATUS, page 3-5)

This signal is asserted (TTL logic level HI) from the Rehostable Adapter ASIC during a non-network transaction or null time (no Rehostable Adapter bus activity). This signal is found on CNTRL_STATUS[16] only during non-network mode (CS_MODE = HI). This signal is asserted during a host transaction involving the MSB of CSR5. This register is the most significant part of the interrupt FIFO, and when read, causes an increment of that FIFO. This signal is available external to the Rehostable Adapter ASIC for possible extension of the interrupt FIFO.

~RST_IN

The Reset Input signal is an active-low TTL-level signal sourced from HOST-specific logic. This master reset signal initializes the Rehostable Adapter into a known state. Rehostable Adapter reset activity is initiated when this signal transitions from low to high.

NOTE: This signal must be implemented by the host

~RX0\RX0 / ~TX0/TX0

The receiver and transmitter connections are all 100 K differential level drive and sense. When using the coax versions, note that the signals are compatible with 100 K but are not actually 100 K "in"s and "out"s. To be safe, these signals need to be buffered. The signals going to the transmit side can be the differential "out" of any 100 K part. The receive-side signals are fed to the inputs of a differential receiver capable of sufficient gain to handle an input down to 150 mV. All these signals must have equal-length paths at propagation time. The maximum recommended skew between the two signals from one node to the next is 1 ns. Therefore, do not use a significant portion of this "budget" in the electronics; leave it for the cable plant (200 ps is recommended).

~RX1\RX1 / ~TX1/TX1

(See ~RX0\RX0 / ~TX0/TX0)

R_W

The ***R_W*** line is a status line to the Rehostable Adapter ASIC. This line specifies read or write "type" cycle to the Rehostable Adapter ASIC chip. ***R_W*** asserted (TTL level HI) indicates this is a read cycle. If this signal is de-asserted (TTL level LOW), then the transaction requested is a write.

RX_F_WR (See CNTRL_STATUS, page 3-5)

The Receiver FIFO write signal is asserted (TTL logic level HI) from the Rehostable Adapter ASIC at the end of a network transaction. This signal is found on the Rehostable Adapter ASIC pin labeled CNTRL_STATUS[20] during network mode (CS_MODE = LOW). This signal is asserted anytime a network transaction writes into the interrupt FIFO. This signal is made available for both monitoring purposes as well as possible Interrupt FIFO width extensions.

RX_INT (See CNTRL_STATUS, page 3-5)

The Receiver Interrupt signal is asserted (TTL logic level HI) from the Rehostable Adapter ASIC during a network transaction. This signal is found on CNTRL_STATUS[17] only during network mode (CS_MODE = LOW). This signal is a status signal meaning, if asserted, that the received message from the network had the interrupt bit set. This signal is mainly used for monitoring purposes.

RX_RETRY (See CNTRL_STATUS, page 3-5)

The Receiver Retry signal is asserted (TTL logic level HI) from the Rehostable Adapter ASIC during a network transaction. This signal is found on CNTRL_STATUS[16] only during network mode (CS_MODE = LOW). This signal is a status signal meaning, if asserted, that the received message from the network had the retry bit set. This means that this message was transmitted more than once from the originating node. This signal is mainly used for monitoring purposes.

STS1,STS2

The Status of Transaction Select lines are TTL level outputs which correspond to the current transaction or, during a null transaction condition, the last transaction which the Rehostable Adapter ASIC is/was executing. These codes are the same as ***TS1***,***TS2***. These outputs become valid just after either BGH or BGN is asserted.

~SW_OE0

The Switch Output Enable 0 is an active low TTL level output from the Rehostable Adapter ASIC. This signal instructs the external switches to output their value onto the MD bus. This activity occurs at reset ONLY.

S_CLK

This serial output clock is constantly running at a rate of CLK/4 [9.375 MHz = 106.6 ns]. This output is CMOS driven TTL levels to and from the host.

S_DATA

This is a bi-directional line used for both output from the LEDS controller, and input from an external device. This signal is CMOS driven TTL levels to and from the host.

S_DIR

This is the S_DATA direction indicator. When this line is a logic LOW, the LEDS controller is in an output mode. Conversely, when this line is HIGH, the LEDS controller is shifting information in from an external device. The timing is set up so that the transition from LOW to HIGH (rising edge) of this signal can be used to clock the shift register data into a buffering register. This signal is CMOS driven TTL levels to and from the host.

TRIG3 (See CNTRL_STATUS, page 3-5)

The TRIG3 signal is asserted (TTL logic level HI) from the Rehostable Adapter ASIC during a host or network transaction. This signal is found on CNTRL_STATUS[20] only during non-network mode (CS_MODE = HI). Specifically, this signal is a trigger output from the Rehostable Adapter ASIC. The Rehostable Adapter ASIC has an extra RAM bit in the internal RAM's ACR[5]. This bit was defined to be ANY shared memory access (host or network). This trigger is a single clock cycle long (26.64 ns). Since this output only occurs during non-network transactions the actual trigger is delayed until CS_MODE = HI to make sure it is seen.

NOTE: Since ACR[5] does not go into the Rehostable Adapter ASIC, this trigger output is only Rehostable-Adapter-ASIC-supported for internal memory

TRIGGER

This signal is a non-serialized/real-time trigger from the Rehostable Adapter ASIC. It is fed from a multiplexer inside the LEDS controller chip. The two possible connections to this line are: External Trigger #1, or External Trigger #2. The control of this multiplexer is a bit generated by an external device. This signal is CMOS driven TTL levels to and from the host.

TS1, TS2

The Transaction Select lines instruct the Rehostable Adapter ASIC as to which type of transaction to perform. The combination of these two lines result in four possible codes that correspond to the transaction types. These code combinations are shown in Table 3-6.

Table 3-6 Transaction Types

TS1	*TS2*	Transaction Types
0	0	Shared Memory
1	0	CSR
0	1	Non-memory Transfer (NON-BOF*)
1	1	Non-memory Transfer (BOF*)

* Beginning of Frame

TSP[2:0]

The Transaction Size and Position lines inform the Rehostable Adapter ASIC how many bytes out of a possible 4 bytes are valid for this Rehostable Adapter ASIC transaction. Note that the Rehostable Adapter ASIC supports only 32-bit read cycles. The only exception to that is for CSR transactions where the actual read of a register causes an activity.

EXAMPLE:

CSR1-a read to either byte will cause a register CLEAR. However, reading CSR0 (same 32-bit location) will NOT cause the clear. The ***TSP*** lines are ignored during all Non-Memory Transfers (000 is assumed [32 bits]). Table 3-7 provides the transaction size and location of the ***TSP*** lines.

Table 3-7 *TSP* Lines

TSP2	TSP1	TSP0	Transaction Size and Location
0	0	0	32 Bits - MD[31:0]
-	0	1	16 Bits - MD[15:0]
0	1	X	16 Bits - MD[31:16]
1	0	0	8 Bits - MD[31:24]
1	0	1	8 Bits - MD[23:16]
1	1	0	8 Bits - MD[15:8]
1	1	1	8 Bits - MD[7:0]

Table 3-8 Rehostable Adapter Signal Capacitance

Capacitance Loading			
Signal	**Max Value**	**Signal**	**Max Value**
ACR[7:0]	17 ρf	MEM_DET	12 ρf
~ACR_DOE	17 ρf	MEM_HIT	5 ρf
~ACR_WE	29 ρf	~MWE	29 ρf
A_D_LTCH	15 ρf	~M_DOE	29 ρf
BUSY	5 ρf	OE.A2B[3:0]	12 ρf
CLK	10 ρf	OID*	17 ρf
CSR_ME[1:0]	5 ρf	RD_CSR5*	5 ρf
CS_MODE	17 ρf	~RST_IN	5 ρf
EXT_PWR	N/A	~RX0/RX0	3 ρf
F_RELAY	5 ρf	~RX1/RX1	3 ρf
HA[31:2]	5 ρf	***R_W***	5 ρf
HACK	15 ρf	RX_F_WR*	5 ρf
HD[31:0]	12 ρf	RX_INT*	5 ρf
HREQ	5 ρf	RX_RETRY*	5 ρf
HREQ_PEND	5 ρf	STS1	5 ρf
INT_ARMED*	5 ρf	STS2	5 ρf
INT_ERROR*	5 ρf	‘SW_OE0	5 ρf
INT_PEND	5 ρf	S_CLK	12 ρf
INT_RXFIFO*	17 ρf	S_DATA	12 ρf
LINK_A_B	5 ρf	S_DIR	12 ρf
L_CD_LED	12 ρf	TRIG3*	5 ρf
L_INSERT_LED	12 ρf	TRIGGER	12 ρf
L_MECHSW	12 ρf	***TS1***	5 ρf
MA[22:2]	17 ρf	***TS2***	5 ρf
MBCS	17 ρf	***TSP***[2:0]	5 ρf
MC[2:0]	17 ρf	~TX0/TX0	N/A
MD[31:0]	41 ρf	~TX1/TX1	N/A

* See CNTRL_STATUS. page 3-5

This page intentionally left blank

4. DESIGN GUIDELINES

4.1 Overview

This chapter discusses software compatibility, design requirements, throughput considerations, and optional design features.

4.2 Software Compatibility

This section highlights areas of the hardware design affecting software ported from another SCRAMNet system. It should be noted however, that software compatibility is not without its cost in hardware. In many cases it may be possible and desirable to deviate from the software specification described in the *SCRAMNet Utilities User Manual*.

4.2.1 Interrupt Vector Register

The interrupt vector register is CSR6 in the SCRAMNet I/O area. The SCRAMNet ASIC or the Rehostable Adapter does NOT provide this register. If this register is to be “software compatible,” it must be externally mapped into the CSR I/O area.

The vector register should be left off user designs because this external mapping would result in additional hardware. It is also possible that the interrupt system on the embedded application will be simpler in design and not involve a vector at all.

4.2.2 Interrupt Pending and the SCRAMNet Interrupt system

If interrupts are implemented on the Rehostable Adapter, the user must determine how this is to be implemented. If rehosted SCRAMNet algorithms are used, the user design must include the handling of the ***INT_PEND*** signal.

There are two choices when implementing interrupts on the Rehostable Adapter: standard and ultra-simple. The standard interrupt implementation path will keep the INT_ARMED bit working the way it is documented in the SCRAMNet Hardware Reference Manual. If the ***INT_PEND*** line is grounded, then the INT_ARMED bit in CSR1 will always be a ‘1’ when proper values are written to CSR0 to enable interrupts. Also, writes to CSR1 to “re-arm” interrupts will not be necessary. This, along with the ultra-simple approach, will be discussed in more detail in paragraph 4.5: Optional Design Features, page 4-11.

4.2.3 SCRAMNet Memory and CSR mapping

The use of the SCRAMNet ASIC address comparator does have an effect on software compatibility. The memory address is specified in CSR10 and CSR11 on some SCRAMNet products. The memory decoding may also be turned off under program control. To use these “software” features, the ASIC comparator must be used. This will require some diodes to program the comparator size, and the connection of the upper address bits to the Rehostable Adapter. Once these design issues are accepted, then the output signal MEM_HIT will reflect a SCRAMNet memory address “hit”.

NOTE: Because embedded systems often have simple address decoding, the user design is responsible for the address decoding of the CSR and memory areas.

4.3 Design Requirements

4.3.1 Transaction Definition

A transaction is defined as a read or write to the Rehostable Adapter by a controller. It centers on setting up signals to define either a CSR or Memory activity, read or write, and the data size of the transaction. The user interface design then performs a handshake with the Rehostable Adapter to start the transaction. Once the Rehostable Adapter has started the transaction, the Rehostable Adapter handles all timing. A specific signal called A_D_LTCH latches all transaction-setup information into the Rehostable Adapter. Upon completion, the Rehostable Adapter raises a signal to signify the end of the transaction.

4.3.2 Transaction Setup Information

THE *R_W* LINE AND TRANSACTION SELECT LINES

The ***R_W*** lines instruct the Rehostable Adapter concerning the direction of a new transaction to the Rehostable Adapter. If the ***R_W*** line is low, it signifies a write to the Rehostable Adapter (data moves to the Rehostable Adapter). When the ***R_W*** line is high, this signifies a read from the Rehostable Adapter (data moves from the Rehostable Adapter).

NOTE: The ***R_W*** line must be set up 0 ns relative to the rising edge of the ***HREQ*** signal. It must be stable before and through the rising edge of the A_D_LTCH signal.

The Rehostable Adapter supports three distinct types of transactions: (The typical SCRAMNet user usually needs only the first two).

- Control/Status Register (CSR)
- Memory
- Non-Memory Network Transfers

CSR accesses are intended to alter control or obtain status information about the SCRAMNet network node. Memory accesses are intended to pass information to other nodes on the network in a shared-memory fashion. Non-Memory Network Transfer accesses are intended to pass information to other nodes on the network in a non-shared memory fashion.

NOTE: Non-Memory Network Transfers do not write into memory space. A non-rehostable adapter will ignore them. For more information contact Systran Corp.

To inform the Rehostable Adapter which type of transaction is to be performed, two transaction select lines are available-***TS1***, and ***TS2***. Table 4-1 shows the values for the *TSx* lines verses the type of transaction selected.

Table 4-1 *TS1*/*TS2* Values

Transaction Type	***TS1***	***TS2***
Memory	0	0
CSR	1	0
Non-memory	0	1
Non-memory (BOF)	1	1

The ***TSx*** lines must be set up 0 ns relative to the rising edge of ***HREQ***. They must be stable before and through the rising edge of A_D_LTCH.

Since the normal interface to the Rehostable Adapter does not involve Non-memory Network Transfers, just ground the ***TS2*** line. The ***TS1*** line then becomes a memory/CSR space selector. This is typically generated by the user's address-decoder logic.

TRANSACTION SIZE AND POSITION LINES

The Transaction Size and Position (***TSP***[2:0]) describe the transactions data size and its position on the ***HD***[31:0] bus. The Rehostable Adapter handles 8-, 16-, and 32-bit writes, and 32-bit read operations. (In general the TSP codes are ignored for read accesses and 32 bits of data are provided.) Table 4-2 describes the size of the data item and its position on the ***HD***[31:0] bus for all values of the TSP codes.

☞ **NOTE**: THERE IS NO BYTE SWAPPING DONE ON THE REHOSTABLE ADAPTER. If this unjustified data path arrangement does not match your host design, you may have to impose a data-size limitation or add byte-lane steering logic in order to properly handle data.

Table 4-2 TSP Codes

Position	***TSP2***	***TSP1***	**TSP0**	**Size**
31:0	0	0	0	32-bit
15:0	0	0	1	16-bit
31:16	0	1	X	16-bit
31:24	1	0	0	8-bit
23:16	1	0	1	8-bit
15:8	1	1	0	8-bit
7:0	1	1	1	8-bit

Although the Rehostable Adapter always returns 32 bits of data for a read transaction, the TSP codes should always reflect the transaction size requested. This is specifically important for 8- and 16-bit CSR access. Some specific 16-bit registers take action when they are read. The SCRAMNet ASIC on the Rehostable Adapter examines the TSP codes in these cases to decide if the transaction requested requires READ/CLEAR or UPDATE action.

NOTE: The ***TSPx*** lines must be set up 0 ns relative to the rising edge of ***HREQ***. They must be stable before and through the rising edge of A_D_LTCH.

HOST ADDRESS BUS

The Host Address bus (***HA**[31:2]*) is used to supply the Rehostable Adapter with an address. The number of ***HA*** lines which actually need to be connected depends on three factors:

- The size of the targeted address bus.
- Whether the user's design will make use of the Rehostable Adapter's address comparator.
- The memory size supported on the Rehostable Adapter.

The targeted address bus must have sufficient addressable space. For example: If the users target design is a Motorola 68000 and the upper address line is A23, then the address lines ***HA**[31:24]* cannot be connected.

If the users design includes the Rehostable Adapter's address comparator, then as many address lines as possible must be connected to the Rehostable Adapter. See page 4-13 for more information the SCRAMNet ASIC address decoder and its application to the Rehostable Adapter.

NOTE: All unused ***HA*** lines MUST be grounded.

If the user's design does not include the Rehostable Adapter's address comparator, this greatly reduces the number of address lines that must be connected to the Rehostable Adapter.

Table 4-3 shows the minimum address connections verses the memory size.

Table 4-3 *HA* Address Connections vs. Memory Size

Memory Size	Address Lines Connected
2 MB	***HA***[20:2]
1 MB	***HA***[19:2]
512 KB	***HA***[18:2]
4 KB	***HA***[11:2]

If only CSR support is desired, then ***HA***[5:2] provides the addressing for CSRs.

NOTE: The ***HAx*** lines must be set up 0 ns relative to the rising edge of ***HREQ***. They must be stable before and through the rising edge of A_D_LTCH.

NOTE: A01 is not present on the Rehostable Adapter. Its functionality is included in the ***TSPx*** lines. For more information, see 5.6.2 Transaction Size and Position Encode, page 5-5.

HOST DATA BUS

The Host Data bus ***HD***[31:0] transfers data between the Rehostable Adapter and the user design. The HD bus is not justified. Byte swapping logic may be required depending on the user data bus size and processor integer format. The Rehostable Adapter is a Big-Endian device. Table 4-4 is derived from Big-Endian formatting of a 32-bit device.

NOTE: For a write transaction the ***HDx*** lines must be set up 0 ns relative to the rising edge of ***HREQ***. They must be stable before and through the rising edge of A_D_LTCH.

Table 4-4 Big-Endian 32-bit Formatting

Data Size	*HD*[31:24]	*HD*[23:16]	*HD*[15:8]	*HD*[7:0]
8-bit	Byte 0 - MSB CSR0	Byte 1 - LSB CSR0	Byte 2 - MSB CSR1	Byte 3 - CSR1
16-bit	Word 0 - CSR0		Word 1 - CSR1	
32-bit	Longword 0 - CSR0[MSB - LSB] CSR1 [MSB - LSB]			

4.3.3 Transaction Start: *HREQ* and HREQ_PEND Handshake

The ***HREQ*** signal is the master control signal which, when asserted (logic ‘1’), instructs the Rehostable Adapter to perform the requested transaction. When seen, the Rehostable Adapter will respond by asserting HREQ_PEND. This signifies that the request from the user’s design has been latched in the Rehostable Adapter. The ***HREQ*** signal should then be de-asserted (logic ‘0’). The ***HREQ*** signal is sampled on the rising edge of every CLK driven from the Rehostable Adapter. Figure 4-1 illustrates this handshake.

No setup and hold times for ***HREQ*** relative to CLK are given because the SCRAMNet ASIC (primary component of the Rehostable Adapter) has a special synchronizer flip-flop in the design for the ***HREQ*** signal.

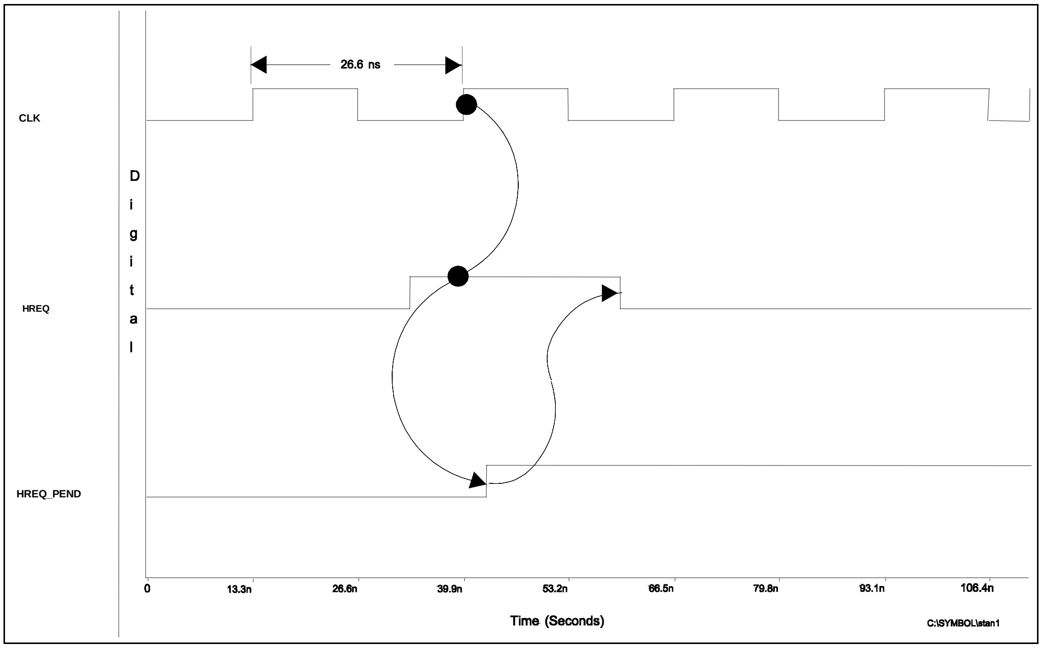


Figure 4-1 *HREQ* and HREQ_PEND Handshake

☞ **NOTE**: Use synchronous logic for the generation of ***HREQ*** since HREQ_PEND may suffer some metastable effects. Do not use it in combinational/synchronous logic.

The following rules apply to the generation of ***HREQ***:

- ***HREQ*** can never be asserted when HREQ_PEND is already asserted.
- Once HREQ_PEND is asserted in response to a ***HREQ***, the ***HREQ*** signal must be de-asserted in no less than three CLK periods (79.8 ns). Otherwise, a second transaction may be issued).

4.3.4 Transaction End: *HACK* Considerations

The ***HACK*** signal is used to generate a "host acknowledge" (***HACK***) signal back to the user logic indicating the requested transaction is completed. This signal is used on the Rehostable Adapter to latch the data during a read activity. ***HACK*** is also used to reset or end a user-based logic transaction. This signal is a positively-asserted single-clock signal (26.6 ns).

☞ **NOTE**: For read transactions on the HDx lines, the data will be available 5.7 ns after the rising edge of ***HACK***. To cause the Rehostable Adapter to drive the HDx bus, the user design must drive the appropriate byte output-enable signal(s)-OE_A2B[3:0] (logic '1'). These are active-high signals corresponding directly to the four byte-lanes [MSB to LSB]. The output-enable time-OE_A2Bx = '1' to HDx valid is 5 ns.

4.4 Throughput considerations

The SCRAMNet memory bus is shared by two entities—user read/write requests, and network write requests. These requests are serviced on a first-come-first-served basis with priority given to network requests if a tie exists. This makes user throughput a statistical problem based upon application-oriented random network traffic. It is necessary to make some assumptions about network-traffic levels in order to work through the numbers.

For this discussion the following throughput levels will be used - no traffic (0 MB/s), light traffic (2 MB/s), heavy traffic (6.5 MB/s), and full traffic (16.7 MB/s).

4.4.1 User Throughput With No Network Traffic

With no network traffic the user requests will always win arbitration. The arbitration time is 1 clock cycle (26.6 ns). This will be added to the access times.

HOST MEMORY READ THROUGHPUT

```
request/time  = arb_time + cycle_time
  cycle_time  = 5 clocks for a memory read (133 ns)
     arb_time = 1 clock (26.6 ns)
 request-time = 26.6 ns + 133 ns = 159.6 ns
         MB/s = (1/(request-time)) * 4
              = 25.06 MB/s (32-bit (4 byte) transfers)
```

HOST MEMORY WRITE THROUGHPUT

```
request-time  = arb_time + cycle_time
  cycle_time  = 9 clocks for a memory write (239.4 ns)
     arb_time = 1 clock (26.6 ns)
```

```
request-time  = 26.6 ns + 239.4 ns = 266 ns
        MB/s = (1/(request-time)) * 4
              = 15.04 MB/s (32-bit (4 bytes) transfers)
```

CSR READ THROUGHPUT

```
request-time  = arb_time + cycle_time
  cycle_time  = 5 clocks for a CSR read (133 ns)
    arb_time  = 1 clock (26.6 ns)
request-time  = 26.6 ns + 133 ns = 159.6 ns
        MB/s = (1/(request-time)) * 4
              = 25.06 MB/s (32-bit (4 bytes) transfers)
```

CSR WRITE THROUGHPUT

```
request-time  = arb_time + cycle_time
  cycle_time  = 4 clocks for a memory write (106.4 ns)
    arb_time  = 1 clock (26.6 ns)
request-time  = 26.6 ns + 106.4 ns  = 133 ns
        MB/s = ( 1/(request-time) ) * 4
              = 30.07 MB/s (32-bit (4 bytes) transfers)
```

4.4.2 User Throughput With Light Network Traffic

With 2 MB/s network traffic, user requests will win arbitration only a percentage of the time. If the network request wins arbitration for the SCRAMNet memory bus, the user transaction will be delayed until the network write is finished.

To model the network traffic, assume a constant level of network throughput from the network. Since the data in the SCRAMNet transmission is 32 bits wide, the arbitrator will see a network write request at the following rate:

```
32-bit traffic rate  = byte rate / 4
                     = (2 MB/s) / 4
                     = 500 K requests/s
                time = 1 / rate
                     = 1/ (500 K requests/s)
                     = 2000 ns
```

Therefore, every 2000 ns the SCRAMNet memory controller will be busy with a network request.

NETWORK WRITE CYCLE TIME

```
cycle_time  = 5 clocks for a network write (133 ns)
```

Therefore every 2000 ns the SCRAMNet memory controller will be busy for 133 ns with a network write cycle.

In terms of SCRAMNet Memory bus Bandwidth utilization this would be -

```
133 ns / 2000 ns = 6.65 %
```

The remaining bandwidth is available to the user accesses, and this is equal to -

```
100 % - 6.65 % = 93.35 %
```

This is used to calculate the host throughput.

HOST MEMORY READ THROUGHPUT AVAILABLE

```
Available bandwidth = request/time / rate
               Rate = request/time / available bandwidth
```

= 159.6 ns / 93.35 %
= 170.96 ns
MB/s = (1/170.96 ns) * 4
= 23.39 MB /sec

HOST MEMORY WRITE THROUGHPUT

Available bandwidth = request/time / rate
Rate = request/time / available bandwidth
= 266 ns / 93.35 %
= 284.95 ns
MB/s = (1/284.95 ns) * 4
= 14.04 MB /sec

CSR READ THROUGHPUT

Available bandwidth = request-time / rate
Rate = request-time / available bandwidth
= 159.6 ns / 93.35 %
= 170.96 ns
MB/s = (1/170.96 ns) * 4
= 23.39 MB /sec

CSR WRITE THROUGHPUT

Available bandwidth = request-time / rate
Rate = request-time / available bandwidth
= 133 ns / 93.35 %
= 142.47 ns
MB/s = (1/142.47 ns) * 4
= 28.08 MB /sec

4.4.3 User Throughput With Heavy Network Traffic

With 6.5 MB/s network traffic the user requests will win arbitration only a percentage of the time. If the network request wins arbitration for the SCRAMNet memory bus, the user transaction will be delayed until the network write is finished.

To model the network traffic, we will assume a constant level of network throughput from the network. Since the data in the SCRAMNet transmission is 32-bits wide, the arbitrator will see a network write request at the following rate:

32 bit traffic rate = byte rate / 4
= (6.5 MB/s) / 4
= 1625 K requests/sec
time = 1 / rate
= 1/ (1625 K requests/sec)
= 615 ns

Therefore, every 615 ns the SCRAMNet memory controller will be busy with a network request.

NETWORK WRITE CYCLE TIME

cycle_time = 5 clocks for a network write (133 ns)

Therefore every 615 ns the SCRAMNet memory controller will be busy for 133 ns with a network write cycle.

In terms of SCRAMNet Memory bus Bandwidth utilization this would be:

133 ns / 615 ns = 21.62 %

The remaining bandwidth is available to the user accesses, and this is equal to:

```
100 % - 21.62 %   = 78.38% %
```

This will be used to calculate the host throughput.

HOST MEMORY READ THROUGHPUT AVAILABLE

```
Available bandwidth = request-time / rate
               Rate = request-time / available bandwidth
                    = 133 ns / 78.38 %
                    = 169.68 ns
               MB/s = (1/169.68 ns) * 4
                    = 23.57 MB /sec
```

HOST MEMORY WRITE THROUGHPUT

```
Available bandwidth = request-time / rate
               Rate = request-time / available bandwidth
                    = 266 ns / 78.38 %
                    = 339.37 ns
               MB/s = (1/339.37 ns) * 4
                    = 11.78 MB /sec
```

CSR READ THROUGHPUT

```
Available bandwidth = request-time / rate
               Rate = request-time / available bandwidth
                    = 159.6 ns / 78.38 %
                    = 203.62 ns
               MB/s = (1/203.62 ns) * 4
                    = 19.64 MB /sec
```

CSR WRITE THROUGHPUT

```
Available bandwidth = request-time / rate
               Rate = request-time / available bandwidth
                    = 133 ns / 78.38 %
                    = 169.69 ns
               MB/s = (1/169.69 ns) * 4
                    = 23.57 MB /sec
```

4.4.4 User Throughput With Full Network Traffic

With 16.7 MB/s network traffic the user requests will win arbitration only a percentage of the time. If the network request wins arbitration for the SCRAMNet memory bus, the user transaction will be delayed until the network write is finished.

To model the network traffic, assume a constant level of network throughput from the network. Since the data in the SCRAMNet transmission is 32 bits wide, the arbitrator will see a network write request at the following rate:

```
32 bit traffic rate  = byte rate / 4
                     = (16.7 MB/s) / 4
                     = 4175 K requests/s
                time = 1 / rate
                     = 1/ (4175 K requests/s)
                     = 239.5 ns
```

Therefore, every 239.5 ns the SCRAMNet memory controller will be busy with a network request.

NETWORK WRITE CYCLE TIME

```
          cycle_time  = 5 clocks for a network write (133 ns)
```

Therefore, every 239.5 ns the SCRAMNet memory controller will be busy for 133 ns with a network write cycle.

In terms of SCRAMNet Memory bus Bandwidth utilization this would be -

```
    133 ns / 239.5 ns = 55.53 %
```

The remaining bandwidth is available to the user accesses, and this is equal to -

```
      100 % - 55.53 % = 44.47 %
```

This will be used to calculate the host throughput.

HOST MEMORY READ THROUGHPUT AVAILABLE

```
  Available bandwidth  = request-time / rate
                 Rate  = request-time / available bandwidth
                       = 159.6 ns / 44.47 %
                       = 358.89 ns
                 MB/s  = (1/358.89 ns) * 4
                       = 11.14 MB /sec
```

HOST MEMORY WRITE THROUGHPUT

```
  Available bandwidth  = request-time / rate
                Rate   = request-time / available bandwidth
                       = 266 ns / 44.47 %
                       = 598.16 ns
                MB/s   = (1/598.16 ns) * 4
                       = 6.69 MB /sec
```

CSR READ THROUGHPUT

```
  Available bandwidth  = request-time / rate
                  Rate = request-time / available bandwidth
                       = 159.6 ns / 44.47 %
                       = 358.89 ns
                  MB/s = (1/358.89 ns) * 4
                       = 11.14 MB /sec
```

CSR WRITE THROUGHPUT

```
  Available bandwidth  = request-time / rate
                  Rate = request-time / available bandwidth
                       = 133 ns / 44.47 %
                       = 299.08 ns
                  MB/s = (1/299.08 ns) * 4
                       = 13.37 MB /sec
```

4.4.5 Conclusions And Comments

These examples demonstrate that the average throughput to the Rehostable Adapter is dependent on network activity. What is not seen here is a Receiver FIFO that exists in the SCRAMNet ASIC. The Receiver FIFO is only three transactions deep but during high throughput levels on the network and host, it is possible to get backed up to three network requests. Since the network requests always win arbitration, the typical wait from a ***HREQ*** to a ***HACK*** may vary significantly under high-throughput conditions.

4.5 Optional Design Features

The Rehostable Adapter has two additional optional design features:

- The SCRAMNet ASIC Interrupt System
- The Rehostable Adapter Address Decoder.

4.5.1 The SCRAMNet ASIC Interrupt System

The Rehostable Adapter supports SCRAMNet interrupts. External hardware is required to modify the signal originating from the SCRAMNet ASIC into whatever is required for the users application. There are two possible implementations for interrupts and the Rehostable Adapter: standard and ultra-simple.

STANDARD INTERRUPT SCHEME

Interrupt flow activities are as follows.

1. The SCRAMNet ASIC is programmed through the CSRs to generate an interrupt(s) based on “events”. Set the *INT_ARMED* bit in CSR1 = ‘1’.
2. The SCRAMNet ASIC, responding to an “Event” will then raise either CNTL_STATUS18 (for a memory based interrupt) or *CNTRL_STATUS17* (for a error based interrupt) when *CS_MODE* = ‘1’.
    - *CNTRL_STATUSx* lines have a different definition when
    - *CS_MODE* = ‘0.
    - *INT_ARMED* status bit in CSR1 will still read ‘1’.
3. The SCRAMNet ASIC on the Rehostable Adapter will continue to generate the appropriate interrupt signal(s) until one of the following occurs: the interrupt stops occurring (this implies that software has removed the condition), or the ***INT_PEND*** signal has a rising edge.
4. If the ***INT_PEND*** signal has a rising edge then the *INT_ARMED* status bit in CSR1 will read ‘0’ and when *CS_MODE* = ‘1’, the *CNTRL_STATUSx* lines will show no interrupt. At this point the ISR should have eliminated the cause of the interrupt, and at the end of the interrupt routine, a write to CSR1 will again set the INT_ARMED bit in CSR1 and, in effect, re-arm the interrupt system to show interrupts through the *CNTRL_STATUSx* lines.

Normally a feedback path is created which ties the generated interrupt to the ***INT_PEND*** line. This will cause the interrupt line generated to de-assert shortly after it is generated. A simple set of circuitry is shown in Figure 4-2. In this circuit, the interrupt bits, which can be included in the user design (*MEMORY_INT*, *ERROR_INT*, or *INTERRUPT*), will be very clean positively-asserted pulses (2 clocks long) whenever the Rehostable Adapter generates an interrupt. The following VHDL code can be synthesized using Viewlogic for the same basic circuitry:

```
INT1:     process
   begin
   wait until prising(CLK);
   if CS_MODE = ‘1’ then
          MEMORY_INT <<= CNTRL_STATUS18;
          ERROR_INT <<= CNTRL_STATUS17
          INT_PEND <<= INTERRUPT;
   end if;
   end process

   INTERRUPT <<= MEMORY_INT or ERROR_INT;
```

NOTE: The ***INT_PEND*** driving circuitry can be connected to user-specific interrupt-acknowledge circuitry; for example, the vector register in a vectored-interrupt system.

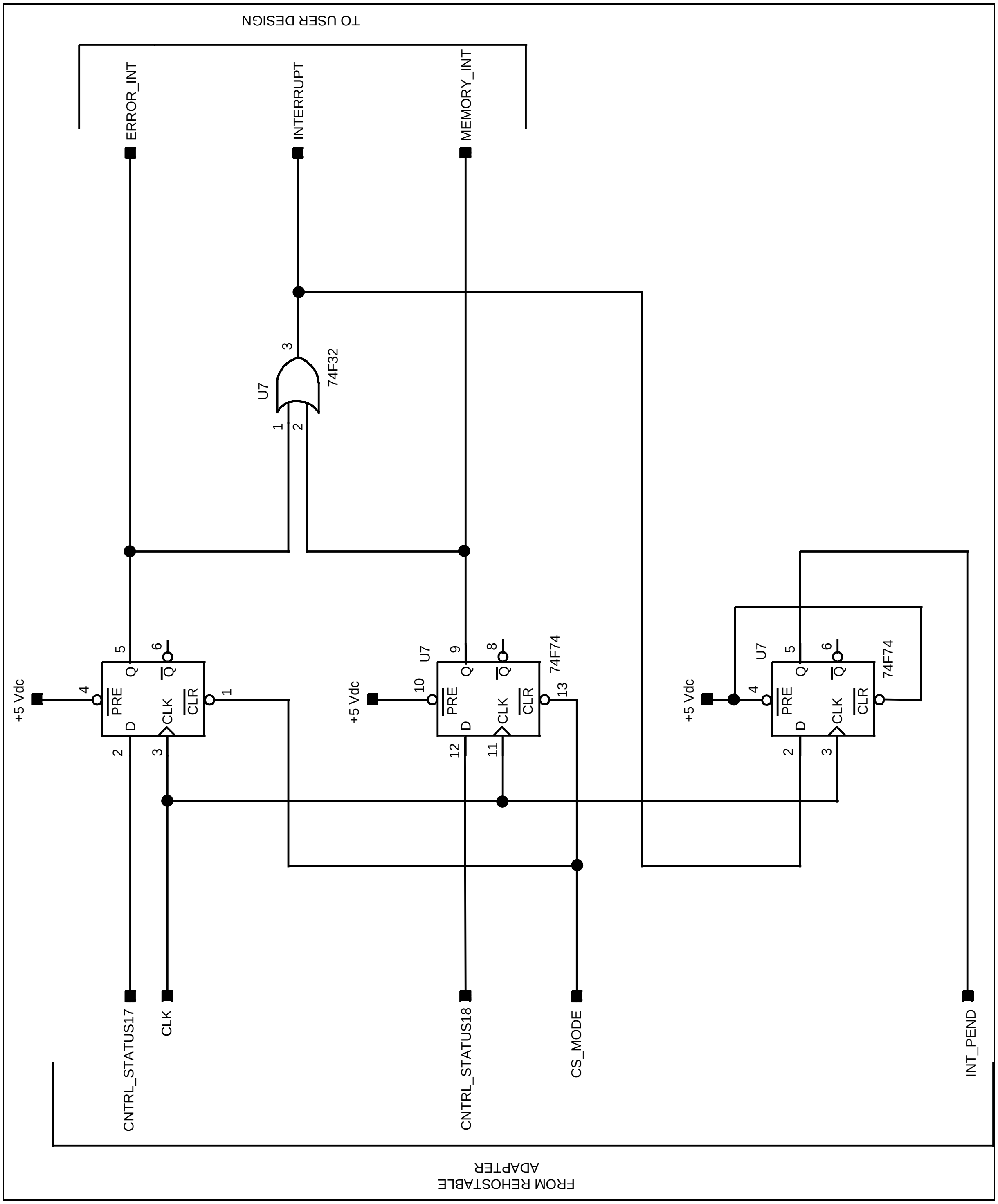


Figure 4-2 Standard Interrupt Logic

ULTRA-SIMPLE INTERRUPT SCHEME

An Ultra-Simple interrupt scheme is also available to the interrupt user. Figure 4-3 shows a reduced version of the interrupt circuitry. This will produce a set of rising-edge pulses on the outputs when the Rehostable Adapter generates an interrupt (this would be appropriate for some micros with edge-level interrupt support).

☞ **NOTE**: The ***INT_PEND*** is not used in this scheme. Therefore, the ***INT_PEND*** line MUST be tied to a logical ‘0’. This implies that the software in the interrupt service routine must stop the generation of interrupts.

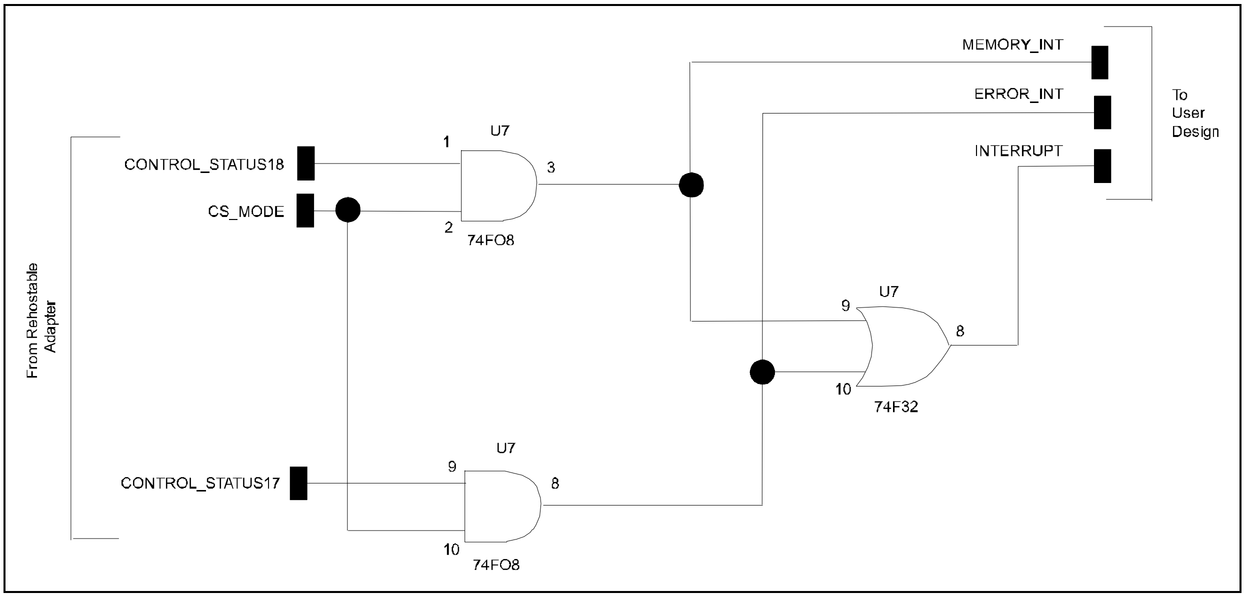


Figure 4-3 Ultra-simple Interrupt Logic

4.5.2 The SCRAMNet ASIC Address Decoder and Its Application to the Rehostable Adapter

The onboard SCRAMNet ASIC address decoder was designed to give an interface designer a re-configurable, fast, and wide-address comparator. In an embedded application this function usually does not require this versatility and is often implemented in a simple PLD. However, if desired, the user can make use of the Rehostable Adapter’s address decoder (in the SCRAMNet ASIC). These on-board comparators exist for user implementation. If the ASIC comparator is used for CSRs as well as memory, then between 2 and 4 octal tristateable buffers are required.

The versatility of using the onboard ASIC memory decoder is not without cost in parts and therefore is not recommended for the simple Rehostable Adapter design.

Figure 4-4 demonstrates one possible way to load configuration data into the SCRAMNet ASIC. The base address of the CSRs is transferred to the Rehostable Adapter via the MD bus. The base address is defined by the switch array. The switch array places the address on the MD bus when ~SW_OE0 is asserted during the SCRAMNet ASIC reset/power-up. In addition to the CSR base address, the bus resolution for both the CSRs and Memory are loaded, as well as the Variable Length Enable (allows PLUS MODE protocols).

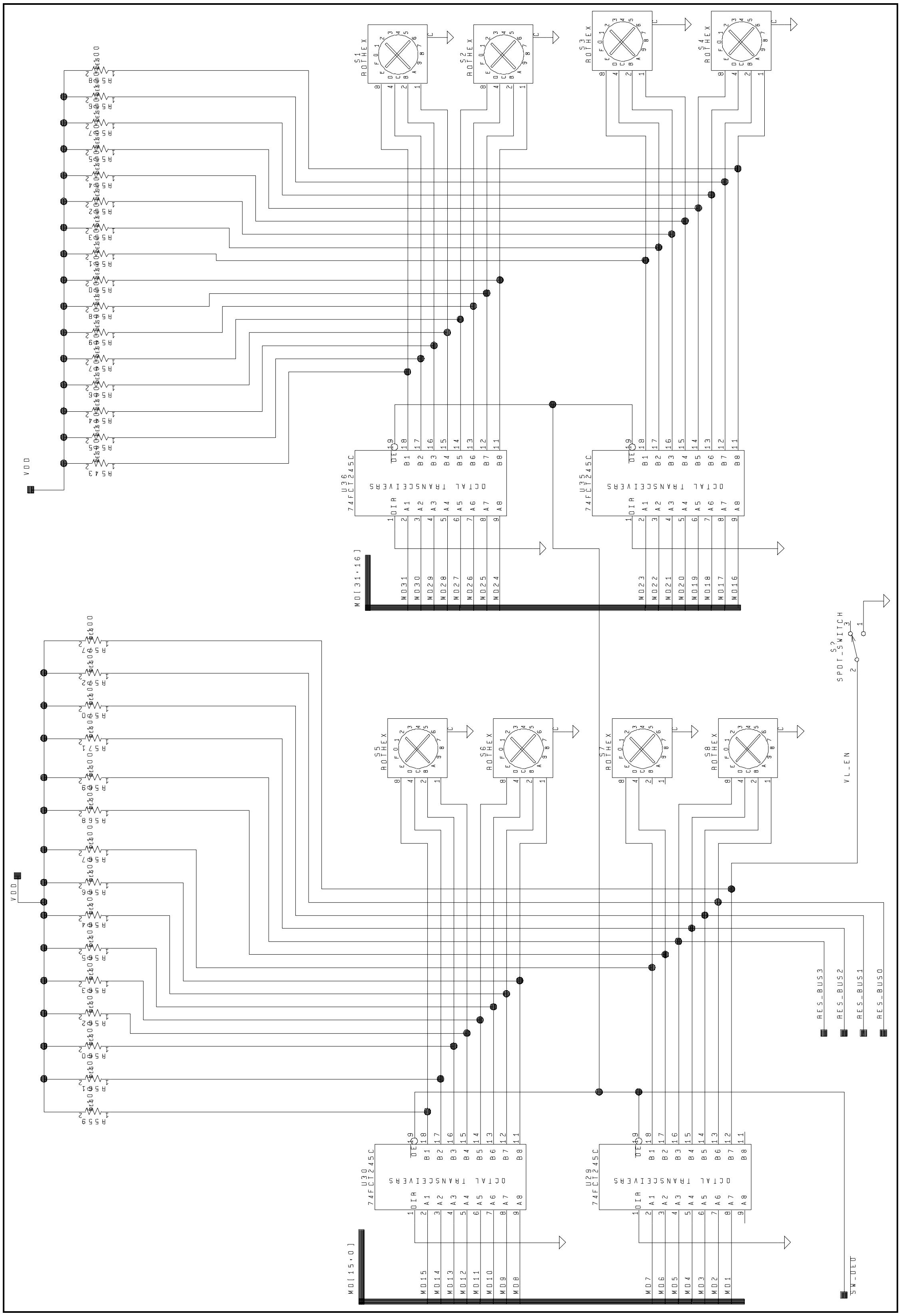


Figure 4-4 Switch 1

For a 32-bit-wide bus address, connect MD31-1. For a 24-bit-wide bus address, connect MD23-1. Likewise, for a 16-bit-wide bus address, connect MD15-1. The 8-bit address decoding is supported on the CSR decoder only.

The address-comparator-value inputs only go down to ***HA***6 (loaded during on a switch read on MD6). The data supplied on MD[5:4] and MD[3:2] is resolution information. These values program the two comparators most significant bit. Table 4-5 shows the options.

Table 4-5 MD Bus Bits Definition During Switch READ

Bits		Definition
MD[31:6]		CSR Base Address
MD[5]	MD[4]	CSR Base Address Modifiers
1	1	***HA***31 32-bit Addressing
1	0	***HA***23 24-bit Addressing
0	1	***HA***15 16-bit Addressing
0	0	***HA***7 8-bit Addressing
MD[3]	MD[2]	Memory Bus Address Modifiers
1	1	***HA***31 32-bit Addressing
1	0	***HA***23 24-bit Addressing
0	1	***HA***15 16-bit Addressing
0	0	Not Supported
MD[1]		Variable -length Enable: “1” = Enable
MD[0]		Reserved

The output of the ASICs comparator is completely asynchronous. The outputs for the CSR decoder are CSR_ME[1:0]. These two lines, after 20 ns of stable address in the ***HA***[31:0] bus, will show if the address is in the CSR range and if the register is internal or external to the Rehostable Adapter. Table 4-6 shows the definitions of the CSR_ME[1:0] lines.

As can be seen in Table 4-6, CSR_ME[0] can be used by itself as an address bit in simple situations since normal Rehostable Adapter implementations will NOT have to deal with external CSRs.

Table 4-6 CSR_ME[1:0] Line Definitions

CSR_ME[1]	CSR_ME[0]	Definition
X	0	Not equals (No address hit)
0	1	Rehostable internal CSR register compare
1	1	External rehostable CSR compare

The internal address comparator for SCRAMNet memory is easier to work with. The SCRAMNet ASIC has a built-in register (which can be initialized from the EEPROM) that defines the memory comparators address plus an enable bit. The only remaining issue is the address bus comparison width. This is defined by Bits MD[3] and MD[2] during the switch read (see Table 4-5) . If the SCRAMNet memory decoder ONLY is used, then one option to configure the address resolution switches is to use diodes; with the cathode side towards the signal ~SW_OE0 and the anode on the appropriate MDx line.

NOTE: All of the ASIC's comparator outputs are asynchronous. They become valid in less than 20 ns after a stable address is placed on the HA bus.

5. REHOSTABLE INTERFACING

5.1 Overview

This chapter discusses the Rehostable Adapter Interface features, system architecture, controller functionality, Rehostable Memory Decoder, memory subsystems, schematics, hardware assembly, and bill of materials.

5.2 Basic Rehostable VME Interface

The Basic Rehostable VME Interface retains as many SCRAMNet features as possible while minimizing development costs. The functionality of the SCRAMNet VME6U board has been reduced to create a simplistic, systematic approach to produce a Rehostable Adapter. The resulting design is a reduced VME interface.

5.3 Supported Features

5.3.1 Write Posting

Write Posting is the decoupling of the host bus with the Rehostable Adapter during write cycles. This involves moving the data to the Rehostable Adapter during write cycles, and ending the host bus cycle at that point even though the Rehostable Adapter has not finished the cycle. This permits a level of concurrent processing—the CPU is busy processing the next instruction, and the SCRAMNet device is processing the write.

☞ **NOTE**: Read cycles are never affected by Write Posting.

5.3.2 Quasi-dual Vector Interrupts

Quasi-dual vector interrupts are supported at a single fixed interrupt level. This reduces the level of complexity by fixing specific items. The base vector is programmable and the second vector equals the base vector plus 1 (The base vector must be an even number).

5.3.3 8–, 16–, And 32–bit Transactions

The interface supports 8–, 16–, and 32–bit transactions to and from VME. All 24–bit transactions are ignored. The support of these transaction sizes, and the knowledge of where transactions occur on the VME data bus, implies a required data-steering logic block between the SCRAMNet device and the data bus.

5.4 Unsupported Features

The onboard Rehostable Adapter ASIC address decoder is not used with the Rehostable VME Interface. Instead, the VME controller is generated with a fixed-address decoder for memory and a switch that allows the CSRs to be mapped to four different I/O locations.

5.5 System Architecture

Figure 5-1 shows the architecture required to satisfy the design requirements. This diagram has six major parts: two buffer blocks, data un-justifier block, VME controller interface block, rehostable block, and the Media Card (“MAC”) block.

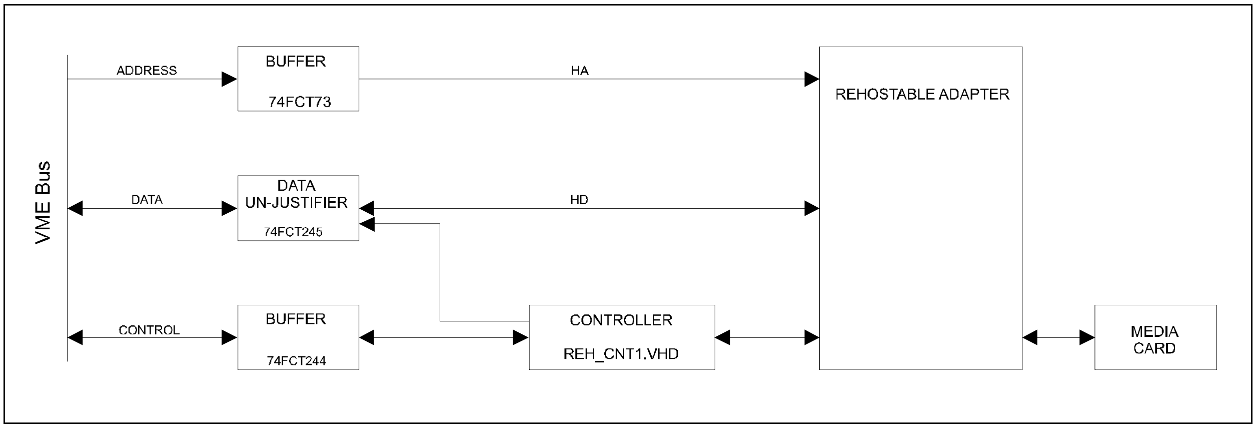


Figure 5-1 Block Diagram 1

5.5.1 Buffer Blocks

Two of the blocks are buffer blocks. They electrically isolate the rehostable design from the VMEbus. They also allow a single point of connection to the VMEbus that is in close proximity to the VME6U connectors (Short stub lengths are a part of the VMEbus Specification). These buffers were made from 74FCT573 latches. The control signals were set so that the part acts as a flow-through buffer. (A 74FCT244 was used for some of the control signals).

5.5.2 Data Un-justifier Block

The third block on the VME data bus also functions as a data un-justifier. VME moves all 8- and 16-bit accesses across the lower 16 bits of its data bus. This is known as justification. Since the Rehostable Adapter is unjustified, some steering logic is included which employs special routing under VME accesses less than 32 bits. Table 5-1 shows the transformation required for a given transaction.

Table 5-1 Data Un-justifier Transformations

Size	VME Data Bus				Rehostable Adapter Data Bus			
	D31:24	D23:16	D15:8	D7:0	D31:24	D23:16	D15:8	D7:0
32-bit	3	2	1	0	3	2	1	0
16-bit even			3	2	3	2		
16-bit odd			1	0			1	0
8-bit even			3		3			
8-bit odd				2		2		
8-bit even			1				1	
8-bit odd				0				0

From the table it can be seen that on-command byte-lane D15-8 needs to be steered to D31-24. Likewise, D7-0 needs to be steered to D23-16 on command. It can also be seen that while not required, both abnormal steering motions could always be asserted together. Schematic drawing B-12 (D32BUF) shows the implementation of this un-justifier/buffer.

The 75FCT245's are used because the path is bi-directional and the following control signals are available for the controller to manipulate.

DATA_DIR	This sets the Data bus direction.
~DB_EN31	This is the active low output-enable for D31-16
~DB_EN16	This is the active low output-enable for D15-D8
~DB_EN8	This is the active low output-enable for D7-D0
~DB_ENSW	This is the active low output-enable for the 74FCT245's steering the D15-D0 part of the VME bus to the D31-D16 part of the host Data bus.

5.5.3 Controller Interface Block

This block interfaces the VME control lines to the Rehostable Adapter. This block handles address decoding, detection of a VME cycle, acknowledgment of a VME cycle, and all interrupt generation capability. Figure 5-2 shows the internal structure of the controller used on this board. An example of the Rehostable VME controller interface is described in Appendix D, page D-25.

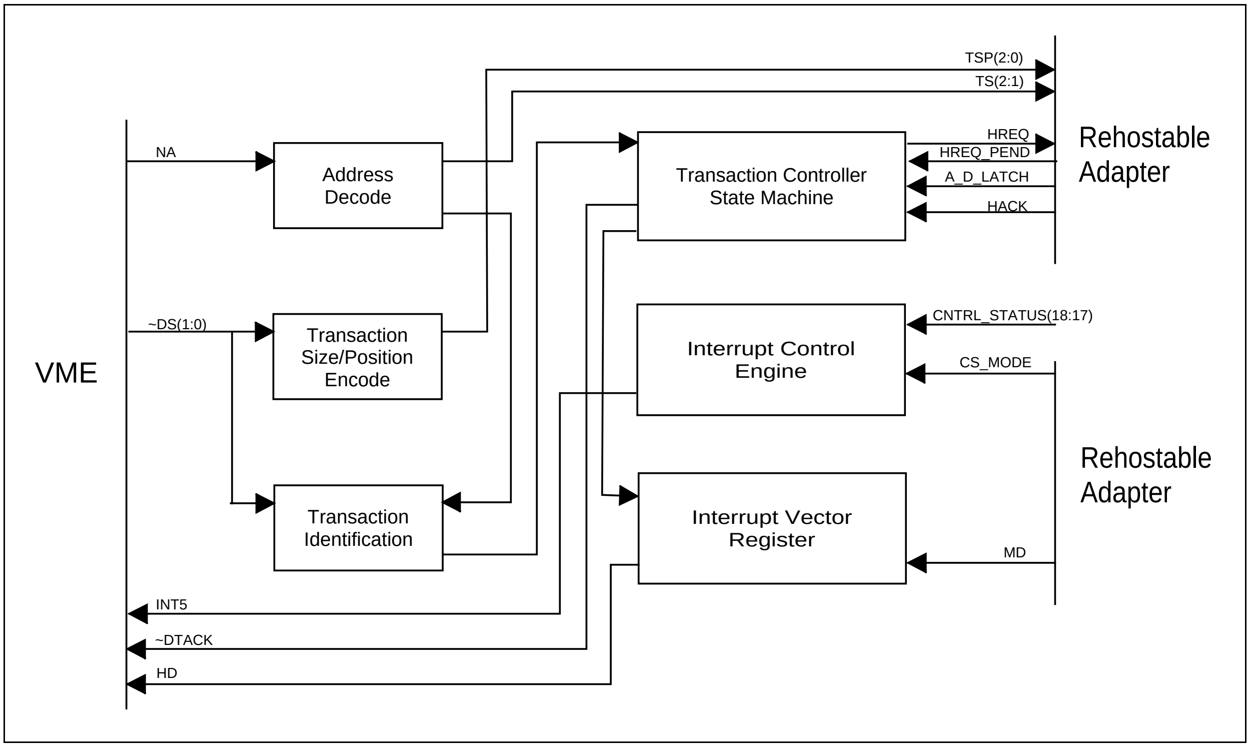


Figure 5-2 Block Diagram 2

5.5.4 Rehostable Adapter Block

The Rehostable Adapter block represents the Rehostable Adapter and its connectors.

5.5.5 Media Card (“MAC”) Interface Block

The Media Card installed at this interface port allows a physical connection to other SCRAMNet nodes.

5.6 Controller

The controller is the heart of the interface. It provides all control outputs and glue logic to interface the SCRAMNet Rehostable Adapter to the VMEbus. The controller is responsible for:

- Address Decode
- Transaction Size and Position Encode
- Transaction Identification
- Transaction Controller
- Interrupt Vector Register
- Interrupt Control Engine

5.6.1 Address Decode

Table 5-2 contains the fixed memory addresses used during internal testing:

Table 5-2 Memory Address Modifiers

Bus	Address Modifiers	Offset	Resource
16-bit address bus	29,2D	0000 *hex*	CSR’s*
24-bit address bus	39,3D	800000 *hex*	Memory

* See Subsection 5.7.1: Using the CSR Decoder

The address modifiers are the VME extensions to the address bus. They specify the resolution of the address bus on VME. On VME the three address bus sizes are three different address buses that share common address lines. The codes listed in Table 5-2 are for appropriate address accesses for the buses listed. For more information see the IEEE bus specification for VME. These address-modifier bits must be compared and found to be equal (and the offset must be equal) for the board to respond.

The following VHDL lines decode the address modifier bits:

```
--    These are from the VMEbus specification
--    Standard is the VME specification for the 24-bit address bus
standard       <=  ASSERTED when (am_bus = X”3d” or am_bus = X”39”) else
                       DE_ASSERTED;
sixteen-bit    <=  ASSERTED when (am_bus = X”2d” or am_bus = X”29”) else
                       DE_ASSERTED;
```

In this interface, the 24-bit “standard” address bus is only 16 MB. Therefore, the board occupies only 8 MB of address space. This allows the following statement to be used to decode the memory space:

```
MEM_ASIC_0      <=  ASSERTED when (standard = ASSERTED and VME_A23 =
                        ASSERTED), else DE_ASSERTED;
```

For the Rehostable’s CSRs, the following line of VHDL was used:

```
--    ASIC’s CSRs are on the 16-bit bus at address 0x0000 – 0x001e
CSR_ASIC_0      <=  ASSERTED when (sixteen_bit = ASSERTED and VME_A(15
                        downto 5) = B”00000000000” else DE_ASSERTED;
```

One external CSR is required as a vector register for the VME interface. This would most likely NOT be needed in an embedded application. The following VHDL decoded the single vector register:

```
--    HA's CSR is on the 16-bit bus at address 0x0020-0x003E
CSR_HA_0         <=  ASSERTED when (sixteen_bit = ASSERTED and VME_A(15
                          downto 5) = B"00000000000" else DE_ASSERTED;
```

NOTE: The CSRs are no longer at a fixed location. See Section 5.7.1: Using the CSR Decoder.

In addition to being used to inform the test board controller that a specific VME transaction is for our test board, the previous comparator outputs will be used to generate the TSx lines to the Rehostable Adapter. The TSx lines are used to instruct the Rehostable Adapter as to what type of action should occur (MEMORY,CSR,etc.). The following VHDL line is used to generate the ***TS1*** line:

```
TS1                    <=   ASSERTED when (CSR_HA = ASSERTED or CSR_ASIC =
                            ASSERTED) else, DE_ASSERTED;
```

For this implementation, the ***TS2*** line is grounded to the Rehostable Adapter. The ***TS2*** line is used to generate Non-Memory Transfers (NMTs) across the SCRAMNet network. For information concerning NMTs, see the SCRAMNet hardware reference manual.

5.6.2 Transaction Size and Position Encode

The Transaction Size and Position (***TSPx***) are encoded with information that represents the data size and its position on the HD bus. Table 5-3 shows the supported VME transactions and their size and the corresponding ***TSPx*** codes.

Table 5-3 VME Transactions and TSPx Codes

D31:24	D23:16	D15:8	D7:0	~DS1	~DS2	A01	~LWORD	*TSP*[2:0]
X	X	X	X	0	0	0	0	000
		X	X	0	0	1	1	001
X	X			0	0	0	1	01X
X				0	1	0	1	100
	X			1	0	0	1	101
		X		0	1	1	1	110
			X	1	0	1	1	111

The following VHDL maps the table (tsp_tbl) onto the ***TSPx*** lines. The code in Table 5-3 is reorganized into inputs versus outputs (where inputs are to the right).

```
stat: process(TSP_IN)
          constant tsp_tbl:vlbit_2d(0 to 6,6 down to 0) :=
--                  From the VME Specification – note that no illegal
                                                  - codes are trapped.
                        (
--                      DDAL_TTT
--                      SS0W_SSS
--                      101R_PPP
--                         **D_210
--                         B"XXX0_000",
                        B"0011_001",
                        B"0001_010",
                        B"0101_100".
                        B"1001_101",
                        B"0111_110",
                        B"1011_111"
                        );
                   begin
                   pla_table(TSP_IN,tsp,TSP_TBL);
                   end process;
```

5.6.3 Transaction Identification

To identify a transaction on the VMEbus, the design must look for activity on the ~DSx lines. The following VHDL line does this:

```
CYCLE <= ASSERTED     when     ((DS0 = ASSERTED or DS1 = ASSERTED) and
                                    n_v_rst = N_DE_ASSERTED)
                      else DE_ASSERTED;
```

So CYCLE will assert when a VMEbus cycle begins. At this point, delay and synchronize the signal to the system clock.

```
dly:      process
          begin
          wait until prising(CLK) or CYCLE = '0'
          if prising(clk) then
                    cycle_delay <= CYCLE;
                    cycle_double_delay <= DE_ASSERTED;
                    start <= DE_ASSERTED;
          end if;
          if CYCLE = '0' then
               cycle_delay <= DE_ASSERTED;
               cycle_double_delay <= DE_ASSERTED;
               start <= DE_ASSERTED;
          end if;
          end process;
```

From this logic a single clock pulse length “start” signal is generated. Note that it is delayed two clock cycles to allow all needed address decoding to become valid before the transaction controller starts.

The last item before the transaction controller "runs" the transaction is to identify the type of transaction. This is done by the following VHDL:

```
VME_BUS_IDLE        <= B"000";
HA_CSR_READ         <= B"010";
INTERRUPT_ACK           <= B"011";
ASIC_WRITE          <= B"100";
ASIC_READ           <= B"101";
NOT_OUR_CYCLE           <= B"111";

--          At the beginning of a VME cycle decide what we're doing...

dec:process
            begin
            wait until prising(CLK) or N_SYSRESET = '0';
            if prising(CLK) then

--          Synchronously....

              if CYCLE = ASSERTED then
                  if start = ASSERTED then
                        if WRITE = DE_ASSERTED and CSR_HA = ASSERTED then
                              CYCLE_DECODE <= HA_CSR_READ;
                        elsif IACK = ASSERTED then
                              CYCLE_DECODE <= INTERRUPT_ACK;
                        elsif WRITE = ASSERTED and (CSR_ASIC = ASSERTED or
                             MEM_ASIC = ASSERTED or CSR_HA = ASSERTED) then
                                          CYCLE_DECODE <= ASIC_WRITE;
                        elsif WRITE = DE_ASSERTED and (CSR_ASIC = ASSERTED or
                             MEM_ASIC = ASSERTED) then
                                          CYCLE_DECODE <= ASIC_READ;
                        else
                                          CYCLE_DECODE <= NOT_OUR_CYCLE;
                        end if;
                  end if;
              else
                        CYCLE_DECODE <= VME_BUS_IDLE;
              end if;
            end if;
            if N_SYSRESET = '0' then
                  CYCLE_DECODE <= VME_BUS_IDLE;
            end if;
end process;
```

At this point the logic has identified one of the following possibilities:

- HA_CSR_READ
- INTERRUPT_ACK
- ASIC_WRITE
- ASIC_READ
- NOT_OUR_CYCLE

The first two, HA_CSR_READ and INTERRUPT_ACK, are handled "fully" by this host interface logic. The next two transactions, ASIC_WRITE and ASIC_READ, are moved through to the Rehostable Adapter logic. The last possibility, NOT_OUR_CYCLE, only occurs if the VME cycle does NOT address either CSRs or memory.

5.6.4 Transaction Controller

The transaction controller is a state machine that runs the interfacing handshakes between the Rehostable Adapter and the VME bus.

The transaction identification logic has identified four separate transactions that need to be implemented by the transaction controller.

- INTERRUPT_ACK
- HA_CSR_READ
- ASIC_WRITE
- ASIC_READ

The following is an algorithmic description of the needed activities during each of the specified transactions. Note that the state numbers are shared through the transaction controller state machine.

INTERRUPT_ACK

```
--    Here the goal is to determine if an interrupt is being
--    generated. If so continue processing. If not, PASS allows the
--    interrupt chain to flow through the board, and we will stick in
--    the starting state forever (until the VME cycle ends. The
--    IPA_CODE is the level-equate-on-address lines A3-1 for a hard-
--    coded interrupt level.
```

STATE

```
------
0       Set data port direction to "read"
        if (IPA_CODE==HARD_CODED_LEVEL) and
                    WERE_GENERATING_INTERRUPT
            and not PASS then
            goto 1
        else
            SET PASS
        end if
--      Next we have determined that we are generating the interrupt
--      but we still must wait for the daisy chain to propagate to
--      us. Note that if someone else takes the interrupt before us
--      we will stick in this state forever (until the VME cycle
--      ends).
1       if I_ACK_IN then
            Drive Vector on HD[7:0]
            Turn on DB_EN8-- VME D07-D00
            SET INT_PEND  -- to remove the rehostable interrupt
                           -- generation
            goto 6
        end if
--      Next we generate DTACK to end the cycle. The machine will
--      remain in this state until the end of the cycle in which
--      everything will be reset.
6       Set DTACK
        goto 6    -- This will continue until the VME master terminates
                  -- the transaction.See defaults state condition.
```

HA_CSR_READ

```
--      In this cycle, since we only have one CSR, we simply gate
--      the data and respond to the bus.
0       Set data port direction to "read"
        goto 1
1       Drive Vector on HD[7:0]
        Turn on DB_EN8 -- VME D07-D00
        goto 6
--      Next we generate DTACK to end the cycle. The machine will remain in this
--      state until the end of the cycle in which everything will be reset.
6       Set DTACK
        goto 6    -- This will continue until the VME master terminates
                  -- the transaction. See defaults state condition.
```

ASIC_WRITE

```
--      This is the most complicated host adapter cycle. Here we set
--      the direction, output enable the data buffers and start the
--      asic arbitration activity. Notice this design uses write
--      posting - that is, it acknowledges the bus before the
--      SCRAMNet ASIC has finished processing the cycle.
--      This design also uses a special A_D_LTCH edge-sampling
--      device to make sure that the synchronous design does not
--      miss the rising edge of A_D_LTCH for the current cycle.

0       Set data port direction to "write"
        goto 1
1       Set DB_EN* according to TSP codes.
        if not HREQ_PEND then
            SET HREQ
            UNCLEAR A_D_LTCH_SMP  -- Arm to start looking for
                                  -- A_D_LTCH from rehost
            goto 2
        end if
2       if HREQ_PEND then
            CLEAR HREQ
            goto 3
        end if
3       if A_D_LTCH_SMP then
            goto 6
        end if
--      Next we generate DTACK to end the cycle. The machine will
--      remain in this state until the end of the cycle in which
--      everything will be reset.
6       Set DTACK
        goto 6    -- This will continue until the VME master terminates
                  -- the transaction. See defaults state condition.
```

ASIC READ

```
--      This is the ASIC read cycle. Note that we still must go
--      through the byte justifier even though the ASIC only reads a
---     size of 32 bits...
0       Set data port direction to "read"
        goto 1
1       Set DB_EN* according to TSP codes.
        if not HREQ_PEND then
            SET HREQ
            UNCLEAR A_D_LTCH_SMP -- arm to look for a A_D_LTCH
                                 -- This is to make sure that the
                                 -- ack from the rehost is in sync
                                 -- with this VME transaction and
                                 -- NOT from a previous “Write
                                 -- Posted” write.
            goto 2
        end if
2       if HREQ_PEND then
            CLEAR HREQ
            goto 3
        end if
3       if A_D_LTCH_SMP and HACK then -- We MUST wait for a HACK on
                                      -- a read....
            SET G_A_to_B
            goto 6
        end if
--      Next we generate DTACK to end the cycle. The machine will
--      remain in this state until the end of the cycle in which
--      everything will be reset.
6       Set DTACK
        goto 6    -- This will continue until the VME master terminates
                  -- the transaction. See defaults state condition.
```

DEFAULT STATES

```
--      When CYCLE goes away, the following values should define the
--      default controller state.
        STATE = 000
        Data Port Direction = "write"
        not PASS
        not INT_PEND
        DB_EN* are inactive
        not DTACK
        not HREQ
        not A_D_LTCH_SMP_CLEAR
        not OE for the host CSR [vector]
```

The VHDL code that implements the previous algorithms can be found in Appendix B. It is labeled as “the main VME Host Adapter state machine”. Appendix B is a complete Altera Compile Report.

5.6.5 Interrupt Vector Register

The interrupt vector register is created simply by the following VHDL:

```
--      Latch the interrupt vector when written to. Note that the
--      interrupt vector which is latched is only 7 bits. The 8th
--      bit is supplied by the int_error bit as to produce different
--      vectors based upon different interrupt types.
ff:     process
        begin
        wait until prising(VEC_WRITE_I);
        md_l <<= MD;
        end process;
--      This makes a 8 bit vector by adding the interrupt is a error status
--      bit.
int_vector <<= md_l & err_int;
```

To read the vector or to dump it to the VMEbus during an interrupt acknowledge cycle, an output-enable signal is created. The transaction controller controls the N_VEC_OE signal. The following VHDL performs the actual gating/tristateing of the output bus.

```
HD  <=       int_vector when N_VEC_OE = N_ASSERTED else
             B"ZZZZZZZZ";
```

To write to the vector register, a write pulse must be generated. This pulse is made from the following VHDL:

```
vector_byte  <=    ASSERTED when (TSP_L = B"000" or TSP_L = B"001" or TSP_L =
                       B"111") else DE_ASSERTED;
VEC_WRITE_O  <=    HACK and MA5 and (busy or (WRITE_L and vector_byte and (not
                       STS2)and STS1));
```

The VEC_WRITE_O pulse is a combination of a ***HACK*** and MA5 (upper CSR area) and status from the Rehostable Adapter, which states that a CSR access is being processed. It is further added with a ***HACK*** during the ASIC EEPROM initialization (BUSY=1). This allows the vector register to be initialized from the serial EEPROM in the Rehostable Adapter. The signal VEC_WRITE_O is externally connected to the signal VEC_WRITE_I. This feeds the actual register clock directly.

5.6.6 Interrupt Control Engine

The interrupt controller for the VME rehostable evaluation board is the “standard” implementation. The following VHDL lines generate the interrupt signal:

```
INT1:   process
        begin
        wait until prising(CLK);
        if CS_MODE = `1 then
            MEMORY_INT <=CNTRL_STATUS18;
            ERROR_INT <=CNTRL_STATUS17
            INT_PEND <=INTERRUPT;
        end if;
        end process
        INTERRUPT <=MEMORY_INT or ERROR_INT;
```

The interrupt signal is used to actually generate the **INTx** signal on the VMEbus. (An open collector driver is used to invert the signal and electrically match it to the VME specification). This interrupt signal is also used in the transaction controller state machine to determine if this VME board is generating an interrupt.

5.7 Memory Decoder

5.7.1 Using The CSR decoder

On the original Rehostable VME Interface design the internal (ASIC) CSRs were located on the 16-bit bus at address 0000 *hex* – 001E *hex*. While the external CSRs were located on the 16-bit bus at address 0020 *hex* – 003E *hex*. To allow the testing of multiple Rehostable adapters within the same VME backplane a decision was made to add a comparator for the decoding of CSRs. With the addition of a switch and an 8-bit comparator up to four different CSR I/O locations can be selected.

The following modifications tie this comparator into the existing comparator in the reh_eval controller. Table 5-4 shows the I/O locations and their switch settings.

Table 5-4 CSR_I_O_Locations and Switch Settings

Bus	Switch (S1) Position	ASIC CSRs	External CSRs
16-bit Address Bus	0	0x0000 - 0x001e	0x0020 - 0x003e
16-bit Address Bus	4	0x0040 - 0x005e	0x0060 - 0x007e
16-bit Address Bus	8	0x0080 - 0x009e	0x00a0 - 0x00be
16-bit Address Bus	C	0x00c0 - 0x00de	0x00e0 - 0x00fe

- HA7 has been lifted from the reh_eval controller and grounded.
- HA6 has been lifted from the reh_eval controller and attached to the NOT EQUAL output from a 74F521 comparator.

5.7.2 Using the On-board Rehostable Adapter Memory Decoder

To allow the memory area to move dynamically, the Rehostable Adapter's on-ASIC memory decoder is used. The MEM_ASIC_I path is normally connected to the MEM_ASIC_O output. This path can be interrupted and the input MEM_ASIC_I can easily be wired to the MEM_HIT connection on the Rehostable Adapter.

The last step is to program the width of the address decoder on the Rehostable Adapter. The width can be 32 bits, 24 bits, or 16 bits. The programming of the size is accomplished through values placed upon the MD lines of the rehostable during a period of time shortly after reset. This time is defined by the output of the Rehostable Adapter pin called ~SW_OE0. The programming in this specific case is accomplished by placing a diode between ~SW_OE0 (cathode connected to ~SW_OE0) and DXX. The MD31-0 lines are pulled up internally, and in this case the diode pulls the appropriate data line on the MD bus to ground. This sets the internal-address comparator width to 24 bits. Now the address comparator comparison value is taken from CSR10 and 11.

For further information concerning the internal-address comparators see the SCRAMNet hardware reference manual.

5.8 Memory Subsystems

The Rehostable Node supports addressing of up to 8 MB of SCRAMNet memory. However, due to the small physical size of the Rehostable Node and the density of static RAM chips, the node is only available with 512 KB, 1 MB, or 2 MB of on-board memory. Designs which need less than 512 KB, or more than 2 MB of SCRAMNet memory must implement an external memory system on the host board. A Rehostable Node without on-board memory and the SCRAMNet Memory Card is available for this purpose. The VME host board design example in this manual includes a connector to which will allow the external SCRAMNet Memory Card to be installed providing the user with 4 or 8 MB of SCRAMNet memory.

This page intentionally left blank

6. SERIAL LEDS SUPPORT CIRCUITRY

6.1 Overview

The LEDS support circuitry consists of a single complex programmable logic device (PLD) on the SCRAMNet core circuitry. The output I/O of this circuitry consists of four wires that are routed to the auxiliary connector and to the Media Card connector. These lines contain a proprietary bi-directional serial-protocol interface with the following features:

- The following SCRAMNet Product LEDS are available.
  - INSERT
  - MSG WAITING
  - CARRIER DETECT
  - ERROR
  - OWN_MESSAGE
  - FORIGN_MESSAGE
  - LINK_AB
- The second MICROWIRE port supported by the SCRAMNet ASIC is available.
  - E_CS1 (output)
  - E_DOUT (output)
  - E_CLK (output)
  - E_DIN (input)
- A single multiplexed line containing external trigger information
  - External Trigger #1 or External Trigger #2 selectable
- A network event counter overflow indicator (Divide by 64).
  - Count OWN_MESSAGES
  - Count FGN_MESSAGES
  - Count ALL_MESSAGES
- Remote interrupt bit.

6.2 Interface

The I/O interface to the LEDS part of the support circuitry is as follows:

S_CLK	The serial output clock constantly running at a rate of CLK/4 [9.375 MHz - 106.6 ns]
S_DATA	A bi-directional line used for both output from the LEDS controller and input from an external device.
S_DIR	S_DATA direction indicator. When this line is logic low, the LEDS controller is in output mode. Conversely, when this line is high the LEDS controller is shifting in information from an external device. The timing is set up so that the transition from low to high (rising edge) of this signal can be used to clock the shift register data into a buffering register.

TRIGGER ....A non-serialized/real-time trigger from the ASIC. It is fed from a multiplexer inside the LEDS controller chip. The two possible connections to this line are External Trigger #1 or External Trigger #2. The control of this multiplexer is a bit generated by an external device.

6.3 Serial Protocol

Figure 6-1 defines the serial protocol. Specifically, when the S_DIR bit is low the LEDS controller chip will shift out 11 bits of information. These 11 bits of information are defined below (MSB to LSB - MSB first):

[10]	E_CS1	Chip select for the external MICROWIRE device
[9]	E_DOUT	Data Out for the external MICROWIRE device
[8]	E_CLK	Clock for the external MICROWIRE device
[7]	OVER	Divide-by-64 overflow indication
[6]	INSERT	INSERT LED
[5]	MW	Message Waiting LED
[4]	CD	Carrier Detect LED
[3]	ERROR	ERROR LED
[2]	OWN_MSG	This LED is active when a node receives its own message back
[1]	FGN_MSG	This LED is active when a node receives a message form another node
[0]	LINK_AB	This LED shows which of the dual links are active [on=B]

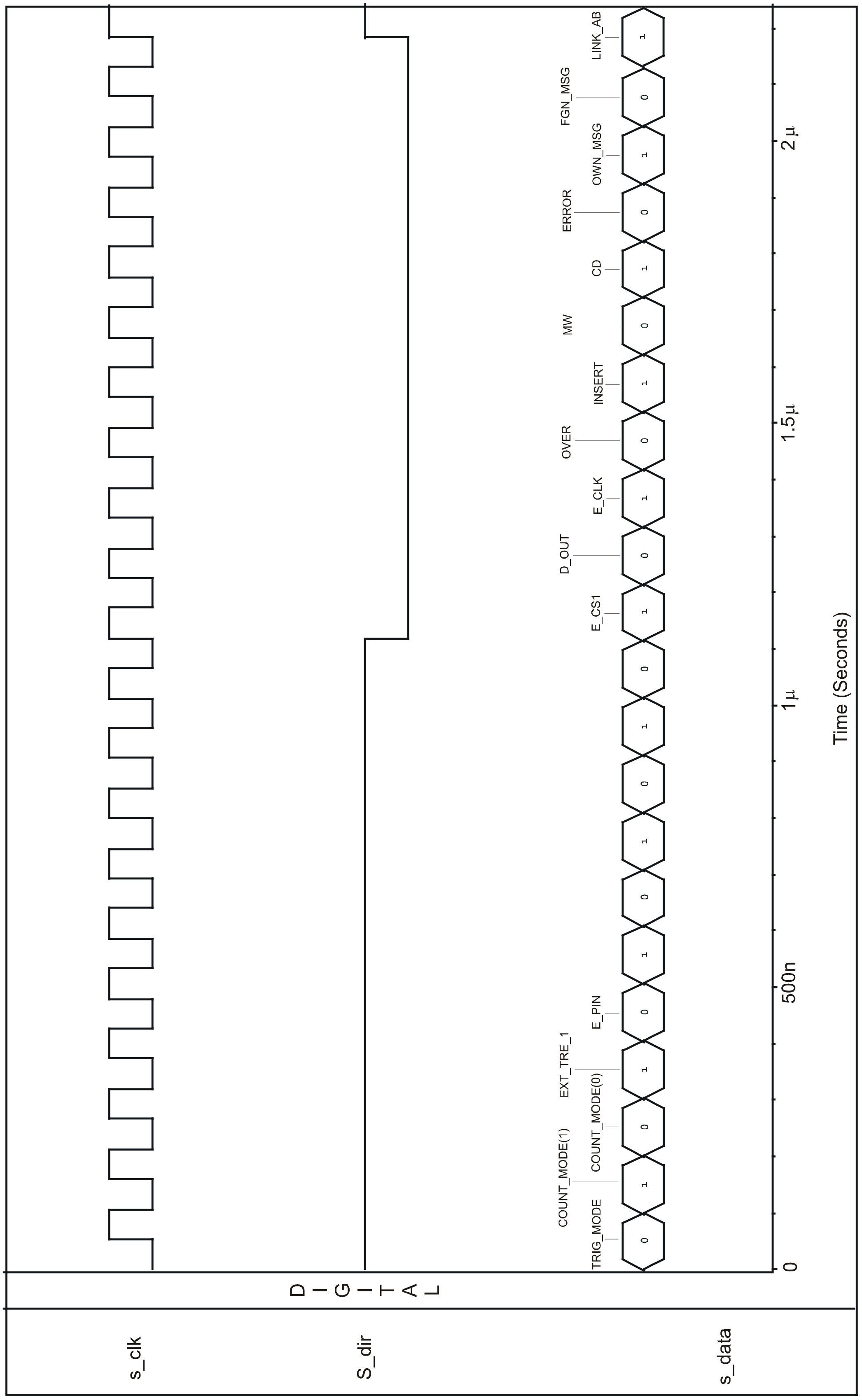


Figure 6-1 Serial Protocol

When S_DIR bit goes from low to high, the incoming shift-register may be latched into a buffer. When the S_DIR bit is high, the LEDS controller will start shifting 11 data bits in from the S_DATA line (note that the LEDS chip will not actively drive the S_DATA line when S_DIR = high). These 11 incoming data bits are defined as follows (MSB to LSB - MSB comes in first):

NOTE: The data is latched on the falling edge of the S_CLK signal.

[10]	trig_mode	This input to the LEDS controller selects which SCRAMNet External trigger to place on the TRIGGER signal line on the auxiliary connector. If this bit is high, then External Trigger #1 is used. If this bit is low, then External Trigger #2 is used.
[9]	count_mode(1)	See count_mode(0) and Table 6-1.
[8]	count_mode(0)	These two bits define various selectable modes in which the divide-by-64 network event counter can work..

Table 6-1 Count_modes

count_mode(1)	count_mode(0)	Explanation
1	1	In this mode the counter counts all network acknowledgements. It also substitutes the current network counter for the following output bits (MSB to SB) CD, ERROR, OWN_MSG, FGN_MSG, LINK_AB.
1	0	In this mode the counter counts other node's network acknowledgements.
0	1	In this mode the counter counts our node's network acknowledgements.
0	0	In this mode the counter counts all network acknowledgements.

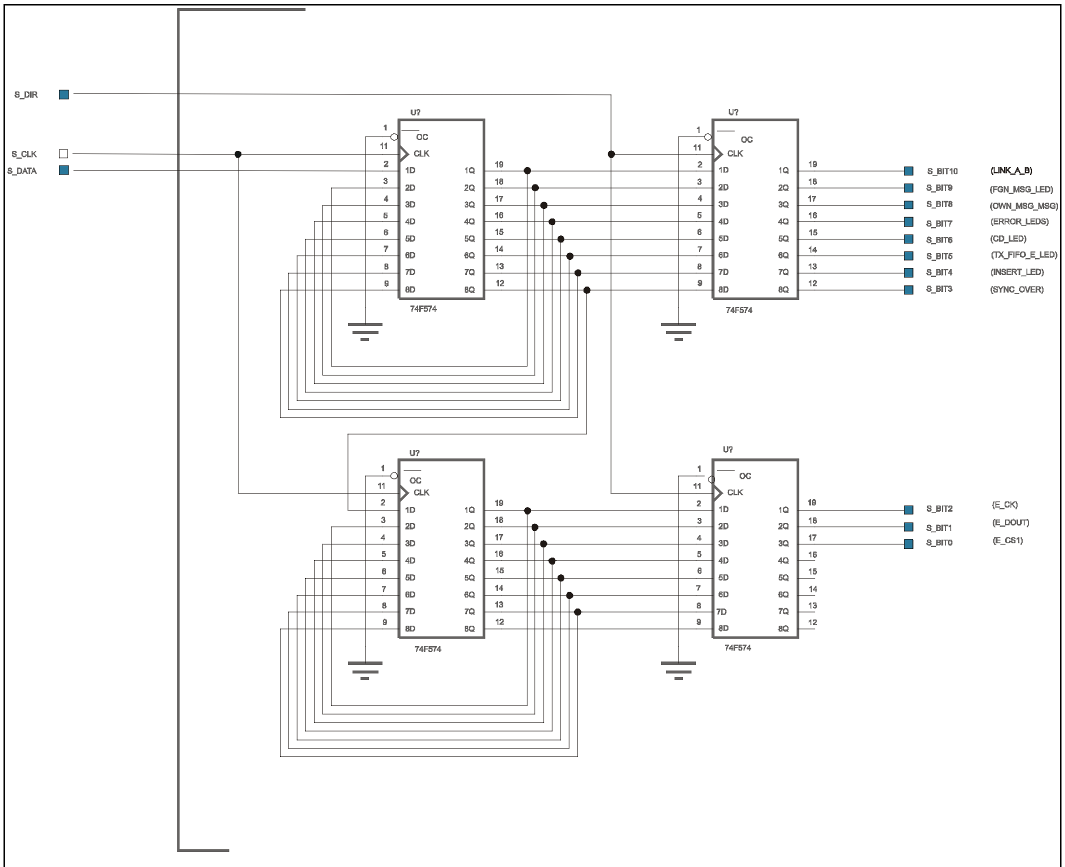
[7]	EXT_TRIG_I	This is an input from the external device that can cause an interrupt on the attached host interface. Its use is non-standard.
[6]	E_DIN_INT	This is the external MICROWIRE device's Data OUT to the ASIC
[5]	(Unused)	These bits are reserved for future use
[4]	(Unused)	These bits are reserved for future use
[3]	(Unused)	These bits are reserved for future use
[2]	(Unused)	These bits are reserved for future use
[1]	(Unused)	These bits are reserved for future use
[0]	(Unused)	These bits are reserved for future use

6.4 User Designs

The LEDS controller was created to facilitate the design of an inexpensive receiver. The simplest design is one using off-the-shelf TTL parts. Figure 6-2 shows a simple receiver circuit using 74x574 octal D flipflops. Here one x574 device is used as a shift register by connecting the Q of one stage to the D of the next. This is done on all stages except the first one, which is connected, to S_DATA. This shift register is clocked by S_CLK. The output of the first x574 is latched by a second x574. The clock on the second x574 is the S_DIR line.

Only the last 8 bits of information is saved/latched from the LEDS controller. This is necessary because the last 8 bits of information correspond to the LED fields from the LEDS controller.

The resister on the S_DATA line can be pulled up or down by the indicated jumper block. This will cause the receiver on the LEDS controller to read either all ones or all zeros. This causes a simple switch to the trigger line. It also has the undesirable effect (when its a `1') of causing the network event counter (internal divide-by-32 binary counter) to be substituted for some LEDS.



NOTE: The serial transmitter (not shown) can be achieved with the same parts plus a few inverters. A simple transmitter for the control of the trigger-mode bit would be a simple pull up/down (I/O) on the S_DATA line. The default mode for the S_DIR mode is zero (low) (i.e., A pull down).

Figure 6-2 Remote Device Serial Receiver

6.5 Simple Receiver Design

The following is a simple Viewlogic synthesizeable VHDL design that may be used to generate a PLD version of the same circuit. This design has been fit and tested in an ALTERA EP610 device.

```
------------------------------------------------------------------
--     The LED Receiver/Driver EPLD
--     V 1.00    3/18/94 (this should fit in a 610)

library pack1076;
use pack1076.pack1076.all;

entity REC is

       port(
              CLK,N_RESET,LOAD:IN vlbit;
              SERIAL_IN:IN vlbit;
              INSERT_LED,TXFIFO_E_LED,CD_LED,ERROR_LED:OUT vlbit;
              OWN_MSG_LED,FGN_MSG_LED,LINK_A_B,sync_over:OUT vlbit
       );

end REC;

architecture desc of REC is

signal shift_reg:vlbit_1d(7 downto 0);
signal hold:vlbit_1d(7 downto 0);
signal reset:vlbit;

begin
--     I Like to work in positive logic.
       reset <= not N_RESET;
sft:   process
       begin
       wait until prising(CLK) or reset = '1';
       if prising(CLK) then
              shift_reg       <=   shiftlum(shift_reg,1);
              shift_reg(0)    <=   SERIAL_IN;
       end if;
       if reset = '1' then
              shift_reg <=('0','0','0','0','0','0','0','0');
       end if;
       end process;

hld:          process
       begin
       wait until prising(LOAD) or reset = '1';
       if prising(LOAD) then
              hold <=shift_reg;
       end if;
       if reset = '1' then
              hold <=('0','0','0','0','0','0','0','0');
       end if;
       end process;

--     From LEDS 5.0

       LINK_A_B              <= hold(0);
       FGN_MSG_LED           <= hold(1);
       OWN_MSG_LED    <= hold(2);
       ERROR_LED             <= hold(3);
       CD_LED         <= hold(4);
```

```
        TXFIFO_E_LED            <= hold(5);
        INSERT_LED              <= hold(6);
        sync_over               <= hold(7);
    end desc;
```

6.6 Full Serial Port Design

Another design that supports the full serial protocol follows. (It is also in synthesizeable Viewlogic VHDL code). The Library “Counters” provide simple synchronous counters. It is available upon request.

```
    -------------------------------------------------------------------
    --  The Digital Netometer EPLD


    --  This design implements the FULL serial interface specification between
    --  the SCRAMNet ASIC based product out of the AUX connector or cabinet
    --  kit interface.  In this design many of the intermediate signals are brought
    --  out of the chip and then brought back in again on separate pins.  This
    --  provides easy trouble shooting and debugging techniques for this design.
    --  It also allows for reuse of the display for other uses.

    --  Targeted, this design fits into a 100 pin ALTERA 7128
CPLD

    --  Revision History
    --  V 1.00  10/25/94
    --  V 1.50  10/26/94    Moved all counters to external pins
    --                      Added FULL Tx / Rx LED interface
    --  V 2.00  11/23/94    Fixed Count C to be 5 bits
    --  V 3.00  12/01/94    Renumbered input bits
    --  V 4.00  12/05/94    Added Inverted clock to serial output process
    library synth;
    use synth.stdsynth.all;

    library counters;
    use counters.counters.all;

    entity DIG_NETO is

        port (
    --      CLK and CLK_2 are connected externally.
            CLK,CLK_2:IN vlbit;
            N_RESET,LOAD:IN vlbit;

    --      S_DATA connection
            SERIAL_IN:INOUT vlbit;

    --      inputs to supply back to the SCRAMNet ASIC based node
            E_DIN_INT,TRIG_MODE,EXT_TRIG_I:IN vlbit;
            COUNT_MODE:IN vlbit_1d(1 downto 0);

    --      Short these 2 together externally
            INPUT_EDGE_O:OUT vlbit;
            INPUT_EDGE_I:IN vlbit;

    --      The 'OVER_X' gets shorted to the 'ENABLE_X' lines
externally
            OVER_A,OVER_B,OVER_C:OUT vlbit;
            ENABLE_A,ENABLE_B,ENABLE_C:IN vlbit;

    --      This should be externally connected to 'SYNC_OVER'
    --      This is the counter input.
```

```
                COUNT_ENABLE:IN vlbit;

--              Short these together externally
                DIV_35_OUT:OUT vlbit;
                D_CNT_ENB:IN vlbit;

--              These are feed forward count 9 overflows for the 3 1/2 digit
--              counter.  The 'D_X_CLEAR_O's are connected to the 'D_X_CLEAR_I's.
                D_A_CLEAR_O,D_B_CLEAR_O,D_C_CLEAR_O: out vlbit;
                D_A_CLEAR_I,D_B_CLEAR_I,D_C_CLEAR_I: in vlbit;

--              These are the latched counter outputs for the 3 1/2 digit counter.
                L_digit_A,L_digit_B,L_digit_C:INOUT vlbit_1d(3 downto 0);
                L_digit_D:INOUT vlbit;

--              These are the 7 Seg outputs for the 3 1/2 digit counter.
                A_SEG,B_SEG,C_SEG:OUT vlbit_1d(6 downto 0);
                D_SEG:OUT vlbit_1d(4 downto 3);

--              These are the serial outputs.
                INSERT_LED,TXFIFO_E_LED,CD_LED,ERROR_LED:OUT vlbit;
                OWN_MSG_LED,FGN_MSG_LED,LINK_A_B,sync_over:OUT vlbit;
                E_CS1,E_CK,E_DOUT: OUT vlbit

                );

end DIG_NETO;

architecture desc of DIG_NETO is

signal Sync_Clear,load_dly:vlbit;
signal input_delayed,input_edge:vlbit;

signal shift_reg:vlbit_1d(10 downto 0);
signal hold:vlbit_1d(10 downto 0);
signal reset:vlbit;

signal zeros: vlbit_1d(5 downto 0);

signal one: vlbit;
signal tmp1,Latch_RESET:vlbit;

signal shift_out: vlbit_1d(4 downto 0);

signal r_countA,r_countB: vlbit_1d(7 downto 0);
signal r_countC: vlbit_1d(5 downto 0);

signal digit_A,digit_B,digit_C,zeros_D: vlbit_1d(3 downto 0);
signal digit_D: vlbit;

signal SCALE: vlbit_1d(5 downto 0);

--    This is for the LED decoders....
--    It provides a table lookup for the 7 Seg outputs.
constant tbl : vlbit_2d(0 to 11, 10 downto 0 ) := (
--
--        2
--      1   3
--        6
--      0   4
--        5
--     bcd(3:0) | seg(6:0)
-----------------+----------
          B"0000_0111111",
          B"0001_0011000",
          B"0010_1101101",
          B"0011_1111100",
          B"0100_1011010",
          B"0101_1110110",
          B"0110_1110111",
          B"0111_0011100",
          B"1000_1111111",
          B"1001_1111110",
          B"101X_XXXXXXX",
          B"11XX_XXXXXXX"
                              );

begin

--   The main goal behind this design is to build a 3 1/2 digit netometer.  Along
--   with this goal, a fully supported 2'nd microwire port is available. The following
--   are the design steps involved with the netometer.

--   Functionally the digital netometer is a frequency counter.  It has an imbedded
--   division factor and update rate circuitry.  Specifically, it is made up of an
--   'update' timing generator, a prescaler to adjust the rate to the update rate, and
--   a 3 1/2 digit bcd counter.  Output latches are used to give a constant output,
--   and 7 Seg outputs are available to directly drive LED displays.

--   One large counter is used to generate a 'slow' rate at which the display is
--   updated and the counters are reset. Another binary counter is used to generate
--   the input prescaleing factor, and finally 3 and 1/2 independent bcd counters are
--   cascaded together to input to the display latches.
--  The size and number of bits involved with the "update" counter and the prescaler
--   counter were determined mathematically.  The following steps are used to
--   compute these.

--   1.   The target update rate of the display is 2Hz.  The rate of the constant
--        clock S_CLK is 9.375 Mhz. To achieve this rate a divide by 4,687,500
--        would be needed.
--==>          A.   2^22 = 4,194,304. This 22 bit counter divider would then result
--             in a 2.2351 Hz update rate.

--   2.   When SCRAMNet is running at 6.5MB/s this is equivalent to a 1.625 Mhz
--        rate. [this is because SCRAMNet transfers 4 bytes at a time] Internal scaling
--        on this network event update rate is a divide by 32 factor. This results in a
--        rate of 50.78Khz being presented to this device for a throughput of 6.50 MB
--        /sec.

--   3.   This means at a reset/update rate of 2.2351 Hz we want a count of 650 on
--        our display.
--        A.   2.2351 Hz = 447.40 ms in time.
--        B.   At 50.78 KHz 22719.34 counts would occur in 447.40ms without any
--             additional scaling.
--        C.   Since the target readout is 650, the scaling factor = 22719.34 /
--             650 = 34.95.
--==>     D.   Rounding this to 35 only gives a .14% error

     reset <=  not N_RESET;
--   Divide by 35 prescaler

--   COUNT_ENABLE is connected to sync_over

--   INPUT_EDGE provides a 1 clock event when a input count is received.
     INPUT_EDGE_O <=   '1' when COUNT_ENABLE = '1' and input_delayed
                    = '0' else
               '0';
```

```
        zeros <=B"000000";

--      Synchronously Loadable Up-Counter
--         procedure counter_L (    signal clk:in vlbit;
--                                  signal enable:in vlbit;
--                                  signal reset:in vlbit;
--                                  signal bus_C:inout vlbit_1d;
--                                  signal load:in vlbit;
--                                  signal input:in vlbit_1d
--                              );

        counter_L(CLK,INPUT_EDGE_I,reset,SCALE,Sync_Clear,zeros);

        Sync_Clear <=     '1' when SCALE = B"100011" else '0';

--      Again create a single clock pulse overflow.
        DIV_35_OUT  <= '1' when Sync_Clear = '1' and load_dly = '0' else '0';

--      This does the synchronous divide by 2^22 [Latch_RESET]
--      This is done by cascading 2 8 bit and one 6 bit counter.

        one <='1';
        tmp1<=  ENABLE_A and ENABLE_B;

--      Latch_RESET is when they all overflow.

        Latch_RESET  ENABLE_A and ENABLE_B and ENABLE_C;
        Latch_RESET <= ENABLE_A and ENABLE_B and ENABLE_C;
        counter(CLK,one,reset,r_countA);
        counter(CLK,ENABLE_A,reset,r_countB);
        counter(CLK,tmp1,reset,r_countC);

--      OVER_x is connected to ENABLE_x

        OVER_A  <= '1' when r_countA = B"11111111" else '0';

        OVER_B  <= '1' when r_countB = B"11111111" else '0';

        OVER_C  <= '1' when r_countC = B"111111" else '0';


--      3 and 1/2 Digit counter...
--      D_CNT_ENB is connected to DIV_35_OUT

        zeros_D <=B"0000";
--      D_A_CLEAR_I is connected to D_A_CLEAR_O

        counter_L(CLK,D_CNT_ENB,reset,digit_A,D_A_CLEAR_I,zeros_D);
        D_A_CLEAR_O <=     '1' when (D_CNT_ENB = '1' and digit_A = B"1001") or
                           Latch_RESET = '1' else '0';

--      D_B_CLEAR_I is connected to D_B_CLEAR_O

        counter_L(CLK,D_A_CLEAR_I,reset,digit_B,D_B_CLEAR_I,zeros_D);
        D_B_CLEAR_O <=     '1' when (D_A_CLEAR_I = '1' and digit_B = B"1001") or
                           Latch_RESET = '1' else '0';

--      D_C_CLEAR_I is connected to D_C_CLEAR_O

        counter_L(CLK,D_B_CLEAR_I,reset,digit_C,D_C_CLEAR_I,zeros_D);
        D_C_CLEAR_O <=     '1' when (D_B_CLEAR_I = '1' and digit_C = B"1001") or
                           Latch_RESET = '1' else '0';

--      This is the process for digit D.  Note that it is only a single bit.
```

```
dig_d:  process
        begin

        wait until prising(CLK) or reset = '1';

        if prising(CLK) then
                if Latch_RESET = '1' then
                        digit_D <= '0';
                else
                        if D_C_CLEAR_I = '1' then
--                              Toggle
                                if digit_D = '1' then
                                        digit_D <='0';
                                else
                                        digit_D <='1';
                                end if;
                        end if;
                end if;
        end if;

        if reset = '1' then
                digit_D <='0';
        end if;
        end process;

--      This Latches the output for the displays

ltch:   process
        begin

        wait until prising(CLK) or reset = '1';
        if prising(CLK) then
                if Latch_RESET = '1' then
                        L_digit_A <=digit_A;
                        L_digit_B <=digit_B;
                        L_digit_C <=digit_C;
                        L_digit_D <=digit_D;
                end if;
        end if;

        if reset = '1' then
                L_digit_A <=B"0000";
                L_digit_B <=B"0000";
                L_digit_C <=B"0000";
                L_digit_D <='0';
        end if;
        end process;

--      These are the Display decoders.....

        pla_table( L_digit_A, A_SEG, tbl );
        pla_table( L_digit_B, B_SEG, tbl );
        pla_table( L_digit_C, C_SEG, tbl );

        D_SEG(4 downto 3)  <=   B"11" when L_digit_D = '1' else B"00";

--      This process is used to create some delayed signals used for single
--      clock edge detection.

dly:    process
        begin

        wait until prising(CLK) or reset = '1';
        if prising(CLK) then
                input_delayed <=COUNT_ENABLE;
                load_dly <=Sync_Clear;
        end if;

        if reset = '1' then
                input_delayed <='0';
                load_dly <='0';
        end if;

        end process;

--      The following is the output direction serial transmitter. This transmits
--      bits on the Falling edge of S_CLK.  Note that these are latched on the falling
--      edge of S_CLK on the SCRAMNet ASIC based product.

--      Also note that 'CLK_2' is externally connected to CLK.  This is to allow CLK to
--      be chosen as the global clock.

SERIAL_IN  <=   shift_out(4) when load = '1' else 'Z';
sft_o:  process
        begin

        wait until pfalling(CLK_2) or reset = '1';

        if prising(CLK) then
                shift_out <=shiftlum(shift_out,1);
                shift_out(0) <='0';

                if LOAD = '0' then
--                      Load output Register...
                        shift_out(4)  <=TRIG_MODE;
                        shift_out(3)  <=COUNT_MODE(1);
                        shift_out(2)  <=COUNT_MODE(0);
                        shift_out(1)  <=EXT_TRIG_I;
                        shift_out(0)  <= E_DIN_INT;
                end if;
        end if;

        if reset = '1' then
                shift_out <=B"00000";
        end if;

        end process;

--      This is the serial receiver.  It serially shifts in bits and latched
--      them on a rising edge of LOAD (S_DIR).

sft:    process
        begin

        wait until prising(CLK) or reset = '1';

        if prising(CLK) then
                shift_reg <=shiftlum(shift_reg,1);
                shift_reg(0) <=SERIAL_IN;

        end if;

        if reset = '1' then
                shift_reg <=B"00000000000";
        end if;
        end process;

hld:    process
        begin

        wait until prising(LOAD) or reset = '1';

        if prising(LOAD) then
                hold <=shift_reg;
        end if;
        if reset = '1' then
                hold <=B"00000000000";
        end if;

        end process;

--      This is a dissection of the latch register to the individual bits.
--      From LEDS 6.0

        LINK_A_B                <= hold(0);
        FGN_MSG_LED             <= hold(1);
        OWN_MSG_LED             <= hold(2);
        ERROR_LED               <= hold(3);
        CD_LED                  <= hold(4);
        TXFIFO_E_LED            <= hold(5);
        INSERT_LED              <= hold(6);
        sync_over               <= hold(7);
        E_CK                    <= hold(8);
        E_DOUT                  <= hold(9);
        E_CS1                   <= hold(10);

end desc;
```

6.7 Cables

Signal integrity is an issue on these high-speed signals (9.375 MHz). For cables less than one meter, there should be no problem. However, for longer length runs, quality cable along with special receiver circuitry should be used to guarantee operation.

This page intentionally left blank

APPENDIX A

CSR DESCRIPTIONS

TABLE OF CONTENTS

A.1 Description

This section describes each Control/Status Register and the function of each bit. The name of each bit is indicative of its set state.

The registers are described using bit 0 as the Least Significant Bit (LSB). For example: Inserting A7C3 *hex* in a 16-bit register would set bits 0, 1, 6, 7, 8, 9, 10, 13, and 15 ON.

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

Table A-1 CSR0

Bits	General SCRAMNet Control (READ/WRITE)
1-0	**Network Communications Mode** – Bit 0 controls the receive enable, and Bit 1 controls the transmit enable.
	00 **None** - In this mode, all communications between the node shared memory and the network is inhibited. The node is still able to pass network traffic but does not receive or transmit any data. Loopback modes are also meaningless unless the Host to Shared Memory write bit is enabled.
	01 **Receive Only** - In this mode, any received message is processed and written to node shared memory. Data written by the host is placed in the node shared memory and in the Transmit FIFO but is not sent out on the network. In this mode, the Transmit FIFO will fill and if the Error Interrupt is enabled, Transmit FIFO full interrupt will be triggered. Before changing modes from Receive-Only to either Transmit-Only or Transmit/Receive, the Transmit FIFO should be cleared. If not, all buffered transmit messages will be sent out on the network.
	10 **Transmit Only** - In this mode, any received message bypasses the shared memory and is passed on. Any message written by the host to node shared memory is transmitted on the network. However, any received message is not written to node shared memory. (Transmissions are subject to data filter characteristics.)
	11 **Transmit/Receive** - In this mode, any received message is processed, written to node shared memory and passed on. Any message written by the host to node shared memory is transmitted on the network. This is the normal operation. (Transmissions are subject to data filter characteristics.)
2	**Redundant Transceiver Toggle** - When this bit is cycled ‘0’, ‘1’, ‘0’, the optional redundant transceiver selected link is changed.
3	**Host Interrupt Enable** - When this bit is set, a received message that is written to node shared memory as an interrupt will generate an interrupt request, and the address will be written to the Interrupt FIFO. This bit must be set in order to receive any interrupts from the network.
4	**Auxiliary Control RAM Enable** - When this bit is set, the ACR bytes are swapped in place of the corresponding least-significant byte of every four-byte word in SCRAMNet memory. The values written to those ACR byte locations will dictate the type of interrupt that will occur when the 4-byte memory location is written into. The ACR has five bits for interrupt control. They are as follows:
	ACR bit 0 - **Receive Interrupt Enable** - Setting this bit generates an interrupt to the host for network interrupt data received in this location.
	ACR bit 1 - **Transmit Interrupt Enable** - Setting this bit generates an interrupt to the network for a host WRITE to this shared memory location.
	ACR bit 2 - **External Trigger 1** - Setting this bit generates a trigger signal to an external connector whenever there is a host read/write access to this shared memory location.
	ACR bit 3 - **External Trigger 2** - Setting this bit generates a trigger signal to an external connector whenever there is a network write to this shared memory location.
	ACR bit 4 - **HIPRO Location Enable** - Setting this bit causes the two 16-bit data or four 8-bit items within the 32-bit address boundary to be transmitted as one 32-bit network message. CSR2[13] must also be set for this action to occur.
5	**Interrupt On Memory Mask Match Enable** - This bit must be set in order for any type of memory interrupt to occur.

Table A-1 CSR0

Bits	General SCRAMNet Control (READ/WRITE)
6	**Override Receive Interrupt Enable Flag** - When this bit is set, an interrupt is generated to the host by any interrupt data received from the network regardless of the status of the ACR Receive Interrupt bit.
7	**Enable Interrupt on Error -** When this bit is set, Interrupt FIFO Full, Protocol Violation, Bad Message and/or Receiver Overflow conditions causes an interrupt request.
8	**Network Interrupt Enable** - This bit must be set to transmit interrupt data to the network.
9	**Override Transmit Interrupt Enable Flag** - When this bit is set, an interrupt is sent out on the network regardless of the status of the ACR Transmit Interrupt bit.
10	**Enable Transmit Data Filter** - When clear, the entire address space is not filtered and the node is capable of transmitting all messages written to the node shared memory by the host on the network. When set, the data-filter function is enabled for the address space above the first 4 K bytes of **SCRAMNet** memory. Bit 11 controls the lower 4 K bytes.
11	**Enable Lower 4 K Bytes For Data Filter -** When set, the lower 4 K bytes of address space is data filtered if bit 10 is also set. When disabled, the address space will not be filtered.
12	**Reset Receive/Transmit FIFO** - This bit must be toggled from ‘0’ to ‘1’ and back to ‘0’ in order to reset the Receive/Transmit FIFO. The R/T FIFO is a temporary high-speed holding area for data flowing through the network. **NOTE**: If the R/T FIFO were to be reset during active network transmissions, the data in the FIFO at that time would be lost and it would cause errors on the downstream nodes in the network ring.
13	**Reset Interrupt FIFO** - This bit must be toggled from ‘0’ to ‘1’ and back to ‘0’ to reset the Interrupt FIFO.
14	**Reset Transmit FIFO** - This bit must be toggled from ‘0’ to ‘1’ and back to ‘0’ to reset the Transmit FIFO.
15	**Insert Node** - This bit controls the nodes communications mode on the network as either a receiver only or a receiver/transmitter. On power-up, this bit is OFF which translates to the receiver-only mode. This allows user-written software (on each host processor on the network) to be initiated from one node whenever the network is started cold. When this bit is ON, the node is “inserted” into the network ring as a receiver/transmitter which is the normal operating mode if the Fiber Optic Loopback CSR2[6] is disabled. This bit is invalid when the Enable Wire Loopback CSR2[7] is ON.

Table A-2 CSR1	
Bits	**SCRAMNet Errors (READ Only with WRITE/RESET for interrupts)** Reading CSR1 will reset the latched error conditions by clearing bits 0,2,4,6,7,8,9,10,11,12,13.
0	**Transmit FIFO Full (Latched)** - When this bit is set, the Transmit-FIFO-Full condition exists. This occurs when there is more data coming from the host to the network than the network can absorb. When the shared memory is full, host WRITEs will be held off by the SCRAMNet host interface logic until the Transmit FIFO is no longer full.
1	**Transmit FIFO Not Empty** - This bit does not represent any type of error condition, but rather just a report on the state of the Transmit FIFO. A ‘0’ represents an empty FIFO, where a ‘1’ indicates at least one message in the FIFO.
2	**Transmit FIFO 7/8 Full (Latched)** - This bit indicates that the Transmit FIFO is 7/8 full. A ‘0’ represents a FIFO that is less than 7/8 full, where a ‘1’ indicates the FIFO is backing up and is more than 7/8 full.
3	Always 0
4	**Interrupt FIFO Full (Latched)** - When this bit is set, the Interrupt FIFO Full error condition exists. Reset the Interrupt FIFO by toggling CSR0[13] to ON then to OFF.
5	**Protocol Violation (Latched)** - When this bit is ON, there has been a signal error at the physical layer (fiber or coax) resulting from noise on the transmission lines or a result of hardware failure. It can be any one of the following: Missing transition for two clock periods on either line, Parity error or a Framing error.
6	**Carrier Detect (Latched)** - This bit is set if the receivers do not detect any or enough output from the previous nodes transmitters. This is usually an indication that the fiber optic lines have become disconnected or there may be dust/dirt where the fiber optic connections have been made. A visual inspection of the network lines will need to be made.
7	**Bad Message (Latched)** - When this bit is set, the hardware has detected an error in the message packet received on the network. If this error persists, it is an indication that a hardware problem on the **SCRAMNet** board may exist.
8	**Receiver Overflow (Latched)** - When this bit is set, the Receive FIFO has received more data than the node is able to process. This condition may indicate a hardware problem on the board.
9	**Transmit Retry (Latched)** - This bit is set if a message returns to the originating node with bit errors. The message is automatically retransmitted indefinitely until it returns without bit errors. This is considered to be an error condition.
10	**Transmit Retry Time-out (Latched)** - This bit is set if a message does not return to the originating node within the time-out value specified in CSR5. The message is automatically retransmitted indefinitely until it returns. This is considered to be an error condition.
11	**Redundant Transmit/Receive Fault (Latched)** - This bit is set if the currently selected optional redundant transceiver has faulted and reverted to the other link. The default value is ‘0’
12	**General Purpose Counter/Timer Overflow (Latched)** - This bit toggles a 16-bit counter/timer. The events to be counted/timed are set using CSR8[9]; CSR9[13]; and CSR9[14]. The output is held in CSR13. The counter/timer can: count errors, count trigger events for triggers 1 and 2, transmit time, network events, free run @ 26.66 ns, and free run @ 1.706 ns with trigger 2 CLEAR.
13	**Current Link (Latched)** - This bit tells which of the optional redundant transceivers is currently selected as the active link. The default value is 1=A.
14	**Interrupts Armed** - During interrupt operation, this bit indicates that the conditions to receive an interrupt are active. If this bit is ‘0’, then the host will receive no interrupts. When CSR1 is written to, then the interrupts-armed bit will return to an active status.

Table A-2 CSR1

Bits	**SCRAMNet Errors (READ Only with WRITE/RESET for interrupts)** Reading CSR1 will reset the latched error conditions by clearing bits 0,2,4,6,7,8,9,10,11,12,13.
15	**Fiber Optic Bypass Not Connected** - This is a status bit concerning the installation of the optional Fiber Optic Bypass Switch. A ‘0’ in this bit indicates that the bypass switch is installed while a ‘1’ indicates it is not installed. Fiber Optic Loopback mode CSR2[6] is dependent upon the Fiber Optic Bypass Switch being installed.

Table A-3 CSR2	
Bits	**General SCRAMNet Control (READ/WRITE)**
5-0	These bits are related to lines connected through the MUX control port and are available to the host interface. They are not required to connect to anything
6	**Disable Fiber Optic Loopback** - When this bit is ‘0’ (power up default), the output of the transmitter is connected by fiber optics directly to the input of the receiver, and the receiver is disconnected from the network. The optional Fiber Optic Bypass Switch must be installed for this mode to be effective. This mode is valid only when the Insert Node CSR0[15] is ON. Set this bit to disable the loopback mode when the node is in use as a part of the network.
7	**Enable Wire Loopback** - When this bit is set, the output of the transmitter is connected by wire directly to the input of the receiver, and the receiver is disconnected from the network. The purpose of this bit is purely diagnostic. This mode is valid only when the Insert Node CSR0[15] is OFF.
8	**Disable Host to Memory Write** - When this bit is set, the host WRITEs are not written to the host node’s shared memory, but are sent out on the network if Transmit CSR0[1] is ON.
9	**Write Own Slot Enable** - When this bit is set, the originating node receives the message slot (or packet) it sent out to the network. This is not the normal procedure but may be used in conjunction with CSR2[10] when it is desired to generate an interrupt to the host, written by the host.
10	**Enable Interrupt On Own Slot** - When this bit is set, a message with the interrupt bit set can be received by the originating node if CSR2[9], is also set. This coupling enables a host processor to interrupt itself (Self Interrupt).
11	**Message Length Limit** - Variable maximum message size: 1024 bytes or 256 byte. It is used in conjunction with CSR2[12], CSR2[14] and CSR2[15] to enable Plus mode communication protocols.

Write-Me-Last/Self-Interrupt Mode Definition

CSR2[10]	CSR2[9]	CSR2[8]	Mode
0	1	1	WRITE ME LAST mode
1	1	0	SELF-INTERRUPT mode
1	1	1	WRITE ME LAST with SELF-INTERRUPT mode

Table A-3 CSR2 (continued)	
Bits	**General SCRAMNet Control (READ/WRITE)**
12	**Variable Length Messages on Network** - When ON, this bit enables variable length messages. It is used in conjunction with CSR2[11], CSR2[14] and CSR2[15] to enable PLUS mode communication protocols (see below).
13	**HIPRO Enable** - When this bit is set, the two 16-bit shortwords associated with the longword addressed at ACR[4], will be transmitted onto the network as one 32-bit longword. The first shortword write will be held until the second shortword write occurs, which results in the 32-bit data value to be written to the network. Exceptions: HIPRO will not work when Disable Host to Memory write CSR2[8] is set. HIPRO will not work when writing two separate shortwords while using interrupts.
14	**Multiple Messages** - This bit allows multiple native messages on the network. It is used in conjunction with CSR2[11], CSR2[12] and CSR2[15] to enable the BURST mode communication protocol (see below).
15	**No Network Error Correction** - This bit is used in conjunction with CSR2[12] and CSR2[14] to enable communication protocols: BURST or PLATINUM mode and the variable length message PLUS (+) mode (see below).

SCRAMNet Protocol Mode Definition

Network Mode	CSR2[15]	CSR2[14]	CSR2[12]	CSR2[11]
	No Error Correction	**Multiple Message**	**Variable Length**	**Message Size Maximum**
BURST	1	1	0	NO MEANING
PLATINUM	0	1	0	NO MEANING
BURST+	1	1	1	1=1024, 0=256
PLATINUM+	0	1	1	1=1024, 0=256

Table A-4 CSR3

Bits	Number of nodes and Tx Node ID (Rx ID) and Transmitted AGE (READ ONLY)
7-0	**Node Number Count** - These bits represent the total number of **SCRAMNet** nodes on the network. The hardware dynamically determines this value. The value ranges from 0 to 255 depending upon the number of nodes actually on the network. **Transmit Age**. This field is also used to READ/WRITE the T_AGE[7:0] field. This register reflects this field when the ID_MUX bit in CSR8[0] is set.
15-8	**Node Identification Number** - These bits represent the **SCRAMNet** node identification number. Each node must have a unique identification number from 0 to 255 for each network ring. The NODE ID need not be in sequential order. **Receive ID**. This field is also used to READ/WRITE the RXID[7:0] field. This register reflects this field when the ID_MUX bit in CSR8[0] is set.

Table A-5 CSR4

Bits	Interrupt FIFO Address (Lower 16 Bits) (READ ONLY)*
15-0	**LSP of the Interrupt Address** - These bits represent the LSP of the interrupt address (A15 - A0). Bits 0 and 1 are always ‘0’ since the addresses are on four-byte boundaries.

Table A-6 CSR5

Bits	Interrupt FIFO Address (Upper 7 Bits) (READ ONLY)*
6-0	**MSP of the Interrupt Address** - These 7 bits represent the MSP of the interrupt address (A22 - A16). When coupled with CSR4, this address represents the SCRAMNet memory location of the interrupt.
13-7	Reserved.
14	**Retry Interrupt FIFO Bit** - This bit is set when an interrupt message is received that has its message retry bit set. This can be checked in the interrupt service routine to guard against double interrupts from the same message if it happens to be retransmitted.
15	Interrupt FIFO Not Empty - When this bit is clear, the Interrupt FIFO is empty. Do not READ CSR4 when this bit is ‘0’. When this bit is set, it signals that CSR5 and CSR4 contain a legitimate interrupt address.

* Writing the Transmit Time-out value to CSR5 stores it in shadow memory. Do not set this value to ‘0’. A value of ‘0’ prevents host-generated data from leaving the Transmit FIFO.

Table A-7 CSR6	
Bits	**Not implemented in the Rehostable Adapter design**
15-0	Reserved.

Table A-8 CSR7	
Bits	**Not implemented in the Rehostable Adapter design**
15-0	Reserved.

Table A-9 CSR8	
Bits	**Rehostable Adapter ASIC Extended Control and Status Register**
0	**ID Multiplex** - When set to 1, CSR3 contains the T_AGE and RXID fields.
1	**Disable Holdoff** - When set, this bit disables the HOLDOFF feature.
7-2	These bits are used for programming the EEPROM.
8	**CSR Reset** - Setting this bit will cause bus errors. On reset, CSRs loads from EEPROM.
9	**General Purpose Counter/Timer Free Run** - Setting this bit causes the GPC to free run at a rate of 37.5 MHz (26.66 ns). This counter mode overrides all other counter mode settings.
10	**Receive Interrupt Override** - When this bit is set, all incoming network messages are treated as interrupt messages.
11	**Mechanical Switch Override** - Normally set to ON. When OFF, Mechanical Switch Loopback Mode is invoked.
14-12	**Memory Size Configuration** - These bits indicate the memory-size code and are used in conjunction with the memory address stored in CSR10 and 11. The memory size is automatically calculated. (See below)
15	Reserved. (Always 1).

Memory Size Configuration

Bit 14	Bit 13	Bit 12	Memory Size
1	1	1	4 KB
1	1	0	128 KB
1	0	1	512 KB
1	0	0	1 MB
0	1	1	2 MB
0	1	0	4 MB
0	0	1	8 MB

Table A-10 CSR9

Bits	Rehostable Adapter Error Interrupt Mask Register
0	Transmit FIFO Full Mask
1	Transmit FIFO not Empty Mask
2	Transmit FIFO 7/8 Full Mask
3	Built In Self Test Stream (BIST) - Internal 82-bit BIST shift register output.
4	Interrupt FIFO Full Mask
5	Protocol Violation Mask
6	Carrier Detect Fail Mask
7	Bad Message Mask
8	Receiver Overflow Mask
9	Transmitter Retry Mask
10	Transmitter Retry Due to Time Out Mask
11	Redundant TX/RX Fault Mask
12	Interrupt on General Purpose Counter/Timer Overflow Mask
13	See Below
14	See Below
15	Fiber Optic Bypass Switch Not Connected Mask

General Purpose Counter/Timer Modes

CSR8[9]	CSR9[14]	CSR9[13]	Counter/Timer Modes
0	0	0	Count Errors
0	0	1	Count Trigs (1&2)
0	1	0	Transit Time
0	1	1	Network Events
1	1	X	Free Run @ 26.66 ns
1	0	1	1.706 µs w/trig 2 CLR

* To enable an On-Error mask, set the bit to ‘1’.

Table A-11 CSR10

Bits	Rehostable Adapter Shared Memory Address (LSW) and ENABLE; replaces physical switches	
0	SMA_ENABLE	**Shared Memory Address Enable**. This bit enables the on-ASIC comparator for shared-memory access.
11-1	-0-	Always zero
12	SMA12	**Shared Memory Address**
13	SMA13	
14	SMA14	
15	SMA15	

Table A-12 CSR11

Bits	Rehostable Adapter Shared Memory Address (MSW) ; replaces physical switches	
0	SMA16	This is the most significant part of the replicated shared memory.
1	SMA17	
2	SMA18	
3	SMA19	
4	SMA20	
5	SMA21	
6	SMA22	
7	SMA23	
8	SMA24	
9	SMA25	
10	SMA26	
11	SMA27	
12	SMA28	
13	SMA29	
14	SMA30	
15	SMA31	

Table A-13 CSR12

Bits	Rehostable Adapter Virtual Paging with ENABLE	
0	VP	**Virtual Paging Enable**. When ON, this bit enables Virtual Paging.
4-1	-0-	Always zero
5	VP_A12	**Virtual Page number.** The significance of this register is dependent on the memory size. (e.g. For 4 MB, only VP_A22 is valid; for 4 KB, VP_A[22:12] are valid.
6	VP_A13	
7	VP_A14	
8	VP_A15	
9	VP_A16	
10	VP_A17	
11	VP_A18	
12	VP_A19	
13	VP_A20	
14	VP_A21	
15	VP_A22	

Table A-14 CSR13

Bits	Rehostable Adapter General Purpose Counter/Timer	
0	RD_COUNT[0]	This is a General Purpose Counter/Timer register. It can be used to count trigger 1 and 2 events, count errors, or other events as programmed by CSR9, bits 13 and 14.
1	RD_COUNT[1]	
2	RD_COUNT[2]	
3	RD_COUNT[3]	
4	RD_COUNT[4]	
5	RD_COUNT[5]	
6	RD_COUNT[6]	
7	RD_COUNT[7]	
8	RD_COUNT[8]	
9	RD_COUNT[9]	
10	RD_COUNT[10]	
11	RD_COUNT[11]	
12	RD_COUNT[12]	
13	RD_COUNT[13]	
14	RD_COUNT[14]	
15	RD_COUNT[15]	

Table A-15 CSR14

Bits	Not implemented in the Rehostable Adapter design
15-0	Not Used

Table A-16 CSR15

Bits	Not implemented in the Rehostable Adapter design
15-0	Reserved.

Table A-17 CSR16

Bits	Not implemented in the Rehostable Adapter design
15-0	Reserved

APPENDIX B

VHDL SOURCE

TABLE OF CONTENTS

B.1 Basic VME Interface

```
--    This is the Basic VME interface the rehostable evaluation
--    Version 1.0 - 7/6/94
--    Version 2.0 - 7/7/94
--    Version 3.0 - 7/8/94
--    Version 3.5 - 7/18/94
--    Version 3.6 - 7/28/94 - Found bug in vector and CSR oe
--    Version 3.7 - 8/1/94  - Changed cycle to be more readable
--    Version 3.8 - 8/15/94  - Fixed the problem of gating during null
--    Version 3.9 - 8/16/94  - Fixed Writes
--    Version 4.0 - 8/16/94  - Added internal reset for combinational reset

library synth;
use synth.stdsynth.all;


entity reh_cnt1 is
   port (   signal CLK:                          in vlbit; -- 37.5Mhz from rehost


--          VME Signals....

            signal N_DS0,N_DS1,N_SYSRESET:       in vlbit;
            signal N_LWORD:                      in vlbit;
            signal N_WRITE,N_IACK:               in vlbit;

            signal A:                            in vlbit_1d(3 downto 1);
            signal VME_A23:                      in vlbit;
            signal VME_A:                        in vlbit_1d(15 downto 5);
            signal AM :                          in vlbit_1d(5 downto 0);

            signal N_IACK_IN:                    in vlbit;

            signal N_IACK_OUT:                   out vlbit;

            signal DTACK:                        out vlbit;


--          Local Data bus and output enables [for interrupt vector]

            signal N_VEC_OE:                     in vlbit;
            signal N_OE_VEC:                     out vlbit;

            signal VEC_WRITE_O:                  out vlbit;
            signal VEC_WRITE_I:                  in vlbit;

            signal HD:                           out vlbit_1d(7 downto 0);


--          ASIC Interface signals....

            signal HREQ:                         out vlbit;
            signal HREQ_PEND:                    in vlbit;
            signal A_D_LTCH:                     in vlbit;
            signal MA5:                          in vlbit;

            signal BUSY,STS1,STS2,HACK:          in vlbit;

            signal TS1:                          out vlbit;
            signal TSP:                          inout vlbit_1d(2 downto 0);
            signal INT_PEND:                     out vlbit;
            signal CS_MODE,CS_17,CS_18:             in vlbit;
            signal G_A_TO_B:                        out vlbit;-- Output enable to rehost


            signal MD:                              in vlbit_1d(7 downto 1);


--          Generate VME Interrupt...

            signal INTERRUPT:                       inout vlbit;


--          VME DATA Bus Justifier and buffer....

            signal DATA_DIR,N_DB_EN8:               out vlbit;
            signal N_DB_EN16:                       out vlbit;
            signal N_DB_ENSW,N_DB_EN31:             out vlbit;


--          Address comparator hit paths.... (_O's are shorted to non _O's)

            signal CSR_ASIC,CSR_HA,MEM_ASIC:in vlbit;
            signal CSR_ASIC_O:                      out vlbit;
            signal CSR_HA_O:                        out vlbit;
            signal MEM_ASIC_O:                      out vlbit;


--          These are the VME Host Adapter's Arbitration and Current State...

            signal CYCLE_DECODE:                    inout vlbit_1d(2 downto 0);
            signal VME_STATE:                       inout vlbit_1d(2 downto 0)

      );


end reh_cnt1;
```

B.1.1 Architecture Body

```
---------------------------------------------------------------------------
-- Architecture body:
---------------------------------------------------------------------------
architecture behv of reh_cnt1 is

signal cycle:                                   vlbit;
signal DS0,DS1,WRITE,IACK:                      vlbit;
signal cycle_Delay,cycle_double_delay,start:    vlbit;

signal HA_CSR_READ,INTERRUPT_ACK:               vlbit_1d(2 downto 0);
signal ASIC_WRITE,ASIC_READ:                    vlbit_1d(2 downto 0);
signal VME_BUS_IDLE,NOT_OUR_CYCLE:              vlbit_1d(2 downto 0);

signal TSP_L:                                   vlbit_1d(2 downto 0);
signal WRITE_L:vlbit;

signal TSP_IN:                                  vlbit_1d(3 downto 0);

signal vector_byte:vlbit;

signal int_vector:                              vlbit_1d(7 downto 0);
signal md_l:                                    vlbit_1d(7 downto 1);
signal err_int:                                    vlbit;

signal standard:                                   vlbit;
signal sixteen_bit:                                vlbit;

signal am_bus:                                     vlbit_1d(7 downto 0);
-- This is to make the compiler happy
-- Since I prefer to work with hex.

signal DIRECTION_WRITE,DIRECTION_READ:  vlbit;
signal DE_ASSERTED,ASSERTED,N_ASSERTED,N_DE_ASSERTED:vlbit;


signal PASS:                                       vlbit;

signal N_A_D_LTCH_SMP_CL,A_D_LTCH_SMP:vlbit;

signal state0,state1,state2,state3,state6:         vlbit_1d(2 downto 0);

signal int_level_ack:                              vlbit;

signal IACK_IN,IACK_OUT:                           vlbit;

signal n_v_rst:                                    vlbit;

begin

--    Makes this easier to read.... (Positive Logic)

DS0 <= not N_DS0;
DS1 <= not N_DS1;
WRITE <= not N_WRITE;
IACK <= not N_IACK;

IACK_IN <= not N_IACK_IN;
N_IACK_OUT <= not IACK_OUT;


--    These are the decoded output states from the arbiter:

VME_BUS_IDLE  <= B"000";
HA_CSR_READ  <= B"010";
INTERRUPT_ACK      <= B"011";
ASIC_WRITE      <= B"100";
ASIC_READ <= B"101";
NOT_OUR_CYCLE     <= B"111";

ASSERTED <= '1';      --    Included for readability
DE_ASSERTED <= '0';

N_ASSERTED <= '0'; --    The 'N_' prefix denotes an active low signal
N_DE_ASSERTED <= '1';


-- This is a synchronous reset needed to force Altera to use the sysreset
-- as the global reset.

st:   process
      begin
      wait until prising(CLK) or N_SYSRESET = '0';
      if prising(CLK) then
            n_v_rst <= '1';
      end if;
```

```
        if N_SYSRESET = '0' then
                n_v_rst <= '0';
        end if;
        end process;


-- This condition denotes the start of a VME cycle from a slave's perspective.

CYCLE <= ASSERTED when ((DS0 = ASSERTED or DS1 = ASSERTED)
                                        and n_v_rst = N_DE_ASSERTED)
   else  DE_ASSERTED;

-- We will de-skew this condition and create an arbitration pulse


dly:    process
        begin
        wait until prising(CLK) or CYCLE = '0';
        if prising(clk) then
                cycle_delay <= CYCLE;
                cycle_double_delay <= cycle_delay;
                start <= cycle_delay and (not cycle_double_delay);
        end if;

        if CYCLE = '0' then
                cycle_delay <= DE_ASSERTED;
                cycle_double_delay <= DE_ASSERTED;
                start <= DE_ASSERTED;
        end if;

        end process;



-- At the beginning of a VME cycle decide what were doing...

dec:    process
        begin
        wait until prising(CLK) or N_SYSRESET = '0';
        if prising(CLK) then

-- Synchronously....

           if CYCLE = ASSERTED then
                if start = ASSERTED then

                        if WRITE = DE_ASSERTED and CSR_HA = ASSERTED then

                                CYCLE_DECODE <= HA_CSR_READ;


                        else if IACK = ASSERTED then

                                CYCLE_DECODE <= INTERRUPT_ACK;


                        else if WRITE = ASSERTED and (CSR_ASIC = ASSERTED or
                                MEM_ASIC = ASSERTED or CSR_HA = ASSERTED) then

                                CYCLE_DECODE <= ASIC_WRITE;
```

```
                  else if WRITE = DE_ASSERTED and (CSR_ASIC = ASSERTED or
                        MEM_ASIC = ASSERTED) then

                        CYCLE_DECODE <= ASIC_READ;

                  else

                        CYCLE_DECODE <= NOT_OUR_CYCLE;

                  end if;
              end if;
            else

              CYCLE_DECODE <= VME_BUS_IDLE;

            end if;

        end if;

        if N_SYSRESET = '0' then

              CYCLE_DECODE <= VME_BUS_IDLE;

        end if;

        end process;



-- These are to make reading the state machine easier...

DIRECTION_WRITE <= '1';
DIRECTION_READ <= '0';
state0 <= B"000";
state1 <= B"001";
state2 <= B"010";
state3 <= B"011";
state6 <= B"110";



-- Interrupt level 5 is hard coded for this design

int_level_ack <=   ASSERTED when (A = B"101") else
                  DE_ASSERTED;


-- This is the main VME Host Adapter state Machine

-- It idles at state0 and waits for a vaild CYCLE_DECODE to begin processing..
-- The path through the state machine is defined by the current state as
-- well as the CYCLE_DECODE value.  For more information on the state
-- machine's program see the file state.pgm file.

st:     process
        begin
        wait until prising(CLK) or N_SYSRESET = '0' or n_v_rst = '0';
        if prising(CLK) then

-- Synchronously....

            if cycle=ASSERTED then
              case v1d2int(VME_STATE) is
```

```
                when 0   =>
                    if CYCLE_DECODE /= VME_BUS_IDLE then

                        if CYCLE_DECODE = ASIC_WRITE then

                            DATA_DIR <= DIRECTION_WRITE;

                        else

                            DATA_DIR <= DIRECTION_READ;

                        end if;


                        if CYCLE_DECODE = ASIC_WRITE or CYCLE_DECODE =
                          ASIC_READ or CYCLE_DECODE = HA_CSR_READ then

                            VME_STATE <= state1;

                        end if;


                        if CYCLE_DECODE = INTERRUPT_ACK and int_level_ack
                          = ASSERTED and INTERRUPT = ASSERTED and PASS
                          = DE_ASSERTED then

                            VME_STATE <= state1;

                        else

                            PASS <= ASSERTED;

                        end if;
                    end if;

                when 1   =>
                    if CYCLE_DECODE = INTERRUPT_ACK and IACK_IN =
                      ASSERTED then

                        INT_PEND <= ASSERTED;
                        N_OE_VEC <= N_ASSERTED;
                        N_DB_EN8 <= N_ASSERTED;
                        VME_STATE <= state6;

                    end if;


                    if CYCLE_DECODE = HA_CSR_READ then

                        N_OE_VEC <= N_ASSERTED;
                        N_DB_EN8 <= N_ASSERTED;
                        VME_STATE <= state6;

                    end if;


                    if CYCLE_DECODE = ASIC_READ or CYCLE_DECODE =
                      ASIC_WRITE then

                        case v1d2int(TSP) is
                            when 0   =>
                    N_DB_EN8 <= N_ASSERTED;
                    N_DB_EN16 <= N_ASSERTED;
                    N_DB_EN31 <= N_ASSERTED;

                when 1   =>

                    N_DB_EN8 <= N_ASSERTED;
                    N_DB_EN16 <= N_ASSERTED;

                when 2   =>

                    N_DB_ENSW <= N_ASSERTED;

                when 4   =>

                    N_DB_ENSW <= N_ASSERTED;

                when 5   =>

                    N_DB_ENSW <= N_ASSERTED;

                when 6   =>

                    N_DB_EN8 <= N_ASSERTED;
                    N_DB_EN16 <= N_ASSERTED;

                when 7   =>

                    N_DB_EN8 <= N_ASSERTED;
                    N_DB_EN16 <= N_ASSERTED;

                when others  =>

                    N_DB_EN8 <= N_DE_ASSERTED;
                    N_DB_EN16 <= N_DE_ASSERTED;
                    N_DB_ENSW <= N_DE_ASSERTED;
                    N_DB_EN31 <= N_DE_ASSERTED;
            end case;


            if HREQ_PEND = DE_ASSERTED then

                HREQ <= ASSERTED;
                N_A_D_LTCH_SMP_CL <= N_DE_ASSERTED;
                VME_STATE <= state2;

            end if;
        end if;

    when 2   =>
        if HREQ_PEND = ASSERTED then

            HREQ <= DE_ASSERTED;
            VME_STATE <= state3;

        end if;

    when 3   =>
        if CYCLE_DECODE = ASIC_READ then
            if A_D_LTCH_SMP = ASSERTED and HACK =
              ASSERTED then

                G_A_TO_B <= ASSERTED;
```

```
                              VME_STATE <= state6;

                        end if;
                  end if;

                  if CYCLE_DECODE = ASIC_WRITE then
                        if A_D_LTCH_SMP = ASSERTED then

                              VME_STATE <= state6;

                        end if;
                  end if;

            when 6   =>
                  DTACK <= ASSERTED;

            when others  =>

                  VME_STATE <= state0;

      end case;
    else

-- This means the VME cycle is over....
      VME_STATE <= state0;
      DATA_DIR <= DIRECTION_WRITE;
      PASS <= DE_ASSERTED;
      INT_PEND <= DE_ASSERTED;
      G_A_TO_B <= DE_ASSERTED;

      N_DB_EN8 <= N_DE_ASSERTED;
      N_DB_EN16 <= N_DE_ASSERTED;
      N_DB_ENSW <= N_DE_ASSERTED;
      N_DB_EN31 <= N_DE_ASSERTED;

      DTACK <= DE_ASSERTED;
      HREQ <= DE_ASSERTED;
      N_A_D_LTCH_SMP_CL <= N_ASSERTED;
      N_OE_VEC <= N_DE_ASSERTED;

    end if;
  end if;

  if n_v_rst= '0' then

      DATA_DIR <= DIRECTION_WRITE;

      N_DB_EN8 <= N_DE_ASSERTED;
      N_DB_EN16 <= N_DE_ASSERTED;
      N_DB_ENSW <= N_DE_ASSERTED;
      N_DB_EN31 <= N_DE_ASSERTED;

      N_OE_VEC <= N_DE_ASSERTED;

  end if;

  if N_SYSRESET= '0' then

      VME_STATE <= state0;
      PASS <= DE_ASSERTED;
      INT_PEND <= DE_ASSERTED;
      G_A_TO_B <= DE_ASSERTED;
```

```
            DTACK <= DE_ASSERTED;
            HREQ <= DE_ASSERTED;
            N_A_D_LTCH_SMP_CL <= N_ASSERTED;

      end if;

      end process;

IACK_OUT <= PASS and IACK_IN;
```

B.1.2 Rising Edge Detector for A_D_LTCH

```
-- This process is a rising edge detector for the A_D_LTCH signal.
-- It is held in clear until the state machine is ready to look for
-- a rising edge of A_D_LTCH.  This is done so that A_D_LTCH = 1 from
-- a earlier pipelined cycle does not confuse the state machine.

a_d:process
      begin
      wait until prising(A_D_LTCH) or N_A_D_LTCH_SMP_CL = '0';
      if prising(A_D_LTCH) then
            A_D_LTCH_SMP <= ASSERTED;
      end if;

      if N_A_D_LTCH_SMP_CL = '0' then
            A_D_LTCH_SMP <= DE_ASSERTED;
      end if;

      end process;
```

B.1.3 Generate TS1 and TS2

```
-- Generate TS1 and TS2
-- TS1 and TS2 are the Transaction type signals to the ASIC.  The types
-- supported by this interface are:
--    TS2       TS1
--     ----       ----
--      0          0   Memory
--      0          1   CSR(s)

-- Note that TS2 = 1 denotes NMT type transactions which are beyond the
-- scope of this simple VME interface.


-- TS2 <= DE_ASSERTED;
-- no NMT's in this interface.....
-- This was removed to free I/O's

TS1 <=   ASSERTED when (CSR_HA = ASSERTED or CSR_ASIC = ASSERTED) else
      DE_ASSERTED;
```

B.1.4 Generate TSP Codes

```
-- Generate TSP codes....
-- The TSP codes are inputs to the ASIC which denote the size and position
-- of the VME transaction.  In this VHDL we build a table of lookup values
-- for the input variables vs the corresponding TSP code.  The PLA_TABLE
-- procedure maps the inputs to the output via the defined table.

-- Input must be an array.
```

```
      TSP_IN <= N_DS1 & N_DS0 & A(1) & N_LWORD;

stat: process(TSP_IN)

      constant tsp_tbl:                         vlbit_2d(0 to 6,6 downto 0) :=
            (
-- From the VME Spec - note that no illegal codes are trapped.
--          DDAL_TTT
--          SS0W_SSS
--          101R_PPP
--          ** D_210
--
            B"XXX0_000",
            B"0011_001",
            B"0001_010",
            B"0101_100",
            B"1001_101",
            B"0111_110",
            B"1011_111"
            );

      begin

      pla_table(TSP_IN,TSP,tsp_tbl);

      end process;
```

B.1.5 Latch TSP Codes and Write Value

```
-- The following latches the TSP codes and the WRITE value.  This is used only
-- in the decoding of the HA provided CSR register write pulse.  When using
-- MA(5) and HACK the user needs to know the byte codes and write value for
-- the current in process transaction.  Note that the TSP codes could reflect
-- a new value for a now pending ASIC transaction.  Pipelining makes this a
-- requirement.

tspl: process
      begin
      wait until prising(A_D_LTCH) or N_SYSRESET = '0';
      if prising(A_D_LTCH) then
            TSP_L <= TSP;
            WRITE_L <= WRITE;
      end if;

      if N_SYSRESET= '0' then
            TSP_L <= B"000";
            WRITE_L <= DE_ASSERTED;
      end if;

      end process;
```

B.1.6 CSR Interrupt Vector Register Write Pulse

```
-- Finally, generate the CSR interrupt vector register write pulse.

vector_byte <=    ASSERTED when (TSP_L = B"000" or TSP_L =
                        B"001" or TSP_L = B"111") else
                  DE_ASSERTED;

VEC_WRITE_O <= HACK and MA5 and (     busy or
                  (WRITE_L and vector_byte and (not STS2)
                   and STS1));
```

B.1.7 Latch Interrupt Vector

```
-- Latch the interrupt vector when written to.  Note that the interrupt
-- vector which is latched is only 7 bits.

ff:   process
      begin

      wait until prising(VEC_WRITE_I);

      md_l <= MD;

      end process;

-- This makes a 8 bit vector by adding the interrupt is a error status bit.

int_vector <= md_l & err_int;

HD <=   int_vector when N_VEC_OE = N_ASSERTED else
        B"ZZZZZZZZ";
```

B.1.8 Watch ASIC Interrupt Lines

```
-- This process watches the ASIC interrupt lines.  If either go active,
-- this process sets the INTERRUPT signal as well as setting the err_int
-- signal to the current value of the interrupt on error line.  This will
-- stay latched until the ASIC releases the interrupt(s) lines.  Note that
-- the CS_MODE bit tells when the control_status bit are valid.

i:    process
      begin

      wait until prising(CLK) or N_SYSRESET = '0';

      if prising(CLK) then

            if INTERRUPT = DE_ASSERTED then
                  if CS_MODE = ASSERTED and (CS_17 = ASSERTED or CS_18 =
                     ASSERTED) then

                        INTERRUPT <= ASSERTED;
                        err_int <= CS_17;

                  end if;
            else
                  if CS_MODE = ASSERTED and CS_17 = DE_ASSERTED and CS_18 =
                     DE_ASSERTED then
                        INTERRUPT <= DE_ASSERTED;

                  end if;
            end if;

      end if;

      if N_SYSRESET = '0' then

            INTERRUPT <= DE_ASSERTED;
            err_int <= DE_ASSERTED;

      end if;

      end process;
```

B.1.9 VME Address Decoding

```
-- This is ultra-simple address decoding for VME.  Note we could have used
-- the ASIC's internal address decoder which would have been much more
-- flexible.


am_bus <= '0' & '0' & AM;


-- These are from the VME bus spec....

standard <=         ASSERTED when (am_bus = X"3d" or am_bus = X"39") else
                    DE_ASSERTED;

sixteen_bit <=      ASSERTED when (am_bus = X"2d" or am_bus = X"29") else
                    DE_ASSERTED;


-- ASIC's CSR's on the sixteen bit bus at address 0x0000

CSR_ASIC_O <= ASSERTED when (sixteen_bit = ASSERTED and
                 VME_A(15 downto 5) = B"00000000000") else
                    DE_ASSERTED;


-- HA's CSR on the sixteen bit bus at address 0x0020

CSR_HA_O <=    ASSERTED when (sixteen_bit = ASSERTED and
                 VME_A(15 downto 5) = B"00000000001") else
                    DE_ASSERTED;


--    Memory is on the standard (24 bit) bus at address 0x800000

MEM_ASIC_O <=  ASSERTED when (standard = ASSERTED and VME_A23 =
                        ASSERTED) else
                     DE_ASSERTED;


end behv;
```

APPENDIX C

SPECIFICATIONS

TABLE OF CONTENTS

C.1 Hardware Specifications

Physical Dimensions:	3.3″ x 4.0″
Weight:	0.11 lbs
Electrical Requirements:	+5 V @ 0.7 Amps
Temperature Range:	
Storage:	-40° to +70°C
Operation:	0° to +40°C
Humidity Range:	
Storage:	0% to 95% (noncondensing)
Operation:	10% to 90% (noncondensing)
Network Line Transmission Rate:	150 million bits/second
Message Length:	
Fixed Length:	82 Bits
Variable Length:	256 or 1024 Data Bytes Maximum
Maximum Nodes on Network Ring:	256
Maximum Node Separation:	
Coax:	30 meters
Standard Fiber:	300 meters
Long Link Fiber:	3500 meters
Shared Memory:	
Standard Size:	0 KB
Optional Sizes:	512 KB, 1 MB and 2 MB on-card 4 MB and 8 MB off-card
Effective Per-Node Bandwidth:	
4 bytes/packet:	6.5 MB/sec
256 bytes/packet:	16.2 MB/sec
1024 bytes/packet:	16.7 MB/sec
Node Latency:	
4 bytes/packet:	250 ns - 800 ns
256 bytes/packet:	250 ns – 16.0 µs
1024 bytes/packet:	250 ns - 61.8 µs

C.2 Part Number

The rehostable Adapter board part number is in the form:

H-AS-DRHSTxxx-00

where:

CODE	DEFINITION
H	Hardware
AS	Top Level Assembly
D	Standard SCRAMNet SC150
RHST	Rehostable Adapter
xxx	**Memory (bytes)**
	512 = 512 K
	01B = 1 M
	02M = 2 M

C.3 Rehostable Adapter Dimensions

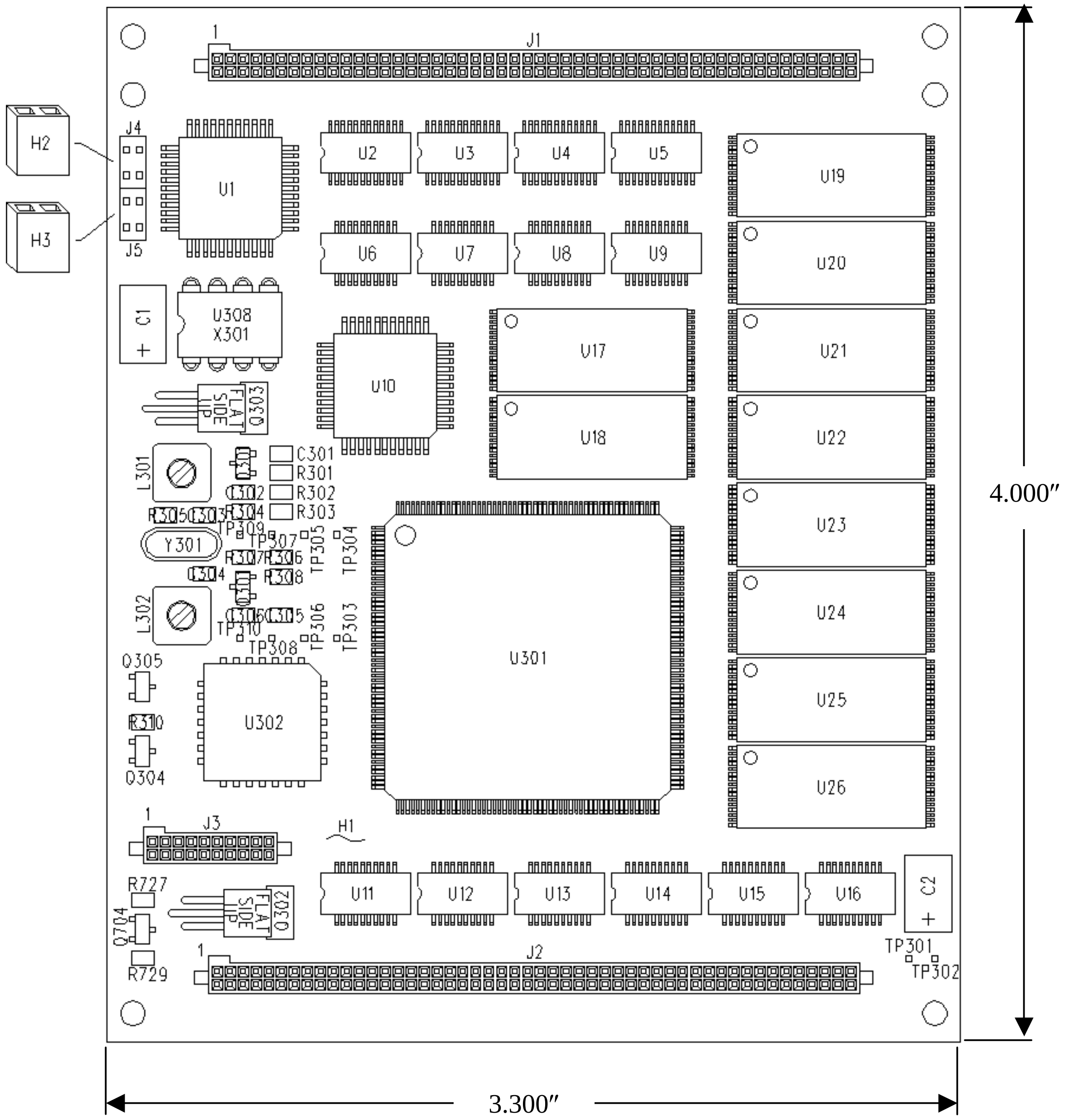


Figure C-1 Rehostable Adapter Dimensions

This page intentionally left blank

APPENDIX D

SCHEMATICS

TABLE OF CONTENTS

D.1 Overview

This Appendix contains the complete schematic set for the Rehostable VME Interface. The VME controller in the schematic set has no visible pin numbers. Therefore a pinout of the VME Controller Interface pinout is included as Figure D-25.

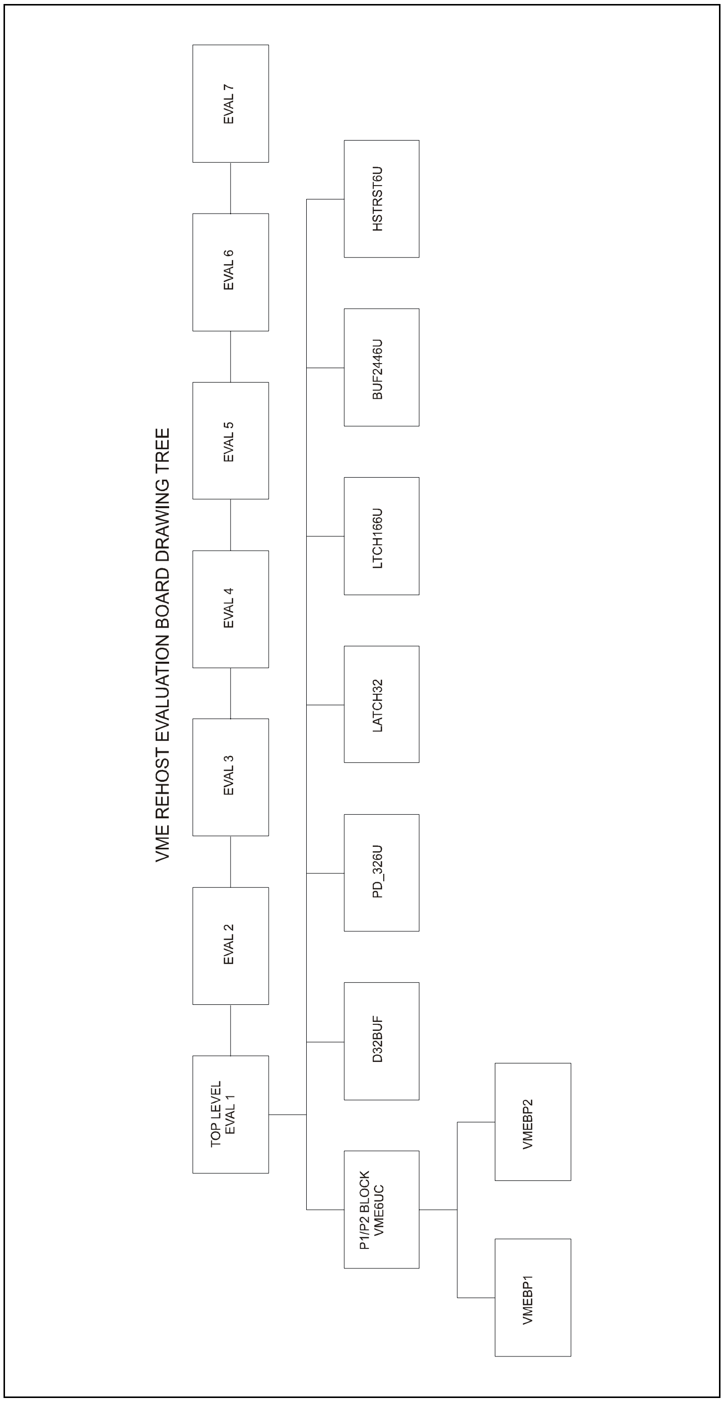


Figure D-1 Rehostable Evaluation Board Drawing Tree

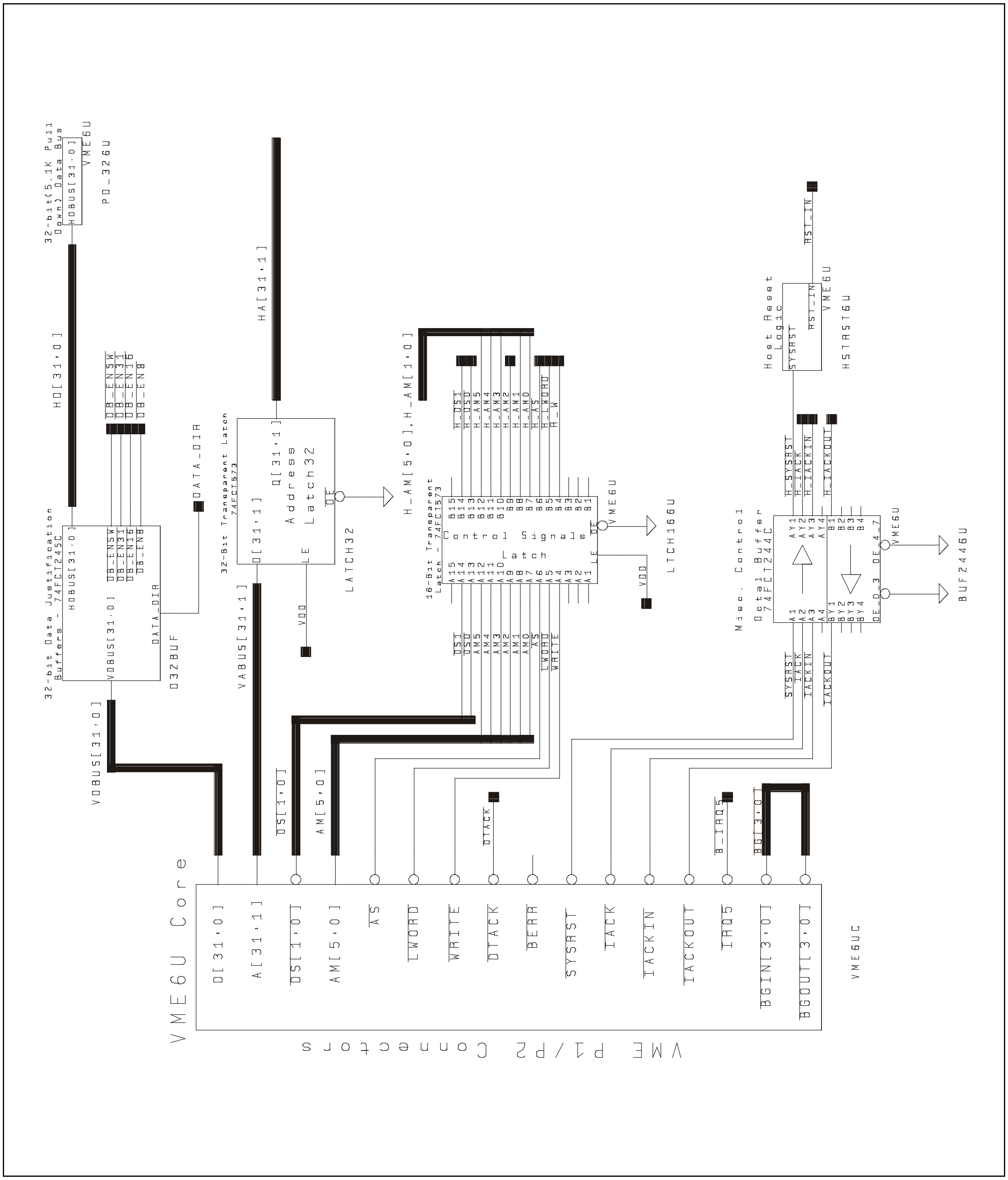


Figure D-2 Top Level Block Diagram (Eval 1)

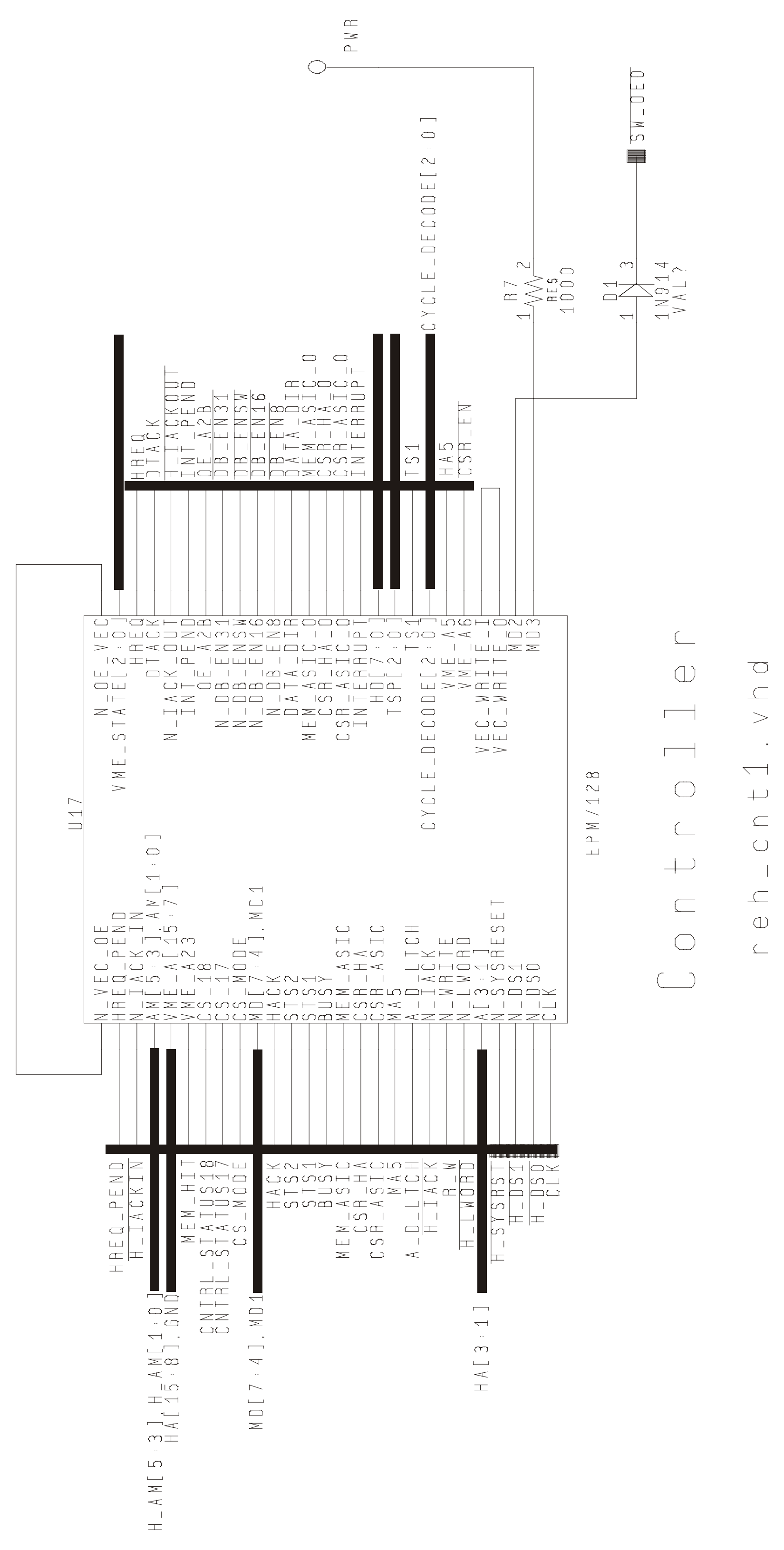
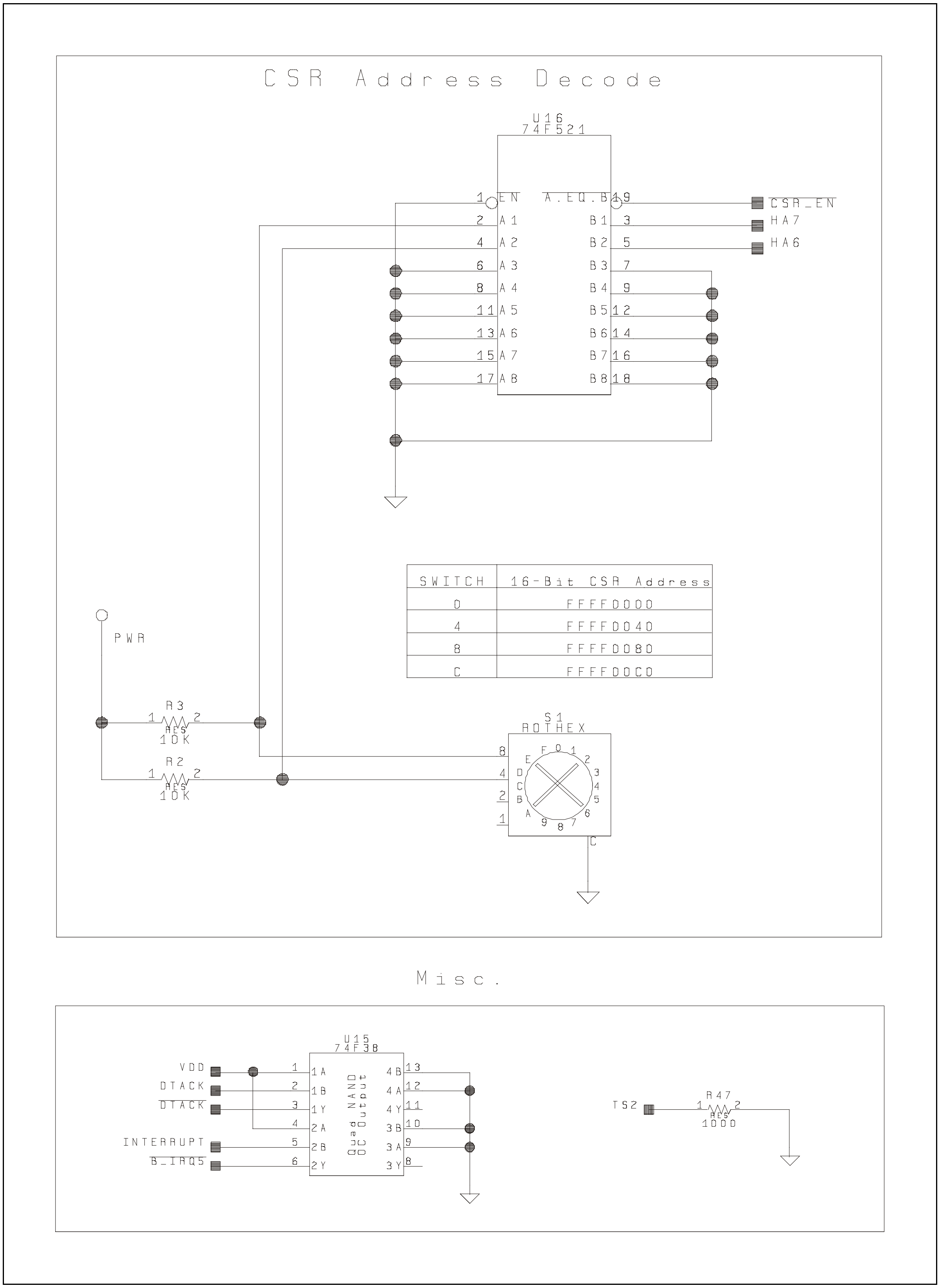


Figure D-3 VME Controller (Eval 2)



SWITCH	16-Bit CSR Address
0	FFFFD000
4	FFFFD040
8	FFFFD080
C	FFFFD0C0

Figure D-4 Address Decode (Eval 2)

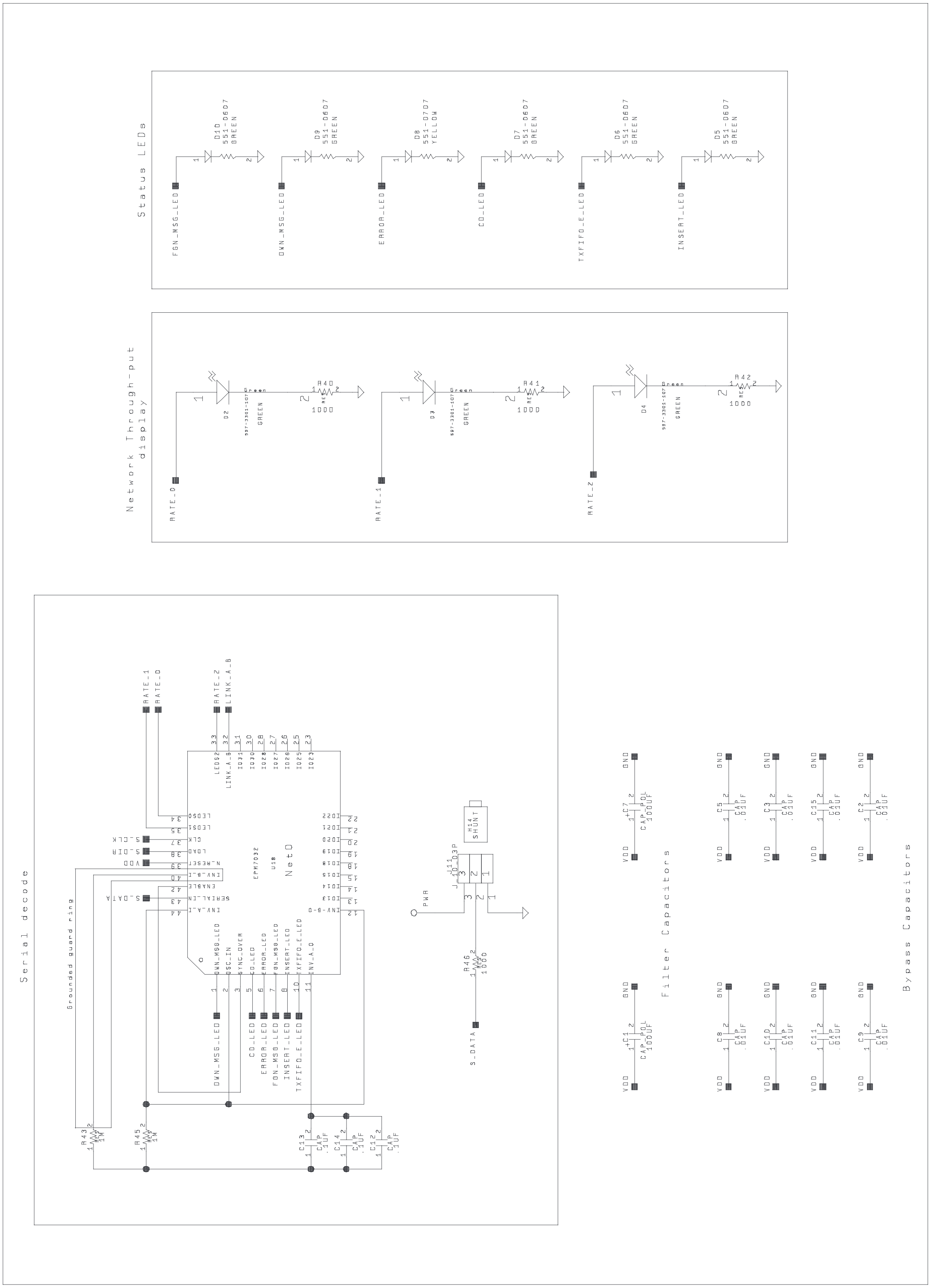


Figure D-5 LED Serial Decode (Eval 3)

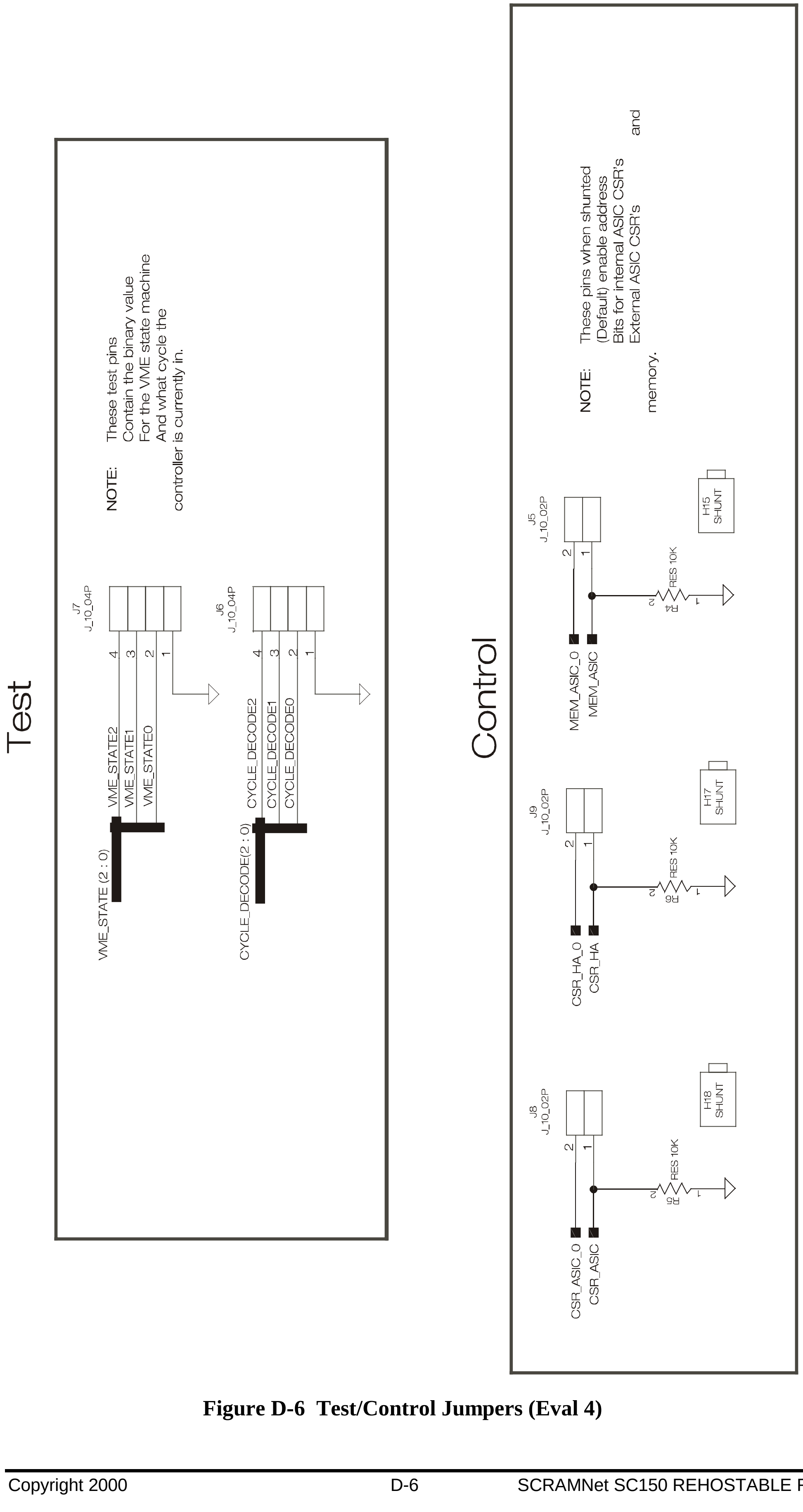


Figure D-6 Test/Control Jumpers (Eval 4)

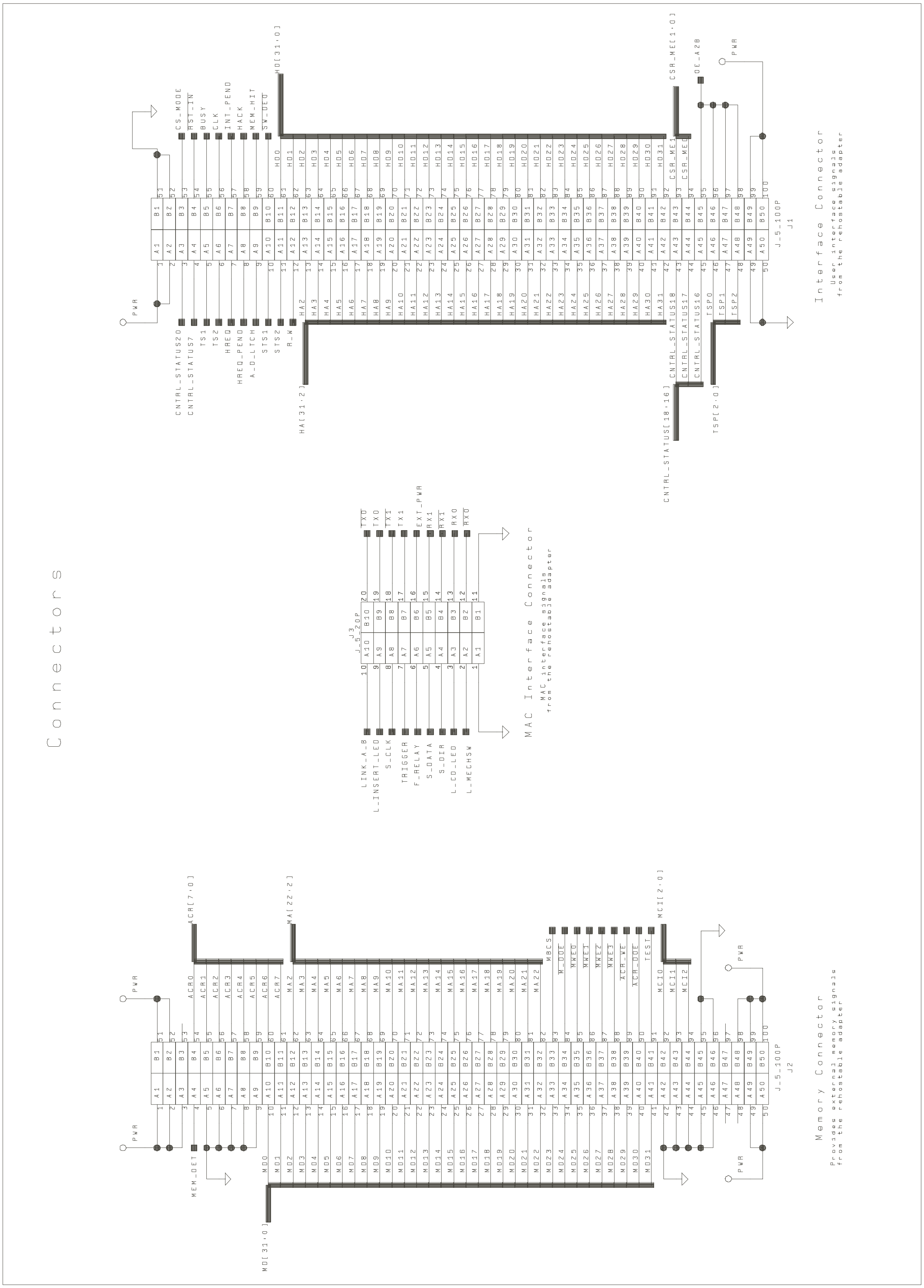


Figure D-7 Rehostable Adapter Mating Connectors (Eval 5)

)

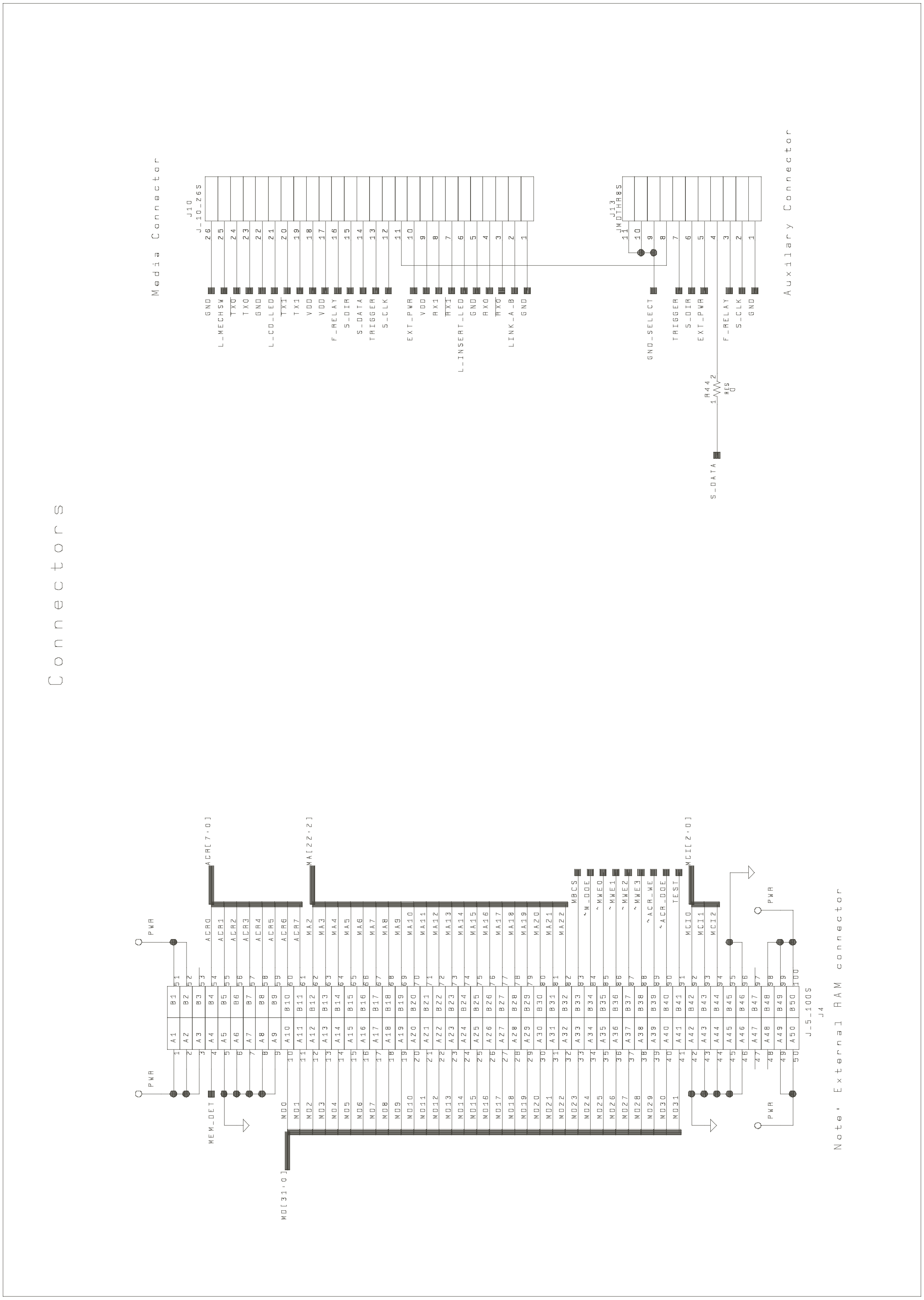


Figure D-8 External RAM/Media Connectors (Eval 6)

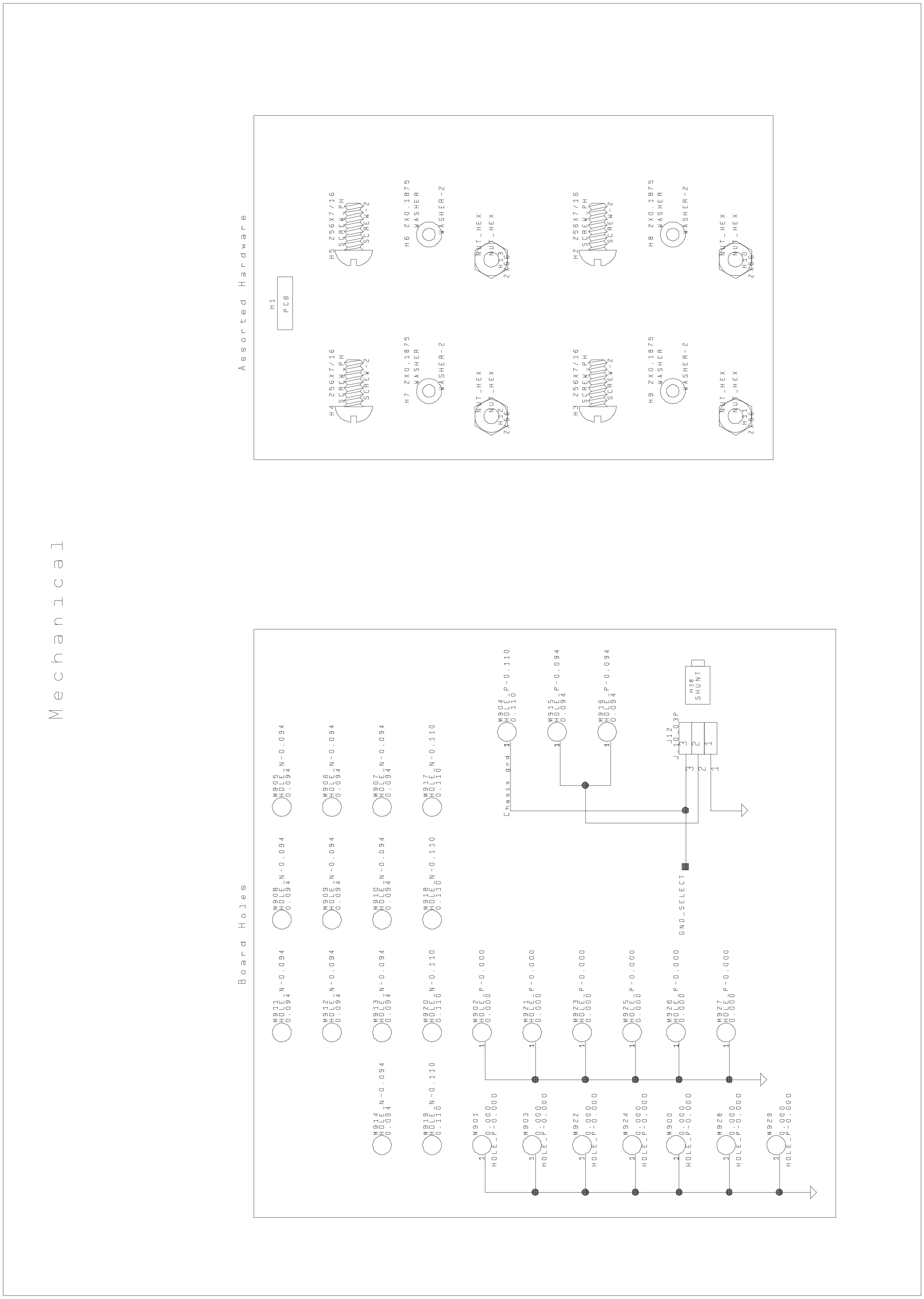


Figure D-9 Mechanical (Eval 7)

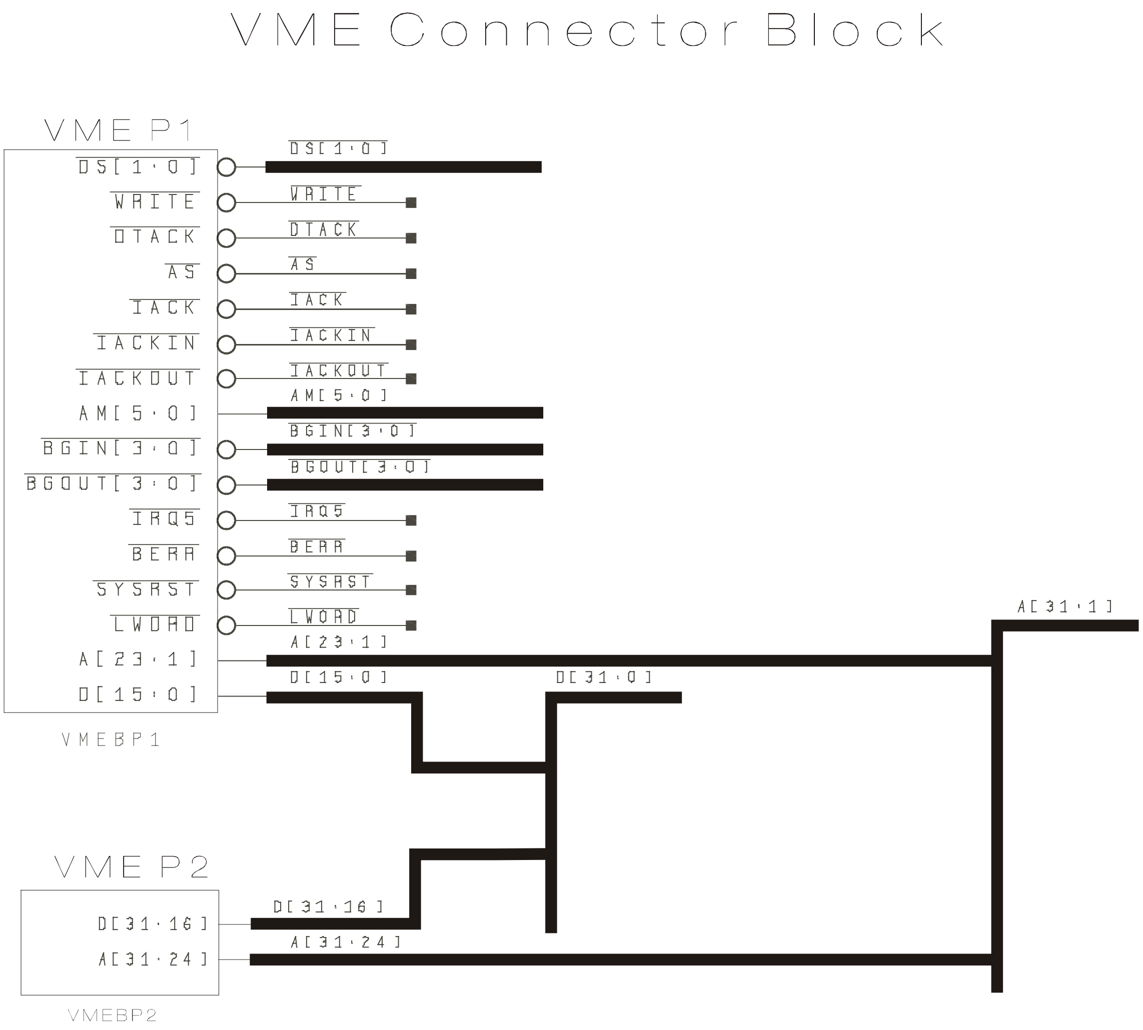
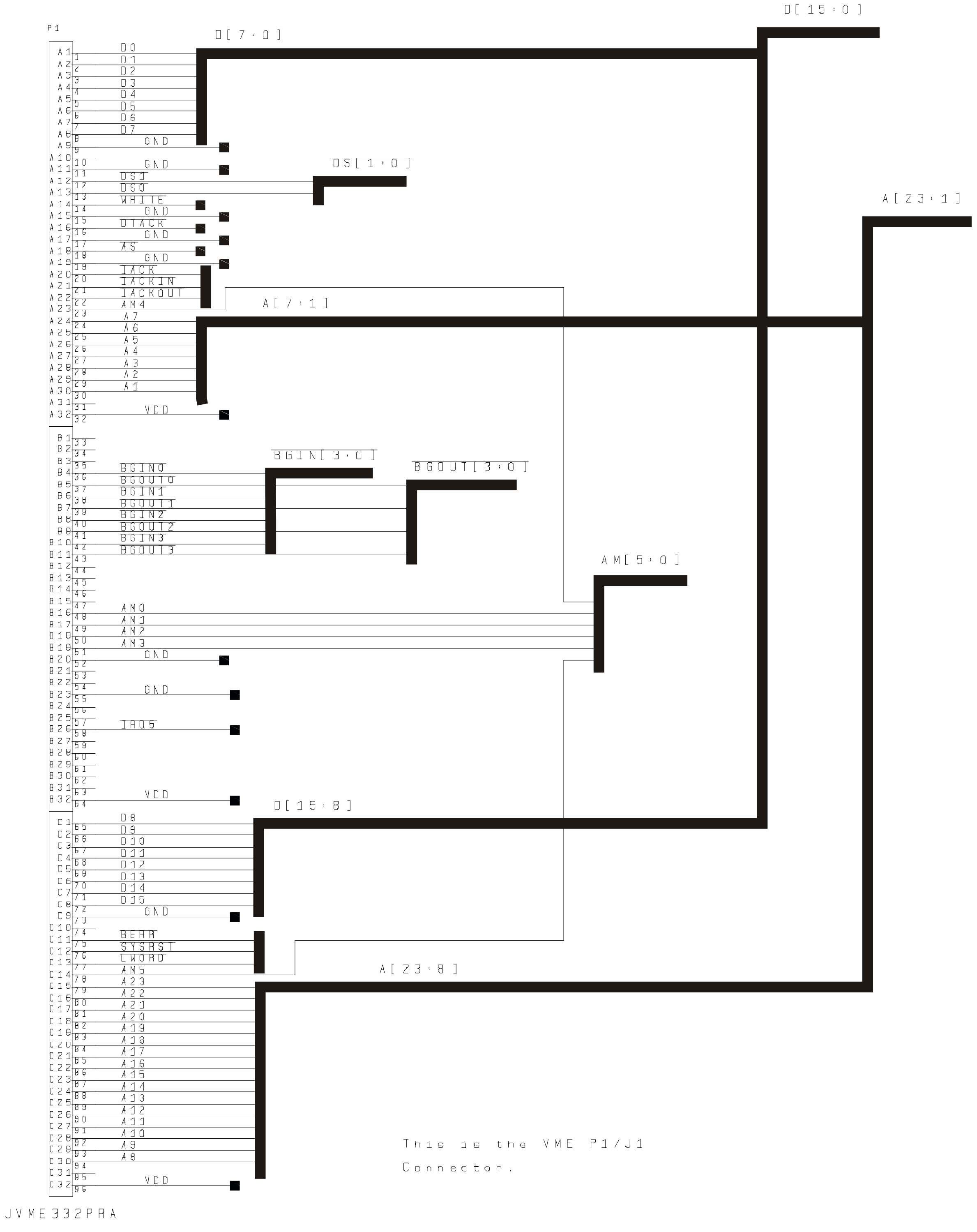


Figure D-10 VME P1/P2 Connector Block Diagram (VME6UC)



This is the VME P1/J1 Connector.

Figure D-11 VME P1 Connector (VMEBP1)

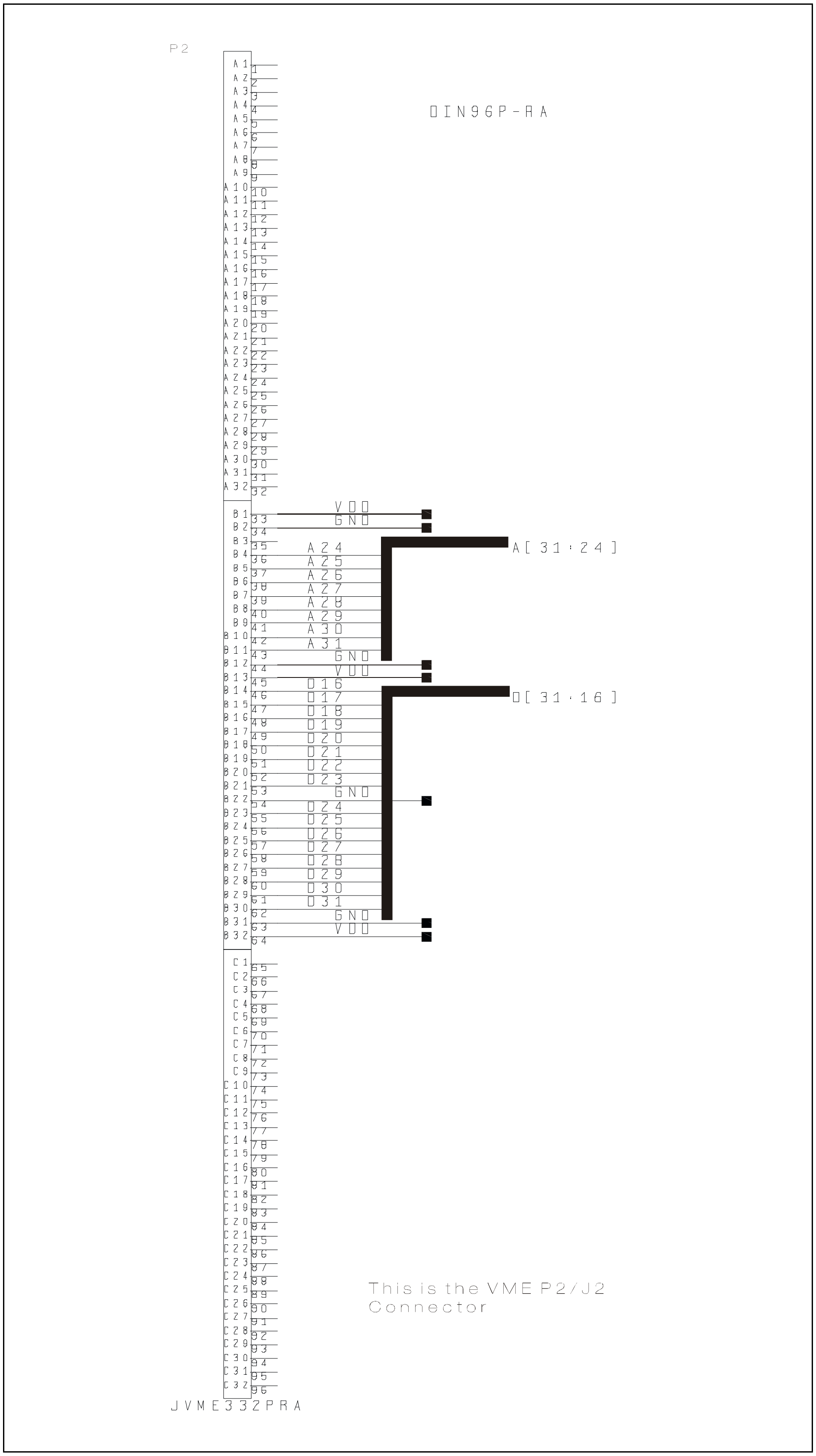


Figure D-12 VME P2 Connector (VMEBP2)

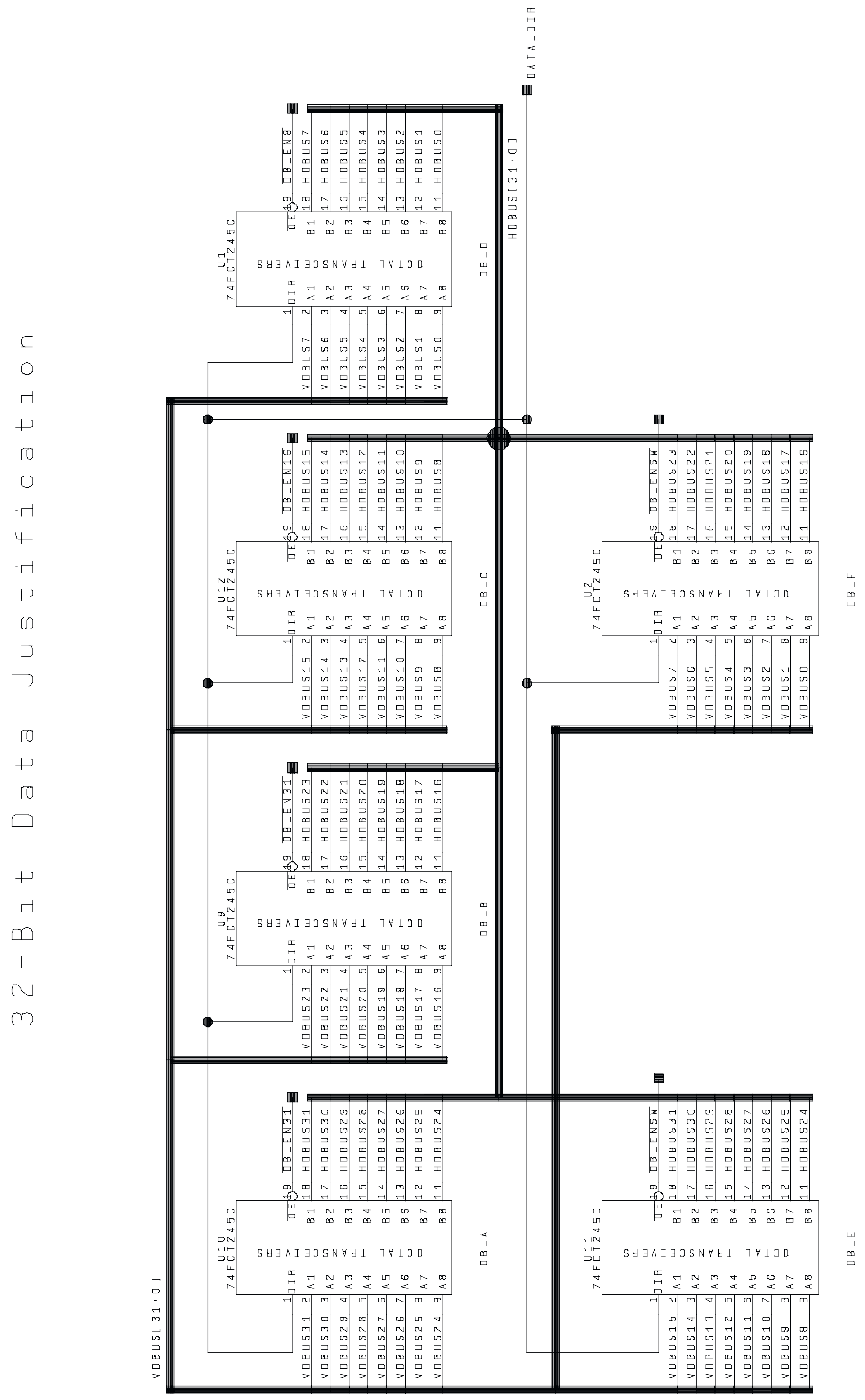


Figure D-13 32-Bit Data Justifier (D32BUF)

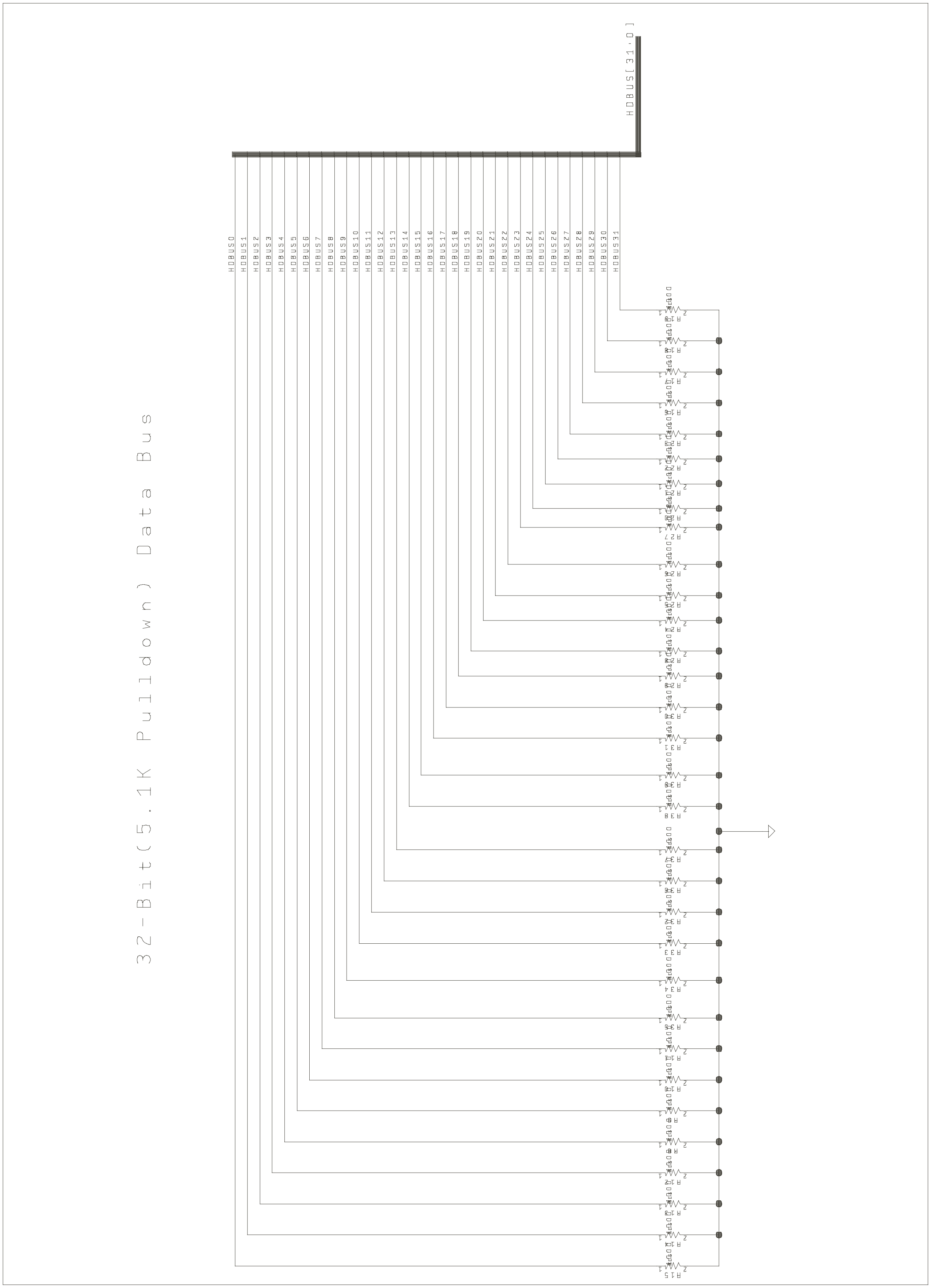


Figure D-14 Data Bus Pull-Down Resistor (PD_326U)

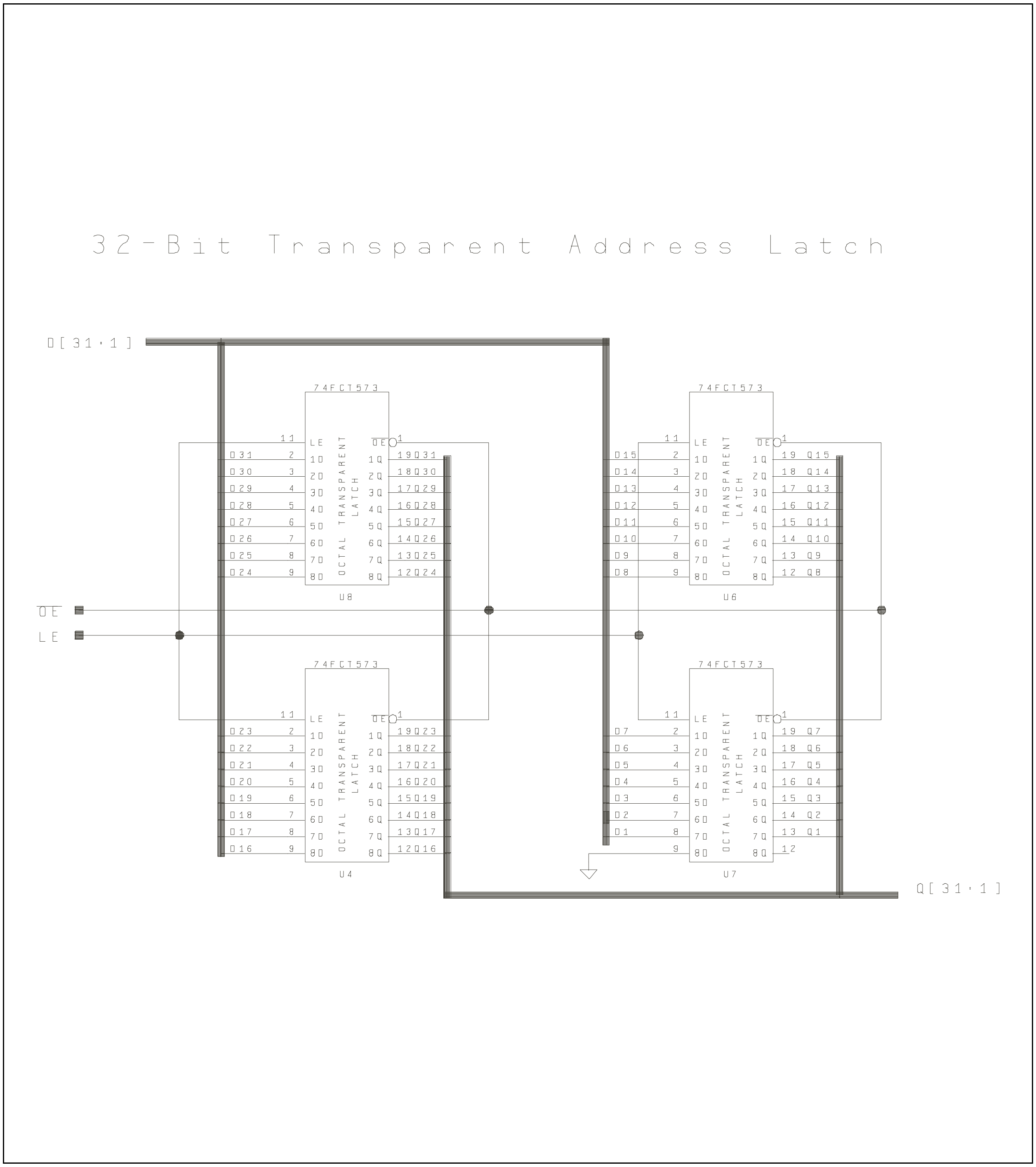
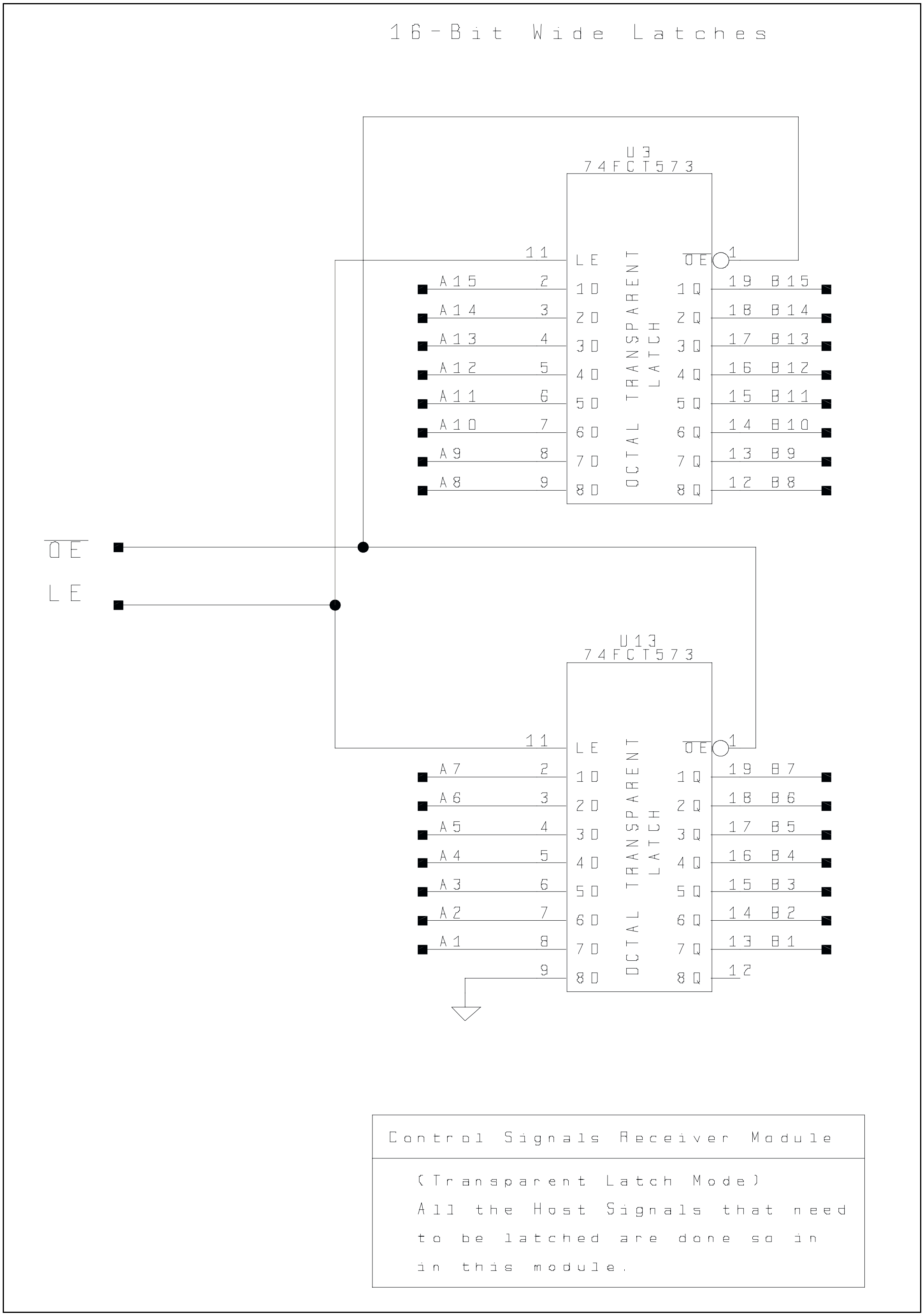


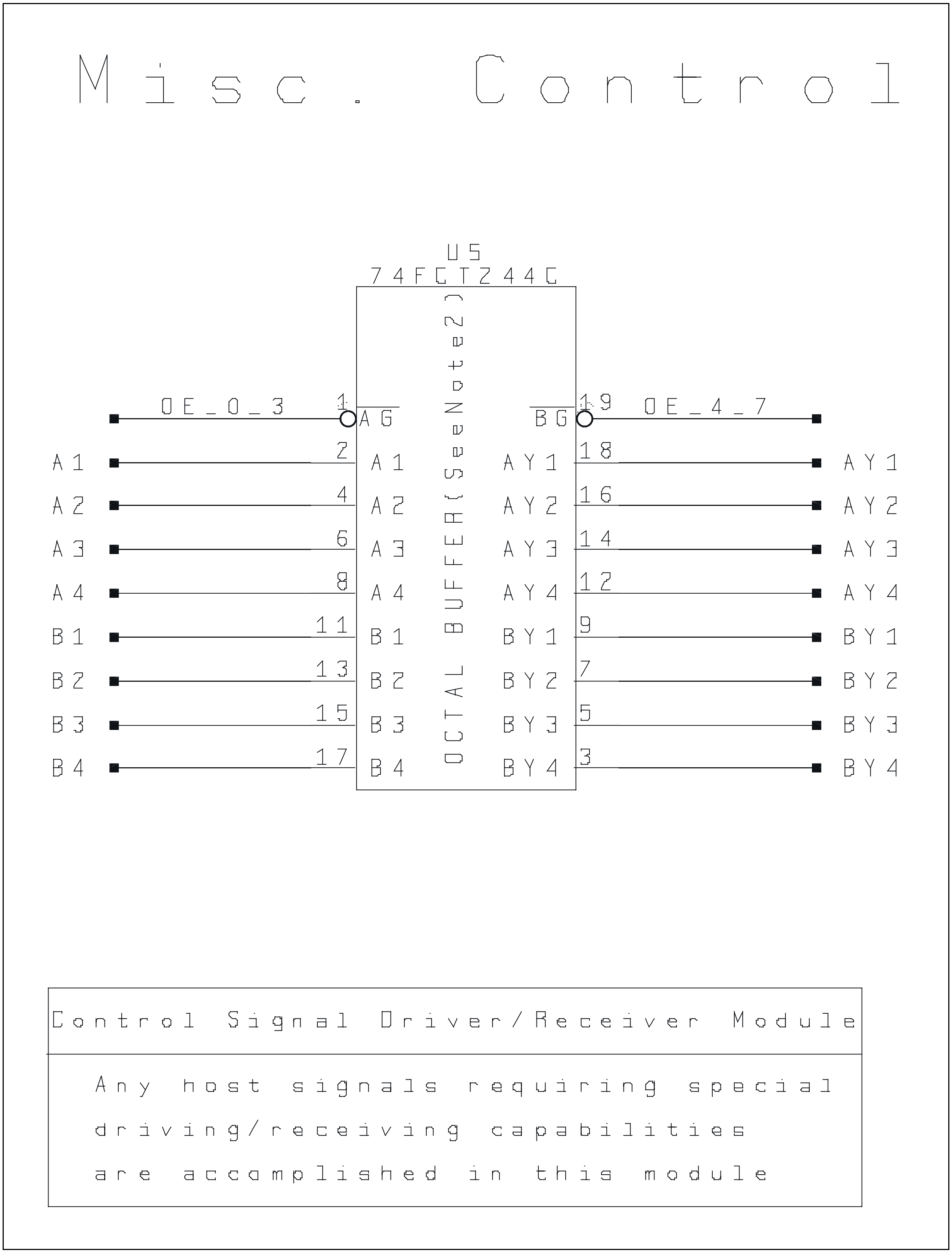
Figure D-15 32-Bit Address Latch (LATCH32)



Control Signals Receiver Module

(Transparent Latch Mode)
All the Host Signals that need
to be latched are done so in
in this module.

Figure D-16 16-Bit Signal Latch (LTCH166U)



Control Signal Driver/Receiver Module

Any host signals requiring special driving/receiving capabilities are accomplished in this module

Figure D-17 Control Buffer (BUF2446U)

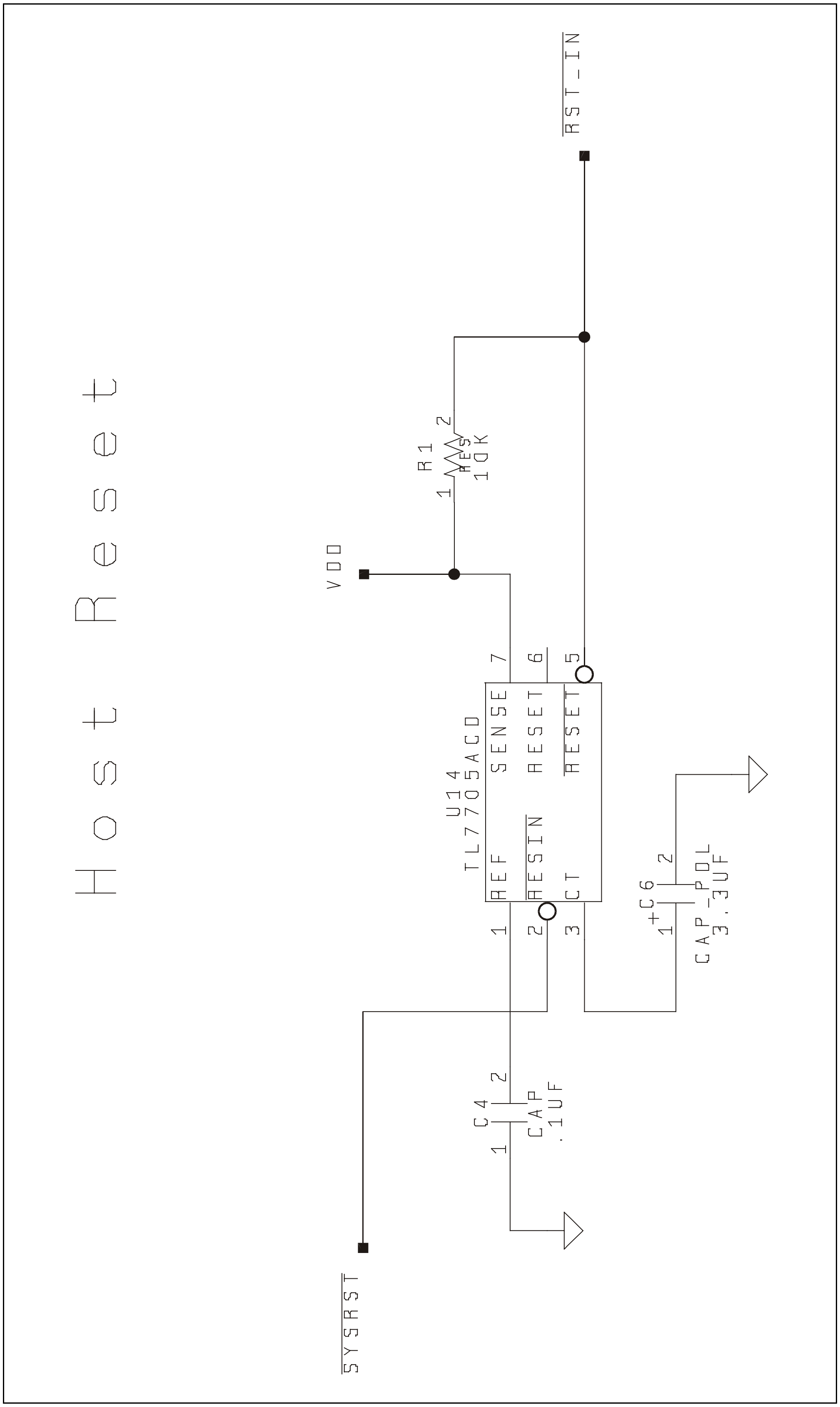


Figure D-18 Host ReReset (HSTRST6U)

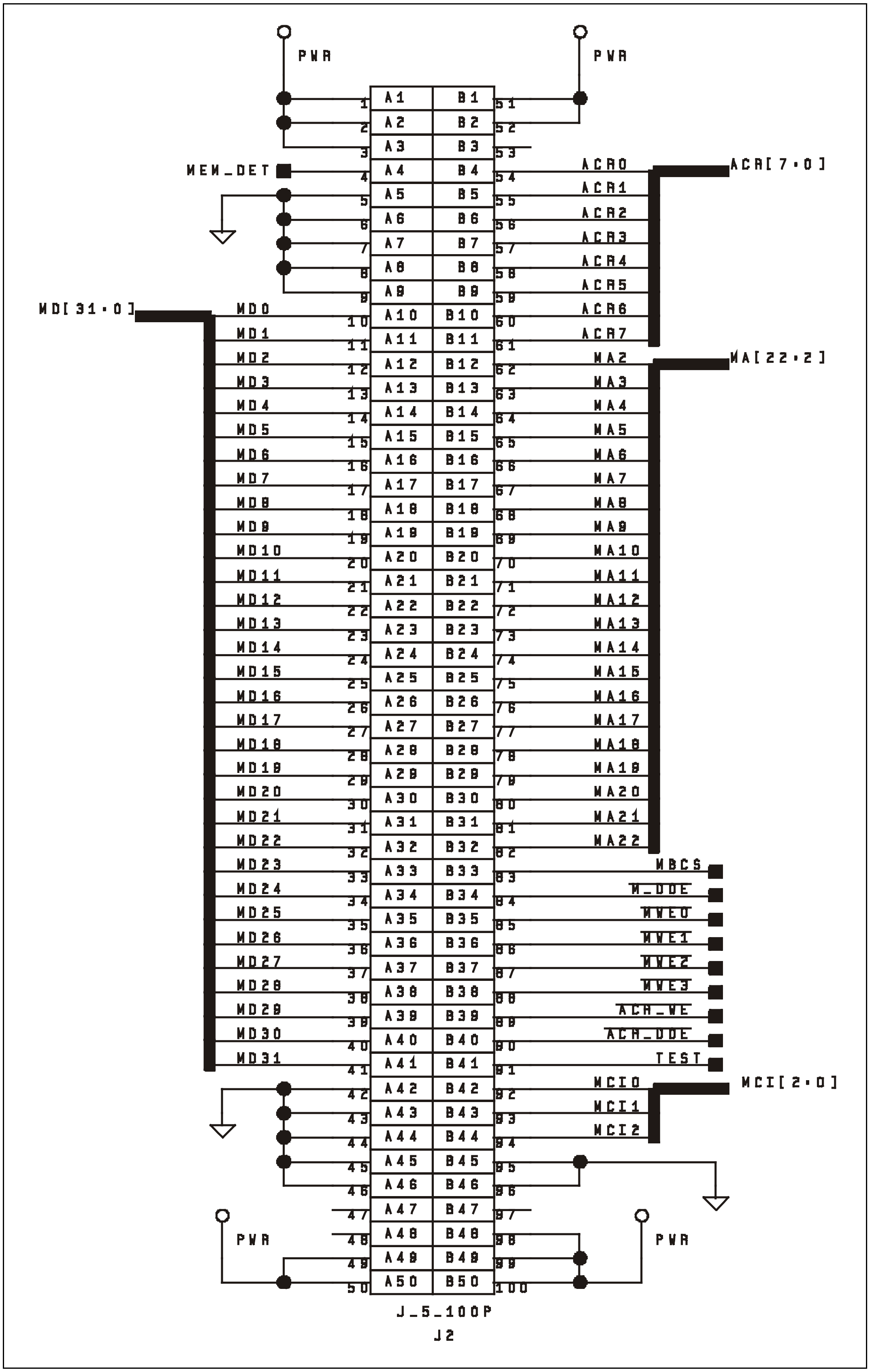


Figure D-19 Rehostable Adapter Memory Connector

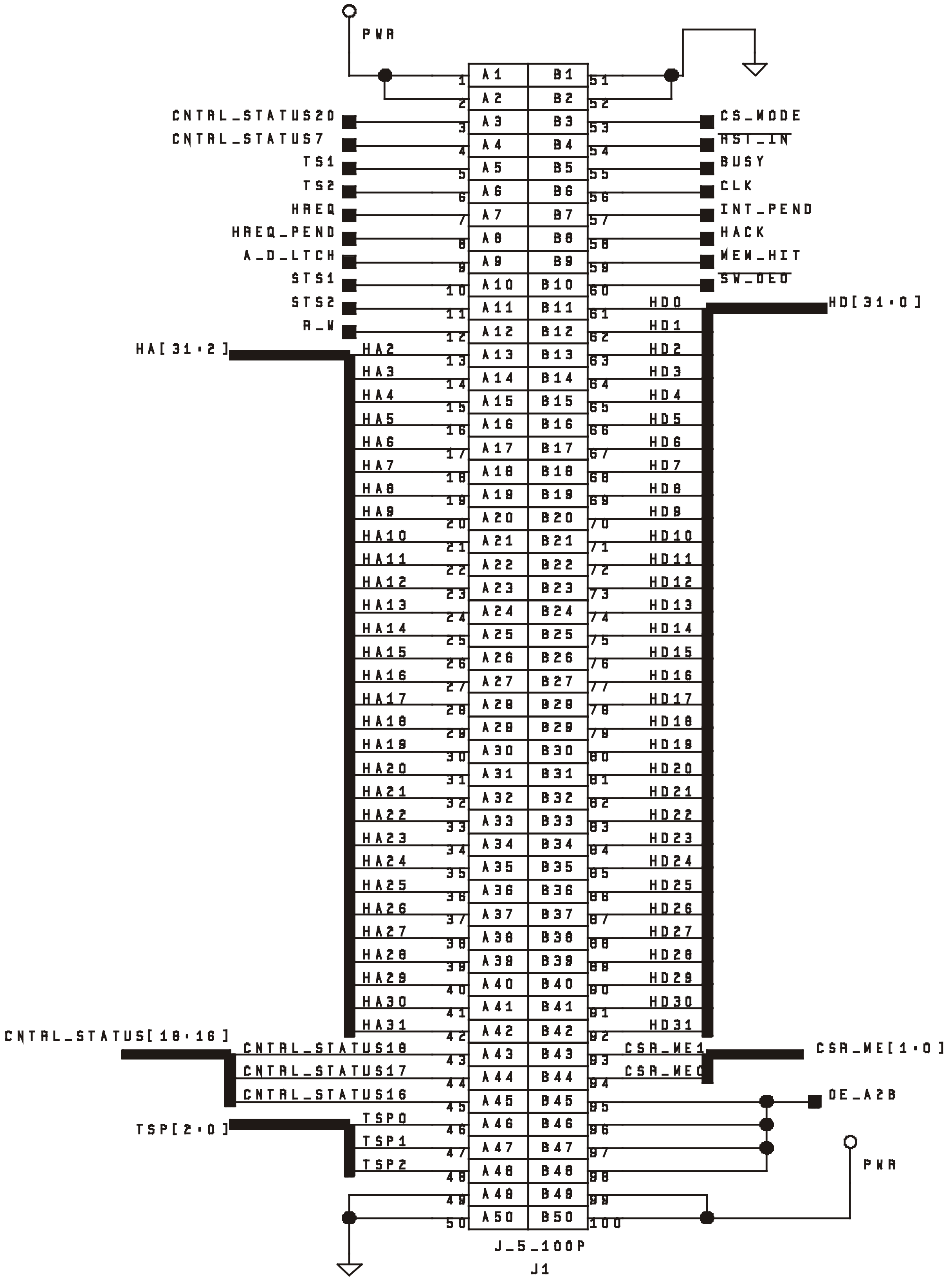
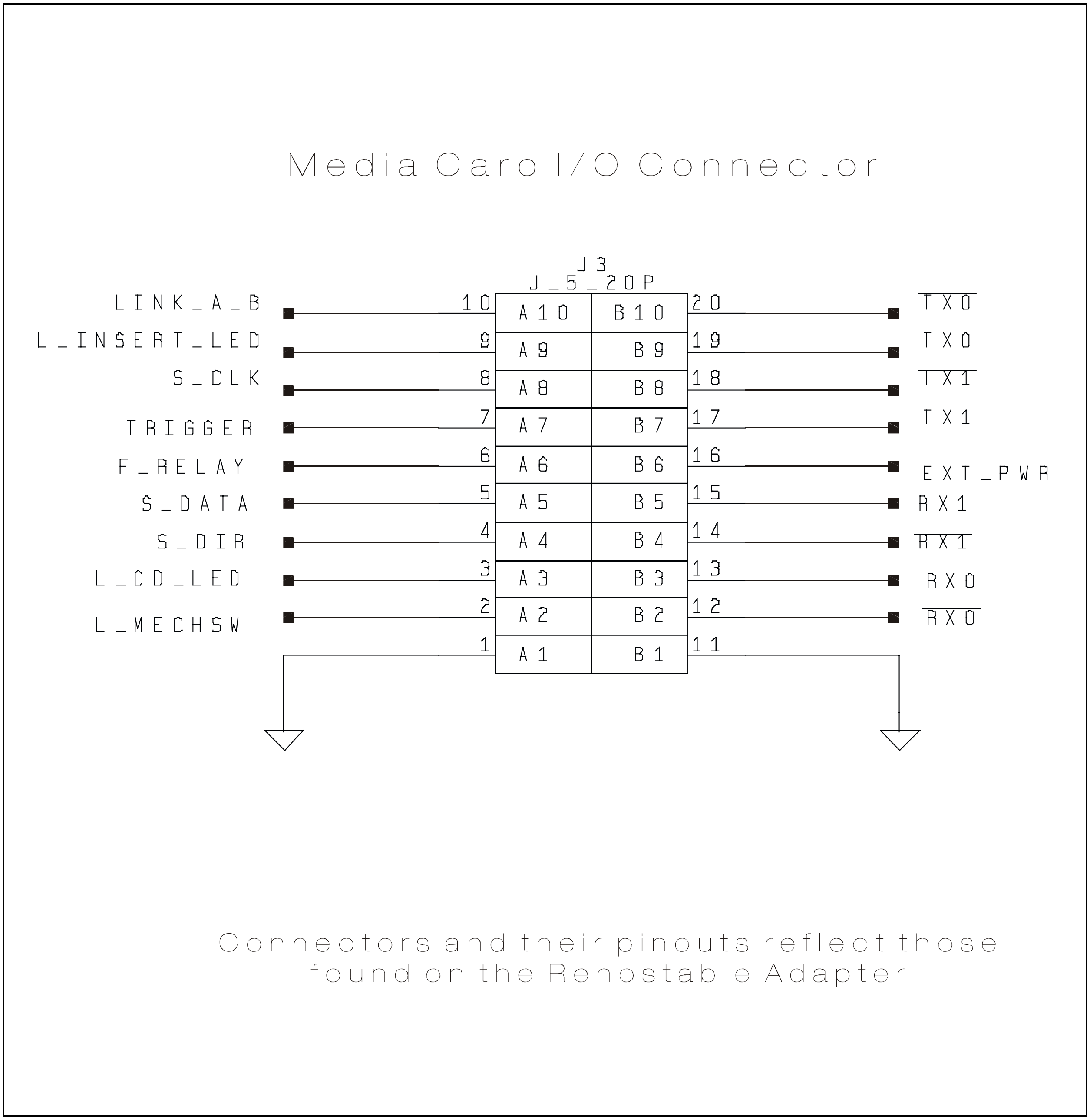


Figure D-20 Rehostable Adapter Interface Connector



Connectors and their pinouts reflect those found on the Rehostable Adapter

Figure D-21 Rehostable Adapter Media Card Interface Connector

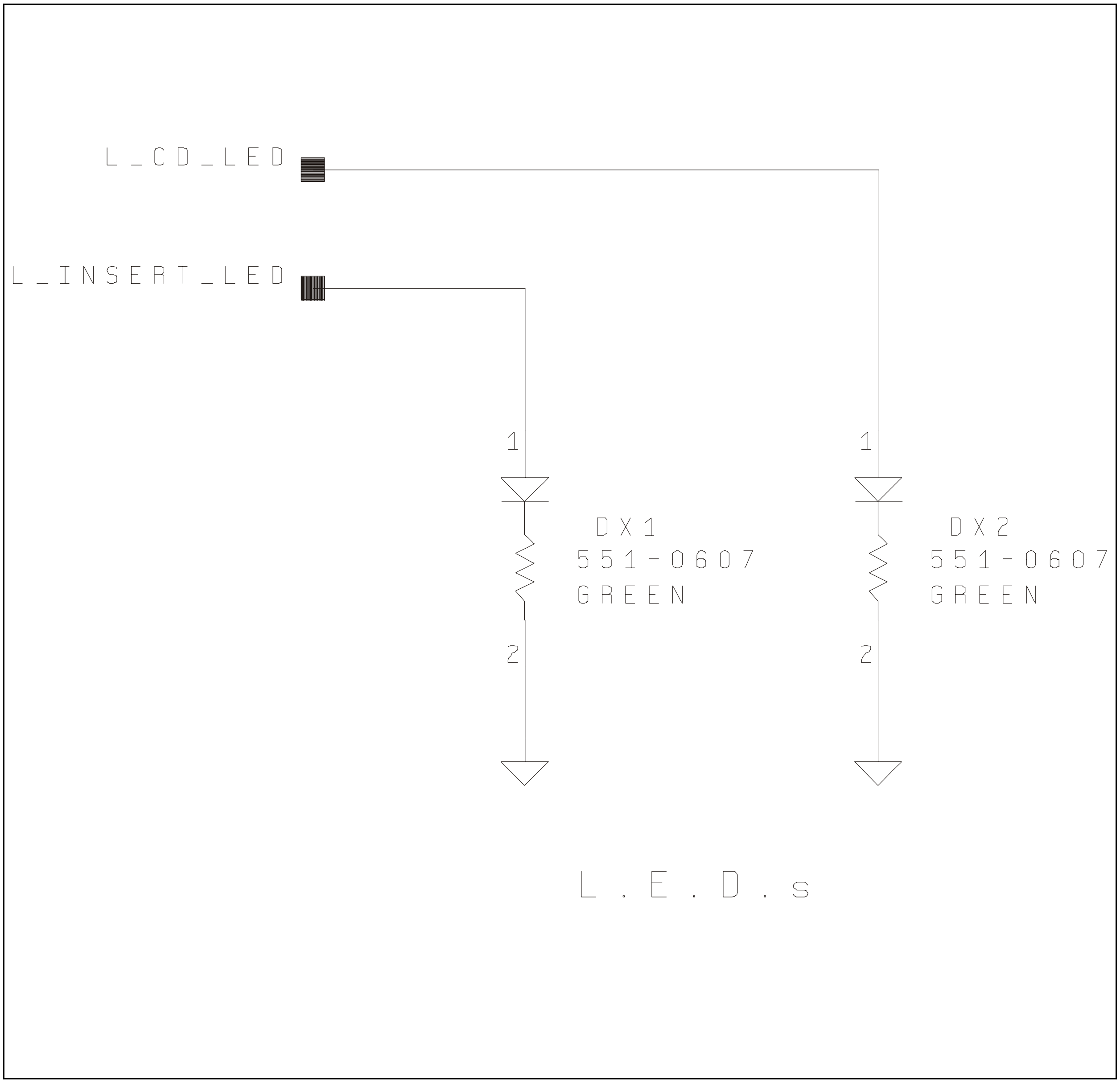


Figure D-22 User Interface LEDs

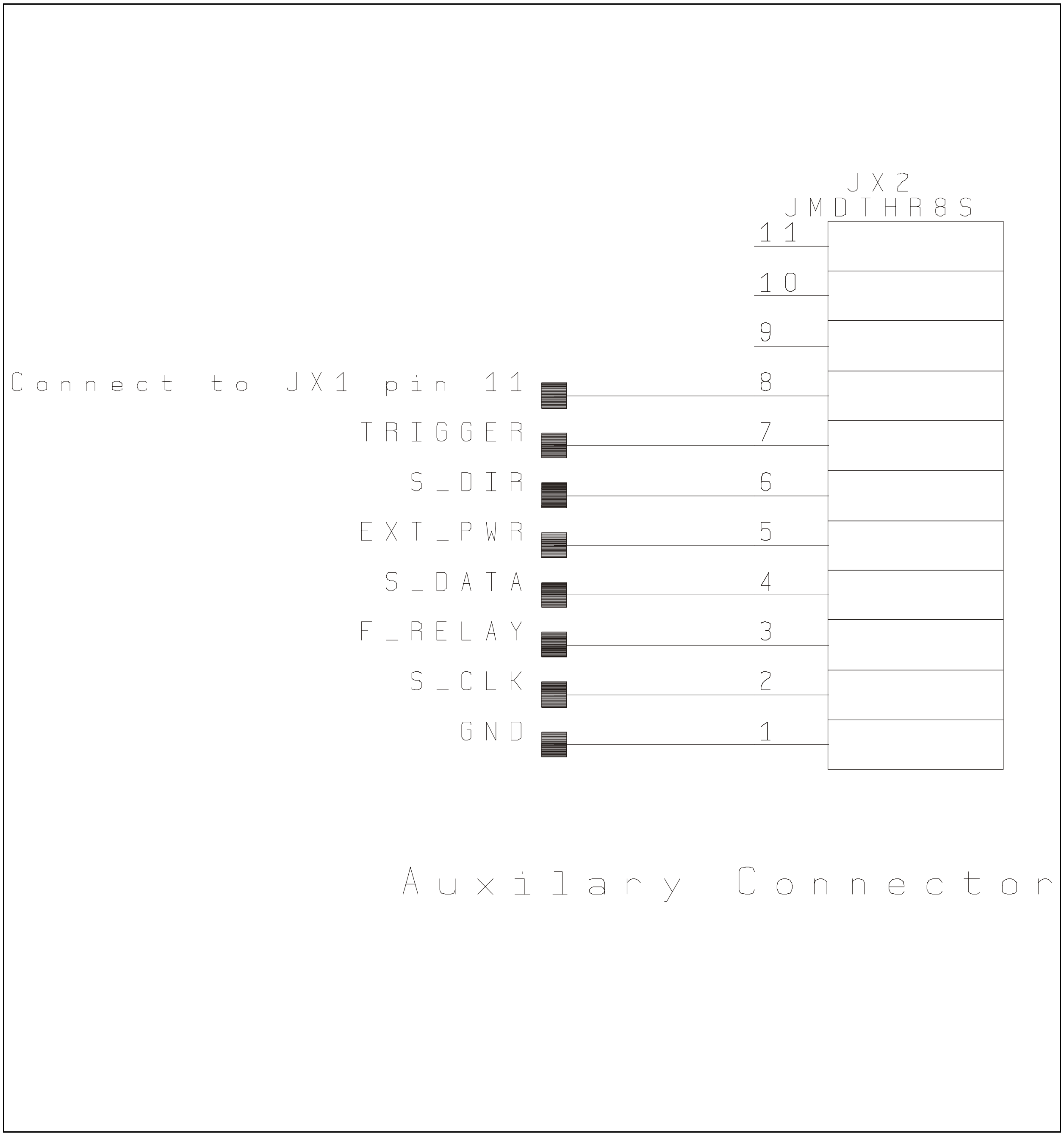


Figure D-23 User INterface Auxiliary Connector

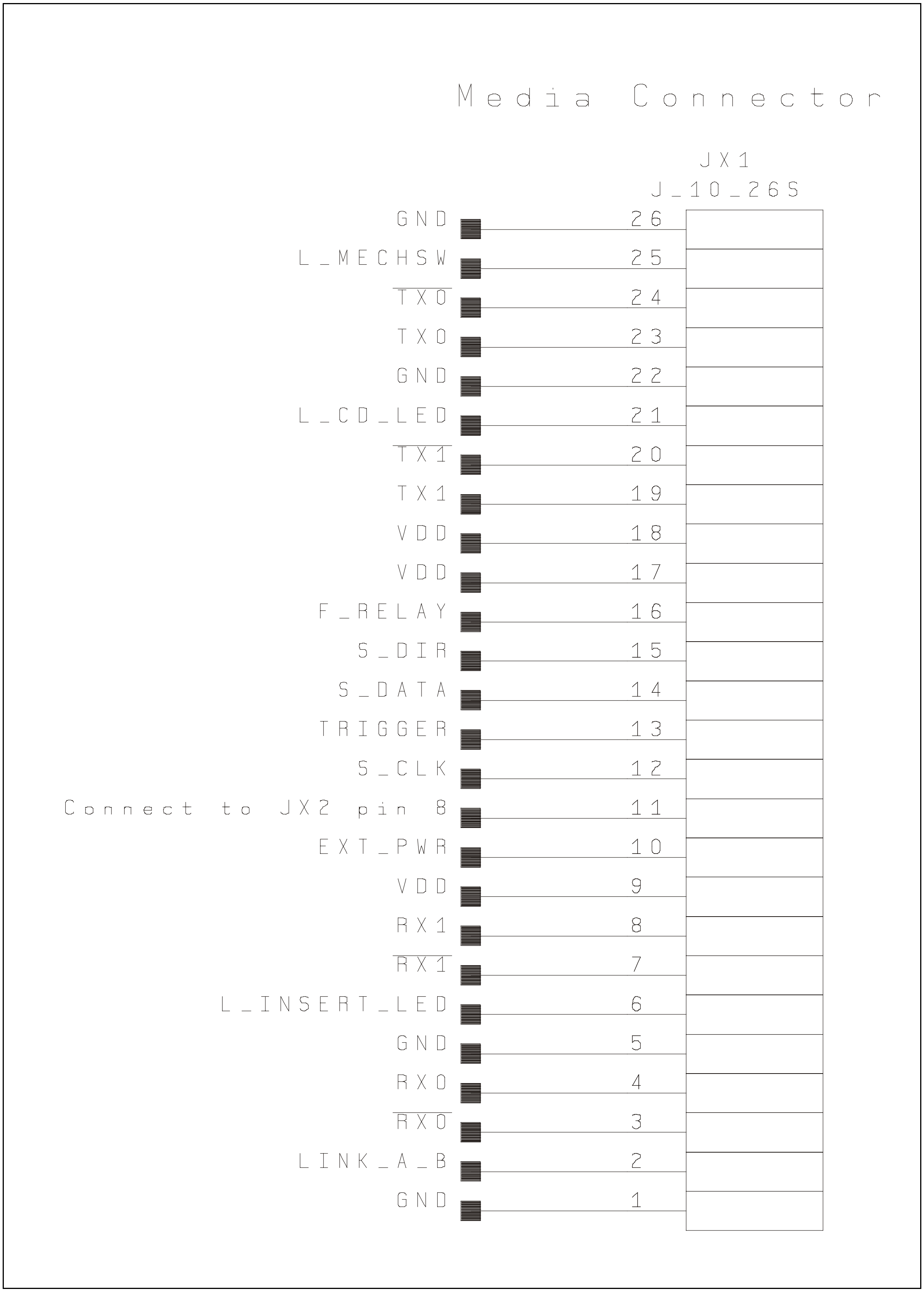
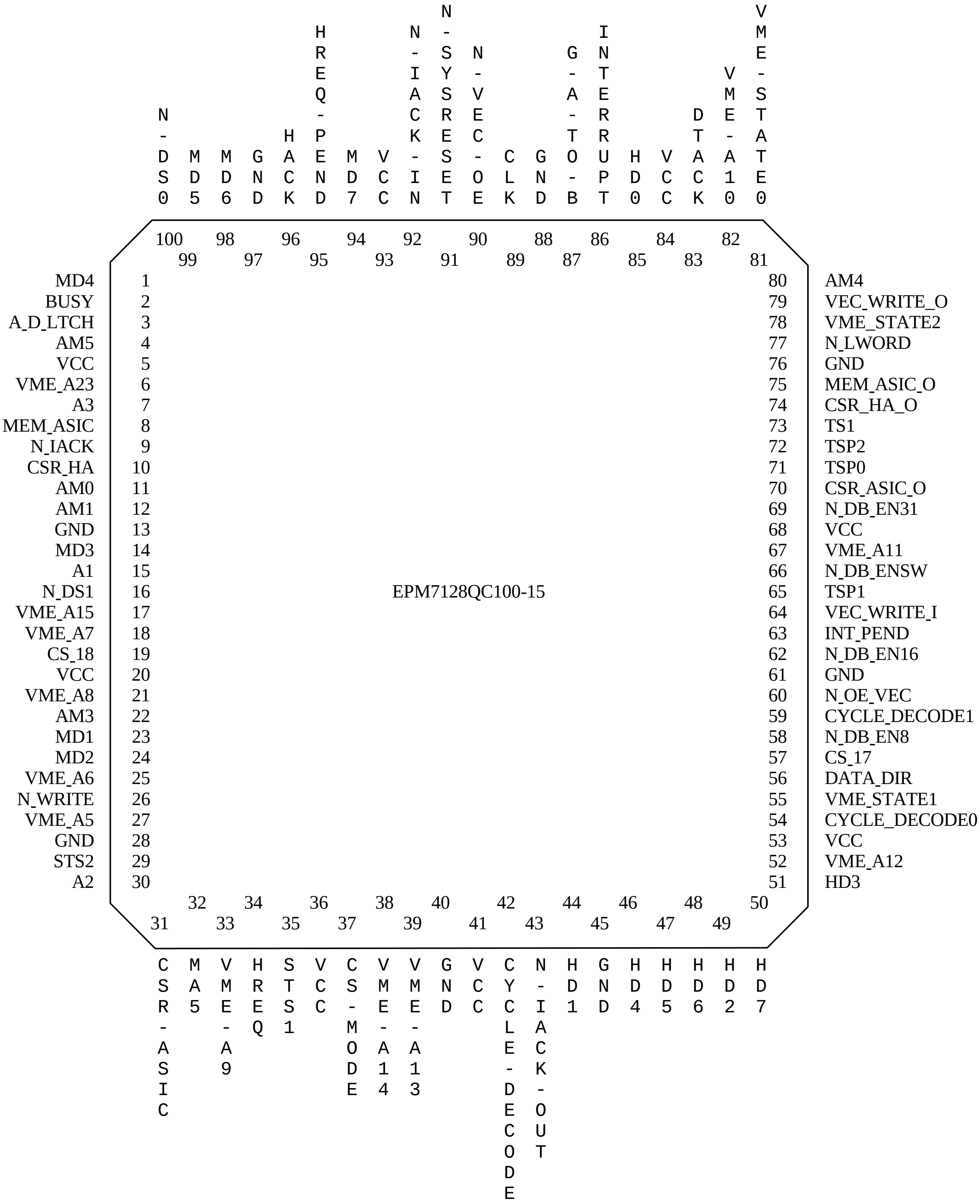


Figure D-24 User Interface Media Access Connector



N.C. = Not Connected.
VCC = Dedicated power pin, which MUST be connected to VCC.
GND = Dedicated ground pin or unused dedicated input, which MUST be connected to GND.
RESERVED = Unused I/O pin, which MUST be left unconnected.

Figure D-25 VME Controller Interface Pinout

This page intentionally left blank

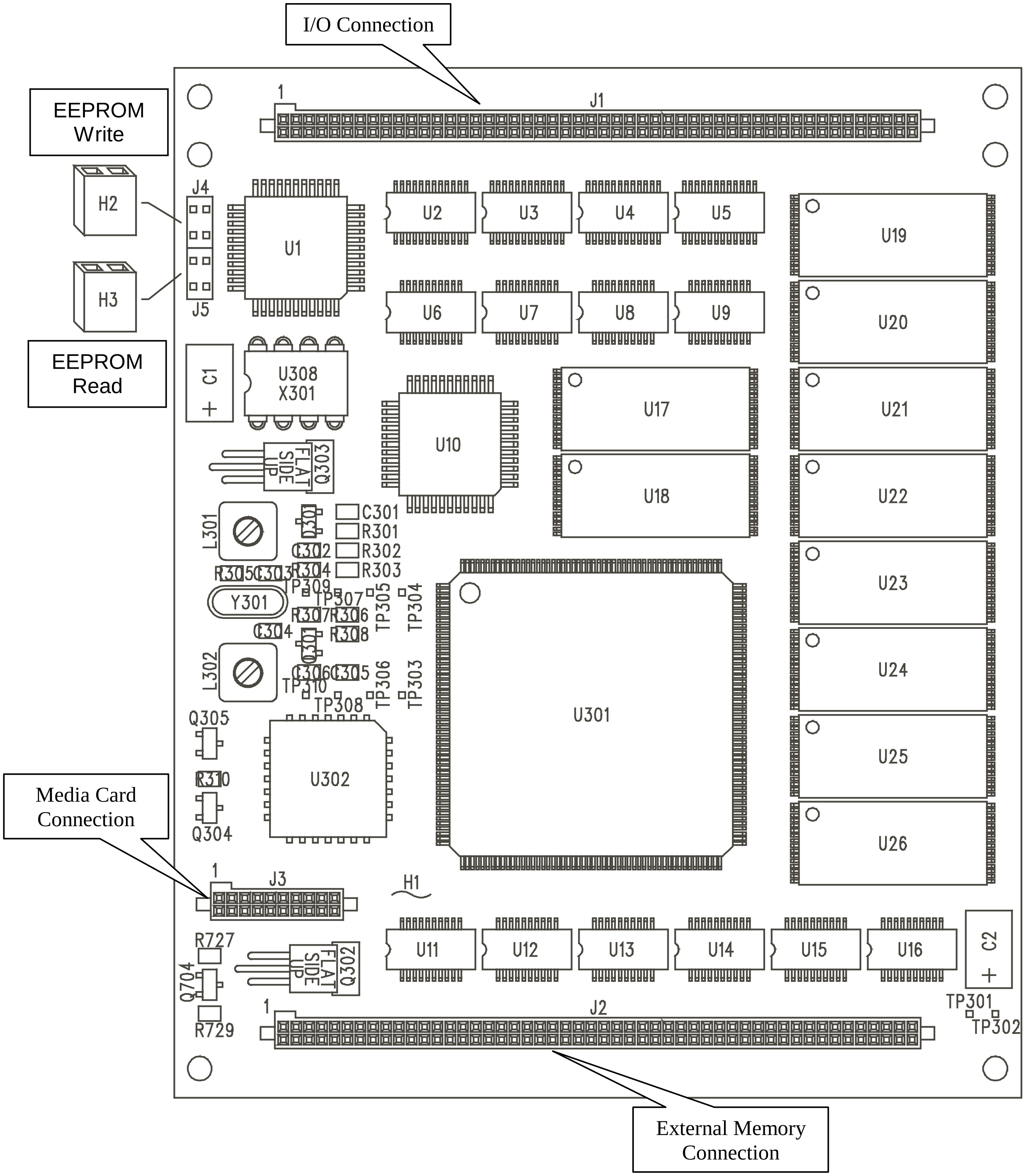
APPENDIX E

HARDWARE ASSEMBLY

TABLE OF CONTENTS

E.1 Overview

This Appendix contains the mechanical assembly drawing with dimensions for creating a design. The Rehostable VME Interface evaluation board is included for completeness.



I/O Connection

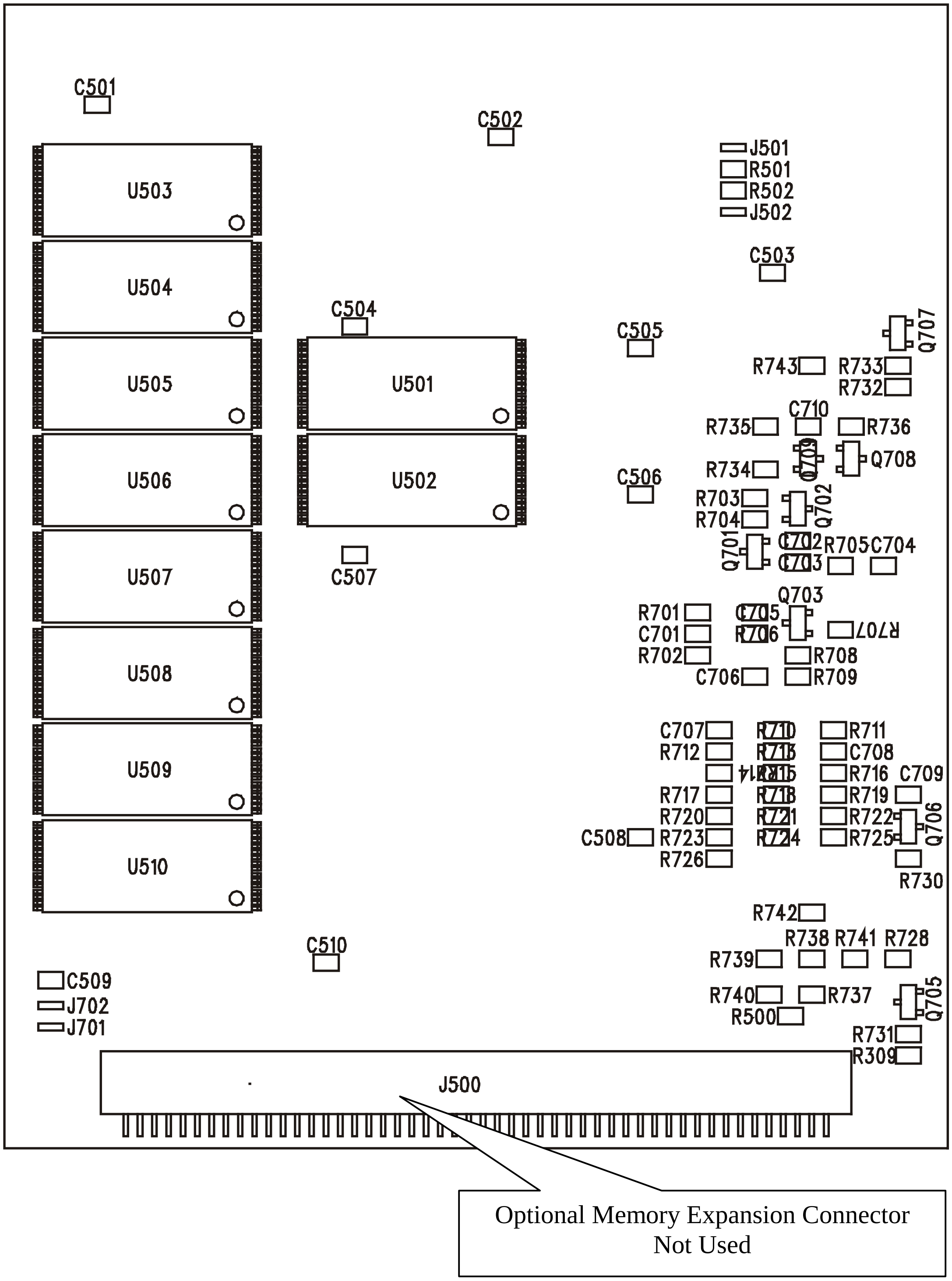
EEPROM Write

EEPROM Read

Media Card Connection

External Memory Connection

Figure E-1 Rehostable Assembly Component Side



Optional Memory Expansion Connector
Not Used

Figure E-2 Rehostable Assembly Solder Side

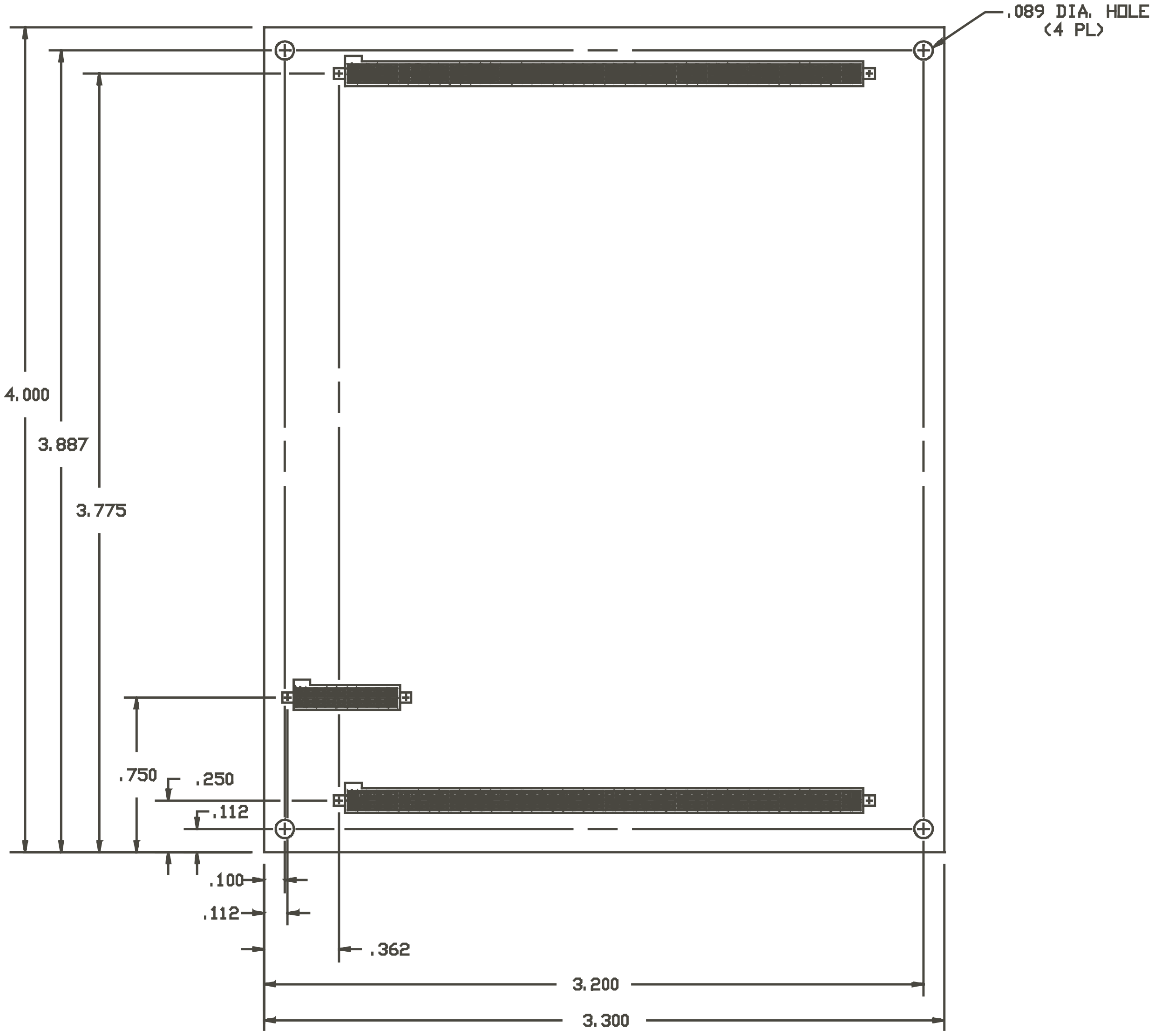
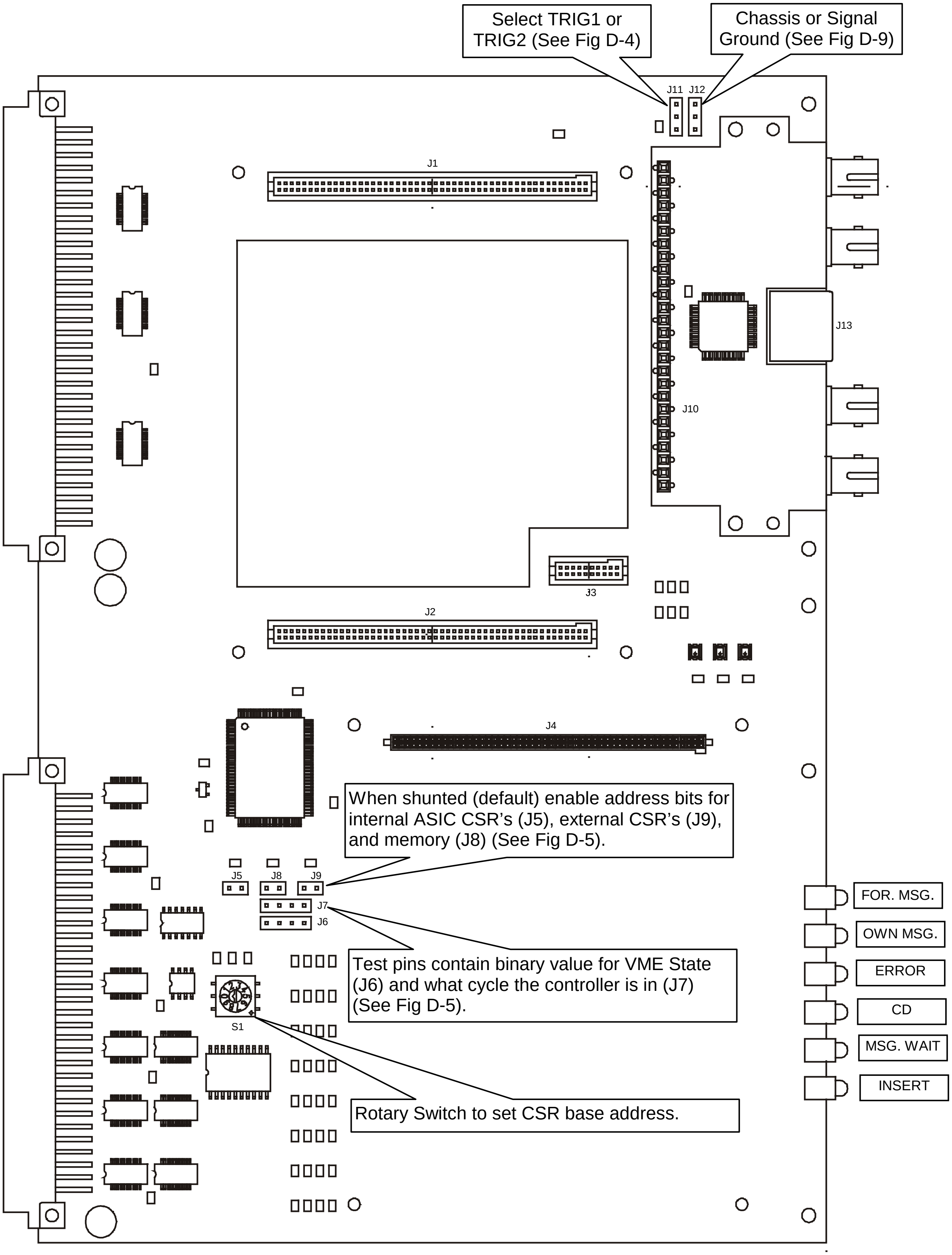


Figure E-3 Rehostable Connections


Select TRIG1 or TRIG2 (See Fig D-4)

Chassis or Signal Ground (See Fig D-9)

When shunted (default) enable address bits for internal ASIC CSR’s (J5), external CSR’s (J9), and memory (J8) (See Fig D-5).

Test pins contain binary value for VME State (J6) and what cycle the controller is in (J7) (See Fig D-5).

Rotary Switch to set CSR base address.

Figure E-4 Rehostable Evaluation Board

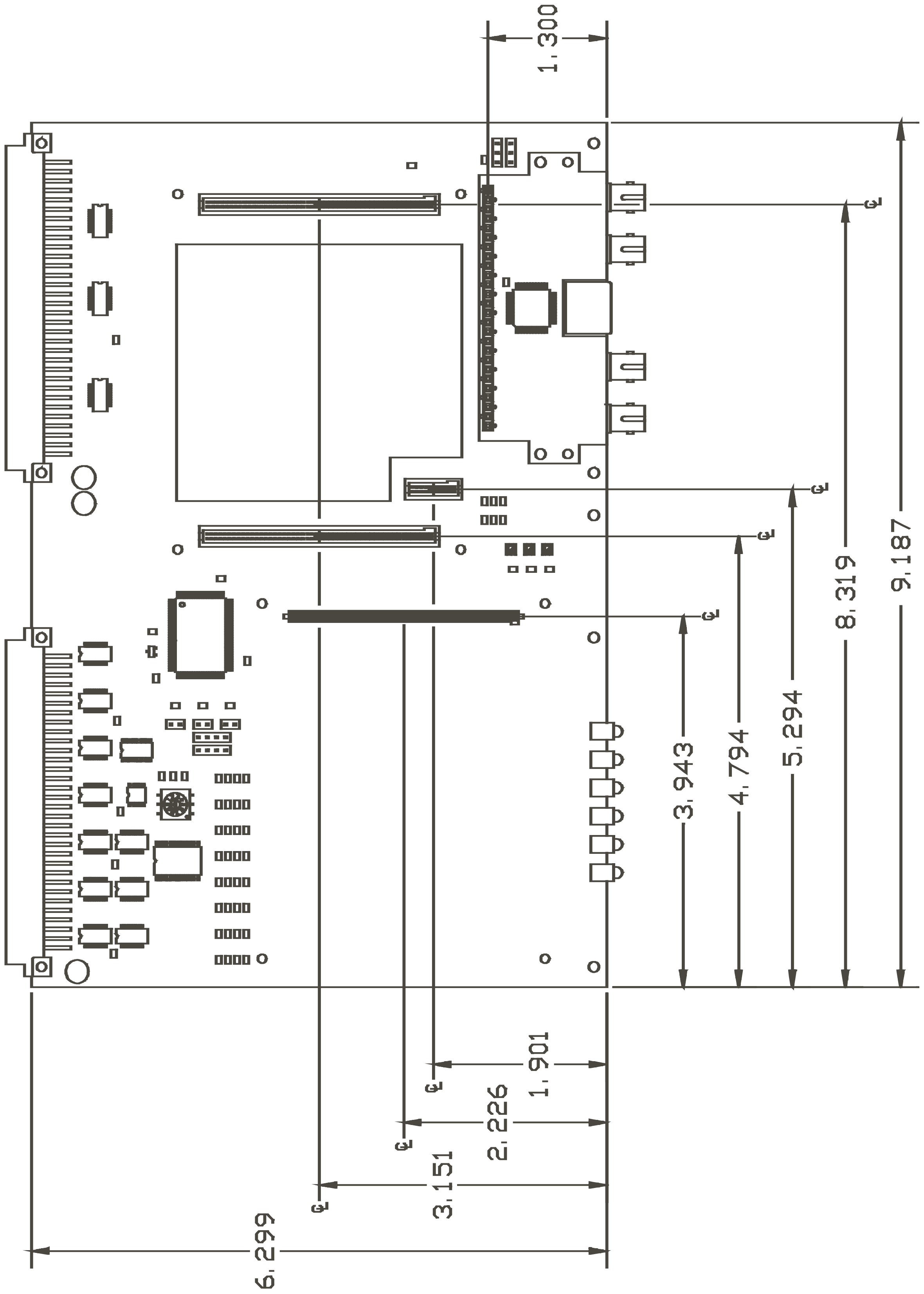


Figure E-5 Rehostable Evaluation Board Dimensions

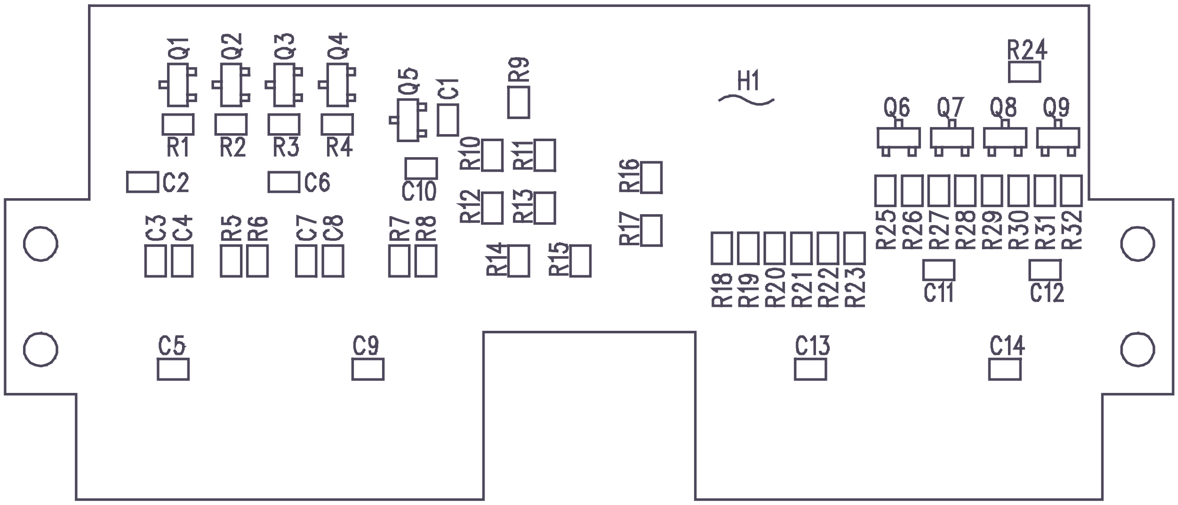


Figure E-6 Media Card (CTRSTDF) Top

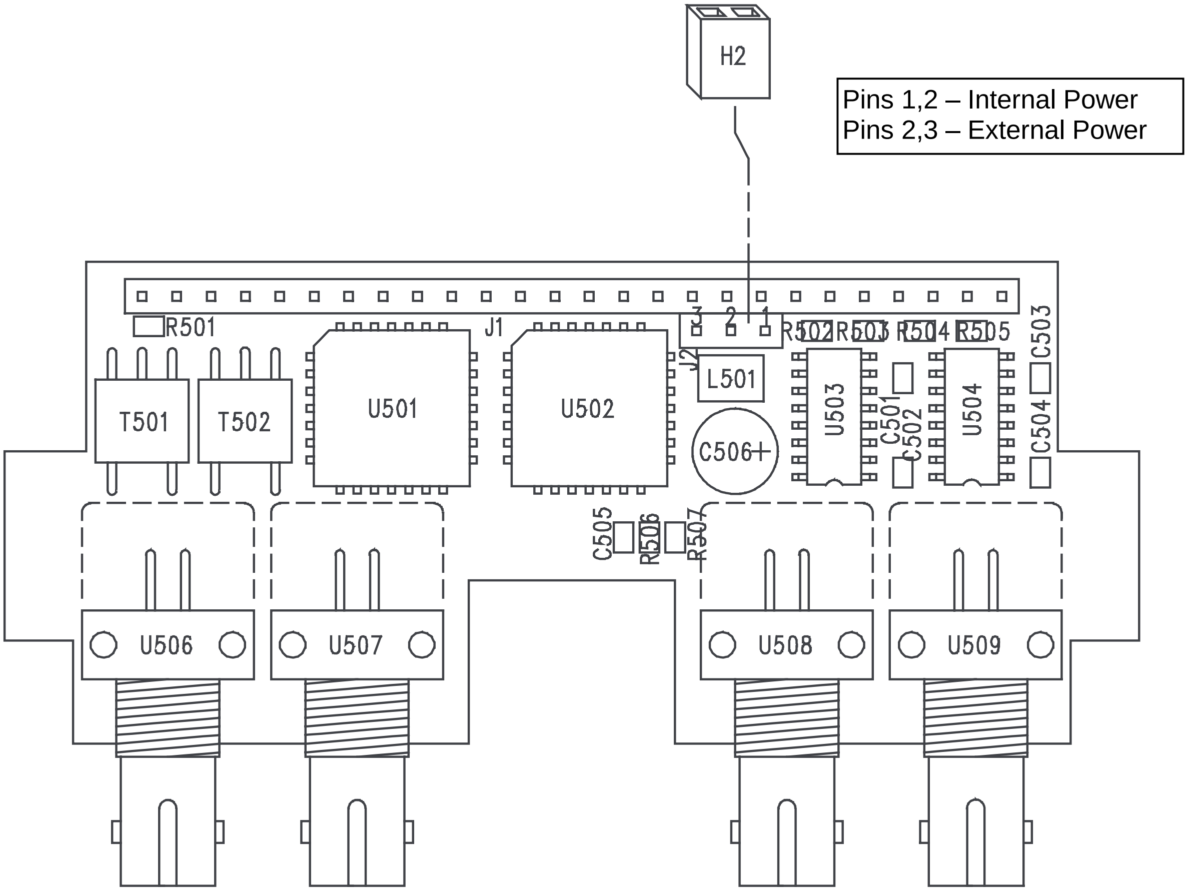

Pins 1,2 – Internal Power
Pins 2,3 – External Power

Figure E-7 Media Card (CTRSTDF) Bottom

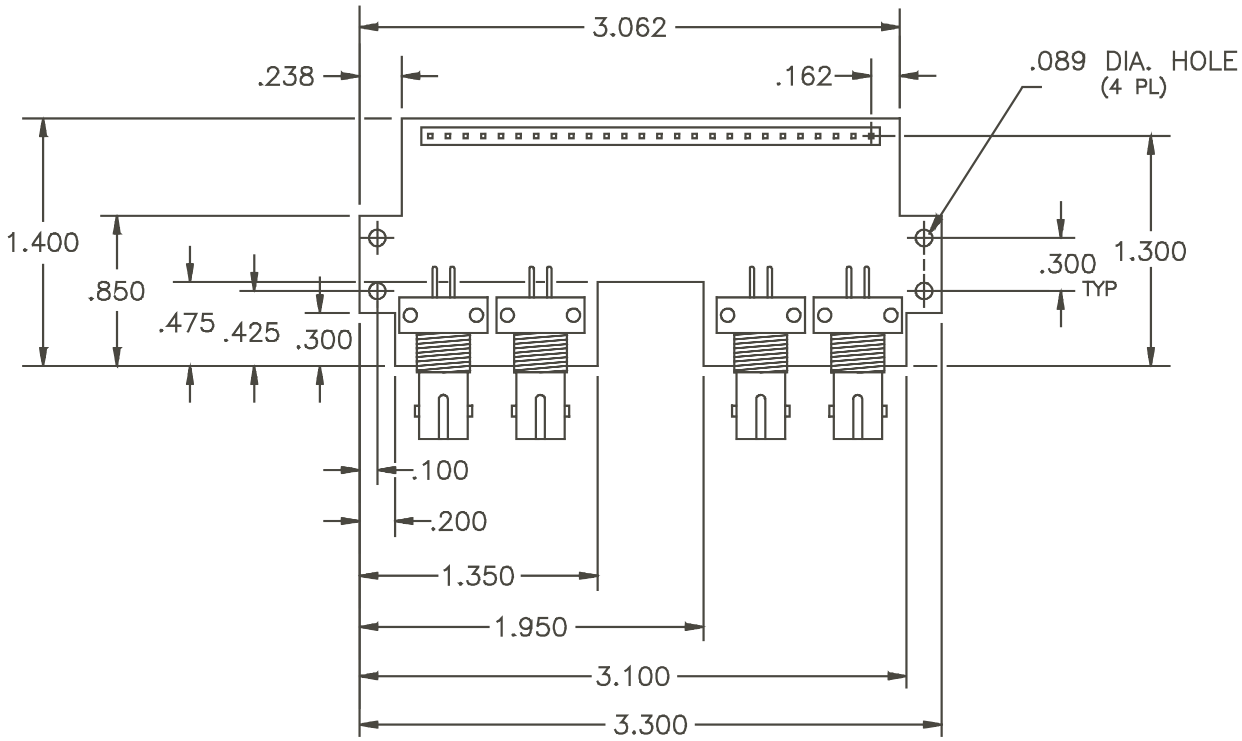
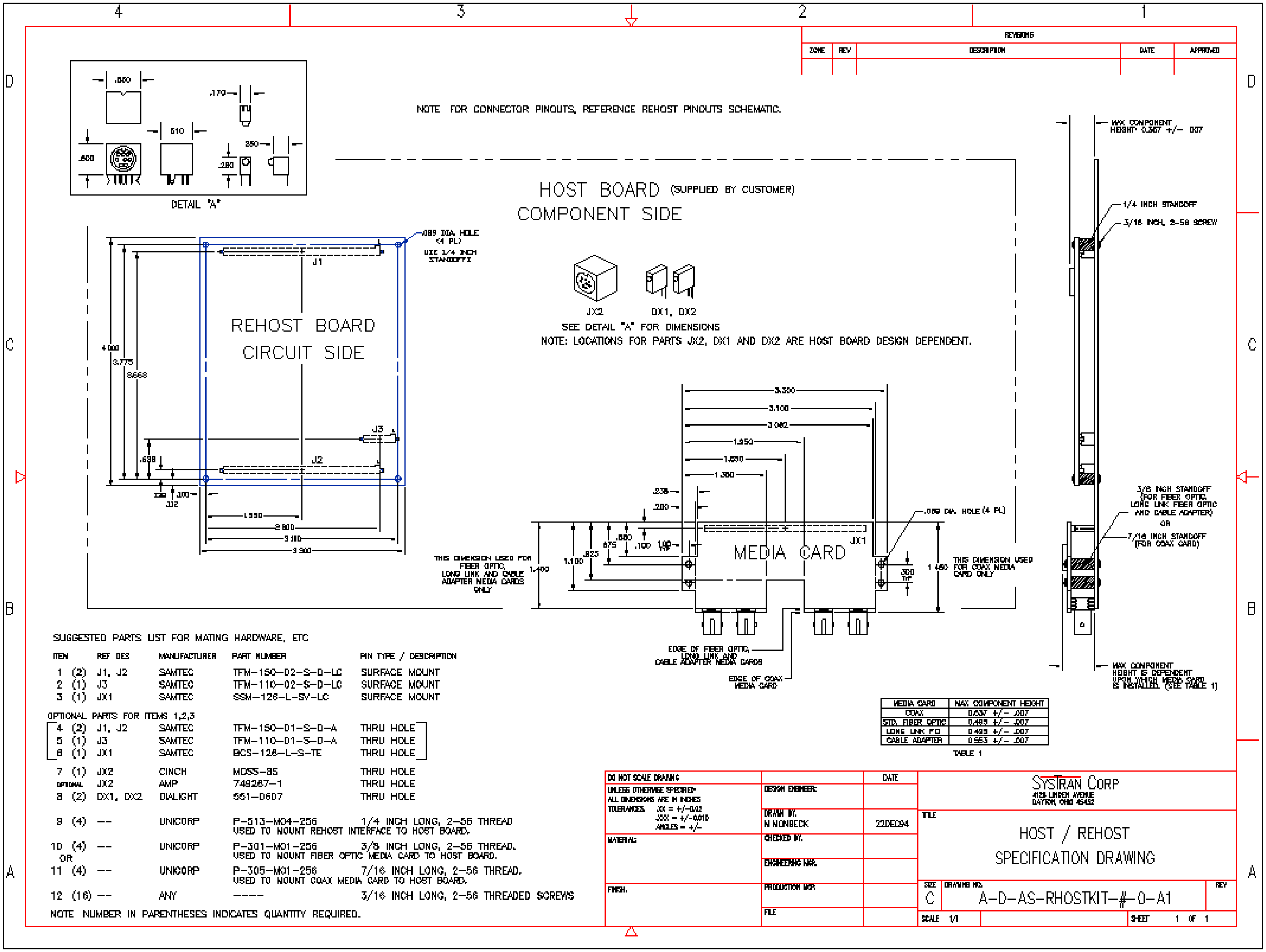


Figure E-8 Media Card (CTRSTDF) Dimensions


NOTE FOR CONNECTOR PINOUTS, REFERENCE REHOST PINOUTS SCHEMATIC.
DETAIL "A"
HOST BOARD (SUPPLIED BY CUSTOMER)
COMPONENT SIDE
JX2
DX1, DX2
SEE DETAIL "A" FOR DIMENSIONS
NOTE: LOCATIONS FOR PARTS JX2, DX1 AND DX2 ARE HOST BOARD DESIGN DEPENDENT.
REHOST BOARD
CIRCUIT SIDE
J1
J2
J3
.089 DIA. HOLE (4 PL)
USE 1/4 INCH STANDOFFS
MEDIA CARD
JX1
.089 DIA. HOLE (4 PL)
THIS DIMENSION USED FOR FIBER OPTIC, LONG LINK AND CABLE ADAPTER MEDIA CARDS ONLY
THIS DIMENSION USED FOR COAX MEDIA CARD ONLY
EDGE OF FIBER OPTIC, LONG LINK AND CABLE ADAPTER MEDIA CARDS
EDGE OF COAX MEDIA CARD
MAX COMPONENT HEIGHT 0.367 +/- .007
1/4 INCH STANDOFF
3/16 INCH, 2-56 SCREW
3/8 INCH STANDOFF (FOR FIBER OPTIC, LONG LINK FIBER OPTIC AND CABLE ADAPTER)
OR
7/16 INCH STANDOFF (FOR COAX CARD)
MAX COMPONENT HEIGHT IS DEPENDENT UPON WHICH MEDIA CARD IS INSTALLED (SEE TABLE 1)
REVISIONS
ZONE
REV
DESCRIPTION
DATE
APPROVED
DO NOT SCALE DRAWING
UNLESS OTHERWISE SPECIFIED:
ALL DIMENSIONS ARE IN INCHES
TOLERANCES .XX = +/-0.02
.XXX = +/-0.010
ANGLES = +/-
MATERIAL:
FINISH:
DESIGN ENGINEER:
DRAWN BY.
N NONBECK
22DEC94
CHECKED BY.
ENGINEERING MGR.
PRODUCTION MGR.
FILE
SYSTRAN CORP
4126 LINDEN AVENUE
DAYTON, OHIO 45432
TITLE
HOST / REHOST
SPECIFICATION DRAWING
SIZE C
DRAWING NO. A-D-AS-RHOSTKIT-#-0-A1
REV
SCALE 1/1
SHEET 1 OF 1

MEDIA CARD	MAX COMPONENT HEIGHT
COAX	0.637 +/- .007
STD. FIBER OPTIC	0.485 +/- .007
LONG LINK FO	0.495 +/- .007
CABLE ADAPTER	0.553 +/- .007

TABLE 1

SUGGESTED PARTS LIST FOR MATING HARDWARE, ETC

ITEM	REF DES	MANUFACTURER	PART NUMBER	PIN TYPE / DESCRIPTION
1 (2)	J1, J2	SAMTEC	TFM-150-02-S-D-LC	SURFACE MOUNT
2 (1)	J3	SAMTEC	TFM-110-02-S-D-LC	SURFACE MOUNT
3 (1)	JX1	SAMTEC	SSM-126-L-SV-LC	SURFACE MOUNT
OPTIONAL PARTS FOR ITEMS 1,2,3				
4 (2)	J1, J2	SAMTEC	TFM-150-01-S-D-A	THRU HOLE
5 (1)	J3	SAMTEC	TFM-110-01-S-D-A	THRU HOLE
6 (1)	JX1	SAMTEC	BCS-126-L-S-TE	THRU HOLE
7 (1)	JX2	CINCH	MDSS-85	THRU HOLE
OPTIONAL	JX2	AMP	749287-1	THRU HOLE
8 (2)	DX1, DX2	DIALIGHT	551-0607	THRU HOLE
9 (4)	--	UNICORP	P-513-M04-256	1/4 INCH LONG, 2-56 THREAD USED TO MOUNT REHOST INTERFACE TO HOST BOARD.
10 (4) OR	--	UNICORP	P-301-M01-256	3/8 INCH LONG, 2-56 THREAD. USED TO MOUNT FIBER OPTIC MEDIA CARD TO HOST BOARD.
11 (4)	--	UNICORP	P-305-M01-256	7/16 INCH LONG, 2-56 THREAD. USED TO MOUNT COAX MEDIA CARD TO HOST BOARD.
12 (16)	--	ANY	----	3/16 INCH LONG, 2-56 THREADED SCREWS

NOTE NUMBER IN PARENTHESES INDICATES QUANTITY REQUIRED.

Figure E-9 Host/Rehost Specification

APPENDIX F

BILL OF MATERIALS

F.1 Overview

This Appendix contains the bill of materials to be used on the designer’s host interface. The parts include connectors, both surface-mount and thru-hole. These connectors are for interfacing to the Rehostable Adapter as well as interfacing to the Media Card and Auxiliary Port. Standoffs and the sizes to be used are represented in the table. Finally, the two diodes are used for status.

Suggested Parts List

	RefDes	Manufacturer	Part Number	Pin Type	* Standoff
1	J1, J2	Samtec	TFM-150-02-S-D-LC	Surface Mount	UNICORP P/N P-513-M04-256 Length 1/4 Thread 2-56
2	J3	Samtec	TFM-110-02-S-D-LC	Surface Mount	
3	Jx1	Samtec	SSM-126-L-SV-LC	Surface Mount	See Note1 below

	RefDes	Manufacturer	Part Number	Pin Type	* Standoff
4	J1, J2	Samtec	TFM-150-01-S-D-A	Thru Hole	UNICORP P/N P-513-M04-256 Length 1/4 Thread 2-56
5	J3	Samtec	TFM-110-01-S-D-A	Thru Hole	
6	Jx1	Samtec	BCS-126-L-S-TE	Thru hole	See Note1 below

7	Jx2	Cinch	MDSS-8S	Thru Hole
8	Jx2	Amp	749267-1	Thru Hole
9	Dx1	Dialight	551-0607	Thru Hole
10	Dx2	Dialight	551-0607	Thru Hole

Note: Item numbers 1, 2, 3, 7, 8, 9, and 10 were used on our in house interface to the Rehost. Items 4, 5, and 6 are the suggested Thru hole parts.

Note1: This Standoff size is determined by the type of MAC card used. For the Fiber Optic MAC a 3/8 x 2-56 standoff will be used. For the COAX MAC a 7/16 x 2-56 will be used.

* This standoff will be used between the Rehost board and the host interface design. This will allow a 1/4 board separation.

Figure F-1 Rehostable Assembly, Component Side

GLOSSARY

ACR	**auxiliary control RAM.** A memory buffer typically used as a data bus width extension for control purposes only. Also referred to as **shadow memory** .
bad message	A message error condition reported by a node's receiver circuitry. This condition is automatically corrected by SCRAMNet hardware.
burst	A protocol where messages are transmitted without error correction to gain higher throughput.
burst+	Also **burst plus**. A variable packet size enhancement for the burst protocol. Maximum packet size may be set to either 256 bytes or 1024 bytes.
carrier wave	An electromagnetic wave that can be modulated, as in frequency, amplitude, or phase, to transmit data, images, sound, or other signals.
FIFO	A data storage method; First In First Out. Also refers to the specific storage area; Transmit FIFO, Interrupt FIFO, etc.
foreign message	A message that is in (passing through) a node other than the one of origin.
insert	The act of placing a node on a network for the purpose of transmitting and receiving messages.
interrupt	An event that changes the normal flow of instruction execution other than an exception or a branch, jump, case or call instruction.
ISR	**interrupt service routine.** A routine executed when a device interrupt occurs.
latched	Data is electrically stored in a circuit until it is needed. A method of coordinating two synchronous events.
longword	Four bytes (32 bits) of data.
loopback	A method of transmitting to the same node's receivers for testing purposes. Applies to both fiber optic and wire media. Also, a test that loops the outgoing signal back to its source.
message packet	See packet.
native message	A message that is received by the node of origin.
node latency	The time delay at a node before a foreign message can be retransmitted.
packet	A message that travels on the network. The minimum packet consists of 81 bits and 1 start bit. The packet includes five fields: Source ID (8 bits), Age (8 bits), Control (3 bits), Data Address (21 bits), Data (32 bits), and 9 parity bits; one for every 8 bits.
physical address	The address used by hardware to identify a location in physical memory or on directly-addressable secondary storage devices (such as disks). A physical memory address consists of a page-frame number and the number of a byte within the page.
platinum	A protocol where messages are transmitted as fast as the system will allow with error correction enabled.
platinum+	(Also platinum plus). A variable packet size enhancement for the platinum protocol. Maximum packet size may be set to either 256 bytes or 1024 bytes.

protocol violation	A signal error at the physical layer (fiber or coax) resulting from noise on the transmission lines or a result of hardware failure. This violation can be any one of the following: • Missing transition for two clock periods on either line • Parity error • Framing error
read cycle	A DTB cycle used to transfer 1-, 2-, 3-, or 4-bytes from a slave to a master. The cycle begins when the master broadcasts an address and an address modifier. Each slave captures this address and address modifier, and checks to see if it is to respond to the cycle. If so, it retrieves the data from its internal storage, places it on the data bus, and acknowledges the transfer. Then the master terminates the cycle.
requester	A functional module that resides on the same board as a master or interrupt handler and requests use of the CTB whenever its master or interrupt handler needs it.
retry	A hardware failure condition reported when the first attempt to send a message around the network has resulted in some type of bit error. The message will be retransmitted indefinitely by the originating node until it is received correctly by the originating node. Valid only in error correction mode (PLATINUM.)
retry time-out	A hardware failure condition reported when the first attempt to send a message around the network is not received by the originating node within the time out period specified in CSR5. The message will be retransmitted indefinitely by the originating node until it is received correctly by the originating node. Valid only in error correction mode (PLATINUM.)
ring time	The time it takes a message to traverse the network ring from the originating node and back again. The time can be calculated by estimating propagation delay at 5 ns per meter of cable plus 250 ns per node. This assumes a best case using 32-bit standard data with no other nodes transmitting. Worst case would be using 800 ns per node. Other times can be calculated using the maximum delays per node when sending variable length data.
rising edge	The time during which a signal makes its transition from low to high.
Rx	Abbreviation for receive or receiver.
shadow memory	See Auxiliary Control RAM (ACR)
shared memory (SM)	**SCRAMNet** memory physically located on the network board. This dual-ported memory is accessible by the host and the network. A host write to shared memory results in a transmitted write to all SCRAMNet nodes at the same relative location.
shortword	16 bits. Also referred to as **halfword**.
signal mnemonics	Terms used to identify signal line events. (1) An asterisk following the name of signals that are level-significant denotes the signal is true/valid when the signal is low. (2) An asterisk following the name of signals that are edge-significant denotes the actions initiated by that signal occur on the falling edge.

time-out----------------------------- Also network time-out. The time written to CSR5 that must elapse before a native message will be retransmitted. The time-out must be a non-zero value.

transaction controller ---------- A state machine that runs the interfacing handshakes between the Rehostable Adapter and the VME bus.

Tx ----------------------------------- Abbreviation for transmit or transmitter.

UAT --------------------------------- A master that sends or receives data in an unaligned fashion.

write posting ---------------------- The decoupling of the host bus with the Rehostable Adapter during write cycles

VME address space -------------	The VME address space varies according to specific VME device and is identified as A16, A24, or A32 space. A32 is the largest address space; it allows up to 4 gigabytes of space using 32 bit addresses. A24 space uses 24 bit addresses, and A16 space uses 16 bit addresses.
VMEbus ----------------------------	A standard bus by which small computers and intelligent peripheral devices can be connected. The term VME stands for Versa Module Eurocard. This non-proprietary bus conforms to the American National IEEE Standard 1014 (ANSI/IEEE std 1014).
write cycle -------------------------	A DTB cycle used to transfer 1-, 2-, 3-, or 4-bytes from a master to a slave. The cycle begins when the master broadcasts an address and address modifier and places data on the DTB. Each slave captures this address and address modifier, and checks to see if it is to respond to the cycle. If so, it stores the data and then acknowledges the transfer. The master then terminates the cycle.

INDEX

This page intentionally left blank