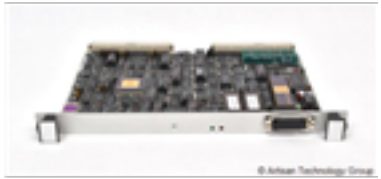
Motorola MVME330

VME Ethernet Controller



In Stock

Like-New / Demo

Open Web Page

https://www.artisantg.com/61700-3

PRELIMINARY INFORMATION

MVME330/D1

AUGUST 1984

MVME330

ETHERNET CONTROLLER

USER'S MANUAL

WARNING

THIS EQUIPMENT GENERATES, USES, AND CAN RADIATE RADIO FREQUENCY ENERGY AND, IF NOT INSTALLED AND USED IN ACCORDANCE WITH THE INSTRUCTION MANUAL, MAY CAUSE INTERFERENCE TO RADIO COMMUNICATIONS. AS TEMPORARILY PERMITTED BY REGULATION, IT HAS NOT BEEN TESTED FOR COMPLIANCE WITH THE LIMITS FOR CLASS A COMPUTING DEVICES PURSUANT TO SUBPART J OF PART 15 OF FCC RULES, WHICH ARE DESIGNED TO PROVIDE REASONABLE PROTECTION AGAINST SUCH INTERFERENCE. OPERATION OF THIS EQUIPMENT IN A RESIDENTIAL AREA IS LIKELY TO CAUSE INTEFERENCE, IN WHICH CASE THE USER, AT HIS OWN EXPENSE, WILL BE REQUIRED TO TAKE WHATEVER MEASURES MAY BE REQUIRED TO CORRECT THE INTERFERENCE.

First Edition

CHAPTER 1

GENERAL INFORMATION

1.1 INTRODUCTION

This manual provides general information, hardware preparation and installation instructions, software preparation and loading instructions, functional description, and support information for the Motorola MVME330 Ethernet Controller. A typical module is shown in Figure 1-1.

For software interfacing information, refer to the KERNEL user's manual. This manual describes the software protocols and formats necessary to enable the MVME330 to communicate with the Ethernet. For more information about Ethernet, refer to Appendix A.

1.2 FEATURES

The features of the MVME330 include:

- 10 MHz, MC68000 Microprocessor Unit (MPU).
- 128K-byte dual-access RAM with parity and no wait states.
- 32K-byte EROM (2 sockets).
- VMEbus (IEEE P1014) A24:D16 master and slave interface for host to MVME330 communications.
- Node address PROM contains a unique address issued by Xerox Corp.
- 2ms timer interrupts the MPU for protocol software timing.
- Local Area Network Controller for Ethernet (LANCE).
  - Descriptor ring buffer management
  - Direct Memory Access (DMA) to local RAM
  - Line access protocol (CSMA/CD)
  - Extensive diagnostics and error reporting to MPU
- Serial Interface Adaptor (SIA).
  - Manchester encoding/decoding
  - Transceiver cable interface
- VMEbus requester.
- VMEbus interrupter with programmable vector.
- VMEbus to onboard processor interrupter.
- Ability to interface across the VMEbus using a serial port for debugging.

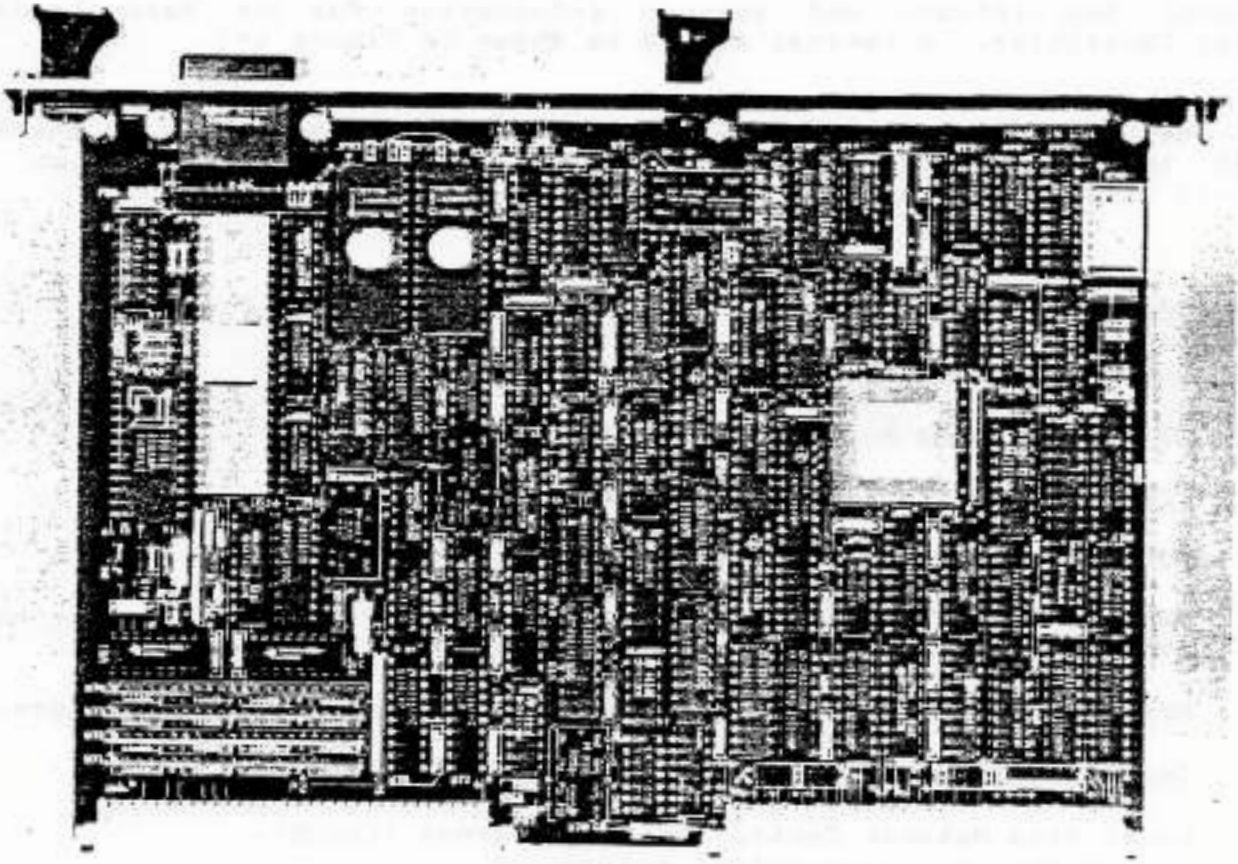


FIGURE 1-1. MVME330 Ethernet Controller

1.3 SPECIFICATIONS

The MVME330 specifications (which are subject to change without notice) are given in Table 1-1.

TABLE 1-1. MVME330 Specifications

CHARACTERISTIC	SPECIFICATIONS
Power requirments	4.4A at +5 Vdc ±5% 0.6 A at +12 Vdc ±5% 0.1 A at -12 Vdc ±5%
Operating temperature	5 degrees C to 50 degrees C
Storage temperature	-40 degrees C to 85 degrees C
Humidity	5% to 95% (non-condensing)
Altitude	0 to 10,000 feet
Reliability	70,000 hours MTBF
Size	6.3 inches x 9.2 inches (16.0 cm x 23.4 cm)
Connectors	
VMEbus	DIN no. 41612C96
Ethernet tranceiver	AMP745094-1

1.4 VME CONFIGURATION

The VME configuration is as follows.

MASTER DATA TRANSFER
- A24:D16
- TOUT = 8 or 16 us
- ADDRESS MODIFIERS = 3E, 3D, 3A, 39, 2D, 29

SLAVE DATA TRANSFER
- A24:D16
- ADDRESS MODIFIERS = 3D, 39

REQUESTER
- Any one of R(0), R(1), R(2), or R(3), (STAT)
- RWD (Release after each MPU VMEbus cycle)

INTERRUPTER OPTIONS
- Any one of I(1), I(2), I(3), I(4), I(5), I(6), or I(7) (STAT)

1.5 GENERAL DESCRIPTION

The MVME330 is a high performance communications processor which provides the physical interface and intelligence necessary to attach information processing devices to Ethernet, a local area network, allowing a high speed exchange of information. Each MVME330 contains node-specific software, as well as industry standard protocol software (FUSION (XNS) or TCP/IP) for exchanging information throughout the network.

The architecture of the MVME330 consists of an MC68000 MPU performing supervisory functions over a LANCE VLSI Ethernet Controller, a closely coupled RAM, ROM for protocol processing code, and bus interface to a host system.

Communication between the bus and the MVME330 is handled by by the Bus Interface Protocol (BIP) software. Communication between the MVME330 and the network's physical medium (the coaxial cable) is handled by the MVME330 kernel software and the LANCE devices. The MVME330 software reference manual describes the related protocols that are required for MVME330 operation. All MVME330 protocols are compatible with the Ethernet 2.0 specification, defined by Xerox, Digital, and Intel.

The MVME330 is installed in the user's system chassis card rack and connected to the chassis backplane (VMEbus). Ethernet interfacing is accomplished by cable interconnections between the MVME330 transceiver connector and the associated Ethernet transceiver equipment.

MVME330's are capable of acting as both system bus master and as slave on the system bus. This means that an MVME330 is capable of writing to and reading from bus memory. A periodic interrupt, generated by the hardware, notifies the software that it is time to refresh memory. This interrupt occurs about every two milliseconds, and results in approximately five percent overhead to perform the software refresh.

1.6 MEMORY CONSIDERATIONS

There are memory usage considerations regarding the LANCE device on the MVME330. The first one regards the location of the memory buffers which LANCE accesses. These buffers must reside on the MVME330 as opposed to other system bus memory. LANCE performs eight-word bursts when writing received Ethernet data to memory or reading data from memory to send on the Ethernet. Ethernet data should be staged in the MVME330 memory between the Ethernet transfer and the system bus transfer. Either the MVME330 or the host processor can manage the transfer of the data between MVME330 memory and user memory.

During LANCE activity, each host access to MVME330 memory can potentially incur up to a seven microsecond wait before being granted access to the memory. The LANCE performs transfers in eight-word bursts, (i.e., LANCE acquires the local data bus and holds onto it for the entire eight-word transfer). Therefore, if seven microseconds exceeds a host's bus timeout interval, the host MVME330 driver should be designed so that the mailboxes reside in host memory and so that the MVME330 performs all data transfers between MVME330 memory and system bus memory. Software is designed with mailboxes in MVME330 memory.

1.7 RELATED DOCUMENTATION

The VMEbus Specification Manual, the MVME330 Kernel Software Reference Manual, and the MVME330 Debugger Reference Manual are applicable to the MVME330. The FUSION User's Manual provides additional information on the installation and use of the application software.

CHAPTER 2

HARDWARE PREPARATION AND INSTALLATION INSTRUCTIONS

2.1 INTRODUCTION

This chapter provides hardware preparation and installation instructions for the MVME330.

2.2 UNPACKING INFORMATION

NOTE

If shipping container is damaged upon receipt, request carrier's agent to be present during unpacking and inspection of equipment.

Unpack equipment from shipping container. Refer to packing list and verify that all items are present. Save packing material for storage or reshipping.

CAUTION

AVOID TOUCHING AREAS OF INTEGRATED CIRCUITRY; STATIC DISCHARGE CAN DAMAGE CIRCUITS.

2.3 HARDWARE PREPARATION

Hardware preparation is as follows.

2.3.1 Standard Jumper Configuration

Each MVME330 is shipped with the jumpers properly configured for standard operation. The jumper headers are labeled on the module for ease of use. A list of jumper headers and their duties is given in Table 2-1.

TABLE 2-1. Jumper Headers

JUMPER HEADER	DESCRIPTION
JP-01	Alternate Transceiver Connection
JP-02	PROM STACK Timing Select
JP-03	PROM Control
JP-04	Resource Timeout Select
JP-05	Module Address Select
JP-08	Interrupt Acknowledge Level Select
JP-09	Interrupt Request Priority Level Select
JP-10	Bus Request Priority Level Select

The standard factory header configurations are listed in Table 2-2. Jumper header locations and standard jumper header configurations are shown in Figure 2-1

TABLE 2-2. Standard Jumper Header Configurations

JUMPER HEADER	JUMPER CONNECTION	
JP-01	None	
JP-02	2-7	3-6
JP-03	2-17	7-12
	3-16	8-11
	5-14	9-10
	6-13	
JP-04	2-3	
JP-05	1-14	6-9
	2-13	8-7
	3-12	
JP-08	4-7	
	2-5	
	3-6	
JP-09	4-11	
JP-10	12-13	19-20
	8-17	21-22
	7-18	23-24

If the installation is standard and the jumpers are properly configured, the installer may proceed to the installation instructions.

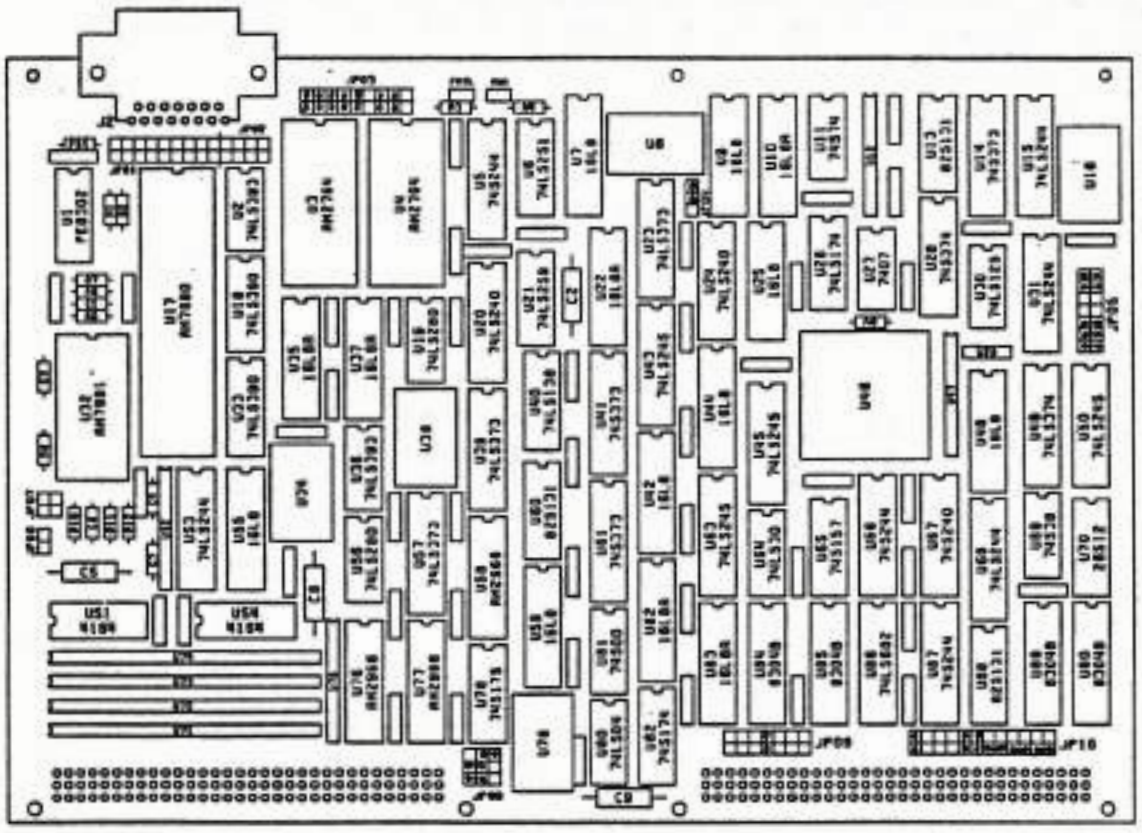


FIGURE 2-1. MVME330 Jumper Header Locations

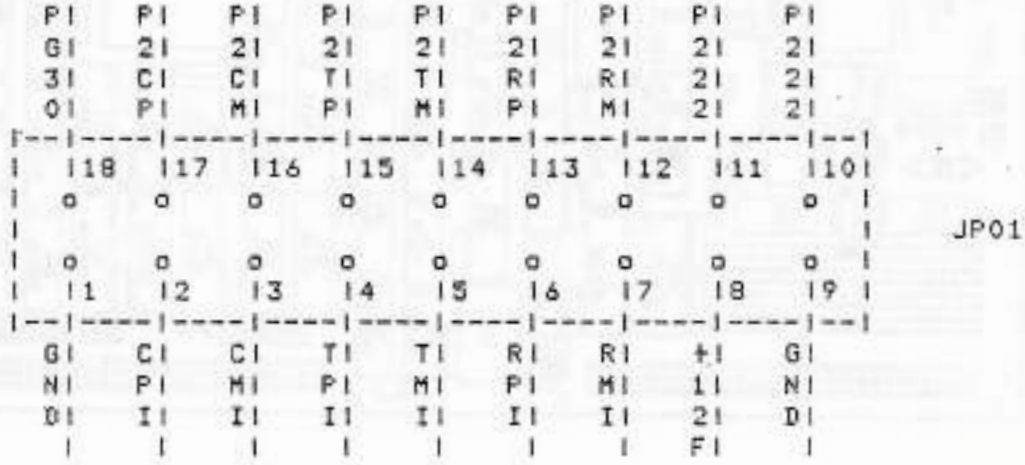
2.3.2 Jumper Header Descriptions

Jumper header descriptions are as follows.

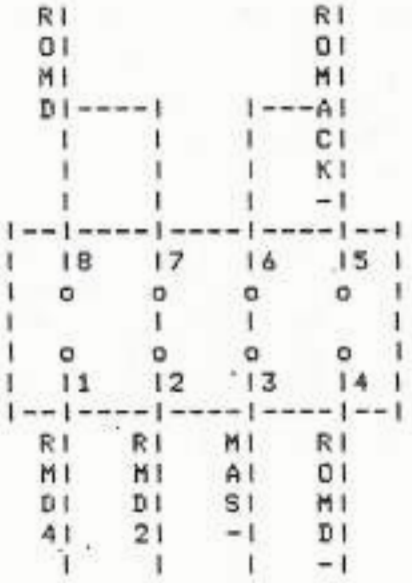
2.3.2.1 Alternate Transceiver Connection (JP-01). If connector P2 of the VMEbus is used to route the signals of the Ethernet transceiver connection, all jumpers on JP01 should be installed. In this case, the transceiver cable will originate from a separate connection instead of the MVME330. The jumpers are installed to connect opposing members of the two rows (i.e., 18-1, 17-2, 16-3, etc.). Header JP01 is illustrated as follows.

NOTE

Do not use J2 if JP01 is used.



2.3.2.2 EPROM DTACK Timing Select (JP02). EPROM DTACK timing is selected according to the speed of the EPROM's used. Factory configuration is for no wait states (JP02 3 to 6, and 7 to 2). Table 2-3 lists JP02 jumpering versus number of wait states. JP02 is illustrated as follows.



JP02

TABLE 2-3. Number of Wait States

NUMBER OF WAIT STATES	JUMPER ON JP02	PROM ACCESS TIME
0	3-6 and 7-2	200 ns maximum
2	4-5 and 7-2	400 ns maximum
4	4-5 and 8-1	

2.3.2.3 EPROM Control (JP03). Header JP03 allows selection of a variety of 24- or 28-pin PROM's. Factory configuration is for the Am2764. Refer to the specification for the PROM to be used for a description of these signals. Table 2-4 provides a description of the jumper outputs (JP03 pin numbers). Header JP03 is illustrated as follows.

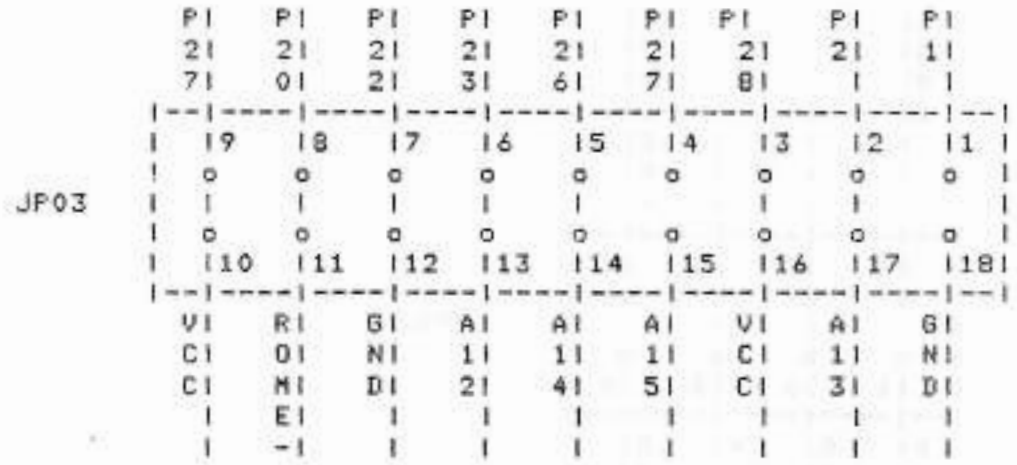
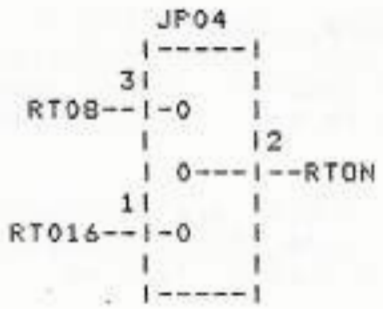


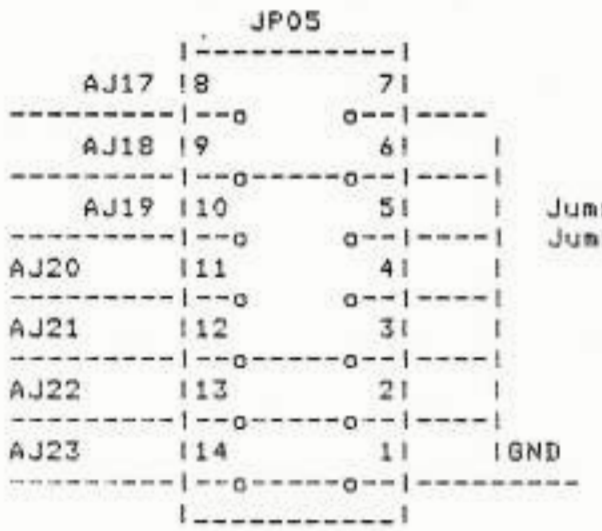
TABLE 2-4. Signal Table

JP03 JUMPER OUTPUT (PIN #)	PROM (27128) CONTROL SIGNAL
1	Vpp
2	A13
3	VCC
4	PGM (Program Enable)
5	A14
6	A11
7	OE (Output Enable)
8	CE (Chip Enable)
9	PGM (Program Enable)

2.3.2.4 Resource Timeout Select (JP04). This header allows selection of either an 8- or 16-microsecond Resource Timeout (RTO). (The module is factory-configured for an 8-microsecond RTO by jumpering pin 3 to pin 2). The resource timeout select header is illustrated as follows.



2.3.2.5 Module Address Select (JP05). User selection of module address. AJ23-17 are compared with VME address bits A23-17 to determine the mode 128K slave response window. Installed jumpers are binary 0 while no jumper is a binary 1. JP05 is illustrated as follows.



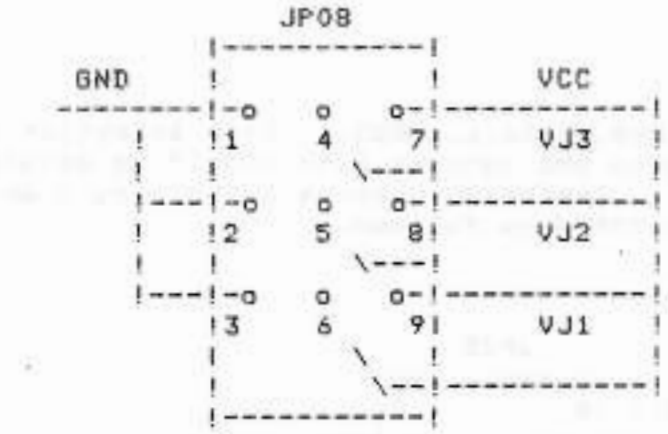
Example:

For DE0000, Jumper pins 3 to 12

For IAXXXX, Jumper pins 6 to 9, 3 to 12, 2 to 13, 1 to 14

2.3.2.6 Interrupt Acknowledge Level Select (JP08). This header in conJunction with header JP09, allows user selection of the VMEbus interrupter level. JP08 is used to select Interrupt Acknowledge (IACK) Levels 1 - 7; the level selected must agree with the interrupt request level configured with JP09.

The three JP08 output lines (VJ1:3) are Jumpered to either GND or +5 Vdc to give 3-bit positive binary code for the IACK level; VJ3 is the most significant bit. JP08 is illustrated as follows.



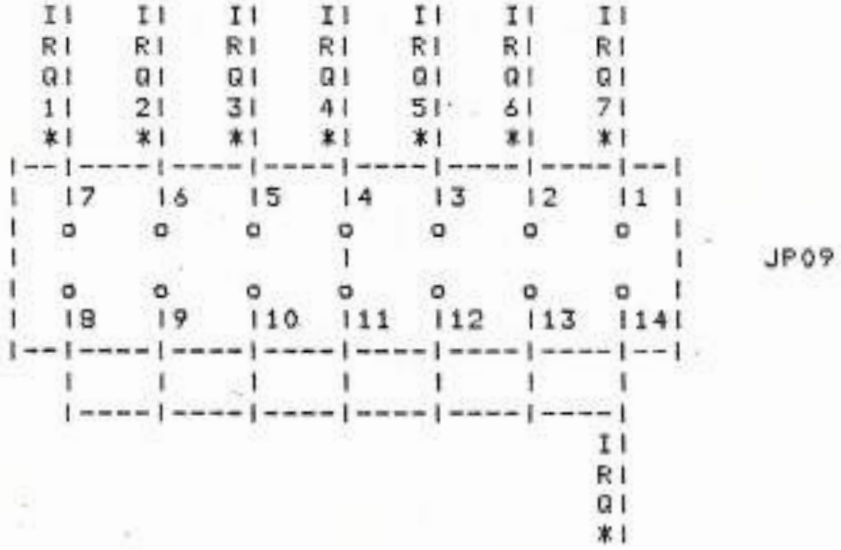
Example:

IACK level 6 is selected by Jumpering to the following pin pairs.

GND		VJn		VCC
1		4	-----	7 -- VJ3
2		5	-----	8 -- VJ2
3	-----	6		9 -- VJ1

2.3.2.7 Interrupt Request Priority Level Select (JP09). This header is used to route the onboard Interrupt Request (IRQ*) signal to one of the seven VMEbus interrupt request lines, IRQ1* - IRQ7*.

The level selected must agree with the interrupt acknowledge priority level configured with JP08. Header JP09 is illustrated as follows.



2.3.2.8 Bus Request Priority Level Select (JP10). This header allows user selection of the bus request priority, which may have the values 0, 1, 2, or 3. When a priority level n is chosen, the following jumpers must be installed.

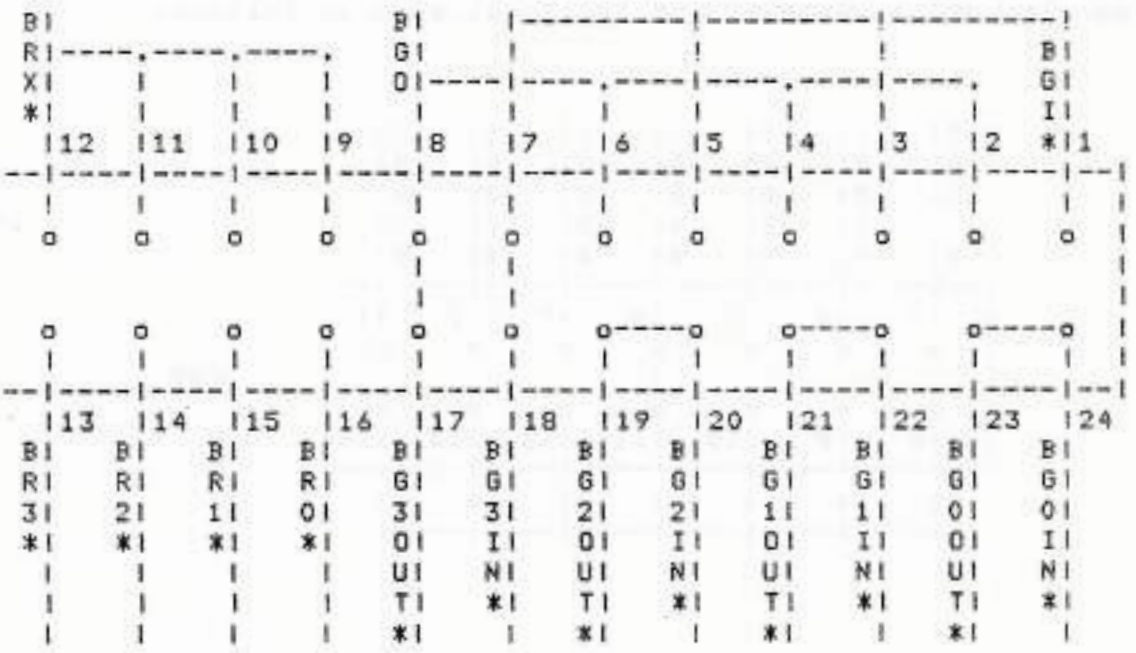
a. The onboard Bus Request (BRX*) signal is tied to the VMEbus Bus Request (BRnv) signal of the chosen level, either BR0*, BR1*, BR2*, or BR3*.

b. The onboard Bus Grant Out (BGO*) signal is tied to the outgoing BGNOUT* signal.

c. The incoming BGnIN* signal is tied to signal BGI*.

d. For the three levels not chosen, the VMEbus BGXIN* signal must be tied to the corresponding BGXOUT* signal.

Header JP10 is illustrated as follows.



Example:

For BR3/BG3

Jumpers on JP10 are: 12-13
8-17
7-18
19-20
21-22
23-24

2.4 INSTALLATION INSTRUCTIONS

2.4.1 Installing The Ethernet Cable

Required items:

- Ethernet cable
- Barrel connectors to join sections of cable
- 2 terminators
- 1 cable ground clamp
- 1 ground wire of appropriate length (AWG rating 2 - 8, insulated)
- Wire strippers
- Electrical tape

Install the Ethernet cable as follows.

a. Plan the network layout. Proper preparation is necessary for a successful installation.

b. Build the cable, using barrel connectors as necessary. If working with sections of cable, each of these sections is terminated by a male terminator. Remove the terminator cap to accept the barrel connector.

NOTE

Total cable length may not exceed 500 meters (1640 feet). Two sections of cable may be joined using a barrel connector.

c. Each free end of the cable must be capped with a terminator.

d. Route the cable per network design.

e. Ground the cable. The cable may only be grounded at one point. One of the terminators is an easily accessible grounding point. Terminate the ground at any existing ground termination point, or at a separately driven ground rod. Clamp a cable ground clamp around the connector chosen as the ground origin. Be careful not to overtighten the screws securing the clamp, or damage to the connector may result.

1. Cut the ground wire to the appropriate length.
2. Strip 1/2-inch of the insulation from each end of the ground wire; insert one end of the wire in the receptacle in the ground clamp.
3. Tighten the screw holding the ground wire in place.
4. Attach the free end of the ground wire to the ground termination point.

f. Wrap all connector junctions and terminators with electrical tape.

2.4.2 Attaching The transceiver

Required items:

- 9/16-inch open-end wrench
- Cable coring tool
- Shield removing tool
- Insulation piercing tool
- Razor knife
- Tap blocks (one per network node)
- Transceivers (one per network node)
- Transceiver cables (one per network node)

NOTE

The following steps must be performed on each network node.

a. The tap block and the transceiver are shipped together in order to protect the transceiver's 'stinger'. Unscrew the tranceiver from the tap block.

NOTE

Handle the tranceiver 'stinger' with care.

b. The Ethernet cable is marked every 2 1/2-meters. The markings indicate the minimum space allowable between transceivers. Clamp the tap block onto the cable at the mark nearest to the node being installed. The threaded hole in the tap block should point toward the node being installed. Use the 9/16-inch open-end wrench to tighten the tap block on the cable.

c. Insert the cable coring tool into the threaded hole in the tap block. Screw the tool in until the threads bottom out, then work it back and forth a few times. This removes the insulator jacket from the cable. (It may also remove some of the braided shielding layer below, this is normal.) Remove the cable coring tool.

d. Insert the shield removing tool into the threaded hole in the tap block. This tool is not threaded. Insert it into the hole and work it back and forth a few times to remove the remaining braided shielding from the cable.

e. Use the razor knife to dig out any remaining shielding from the hole in the tap block.

CAUTION

IF ANY BRAIDED SHIELDING REMAINS IN THE HOLE, THE TRANSCEIVER CAN SHORT THE COAXIAL CABLE AND CRASH THE NETWORK.

f. Insert the inner insulation piercing tool into the threaded hole in the tap block. Screw the tool down until the threads bottom out, then unscrew and remove it. This drills a tiny hole in the cable.

g. If not already done, install an "O" ring onto the transceiver threads.

CAUTION

HANDLE THE STINGER WITH CARE.

h. Insert the transceiver threads into the threaded hole in the tap block, and screw it down until finger tight.

CAUTION

DO NOT OVERTIGHTEN THE TRANSCEIVER.

i. Connect the transceiver cable to the transceiver. Be sure the locking plate is in the closed position. Extend the cable to the node being installed, according to your network plan.

The user may wish to anchor the transceiver and cable to an immobile object for support. If this is done, leave some slack in the cable. This will prevent unintentional pulling of the transceiver cable from pulling (and possibly damaging) the Ethernet cable.

2.4.3 Installation

a. Power down the system.

b. Remove the long covers on the VME backplane.

c. Attach jumpers to the jumper headers on the VME/10 backplane. The MVME330 needs bus request and interrupt continuity. Jumpers should be applied to provide continuity of the bus. Jumpers are not necessary for slots that contain modules.

d. Make sure the system has been upgraded with the proper PROM's to allow it to address the bus.

e. Slide the MVME330 in the proper slot on the chassis. Make sure it is properly seated on the VMEbus.

f. Power up the system.

g. To perform a memory test on the MVME330, Enter MM DE1000;W and enter data. Review the data. If the data is not being stored or refreshed, examine the connection to the bus, emulator and power supply. If everything is stored properly, the MVME330 is functioning correctly.

h. Boot the system

Example:

```
TENBUG 1.2>BO
```

CHAPTER 3

OPERATION

3.1 INTRODUCTION

This section provides an overview of hardware operation, including initialization and diagnostic operation.

3.2 INITIALIZATION PROCEDURE

Upon power up, or system reset, the MVME330 will execute a series of ROM based tests to determine that the module is functioning properly.

Upon successful completion of the tests, the fail LED will be turned off. These tests include a LANCE register and loopback test, an MPU test, a memory test for the 128K dynamic memory, EPROM checksum test, and a control and status register test.

Refer to the FUSION User's Manual for software initialization information.

3.3 OPERATION

For information on the specific protocol in use, refer to the FUSION User's Manual.

CHAPTER 4

FUNCTIONAL DESCRIPTION

4.1 INTRODUCTION

This section provides overall block diagram level and operational descriptions for the MVME330.

4.2 DESCRIPTION

4.2.1 MC68000 MPU

The onboard processor is a 10 MHz MC68000 Microprocessor Unit (MPU). Processor responsibilities include moving commands and data to and from system memory, responding to and generating bus interrupts, executing upper layers of the selected protocol, and running the optional Kernel firmware which controls LANCE, implements layer 2 Ethernet protocol, refreshes memory with software refresh, and supplies timer functions. See Figure 5-2 for MVME330 Memory Map.

4.2.2 Node Address PROM

Every Ethernet station (host and MVME330) has been assigned a worldwide 48-bit address by the Ethernet Address Administration Office. The address is used for station identification on all data transmissions. This address resides in the node address PROM and is initialized by the application protocol software during the negotiation procedure.

4.2.3 EPROM Sockets

The MVME330 has two 28-pin sockets for EPROM. These sockets accept 16K x 8, 8K x 8, and 4K x 8 EPROM of various access timers.

4.2.4 RAM

There are 128K-bytes of dynamic memory with parity. The memory causes no wait states when accessed from the onboard MC68000 MPU. Software refresh occupies 5.0% of the MPU bandwidth.

Parity error is reported as an interrupt. The dynamic memory is accessible from the system bus, from LANCE, and from the MC68000 MPU. The VMEbus has highest access priority, followed by the LANCE and MPU with lowest priority.

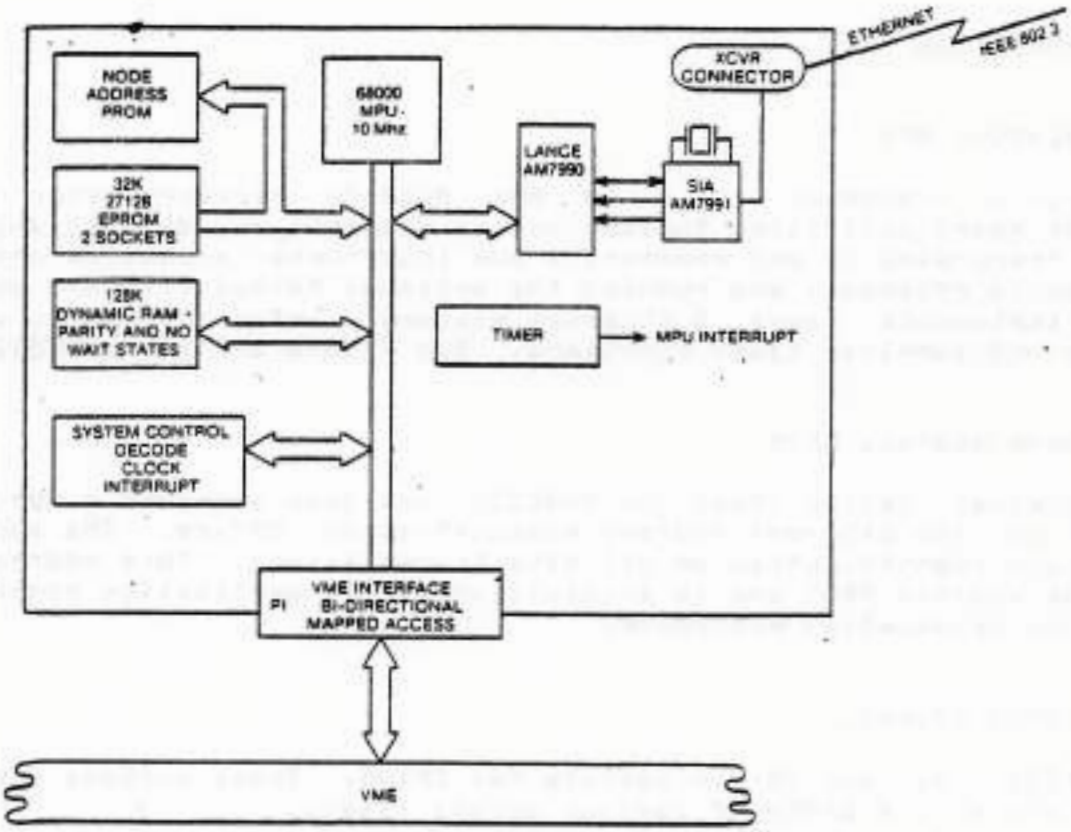


FIGURE 4-1. MVME330 Block Diagram

4.2.5 Local Bus Interface

A single onboard address and data bus is shared between LANCE, RAM and ROM, system bus interface, and the MPU. Bus requests to the MPU cause it to yield this local bus for VMEbus to MVME330 access, and for LANCE Direct Memory Access (DMA) access to onboard memory.

While one resource is using the local bus, all other resources are inactive. For example, a VMEbus access to onboard memory could be held off for more than seven microseconds, while LANCE completed eight DMA cycles. Similarly, the MPU can complete no work during LANCE DMA's. At 120 packets/sec with 1500 bytes per packet, and an average of 800 nsecs per word DMA access, the LANCE DMA will take approximately 7.4% of the local bus bandwidth. With 5% of the bus used for software refresh, and negligible VMEbus access, this leaves approximately 88% of the local bus for the MC68000 MPU. LANCE DMA use of the local bus is sporadic, occupying a large percentage of the bus when in use. During the active reception or transmission of data packets on the 10 megabit Ethernet, LANCE alone uses more than 50% of the local bus bandwidth. There is a limit to the number of back-to-back packets that can be processed by the lower level protocol before upper level processing has time to complete.

The MVME330 buffers as many back-to-back packets as there is receive data buffer space available, without the loss of status or data.

The MVME330 MPU moves data on and off the MVME330. The host processor tells the MVME330 where it wants the data, and the MPU moves it there. If the host processor attempted to get the data, it could conflict with the onboard memory with LANCE DMA, producing inefficiencies.

4.2.6 Clock Timer

A 2 ms timer causes a periodic interrupt to the onboard MC68000 MPU for protocol processing software timers and software refresh functions.

4.2.7 Local Area Network Controller For Ethernet (LANCE) Am7990

LANCE features a MO megabit data rate, a MC68000 MPU-compatible interface, a 16-bit data bus, multiplexed address/data bus, a DMA controller with 24-bit addressing, a sophisticated buffer management structure, a 48-byte SILO data buffer, diagnostic aids, three modes of receive addressing comparisons, CSMA/CD network access algorithm, and extensive error reporting.

The 48 bytes of internal SILO buffer considerably increase the initial response time required for a DMA transfer request. Once the DMA has started, on average of one word must be transferred every 1.6 microseconds to sustain the 10 megabit data rate.

LANCE readily interfaces with the MC68000 MPU. A complication of interfacing with the MC68000 MPU is a multiplexed address data bus, which requires external latches for these lines. LANCE implements the signals for controlling these latches. The onchip DMA controller places LANCE in the high performance category of Ethernet controllers. Available diagnostics include internal or external loopback tests, and CRC logic check and data path validation.

The three modes of network receive address comparison are a full comparison of all 48 destination address bits called the physical mode, a logical mode which puts the 48 bits through a hash filter to determine 1 of 64 logical types called multicast mode, and finally the promiscuous mode where it receives all packets, regardless of address. LANCE implements the full CSMA/CD network access algorithm. Upon detection of a collision, it sends a Jam signal and then follows a backoff algorithm before attempting to transmit again. After 16 successive collisions, it reports an error. Other reported errors include babbling transmitter (transmission of more than 1518 bytes), collision detection circuitry nonfunctional, missed packet due to insufficient buffer space, and a memory timeout (DMA memory access not complete in a reasonable amount of time). These errors cause an interrupt to be generated. Individual packet errors include CRC error, framing error, and SILO overflow or underflow error.

Buffer management features include a defined circular queue of buffer descriptors called descriptor rings. Up to 128 buffers may be queued awaiting processing by LANCE at any one time. Data buffers may be chained to handle long packets in multiple data buffer areas. LANCE controls an 'own' bit for each buffer which signals when the buffer has been filled or emptied and is available for action by the processor.

4.2.8 Serial Interface Adapter (SIA)

The SIA performs the Manchester encoding/decoding necessary for interfacing LANCE to Ethernet. It is compatible with standard Ethernet bus transceivers operating at 10 megabits/sec. The decoder acquires the clock and data within six bit times (600 nsec). It features guaranteed carrier detection and collision detection threshold limits and transient noise rejection. The receiver decodes Manchester data with up to plus or minus 20 nsec clock jitter, which represents 1/5 of a bit time.

4.2.9 Tranceiver Connector

The transceiver connection on the MVME330 is a 15-pin conductor connection (MIL-C-24308) that connects the MVME330 to the transceiver cable. The connector is female, with a slide latch assembly.

4.2.10 Bus Requester

The bus requester, and the controller as bus master follow the VMEbus specifications. The bus request level is Jumper-selectable and the bus request daisy-chain is supported on all levels. As bus master, the VME controller supplies 24-bits of address. The VMEbus requester arbitrates on every transfer. Bus Clear (BCLR*) and Bus Release (BREL*) are not supported. System Reset (SYSRESET*) clears any outstanding bus requests.

4.2.11 Interrupter

The interrupter is compatible with the VMEbus specification. The interrupt level is Jumper-selectable. The interrupt vector is dynamically programmable with any 8-bit vector. The onboard MC68000 MPU sets this vector which causes the interrupt. The interrupter supports the interrupt acknowledge daisy-chain.

4.2.12 Bus To Module Signaling Interrupt

Any host access to the upper 512-bytes of the 128K-byte memory window on the MVME330 will cause an interrupt to the MPU on the MVME330. Access to this window does not constitute an MVME330 memory access as the host will not be forced to wait for data transfer acknowledge. Read will result in no data, write will not alter memory, response is guaranteed to the host and an interrupt is generated to the MVME330 MPU.

4.2.13 Memory Map

The MVME330 subtends 128K-bytes of VME memory space. The module 128K response address is jumper-selectable anywhere in the system bus address space. The DRAM accessable from the VMEbus is identical to the MVME330 DRAM between F01000 and F1FE00. As bus master, the MVME330 may access most of the bus address space (See Memory Map, Figure 5-12). The onboard memory allocation is flexible, however, the MC68000 MPU interrupt vectors must be in RAM. This necessitates a ROM/RAM switch after reset.

4.2.14 Kernel Firmware

The MVME330 Kernel supplies functions which control and monitor the hardware features of the MVME330. The Kernel manages LANCE status registers and descriptor rings, performs software refresh and timer functions, and manages interrupts, allowing protocol software to be written in a high-level language such as Pascal or C. The Kernel allows downloading over Ethernet or directly over the VMEbus from the host, and gathers running statistics on all conditions reported by LANCE.

4.2.15 KNLBUG Debugger

A ROM based debugger is available for the MVME330. It is a full debugger with multiple breakpoints, memory examine, modify, test and move, an assembler, disassembler, download capabilities, and includes software memory refresh. For more information, refer to the MVME330 Kernel User's Manual.

CHAPTER 5

SUPPORT INFORMATION

5.1 INTRODUCTION

This chapter provides connector signal descriptions, parts, parts location diagrams, and schematic diagrams for the MVME330.

5.2 CONNECTOR SIGNAL DESCRIPTIONS

5.2.1 VME Pin Assignments

The MVME330 is inserted in the VMEbus P1 and P2 backplane sockets. The MVME330 connector P1 signals are identified in Table 5-1 by pin number, row number, signal mnemonic, and signal name and description. Table 5-2 identifies connector P2 signals.

5.2.2 Transceiver Cable Pin Assignment

By specification, the MVME330 has a 15-pin female connector (MIL-C-24308 or commercial equivalent) with a slide latch assembly. Transceiver connector pin assignments are given in Table 5-3.

TABLE 5-1. Connector P1 VMEbus Pin Assignments

ROW NUMBER	PIN NUMBER	SIGNAL MNEMONIC	SIGNAL NAME AND DESCRIPTION
A	1-8	DV00-DV07*	DATA bus (bits 0-8) - Bidirectional data lines that provide data transfer to the VMEbus.
A	9,11, 15,17, 19	GND	GROUND
A	10	SYSCLK	SYSTEM CLOCK - A constant 16 MHz input clock signal that is independent of processor speed. This signal is generated by the host processor and used for general timing.
A	12	DS1	DATA STROBE 1 - An active low bidirectional signal that indicates a data transfer will occur on data lines DV08-DV15.
A	13	DS0	DATA STROBE 0 - An active low bidirectional signal that indicates a data transfer will occur on data lines DV00-DV07.
A	14	WRITE	WRITE - A bidirectional signal that defines the cycle type as read or write. A high level indicates a read operation; a low level indicates a write operation.
A	16	DTACK	DATA TRANSFER ACKNOWLEDGE - An active low bidirectional signal that indicates valid data is available on the data bus during a read cycle, or that data has been accepted from the data bus during a write cycle.
A	18	AS	ADDRESS STROBE - An active low bidirectional signal indicates a valid address is on the address bus.
A	20	IACK	INTERRUPT ACKNOWLEDGE - An active low input signal that indicates the bus master is responding to an interrupt request and expects a vector to be placed on the data bus.
A	21	IACKIN	INTERRUPT ACKNOWLEDGE IN - An active low input signal. IACKIN* and IACKOUT* signals form a daisy-chained IACK*. The IACKIN* signal indicates to this module that a vector fetch is in progress.

TABLE 5-1. Connector P1 VMEbus Pin Assignments (cont'd)

ROW NUMBER	PIN NUMBER	SIGNAL MNEMONIC	SIGNAL NAME AND DESCRIPTION
A	22	IACKOUT	INTERRUPT ACKNOWLEDGE OUT - An active low output signal. IACKIN* and IACKOUT* signals form a daisy-chained IACK*. The IACKOUT* signal indicates to the next module that a vector fetch is in progress.
A	23	AM4	ADDRESS MODIFIER (bit 0-3) - Three-state driven line(s) that provide(s) additional information about the address bus, such as size, cycle type, and/or Data Transfer Bus (DTB) master identification. The module responds to data space address modifiers only.
A	24-30	AV07- ----	ADDRESS bus (bits 1-7) - Bidirectional address lines that specify memory addresses.
A	31	-12V	-12 Vdc Power - Used by LANCE devices.
A	32	+5V	+5 Vdc Power - Used by the MVME330.
B	1	BBSY	BUS BUSY - An active low bidirectional signal generated by the current bus master to indicate bus usage.
B	2	NA	Not applicable to the MVME330.
B	3	ACKFAIL	AC FAILURE - Open collector driven signal which indicates that the AC input to the power supply is no longer being provided or that the required input voltage levels are not being met.
B	4, 6, 8, 10,	BG0IN BG1IN BG2IN BG3IN	BUS GRANT (0-3) IN - Active low input signals. Bus grant in and out signals form a daisy-chained bus grant. The bus grant in signal indicates to this module that it may become the next bus master.
B	5, 7, 9, 11,	BG0OUT BG1OUT BG2OUT BG3OUT	BUS GRANT (0-3) OUT - Active low output signals. Bus grant in and out signals form a daisy-chained bus grant. The bus grant out signal indicates to the next module that it may become the next bus master.
B	12-15	BR0-BR3	BUS REQUEST (0-3) - Active low output signals which indicate that a module in the daisy-chain requires access to the bus.

TABLE 5-1. Connector P1 VMEbus Pin Assignments (cont'd)

ROW NUMBER	PIN NUMBER	SIGNAL MNEMONIC	SIGNAL NAME AND DESCRIPTION
B	16-19	AM0-AM3	ADDRESS MODIFIER (bits 0-3) - Three- state driven address bus, such as size, cycle type, and/or DTB master identification. The module responds to data address modifiers only.
B	20,23	GND	GROUND
B	21,22	NA	Not applicable to the MVME330.
B	24-30	IRQ7-IRQ1	INTERRUPT REQUEST (1-7) - Active low output signals that generate a prioritize interrupt. Level 7 is the highest priority.
B	31	+5VSTDBY	
B	32	+5V	+5 Vdc Power - Used by the MVME330.
C	1-8	DV08-DV015	DATA bus (bits 8-15) - Bidirectional data lines that provide data transfer to the VMEbus.
C	9	GND	GROUND
C	10	SYSFAIL	SYSTEM FAIL - An active low bidirectional signal that indicates a module has failed the test program.
C	11	BERR	BUS ERROR - An active low input signal that indicates an unrecoverable error has occured and the bus cycle must be aborted.
C	12	SYSRESET	SYSTEM RESET - An active low input signal that will cause the system to be reset.
C	13	LWORD	LONGWORD - Three-state driven signal to indicate that the current transfer is a 32-bit transfer.
C	14	AM5	ADDRESS MODIFIER (bit 5) - Three-state driven on the address bus, such as size, cycle type, and/or DTB master identification. The module responds to data address modifiers only.

TABLE 5-1. Connector P1 VMEbus Pin Assignments (cont'd)

ROW NUMBER	PIN NUMBER	SIGNAL MNEMONIC	SIGNAL NAME AND DESCRIPTION
C	15-30	AV23-V08	ADDRESS BUS (bits 8-23) - Bidirectional address lines that specify memory addresses.
C	31	+12Vdc	+12 VDC Power - Used by LANCE devices.
C	32	+5Vdc	+5 Vdc Power - Used by the MVME330.

NOTES:
1. All signal directions are with respect to the MVME330.
2. The bus master is the device on the VMEbus that is in control of the transaction at the present time.
3. NA denotes signal pins not applicable to the MVME330.

TABLE 5-2. Connector P2 VMEbus PIN Assignments

ROW NUMBER	PIN NUMBER	SIGNAL MNEMONIC	SIGNAL NAME AND DESCRIPTION
A	All	NA	Not applicable to the MVME330.
B	1,13,32	+5V	+5 Vdc Power - Used by the MVME330.
B	2,12 22,31	GND	GROUND
B	Remaining	NA	Not applicable to the MVME330.
C	1	XRES	EXTERNAL RESET - This signal, when low, causes the MPU to be reset.
C	2	ABOP	This signal, when low, generates a level 7 interrupt, causing the software to abort.
C	Remaining	NA	Not applicable to the MVME330.

NOTES:
1. All signal directions are with respect to the MVME330.
2. The bus master is the device on the VMEbus that is in control of the transaction at the present time.
3. NA denotes signal pins not applicable to the MVME330.

TABLE 5-3. Transceiver Connector Pin Assignments

Pin Number	Signal Name
1	Shield (see note)
2	Collision Presence
3	Receive +
4	Ground
5	Transmit +
6	Power
7	Reserved
8	Reserved
9	Collision Presence -
10	Receive -
11	Reserved
12	Transmit +
13	Power Return
14	Reserved
15	Reserved

NOTE: Shield must be terminated to connector shell as well as pin 1 on both ends, and must not be connected to any other pin. No connections are allowed on the reserved pins.

5.3 PARTS FOR THE MVME330

Components for the MVME330 are shown in Figure 5-1, Parts Locator Diagram. Figure 5-1 reflects the latest issue of hardware at time of printing.

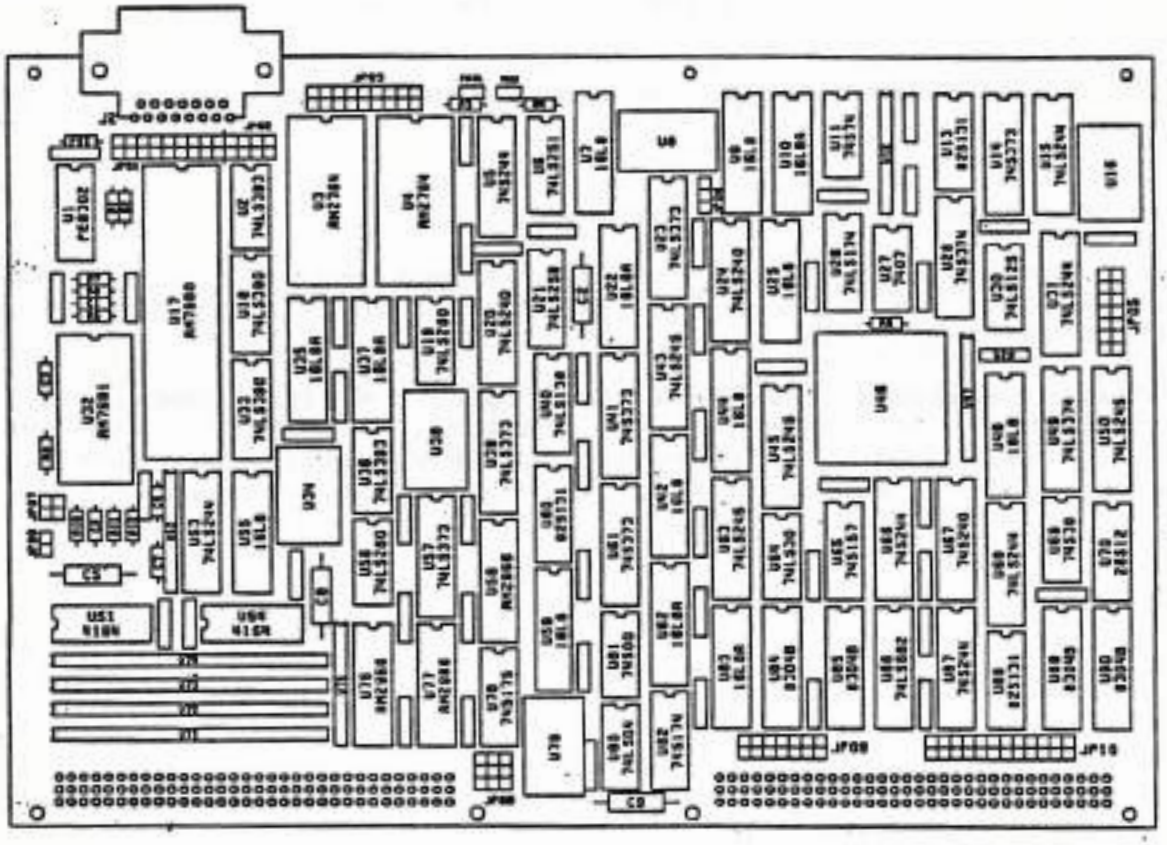


FIGURE 5-1. Parts Location Diagram

5.4 ASSOCIATED CABLES FOR THE MVME330

Associated cables for the MVME330 are as follows.

1 MALE 15 conductor 'D' subminiature type (MIL-C-24308 or equivalent) conductor with locking posts.

1 FEMALE 15 conductor 'D' subminiature type (MIL-C-24308 or equivalent) conductor with a slide latch assembly.

1 standard transceiver cable: Four stranded, shielded, twisted pair conductors plus an overall shield and insulating jacket.

5.5 OUTLINE DRAWING, SCHEMATICS, AND MEMORY MAP

Figure 5-2 shows the overall outline dimensions for the MVME330. The schematic diagrams are given in Figures 5-3 through 5-12 and the system memory map is shown in Figure 5-13.

The schematic diagrams represent the latest design. Occasionally, minor component changes are made at the factory. Therefore, when replacing a component, always use the same value as the defective component even though the schematic diagram may indicate a different value or type.

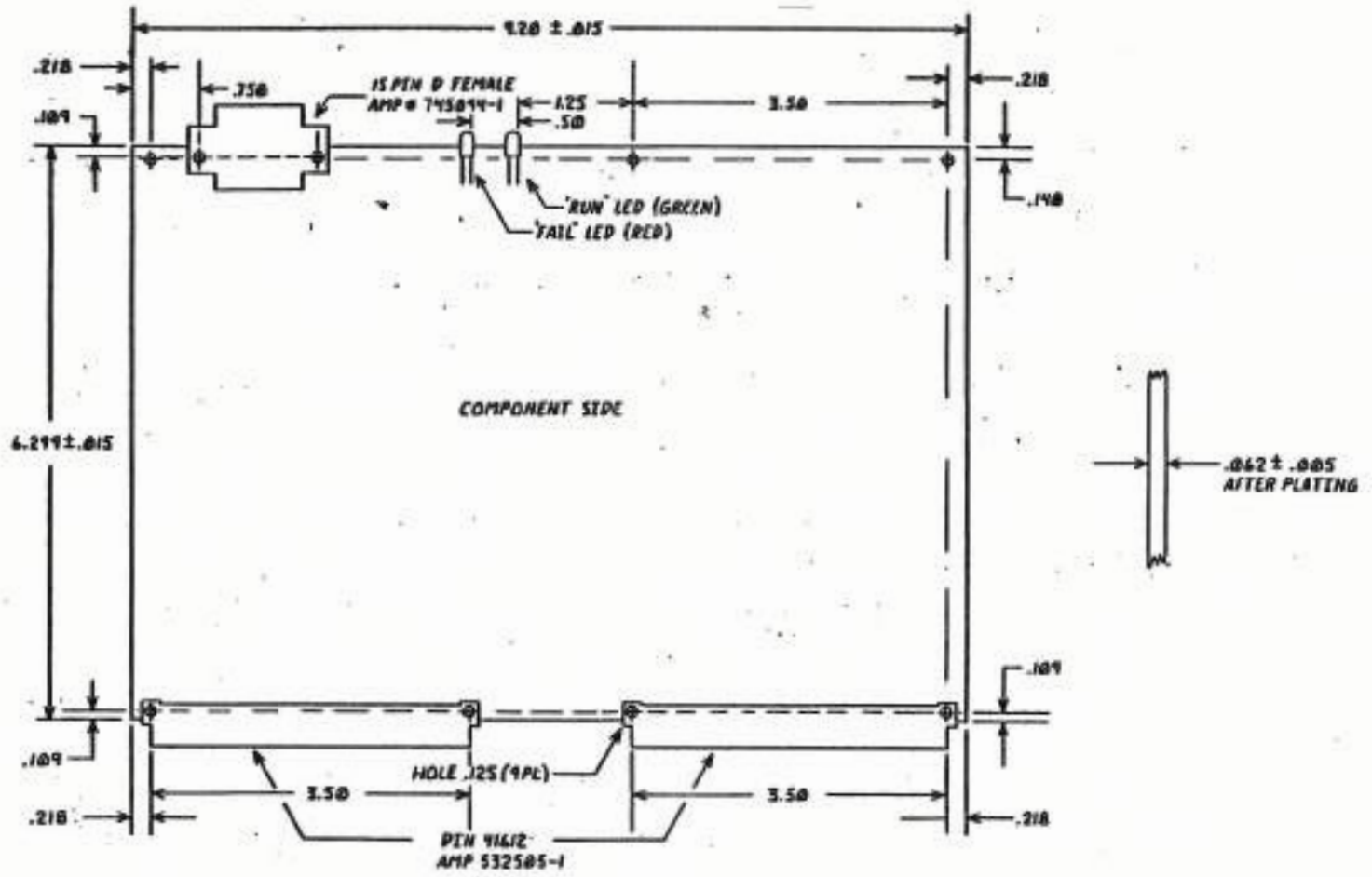


FIGURE 5-2. MVME330 Outline Drawing

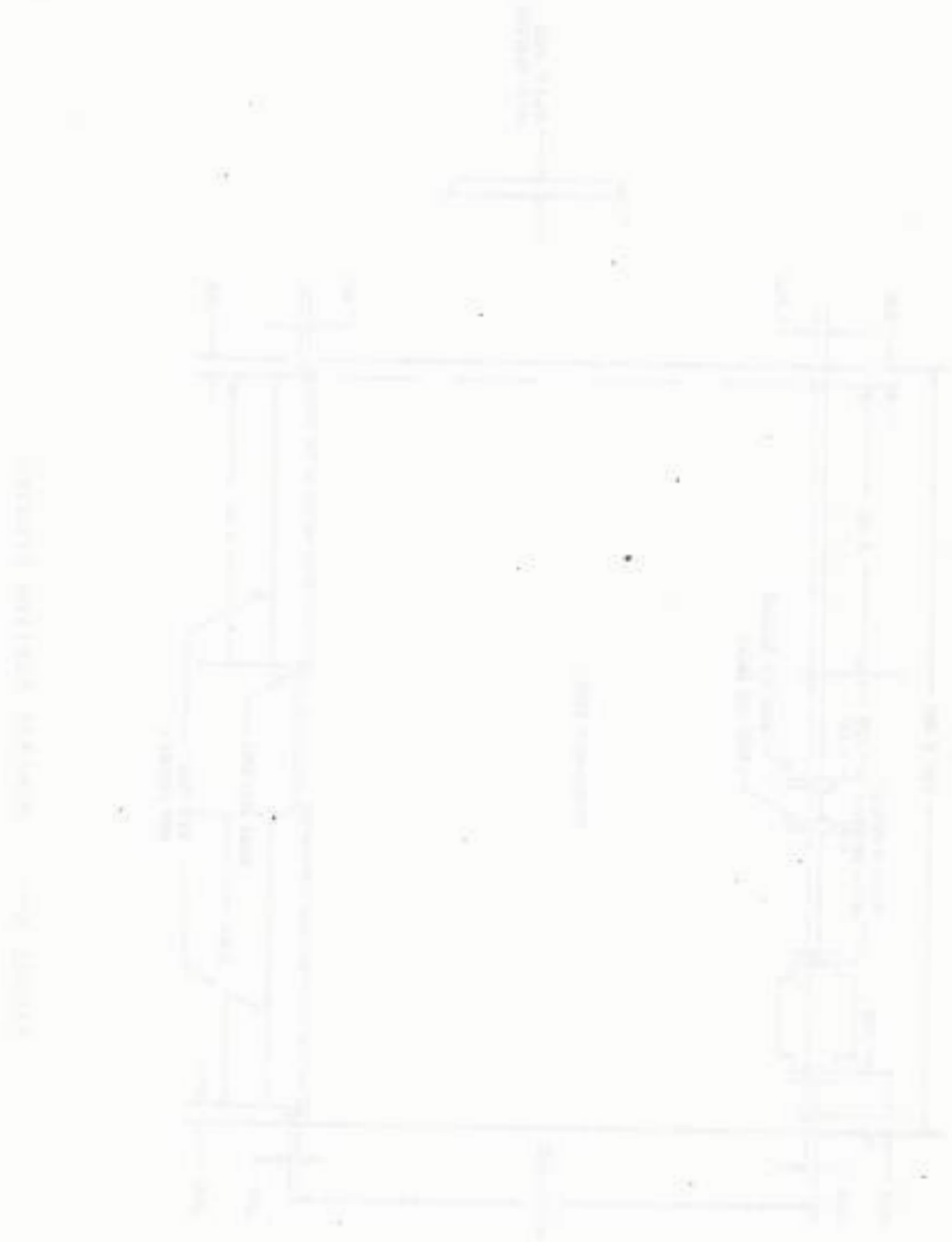


FIGURE 5-3. VME Address Path/Data Path, Schematic Diagram

FIGURE 5-4. VME SLAVE/Interrupter Control, Schematic Diagram

FIGURE 5-5. VME RWD Requester/Master Control, Schematic Diagram

FIGURE 5-6. MPU Address Decode and PROM, Schematic Diagram

FIGURE 5-7. MPU and MPU Bus Control Utilities, Schematic Diagram

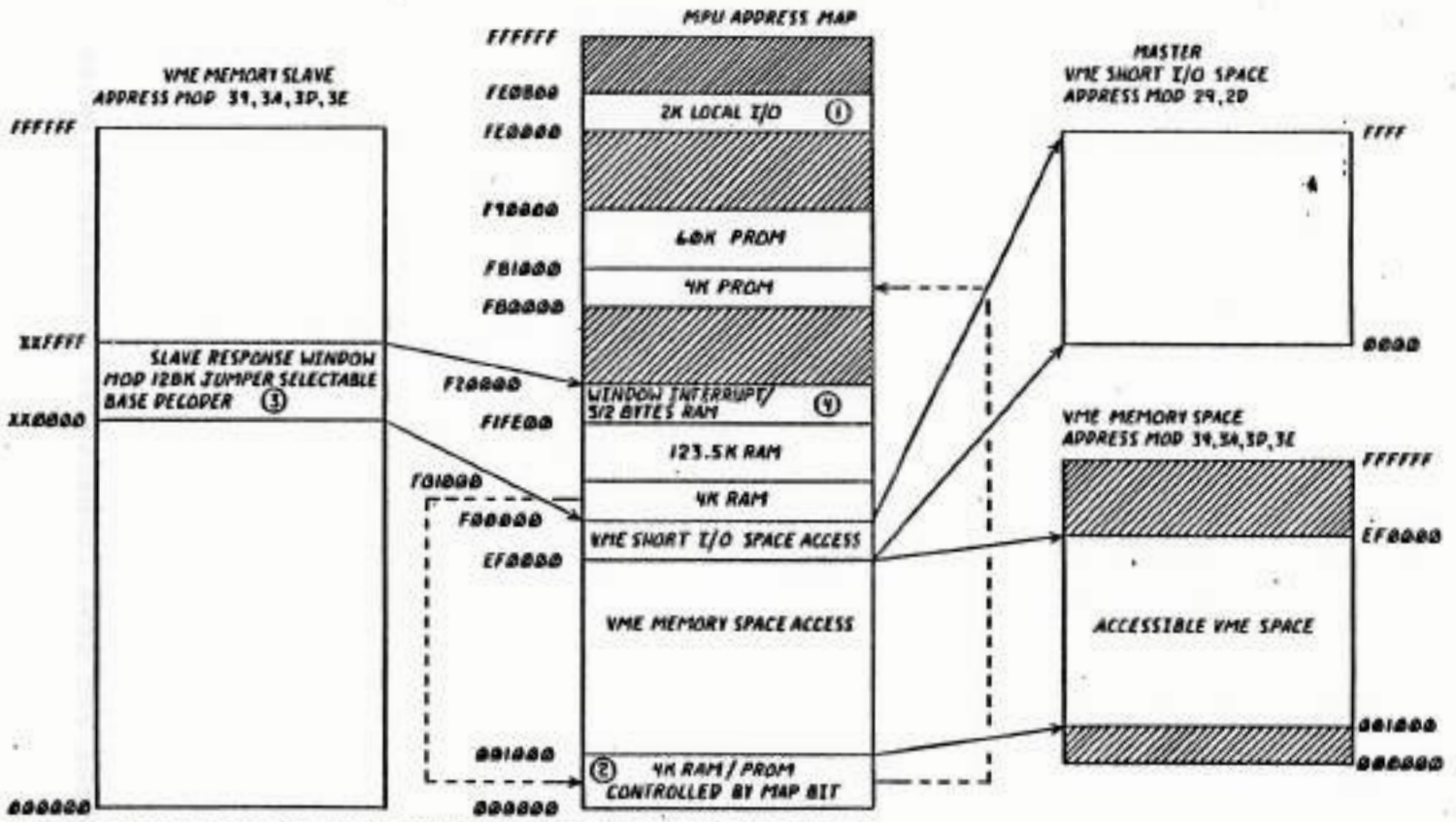
FIGURE 5-8. Local I/O and System Clock, Schematic Diagram

FIGURE 5-9. LANCE and SIA, Schematic Diagram

FIGURE 5-10. DRAM Control, Schematic Diagram

FIGURE 5-11. Prototype DRAM, Schematic Diagram

FIGURE 5-12. VMEbus and Comm Connectors, Schematic Diagram



NOTES:
1. LOCAL I/O SPACE 2MS TIMER, LANCE I/O REGISTERS, MPU CONTROL REGISTERS, ETC.
2. MAP BIT BRINGS 4K OF RAM @ F00000 DOWN TO 000000 AND MOVES 4K OF PROM TO F00000. RESET ON POWER UP.
3. MPU CANNOT ACCESS ITS OWN SLAVE RESPONSE WINDOW IN VME ADDRESS SPACE.
4. VME SLAVE ACCESS TO THIS 512 BYTE SEGMENT CAUSES AN AUTO VECTOR INTERRUPT TO MPU WITHOUT ACCESSING THE INTERNAL MPU BUS. THIS SEGMENT IS THE TOP 512 BYTES OF RAM FOR THE MPU.

FIGURE 5-13. MPU Memory Map

APPENDIX A

BUS INTERFACE PROTOCOL (BIP)

A.1 INTRODUCTION

This appendix defines the protocol used by a processor, or host, to communicate with an MVME330 in order to manage network communications. It does not define the functions performed on either side of the interface before or after receiving the data. It defines a method of passing data within the context of a communication protocol and is the definitive document for writing an I/O driver for the MVME330.

The MVME330 performs several levels of telecommunication protocol processing to offload the burden from the host. The Bus Interface Protocol (BIP) is a host-to-MVME330 software interface which provides an efficient mechanism for controlling the offloaded protocol, without using any protocol-specific features. In this way, a protocol running on the MVME330 may be changed while the operating system I/O calls remain the same. The user specifies the protocol and operations using system I/O calls, and the host I/O driver forwards these requests to the MVME330.

The first protocol planned for the MVME330 will be the Xerox Network System's Sequenced Packet Protocol, including all lower layers. Future protocols to be implemented using the software interface defined herein include ECMA-72 Transport layer and ARPAnet's TCP/IP.

A.1.1 General Description of Communication Protocols and Usage

The Open Systems Interconnection (OSI) seven layer model for communications protocols provides a useful frame of reference for a discussion of protocols. Briefly, starting from the lowest layer, the seven layers are: the physical layer, in this case Ethernet 2.0; the data link layer, defining the control of the transmission and the lowest level addressing formats; the network layer, which provides for end-to-end addressing and multiple network access; the transport layer, which provides reliable sequenced message delivery; the session layer, defining the steps used during a communication session; the presentation layer, which translates, if necessary, data formats so that dissimilar devices can communicate with each other; and finally the application layer, which is basically the function which the user wishes to perform using the lower levels, such as file transfer.

Since the session layer deals closely with the operating system in the sense that it must cause tasks to execute upon request, while the transport layer deals more with the actual transmission and reception of data, this is a good breaking point between those functions executed in the host and those functions performed by the front-end processor. The transport layer performs the functions of reliable transmission and message acknowledgement, and when a transport layer message is not acknowledged within some protocol-specific interval, the message is retransmitted. Thus, not only must outgoing messages be passed down to the network layer for routing, but they must also be saved for possible retransmission until they are acknowledged by the destination or transport layer. Given that the MVME330 has a timer function and the ability to read and write host memory, it is an ideal device to implement the transport layer, as well as the lower levels for Ethernet.

Traditionally, the 'I/O Driver' has been the code which deals directly with a peripheral I/O device. More specifically, it has been the code which interfaces an I/O device to an operating system, presenting commands and data to a peripheral I/O device. This is the case with the MVME330 as well; however, the MVME330 performs operations and possibly transformations on the data, such as appending protocol-specific data headers to outgoing messages before they appear on the media, which in this case is Ethernet. The important point is that the I/O driver doesn't know anything about the content of the data; it is merely concerned with the exchange of the data between the user task in the host and the peripheral device. If the I/O driver can be defined such that the particular protocol in use need not be known, a separation of function ensues, allowing modular software development. This is the case with the MVME330. When using this document to develop protocols, it is recommended that the developer bear in mind this separation of functions.

4.2 ENVIRONMENT AND INTERFACES

The MVME330 software interface enables host processor to control Ethernet communications and protocol processing in an MVME330. The host is the controlling device since it initiates all activity and determines when data transfers take place between itself and the MVME330. Most common operating systems which are capable of handling asynchronous communications should be able to support an MVME330 using this software interface protocol.

Since the MVME330 contains an MC68000 MPU, it is capable of more intelligent activity than simply acting as a front-end protocol Microprocessor Unit (MPU). For example, two MVME330's can be configured on a common system bus as a protocol gateway from one Ethernet to another. Nothing in the original design precludes such a configuration; indeed, this is a configuration planned as an area for future development. Other types of gateways can be imagined also, which may call for the MVME330 to act as an intelligent bus master, controlling various sorts of devices and device controllers on a system bus.

A.2.1 Interface Characteristics

The MVME330 software interface can be characterized as simple yet flexible. The number and location of the I/O mailboxes is negotiated at start up time, allowing the host to specify its preference for their location. Once the I/O mailbox negotiation is complete, the I/O mailboxes are used to exchange data during a transaction. The MVME330 or the host will move whatever data is described through the use of the I/O mailbox. This allows the user to specify the protocol family (such as XNS) and level (e.g., sequenced packet) desired, thus allowing the I/O driver to be protocol independent while allowing the user access to all protocol levels present on the MVME330. Additionally, each side of the interface has the ability to request the other side to interrupt or not, upon completion of the transaction, so interrupt overhead can be finely tuned for a specific host-MVME330 combination.

As a result of the mailbox negotiation phase, two sets of I/O mailboxes are defined: one set for host-to-MVME330 requests and one set for MVME330-to-host requests. This provides a symmetric implementation, allowing for multiple MVME330 and other configurations. This fact, along with the level of indirectness provided by placing addresses of data structures in the mailboxes provides maximum flexibility, configurability, and expandibility.

A.3 FUNCTIONAL SPECIFICATION

The software protocol described in this document binds together a host device driver and a communication protocol program running in an MVME330.

A.3.1 Software Configurations and Modes of Operation

The MVME330 Kernel resides in ROM on the MVME330 module. Application programs, including the IP I/O executive and the protocol application program, may be downloaded into MVME330 memory and started in cooperation with the Kernel. Alternatively, they too may reside in ROM.

Three modes of operation allow for the downloading of an application program, the negotiation of communication parameters, and the exchange of Ethernet packets.

The MVME330 software operates in three major modes:

a. Initialization mode
b. Negotiation mode
c. Operation mode

A.3.2 Initialization Mode

When the MVME330 is reset, for instance upon first application of power, the ROM-based Kernel prepares the MVME330 to receive the application protocol software that will reside in MVME330 RAM. Once the Kernel has given permission, the application protocols will enter the MVME330 RAM in one of these ways:

- Downloaded from the host
- Uploaded from the network
- Transferred from the ROM

In all three cases, the application software will be moved to MVME330 RAM, and it will issue a command to the MVME330 Kernel to start the application running. (Permission to load is not granted if an application has already been loaded.)

A.3.3 Negotiation Mode

When the application code is first started, usually upon being loaded into the MVME330, it prepares to negotiate with the host so as to determine the characteristics desired for operation mode. The host uses a mailbox that is defined by the MVME330 for use during negotiations, and initializes all values in the mailbox, as well as modifying two addresses in the firmware. The application program examines these definitions and responds by assigning a unique host number and filling in any default values. One of the negotiated items is the location (and number) of other mailboxes to be used in the operation mode.

A.3.4 Operation Mode

When both the host and the MVME330 have completed negotiations, normal operations take place by means of the newly negotiated mailboxes. Requests to open, read, and write can be made to the MVME330 application program.

A.4 BIP OPERATIONS AND COMMANDS

A.4.1 Initialization

The MVME330 provides three different methods for initializing the applications program code in the MVME330 memory:

a. Downloading Mode
b. Uploading Mode
c. Resident Mode

A.4.1.1 Downloading_Mode. When the MVME330 is reset, for instance upon first application of power, the ROM-based Kernel gains control and prepares the device for downloading from a host processor. Using the program 'ENPLD' (for MVME330 LOAD), the host acquires permission to download from the MVME330 Kernel, loads the application program code into the MVME330 memory, and instructs the Kernel to start the application running. (Permission to download is not granted if an application has already been loaded.)

A.4.1.2 Uploading_Mode. To be supplied

A.4.1.3 Resident_Mode. To be supplied

A.4.1.4 Initialization_Procedure. In order to begin communication, the MVME330 and the host must agree on a series of address locations that provide a common ground for data sharing. There are five physical bus addresses, residing in the MVME330, which have been arbitrarily assigned. They are described using the format 'XXXXnnnn' which indicates that they are offsets from the starting address of the MVME330. XXXX is defined individually for each operating system and bus structure. The following are the five locations and their values. Figure A-1 illustrates the initialization address blocks.

a. Kernel JUMP command location XXXX1000
b. Firmware start address location XXXX1004
c. Default negotiation mailbox location XXXX1008
d. Default host mailbox location XXXX1018
e. Default MVME330 mailbox location XXXX1038

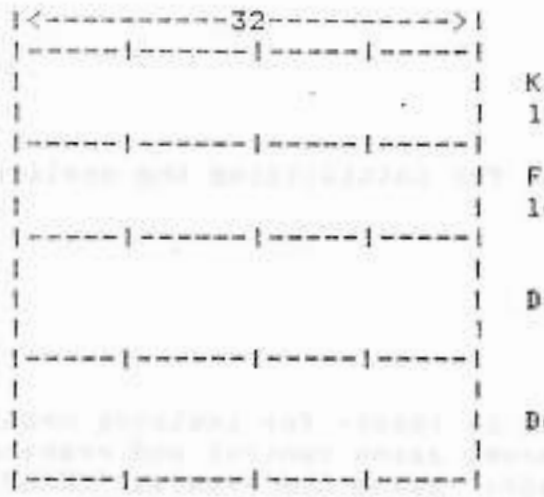


Figure A-1. Initialization Address Blocks

a. Kernel JUMP Command Location XXXX1000

This address defines the location that the MVME330 Kernel is continually monitoring whenever the MVME330 is reset. The host will load the command 'JUMP' into this location when it is time for the MVME330 Kernel to begin its role in the initialization.

This value is originally set to 0 by ENPLD. The host will change the value to 80800000 to begin the execution of the firmware during the negotiation procedure. After the firmware has been started, the MVME330 Kernel will reset this location as necessary for operation.

b. Firmware Start Address Location XXXX1004

The MVME330 RAM address defines the location to which the MVME330 Kernel is to 'JUMP' when it begins its role in the initialization.

This value is originally set to the internal MVME330 address of the main entry point of the protocol firmware by ENPLD. After initialization, this address is used as a Kernel/firmware communciation pointer.

c. Default Negotiation Mailbox Location XXXX100B

This address defines the location that the host and the MVME330 will use for the negotiation communication. Each will place information into and remove information from a mailbox at this point.

During initialization, the physical bus address is set by the device drivers based upon values that were established during the original system configuration.

d. Default MVME330 Mailbox Location XXXX1038

This address defines the location that will be used for all data transfers and communcations from the host to the MVME330, if no other address is specified by the host.

During initialization, this value is set by the MVME330 based upon values that were established during the original system configuration.

A.4.2 NEGOTIATION

The negotiation procedure establishes a firm list of mailboxes for use by the MVME330 and the host. These values are transferred from one to another via the negotiation mailbox. Figure A-2 illustrates the negotiation mailbox.

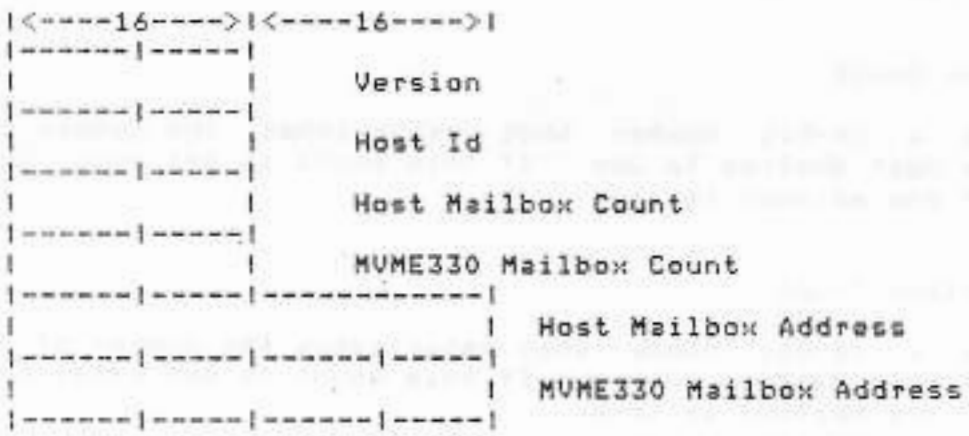


FIGURE A-2. Negotiation Mailbox

A.4.2.1 Version Number

This value defines a 16-bit number that has been set by the host for a two-fold purpose. The version numbers are unique for each release and are stored in the header files. The version number is compared to the value in the OS Kernel to determine if this release of the protocol software can operate on this OS. First, it is used (by a test-and-set protocol) by each host to arbitrate access to the negotiation mailbox. Secondly, it allows the MVME330 to determine the byte-swap nature of the host.

A.4.2.2 Host ID

This value defines a 16-bit number that is set by the MVME330 during the negotiation procedure to uniquely identify each host on the network. The value placed in the mailbox by the MVME330 will be used to identify all ensuing I/O requests by the host.

A.4.2.3 Host Mailbox Count

This value defines a 16-bit number that establishes the number of host mailboxes that the host desires to use. If this count is set equal to zero, the default value of one mailbox is used.

A.4.2.4 MVME330 Mailbox Count

This value defines a 16-bit number that establishes the number of MVME330 mailboxes that the host desires to use. If this count is set equal to zero, the default value of one mailbox is used.

A.4.2.5 Host Mailbox Address

This value is a 32-bit physical bus address that is set by the host to define the starting location of the host mailboxes. If this address is set equal to zero, the default value that the MVME330 specified in the intialization addresses is used.

When multiple host mailboxes are established, their addresses are assumed to be contiguous, starting with the location specified in this field.

A.4.2.6 MVME330 Mailbox Address

This value is a 32-bit physical bus address that is set by the host to define the starting location of the MVME330 mailboxes. If this address is set equal to zero, the default value that the MVME330 specified in the initialization addresses is used. When multiple mailboxes are established, their addresses are assumed to be contiguous, starting with the location specified in this field.

A.4.2.7 Negotiation Procedure

In the negotiation mailbox, the host sets the version number and the host ID to zero, the host and MVME330 mailbox counts to the appropriate values, the host and MVME330 mailbox addresses to the corresponding locations and the firmware start address to the firmware entry point address. The host then sets the Kernel JUMP command to 80800000.

The MVME330 Kernel recognizes this new command and begins the intialization process. The Kernel will read the firmware start address, reset the command in the JUMP field, signaling the host that it has received the command, load the address for the starting subroutine that will be used to intialize the LANCE and build the reference tables that link the Kernel command to the LANCE in the second field, and then jump to the host specified address to begin execution. When the MVME330 has processed all of the values, it places the host ID in field two of the negotiation mailbox. The host, sensing the change in the ID, completes the negotiation procedure by transferring and starting the host ID for identifying future transactions.

A.4.3 DATA TRANSFER

The data transfer procedure is the most used BIP operation. It facilitates the movement of data streams from the host to the MVME330 RAM and vice versa. It performs these transfers through mailboxes that were established during the negotiation procedure. Figure A-3 illustrates the data transfer mailbox.

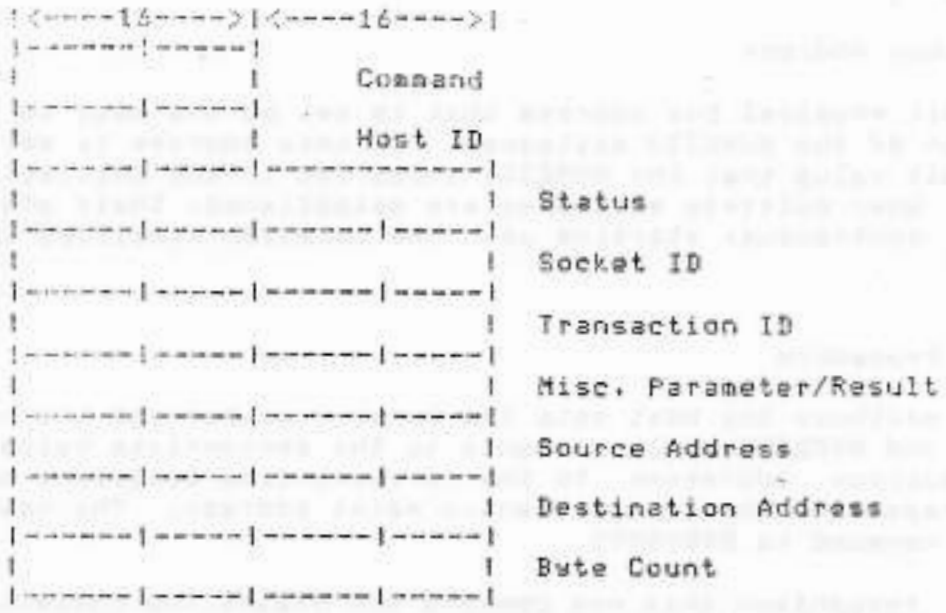


FIGURE A-3. Data Transfer Mailbox

A.4.3.1 **Command.** This value is a 16-bit command code that is loaded alternately by the host and the MVME330. Table A-2 lists the commands that are used by the host to instruct the MVME330. Table A-3 lists the commands that are used by the MVME330 to instruct the host. The value is placed in the command field to initiate the action.

TABLE A-2. Host to MVME330 Command

COMMAND	VALUE
IDLE	0
OPEN	1
CLOSE	2
READ	3
WRITE	4
IOCTL	5
NOTIFY	6
OK	7
YOURS	8
ABORT	9

TABLE A-3. MVME330 to Host Commands

COMMAND	VALUE
IDLE	004
PENDING	814
IN	824
OUT	834
WAKEUP	864

By alternately using these commands, the MVME330 and the host can communicate and transfer data. There interaction of the commands is illustrated in the mailbox state transition diagram, Figure A-4.

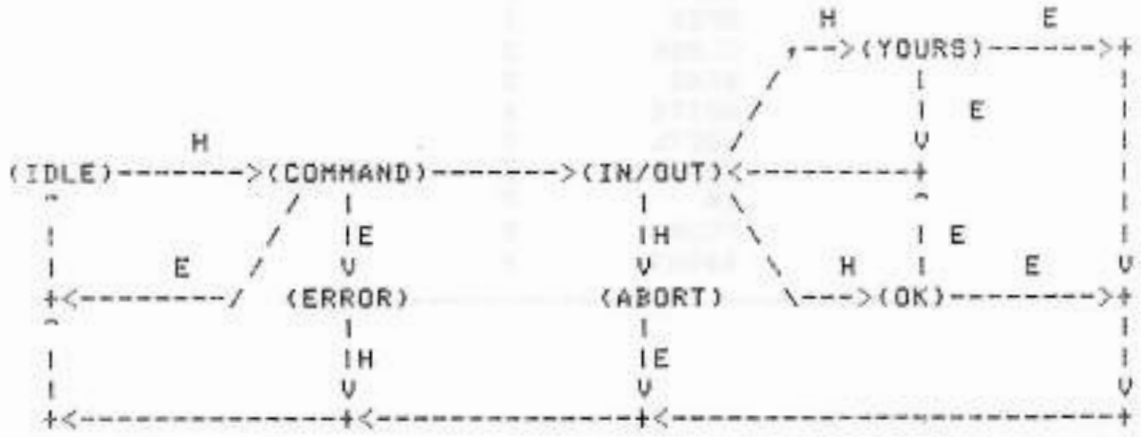


Figure A-4. Mailbox State Transition Diagram

a. IDLE (0 or 00H)

IDLE is the initial and final state. The mailbox state is set to IDLE when the mailbox is created.

b. Action Commands

1. OPEN

OPEN is issued by the host to establish a connection on the network. No data is transferred. Returns a DONE.

2. CLOSE

CLOSE is issued by the host to terminate a connection on the network. No data is transferred. Returns a DONE or a PENDING.

3. READ

READ is issued by the host to notify the MVME330 that it is ready to receive data from the MVME330.

4. WRITE

WRITE is issued by the host to notify the MVME330 that it is ready to send data to the MVME330.

5. IOCTL

IOCTL is issued by the host to notify the MVME330 that network control information must be exchanged.

6. NOTIFY

NOTIFY is issued by the host when it receives a pending command from the MVME330. NOTIFY requests the MVME330 to interrupt the host upon the completion of a particular address conversion that will occur in the future.

7. OK

OK is issued by the host when it has performed a data transfer from the MVME330. The host will also return the actual byte count of the data so the MVME330 can determine whether another transfer is needed to move the entire data.

8. YOURS

YOURS is issued by the host to notify the MVME330 to perform the data transfer on its own. Before issuing the command, the host also translates the addresses of the data to physical bus addresses. This command is only used on those systems where the MVME330 is capable of performing transfers without host assistance.

9. ABORT

ABORT is issued by the host when the MVME330 has requested impossible IN or OUT address. The MVME330 will return the mailbox to an idle state when it receives this command.

10. PENDING

PENDING is issued by the MVME330 when the host has requested a READ or WRITE command and the MVME330 cannot immediately return an IN or OUT answer. Upon receiving a PENDING, the host may issue a NOTIFY or an ABORT.

11. IN

IN is issued by the MVME330 in response to the host request for a WRITE. The IN signifies that the MVME330 has provided an MVME330 source address for the data transfer.

12. WAKEUP

WAKEUP is issued by the MVME330 in response to a previously issued PENDING/NOTIFY sequence. The WAKEUP signifies that the address conversion process is complete and the host may continue its role in the transfer.

A.4.3.2 Host_ID. This value is the 16-bit number that was set by the MVME330 during the negotiation procedure. It is used by the host to identify all I/O requests.

A.4.3.3 Status. This value is a 32-bit completion status that is set by the MVME330. If no error occurs, a file descriptor or data length is returned. The error numbers are those assigned by XNS to signal XNS software. The action on the error codes are taken by the XNS software on the host.

A.4.3.4 Socket_ID. This value is a 32-bit socket identifier that is assigned by the MVME330 in response to an OPEN command by the host. It is used by the host on every transfer to identify the socket that is being used.

A.4.3.5 Transaction_ID. This value is a 32-bit quantity that is supplied by the host to identify each transaction. It is only used by the host.

A.4.3.6 Miscellaneous_Parameter. This value is a 32-bit quantity that is used by the OK, IN or OUT command to return arguments.

A.4.3.7 Source_Address. This value is a 32-bit address that is supplied alternately by the host and the MVME330 to identify the location of data to be moved in a transfer.

A.4.3.8 Destination_Address. This value is a 32-bit address that is supplied alternately by the host and the MVME330 to identify the location to which data will be moved in a transfer.

A.4.3.9 Byte_Count. This value is a 32-bit transfer byte count that is supplied alternately by the host and MVME330 to define the length of the data to be transferred.

A.4.3.10 Data_Transfer_Procedure. The process for communicating a command from the host to the MVME330 using the MVME330 mailboxes that were established during the negotiation procedure is a series of simple steps.

a. Wait for mailbox command to become IDLE.

b. Set the HOST ID.

c. Set the socket ID as returned from a previous OPEN, or zero if this an OPEN.

d. Set the transaction ID to any value to be returned as an identifier of this transaction.

e. Set the miscellaneous parameter as required by the particular command.

f. Set the source or destination address and byte count for the user buffer.

g. Set command

h. Interrupt the MVME330.

i. Wait for the command code to change.

   1. If the command becomes IN, the MVME330 has provided an MVME330 destination address. Either perform the copy and set the command code to IDLE, or convert the host source address to a bus physical address and set the command code to YOURS and wait for the command to become IDLE.

   2. If the command becomes OUT, the MVME330 has provided an MVME330 source address. Either perform the copy and set the command code to IDLE, or convert the host destination address to a bus physical address and set the command code to YOURS and wait for the command to become IDLE.

   3. If the command becomes PENDING, set the command to NOTIFY and the MVME330 will issue an interrupt upon later completion. Wait for the command to become IDLE. Alternatively, set the command to ABORT and return the status to the user or automatically schedule a re-entry later.

4. If the command becomes IDLE, return the status to the user.

The process for communicating a command from the MVME330 to the host using the host mailboxes that were established during the negotiation procedure is also a series of simple steps.

3. If the command code is WAKEUP, use the mailbox contents to complete a previously issued PENDING/NOTIFY request.

The process for communicating a command from the MVME330 to the host using the host mailbox(es) that was (were) established during the negotiation procedure is also a series of simple steps. See Figure A-5.

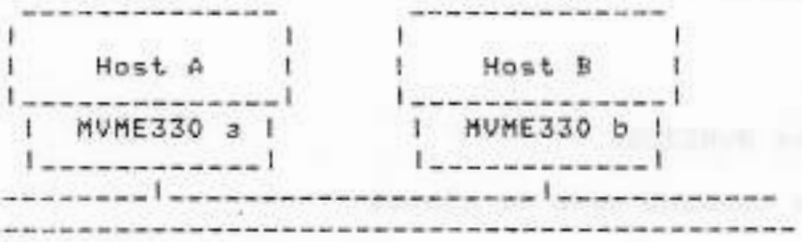


FIGURE A-5. Host Data Transfer

1. Host A wants to transfer data to host B. Host A (A) notifies the MVME330 a (a) to open a socket.

2. A returns the socket number that was opened.

3. A issues an OPEN command on that socket.

4. Host B(B) continually has an outstanding READ on its courier socket (see NOTE).

5. MVME330 b(b) receives the OPEN command on the courier (well-known) socket.

6. b assigns a new socket for communication.

7. b notifies B with a WAKEUP command.

8. B places a READ to the new communication socket.

NOTE

If no outstanding READ exists and the socket is not asynchronous, B will occasionally sample incoming sockets looking for incoming packets (known as polling) If no outstanding READ exists and the socket is asynchronous, b will notify B by interrupting with a "SIGNAL" command.

A.4.4 RUN-TIME PROCEDURES

The first commands executed by FUSION must be xbs-open, xds-open, xms-flip, xpc-open and xsp-open. These commands define the socket connections, when using the run-time library functions.

A.4.5 Restrictions On Use

a. Negotiations must be satisfactorily completed prior to issuing I/O requests.

b. The Host device driver must wait for the completion status upon a CLOSE command to insure that all I/O on the system bus is complete.

A.4.6 Error Handling and Recovery

Error conditions can be easily dealt with, with due respect given to the restrictions above. Synchronous errors may result from invalid socket ID's, asynchronous errors may result from network problems.

A.4.7 Host/BIP Interaction

A.4.7.1 Initialization____Addresses. Paragraph A.4.1 describes the initialization addresses that are used to define the negotiation mailboxes, as well as begin system initialization. The value of these addresses is a function of where the host operating system expects the bus memory to be located. The addresses are initialized through the use of jumpers on the MVME330. For a discussion of how the jumpers are to be set, refer to Chapter 2.

A.4.7.2 Address__Usage. The addresses are used by BIP to perform data transfers from the host to the MVME330 and vice versa. BIP works with two different addresses:

a. Physical bus address

b. MVME330 RAM Memory Access.

See Figure A-6. Data resides at point X in the MVME330 RAM. When the MVME330 MPU needs to access the data at X by using address B. When the host wants to access the data at X, it will use address A. Although both address A (the physical bus address) and address B (the local address) point to the same memory location, they are different values.

Each address is 32 bits. The lower 17 bits are the same for the local address and the bus address. It is the higher 15 bits that vary from system to system. These 15 bits are what the user sets with jumpers at configuration time, in accordance with how the host operating system has configured the bus.

At execution time, it is the BIP responsibility to translate one address to another.

Although room for a 32-bit address has been provided, most systems only use a 24-bit address. For these systems, the lower 17 bits are the same and the higher 7 bits are system dependent.

A.5 SYSTEM DEPENDENT CONSTRAINTS

For the VERSAdos system, the upper level bits are set to 00DE.

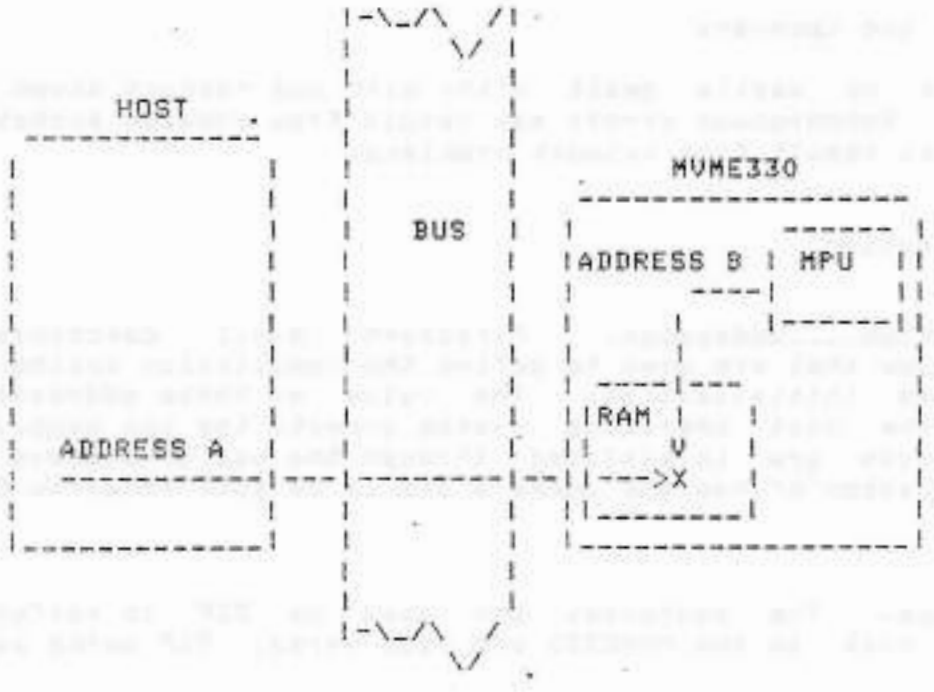


FIGURE A-6. Host/MVME330 Relationship

APPENDIX B

ETHERNET

B.1 INTRODUCTION

The Ethernet local area network provides a communication facility for high-speed data exchange among computers and other digital devices located in a moderate-sized geographic area. Its primary characteristics include:

Physical layer:

Data rate: 10 million bits/sec

Maximum station separation: 2.8 kilometers

Maximum number of stations: 1024

Medium: shielded coaxial cable, base band signaling

Topology: branching non-rooted tree

Date link layer:

Link control procedure: fully distributed peer protocol, with statistical contention resolution (CSMA/CD)

Message protocol: variable size frames, "best-effort" delivery

The Ethernet, like other local area networks, falls in a middle ground between long-distance, low-speed networks that carry data for hundreds or thousands of kilometers, and specialized, very high-speed interconnections that are generally limited to tens of meters. The Ethernet is intended primarily for use in such areas as office automation, distributed data processing, terminal access, and other situations requiring economical connection to a local communication medium carrying bursty traffic at high-peak data rates. Situations demanding resistance to hostile environments, real-time response guarantees, while not specifically excluded, do not constitute the primary environment for which the Ethernet is designed.

The precursor to the Ethernet 2.0 was the "Experimental Ethernet," designed and implemented by Xerox in 1975, and used continually since by thousands of stations. The Ethernet 2.0 builds on that experience, and on the larger base of the combined experience of Digital, Intel, and Xerox in many forms of networking and computer interconnection.

In specifying the Ethernet, the developers specify precise detailed definitions of the lowest two layers of an overall network architecture. It defines what is generally referred to as a link-level facility. It does not specify the higher level protocols needed to provide a complete network architecture. Such higher level protocols would generally include such functions as internetwork communication, error recovery, flow control, security measures (e.g., encryption), and other higher level functions that increase the power of the communication facility and/or tailor it to specific applications. In particular, it should be noted that all error recovery functions have been relegated to higher level protocols, in keeping with the low error rates that characterize local area networks.

One of the main objectives of Ethernet is compatibility. It is intended that every implementation of the Ethernet be able to exchange data with every other implementation. It should be noted that higher level protocols raise their own issues of compatibility over and above those addressed by the Ethernet and other link-level facilities. This does not eliminate the importance of link-level compatibility, however. While the compatibility provided by the Ethernet does not guarantee solutions to higher level compatibility problems, it does provide a context within which such problems can be addressed, by avoiding low-level incompatibilities that would make direct communication impossible.

B.1.1 Functional Model Of The Ethernet Architecture

There are two important ways to view the Ethernet design, corresponding to:

a. Architecture - emphasizing the logical divisions of the system, and how they fit together.

b. Implementation, emphasizing the actual components, and their packaging and interconnection.

Figure B-1 illustrates these two views as they apply to a typical implementation, showing how each view groups the various functions.

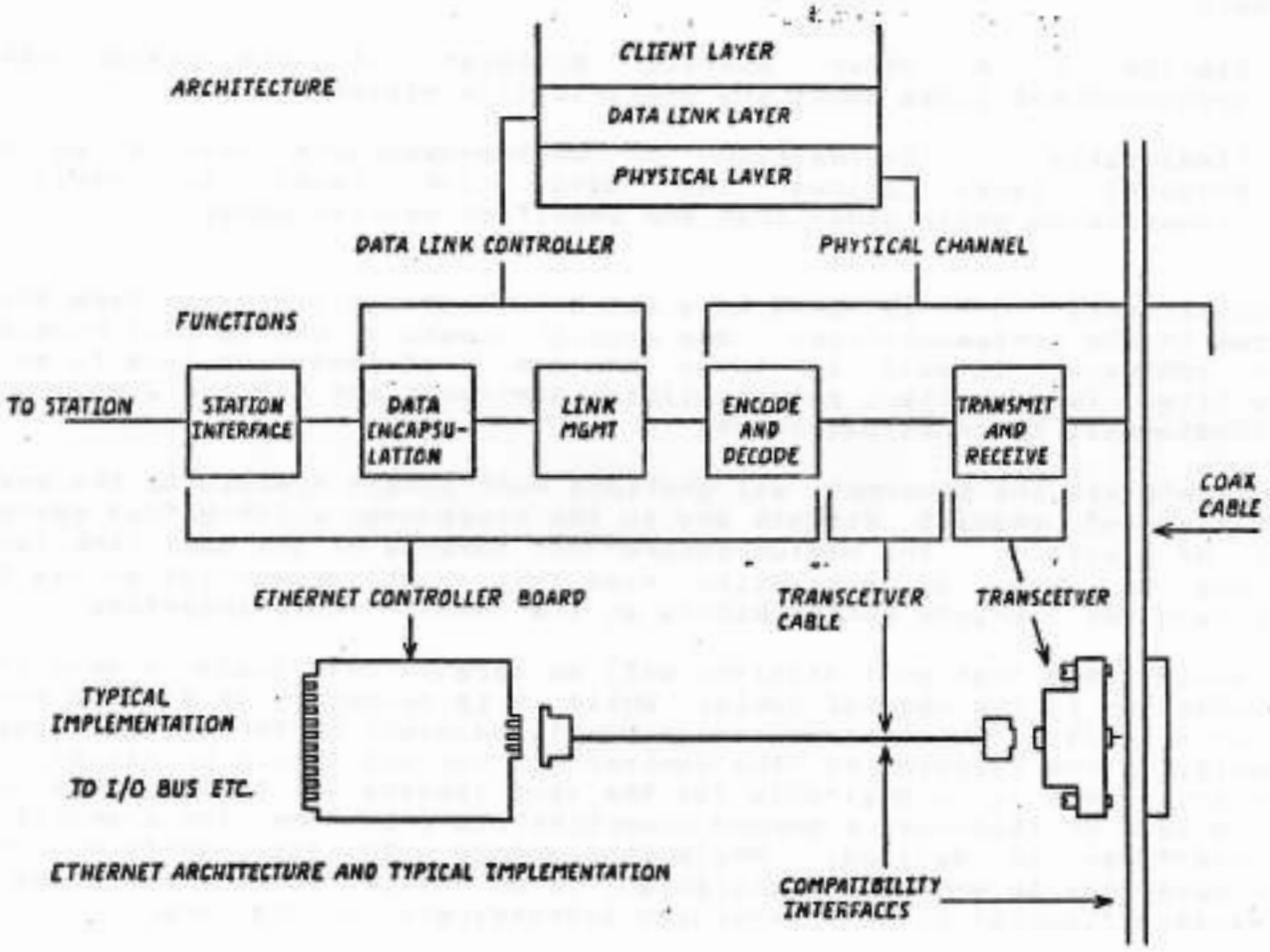


FIGURE B-1. Ethernet Architecture and Typical Implementaion

This introduction is organized along architectural lines, emphasizing the large-scale separation of the Ethernet system into two parts: the data link layer and the physical layer. These layers are intended to correspond closely to the lowest layers of the ISO Model for open systems interconnection. Architectural organization of the specification has two main advantages:

a. Clarity - A clean overall division of the design along architectural lines makes the specification clearer.

b. Flexibility - Segregation of medium-dependent aspects in the Physical Layer allows the Data Link Layer to apply to transmission media other than the specified coaxial cable.

The architectural model is based on a set of interfaces different from those emphasized in the implementations. One crucial aspect of the design, however, must be addressed largely in terms of the implementation interfaces -- compatibility. Two important compatibility interfaces are defined within what is architecturally the physical layer.

To communicate via the Ethernet, all stations must adhere rigidly to the exact specification of coaxial signals and to the procedures which define correct behavior of a station. The medium-independent aspects of the data link layer should not be taken as detracting from this point; communication via the Ethernet requires complete compatibility at the coaxial cable interface.

It is anticipated that most stations will be located some distance away from their connection to the coaxial cable. While it is necessary to place a small amount of circuitry (the transceiver) directly adjacent to the coaxial cable, the majority of the electronics (the controller) can and should be placed with the station. Since it is desirable for the same transceiver to be usable with a wide variety of stations, a second compatibility interface, the transceiver cable interface, is defined. While conformance with this interface is not strictly necessary to ensure communication, it is highly recommended, since it allows maximum flexibility in intermixing transceivers and stations.

B.1.1.1 Layering. The major division in the Ethernet architecture is between the physical layer and the data link layer, corresponding to the lowest two levels in the ISO model. The higher levels of the overall network architecture, which use the data link layer, will be collectively referred to as the "client layer" since the identity and function of higher level facilities are outside the scope of this specification. The one exception is a distinguished client of the data link layer called the network management system, for which the specification specifies a special interface and a configuration testing protocol. The intent is that the Ethernet physical and data link layers support the higher layers of the ISO model (network layer, transport layer).

The basic structure of the layered architecture is shown in Figure B-2.

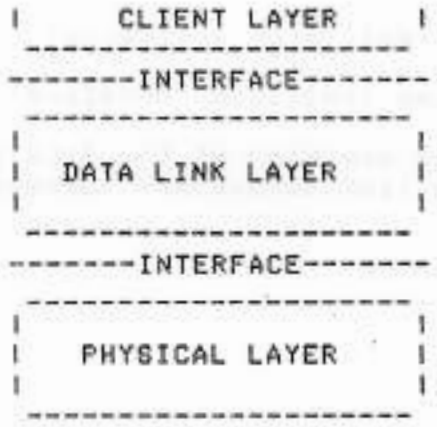


FIGURE B-2. Architectural Layering

B.1.1.2 Data Link Layer. The data link layer defines a medium-independent link level communication facility, built on the medium-dependent physical channel provided by the physical layer. It is applicable to a general class of local area broadcast media suitable for use with the channel access discipline known as carrier-sense multiple-access with collision-detection (CSMA-CD). Compatibility with noncontention media (e.g., switched lines, token-passing rings) is not addressed in this specification.

The data link layer described here is intended to be as similar as possible to that described in the ISO model. In a broadcast network like the Ethernet, the notion of a data link between two network entities does not correspond directly to a distinct physical connection. Nevertheless, the two main functions generally associated with a data link control procedure are present.

a. Data encapsulation/decapsulation

   . framing (frame boundary delimitation)

   . addressing (handling of source and destination addresses)

   . error detection (detection of physical channel transmission errors)

b. Link management

   . channel allocation (collision avoidance)

   . contention resolution (collision handling)

This split is reflected in the division of the data link layer into the data encapsulation sublayer and the link management sublayer (see Figure B-3).

LAYER FUNCTIONS

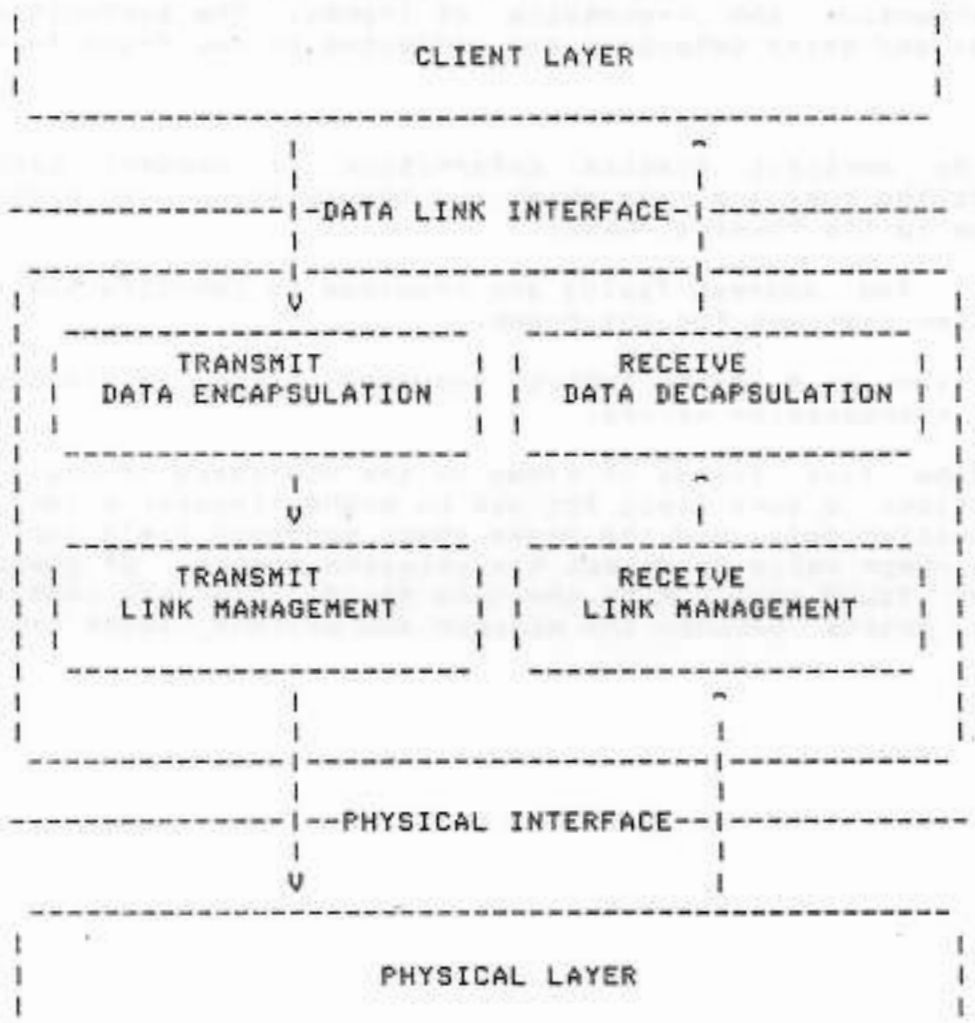


FIGURE B-3. Data Link Layer Function

B.1.1.3 Frame_Format. The data encapsulation function of the data link layer comprises the construction and processing of frames. The subfunctions of framing, addressing, and error detection are reflected in the frame format as follows:

a. Framing - No explicit framing information is needed, since the necessary framing cues (carrier sense and transmitting) are present in the interface to the Physical Layer.

b. Addressing - Two address fields are provided to identify the source and destination stations for the frame.

c. Error Detection - A frame check sequence field is provided for detection of transmission errors.

Figure B-4 shows the five fields of frame -- the addresses of the frame's source and destination, a type field for use by higher layers, a data field containing the transmitted data, and the frame check sequence field containing a cyclic redundancy check value to detect transmission errors. Of these five fields, all are of fixed size except the data field, which may contain any integral number of octets between the minimum and maximum values specified below.

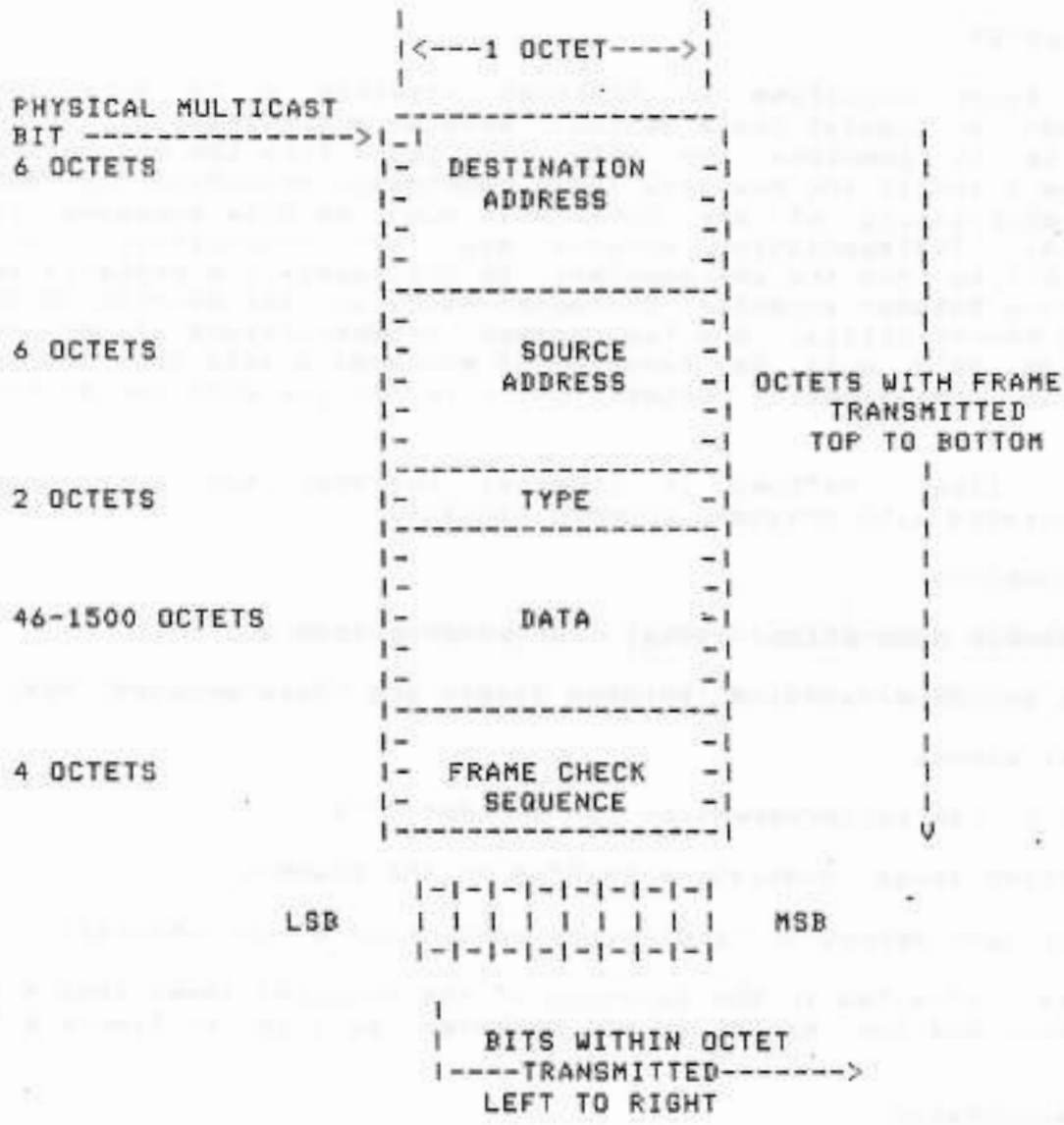


FIGURE B-4. Fields of Frame

The octets of a frame are transmitted from top to bottom, and the bits of each octet are transmitted from left to right (see Figure B-4).

In terms of the ISO model, the Ethernet data link layer provides a multi-endpoint connection between higher-layer entities wishing to communicate. The connection provided is called a data link, and is implemented between two or more data link layer entities called data link controllers via a physical layer connection called the physical channel.

B.2 PHYSICAL LAYER

The physical layer specified in Ethernet provides a 10 Mbps physical channel through a coaxial cable medium. Because one purpose of the layered architecture is to insulate the data link layer from the medium-specific aspects of the channel, the physical layer completely specifies the essential physical characteristics of the Ethernet, such as data encoding, timing, voltage levels. Implementation details are left unspecified, to retain maximum flexibility for the implementer. In all cases, the criteria applied in distinguishing between essential characteristics and implementation details is guaranteed compatibility. Any two correct implementations of the physical layer specified here will be capable of exchanging data over the coaxial cable, enabling communication between their respective stations at the data link layer.

The physical layer defined in Ethernet performs two main functions generally associated with physical channel control:

a. Data encoding

- preamble generation/removal (for synchronization)
- bit encoding/decoding (between binary and phase-encoded form)

b. Channel access

- bit transmission/reception (of encoded data)
- carrier sense (indicating traffic on the channel)
- collision detection (indicating contention on the channel)

This split is reflected in the division of the physical layer into the data encoding sublayer and the channel access sublayer, as shown in Figure B-5.

B.3 NETWORK MANAGEMENT

The Ethernet data link layer provides an interface to a network management system. A network management system provides services which are necessary for network planning, operations, and maintenance.

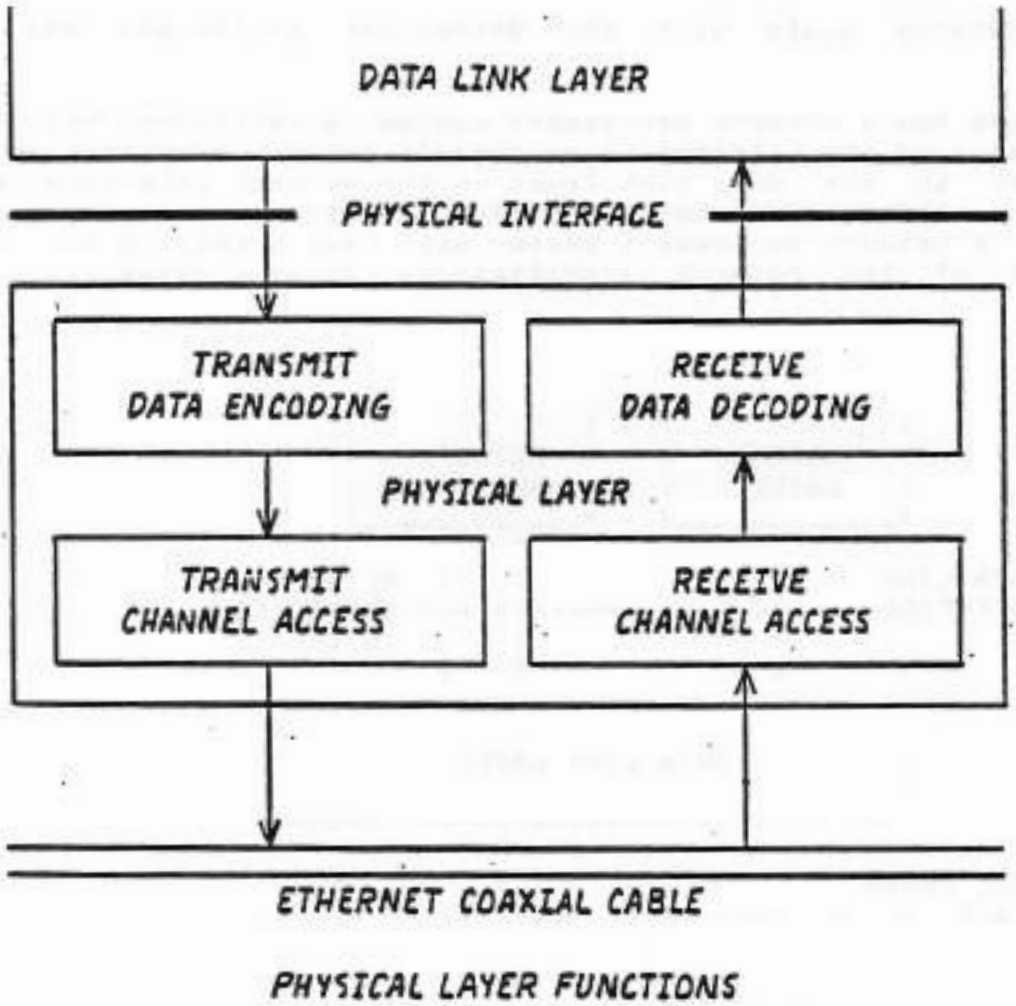


FIGURE B-5. Physical Layer Functions

Network planning gathers usage information to help the users determine when and how to expand the network.

Network operations deal with normal, day-to-day functions, such as initialization, monitoring, and performance optimization.

Network maintenance deals with the detection, isolation, and repair of faults.

Figure B-6 shows how a network management system is positioned with respect to the other layers of the Ethernet in an overall network architecture. It uses two interfaces to the data link layer -- the general interface used by all other clients, and a dedicated one called the network management interface. In addition, a network management system will have access to each one of the higher layers of the network architecture, in many cases via a dedicated interface.

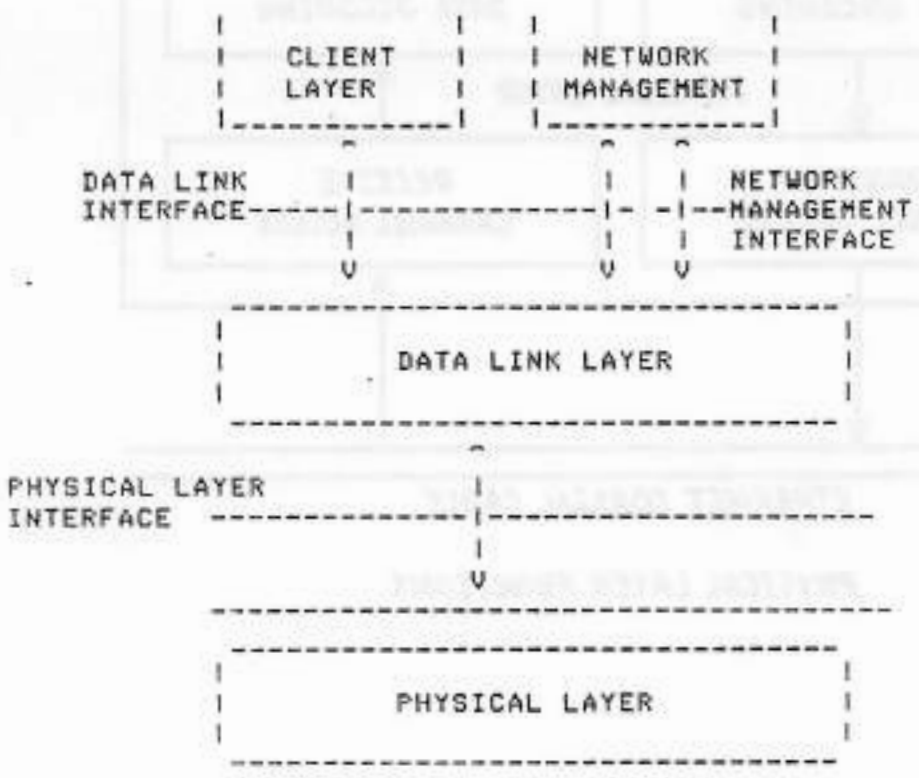


Figure B-6. Architecture Including Network Management Interface

a. Configuration control

- initiating, suspending, and resuming data link operation
- setting the station's physical address
- setting the station's addressing modes (normal, promiscuous, multicast)

b. Observation of

- the values of certain data link state variables and parameters
- activity data (frames sent and received, bad frames, collisions)
- error conditions (operation of carrier sense and collision detect)

Using the general client-to-data link interface and a Configuration testing protocol, network management can ascertain the operability of other stations on the network.

APPENDIX C

PROGRAMMING GUIDE

C.1 INTRODUCTION

If the user is going to use the MVME330 module for a non-standard implemetation, the MVME330 can be programmed through it specific control/status registers. If it is a standard implementation, the user can ignore this section.

The MVME330 local I/O space contains the control and status for MVME330 specific resources as well as access to the LANCE registers and the network parameters PROMS.

C.2 MVME330 CONTROL/STATUS REGISTERS

The MVME330 has 11 control/status registers. Details for usage and purpose of these registers appear in the following text.

NOTE

XX is the module base address, in this instance FE.

C.2.1 Vector Register (VECTR, XX0081)

This register, when written with the desired VME interrupt vector, causes an interrupt to the host at that vector. When read, this register returns the most significant 8-bits of the selected slave response base address (A23-A16). This value enables MVME330 operational software to know what its slave response address is, and thus discriminate between internal and external addresses. This register is also accessible at odd addresses from XX0083 to XX00FF.

C.2.2 Control Status Register (CSR, XX00A1)

The CSR contains four system control and status indicators. It is also accessible at odd locations from XX00A3 to XX00BF. Bit definitions for the CSR are given in Table C-1.

Table C-1. Control Status Register Bit Definitions

BIT	RW	SIGNAL MNEMONIC	SIGNAL NAMD AND DESCRIPTION
7	R	VIRQ	Vectored Interrupt Request. This bit is set when the VECTR is written and clears when the interrupt transaction on the VMEbus is complete. Another VME interrupt transaction cannot take place until this bit clears.
6	RW	FAIL	FAIL. This bit is set when the MVME330 is reset or if the MPU halts. It is also programmable to the reset state and drives SYSFAIL* on the VMEbus. When FAIL is on, the red LED (FAIL) is on and the green LED (RUN) is off. When FAIL is off the red LED is off and the green LED is on.
5	RW	RESET	System RESET. This bit enables the MVME330 to reset the host and all other VMEbus resources. When set, this bit causes SYSRESET* to be activated but blocks it from resetting onboard resources. RESET must be cleared by the MVME330 program in order to release SYSRESET*.
4	R	TIMER	TIMER. This bit indicates when the 2 millisecond timer has gone off. It is cleared when EXR is written to. The TIMER and all exceptions cause a level 7 autovector interrupt. The EXR must be cleared prior to executing an RTE.
3-0		unused	Undefined

C.2.3 Interrupt Enable Register (IER, XX00C1)

This is a single bit (bit 7) register used to enable all interrupts to the MPU. It is cleared by an MVME330 reset and is programmable by the MPU. IER is also accessible at XX0091. Bits 6-0 are undefined.

C.2.4 Transmit Interrupt (TIR, XX00C3)

This bit (7) causes a level 4 autovectored interrupt request. The level 4 request will persist until the bit is cleared by software or a reset. TIR is also accessible at XX00D3. Bits 6-0 are undefined.

C.2.5 Receive Interrupt (RIR, XX00C5)

This bit (7) causes a level 5 autovectored interrupt request. The level 5 request will persist until the bit is cleared by reset or software. RIR is also accessible at XX00D5. Bits 6-0 are undefined.

C.2.6 Utility Interrupt (UIR, XX00C7)

This bit (7) causes a level 1 autovectored interrupt request. The level 1 request will persist until the bit is cleared by software or a reset. UIR is also accessible at XX00D7. Bits 6-0 are undefined.

C.2.7 Map Register (MAPR, XX00CF)

The MAPR is a one bit control register (bit 7) for control of whether RAM (MAPR = 1) or ROM (MAPR = 0) appears in the first 4K-bytes of Microprocessor Unit (MPU) memory space.

This register can be cleared/set by the program and cleared by reset. Bits 6-0 are undefined. MAPR is also accessible at XX00DF.

C.2.8 Exception Register (EXR, XX00E1)

The EXR contains 4-bits which indicate the various types of exceptions. All EXR bits and the TIMER bit are simultaneously cleared by any write to the EXR. The EXR is also accessible at odd locations between XX00E3 and XX00FF. Bit definitions for the EXR are given in Table C-2.

Table C-2. Exception Register Bit Definitions

BIT	RW	SIGNAL MNEMONIC	SIGNAL NAME AND DESCRIPTION
7-4	-	unused	Undefined
3	RC	ACLO	AC line voltage low. This bit is set when ACFAIL* is active on the VMEbus. It causes a level 7 autovector interrupt request.
2	RC	ABO	Abort. This bit is set when the external software abort switch is closed. It causes a level 7 autovector interrupt request.
1	RC	PER	Parity Error. Indicates a RAM parity error occurred on a read operation. It causes a level 7 autovector interrupt request.
0	RC	RTO	Resource Timeout. This bit indicates that the reason for a bus error condition was resource timeout. A resource timeout occurs if access to an on or offboard resource takes longer than 8 us (16us - Jumperable). It causes a level 7 autovector interrupt request.

C.2.9 Unused Registers

The single bit (bit 7) read/write registers at FE00C9, FE00CB, and FE00CD are not assigned a specific use and may be used for signaling or future enhancements to the MVME330. Bits 6-0 are undefined. These registers may also be accessed at FE00D9, FE00DB, and FE00DD, respectively.

C.2.10 LANCE Data Register (LDR, FE0200)

The LDR is a read/write word-only access port to the LANCE control/status registers CSR0, CSR1, CSR2, and CSR3. The specific meaning of bits and fields in these registers this port are detailed in the AMD Am7990 Specification. This port is also accessible at addresses (FE0200 + 4*n) where 0<n<09H.

C.2.11 LANCE Address Register (LAR, FE0202)

The LAR is used to select which of the four LANCE CSR's is to be accessed via the LANCE Data Register port. Only bits 0 and 1 have significance. Bits 15-3 read 0. The use of this register and CSR3-CSR0 are detailed in the AMD Am7990 specification. This port is also accessible at addresses <FE0202 + 4*n> where 0<n<80x.

C.2.12 Network Parameters PROM (NPP, FE0400)

The Network Parameters PROM is a 512 word X 4 bit PROM accessible between FE0400 and FE07FE. Bits 3-0 of each word have significance while bits 15-4 are undefined. The contents of the NPP should include the station address assigned to the MVME330.

C.3 INTERRUPTS

The MVME330 receives interrupt requests from its internal as well as external resources. Most of the interrupts are autovectored for hardware simplicity. The sources, priority and type of interrupts are described in Table C-3 below.

TABLE C-3. Interrupts

LEVEL	NAME	FUNCTION
7A	TIMER	Real time clock, refresh timeout
7A	ACLO	Power failing
7A	ABO	Software abort button
7A	PER	RAM parity error
7A	RTO	Resource timeout
6A	LANCE	LANCE interrupt

TABLE C-3. Interrupts (cont'd)

LEVEL	NAME	FUNCTION
5A	RINT	Receive interrupt
4A	TINT	Transmit interrupt
3A	BINT	Host to MVME330 interrupt (self-clearing)
1A	UINT	Utility interrupt

NOTES: 1. 'A' denotes Autovector.

2. All of the above interrupts except BINT require the MPU to clear the interrupting condition before RTE. All interrupts are enabled/disabled by writing a I/O to IEN (FE00Ch, Bit 7).