Ethernet Network Interface Module

M/N 57C440A

Instruction Manual J-3696-2

RELIANCE ELECTRIC

The information in this user's manual is subject to change without notice.

DANGER

ONLY QUALIFIED ELECTRICAL PERSONNEL FAMILIAR WITH THE CONSTRUCTION AND OPERATION OF THIS EQUIPMENT AND THE HAZARDS INVOLVED SHOULD INSTALL, ADJUST, OPERATE AND/OR SERVICE THIS EQUIPMENT. READ AND UNDERSTAND THIS MANUAL AND OTHER APPLICABLE MANUALS IN ITS ENTIRETY BEFORE PROCEEDING. FAILURE TO OBSERVE THIS PRECAUTION COULD RESULT IN BODILY INJURY.

CAUTION: This module contains static-sensitive components. Careless handling can cause severe damage. Do not touch the connectors on the back of the module. When not in use, the module should be stored in an anti-static bag. Failure to observe these precautions could result in damage to or destruction of the equipment.

Table of Contents

Appendices

List of Figures

1.0 INTRODUCTION

The products described in this manual are manufactured or distributed by Reliance Electric Industrial Company.

This manual describes the Reliance AutoMax TCP/IP Ethernet networking package. It gives AutoMax™ processors access to TCP/IP Ethernet™ local area networks. The package consists of the AutoMax Ethernet Network Interface (ENI) module and software. Cabling is provided by the user.

The ENI module is a high performance communication processor which provides the physical interface and communication intelligence necessary to connect AutoMax processors to TCP/IP Ethernet local area networks (LANs). The physical interface complies with Ethernet 2.0 and IEEE 802.3 standards. Communication is implemented with the TCP/IP network protocol, which is an internationally-recognized industry standard for computer networking.

The ENI software is incorporated in the AutoMax Programming Executive (Version 2.1 and later). It provides the application task interface to the ENI module. The interface is implemented as a set of functions in AutoMax BASIC that are modeled after the UNIX SOCKETS library.

The ENI module can be used only in racks on which the AutoMax operating system with the Ethernet option has been loaded. See the AutoMax Programming Executive instruction manual (J-3684) for information on loading the AutoMax operating system with the Ethernet option.

This manual describes the ENI module, ENI software, ENI module installation, diagnostics, and troubleshooting instructions. A glossary of terms is provided in Appendix D.

1.1 Overview

The AutoMax TCP/IP Ethernet interface provides a reliable and powerful network interface supporting communication between application tasks residing in remote AutoMax processors and/or host computers. A model of the interface is shown in figure 1.1. The components of the model are described below.

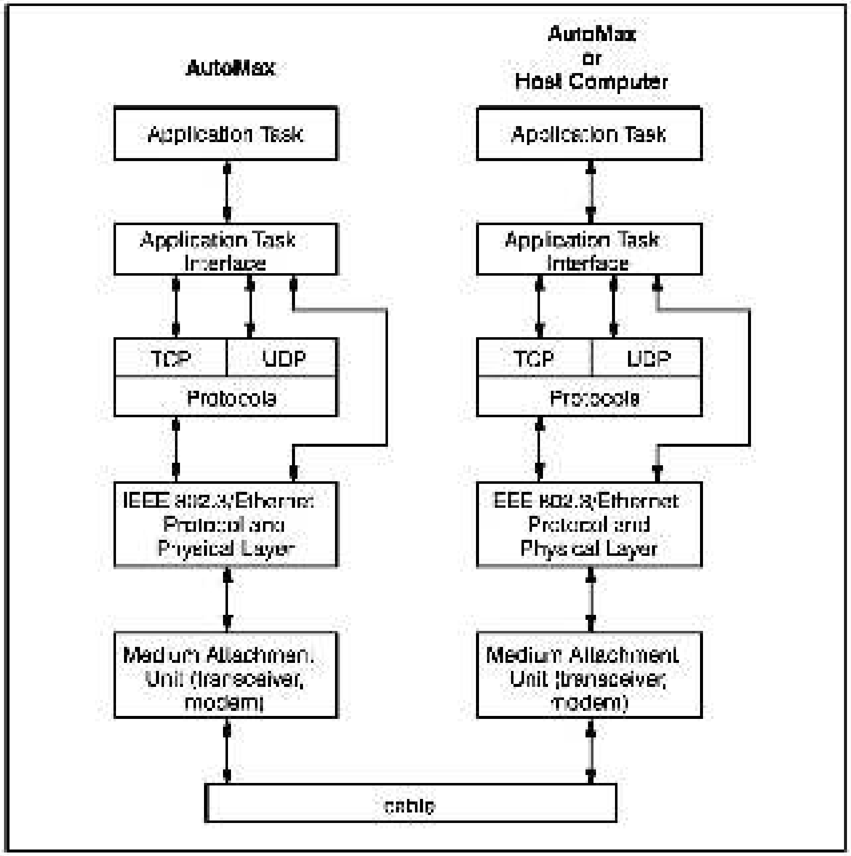


Figure 1.1 - ENI Model

The network interface is modeled as a set of communication protocol layers located one above the other.

1.1.1 Application Task Interface

The **Application Task Interface** (ATI) is a set of function calls in AutoMax BASIC that support the remote task-to-task communication function. The ATI provides a choice of three types of communication services to implement this function. These services are called TCP, UDP, and raw Ethernet. Up to 64 sockets can be created on each ENI, and any one of the three services can be assigned to a socket. Multiple tasks can be handled by the ENI, but the tasks must all reside in the left-most processor in the rack since the ENI communicates only with the left-most processor.

The ENI module transmits and receives at 10 Mbits per second. The actual speed at which data can be moved from one station to another is a function of the protocol used, how fast the AutoMax processor can give a message to the ENI module, and the speed of the host at the other station. Of the three protocols supported, TCP has the most overhead and raw Ethernet has the least overhead.

1.1.2 Communication Protocols

Three communication protocols are available. **Transmission Control Protocol** (TCP) provides a reliable communication channel between two tasks. **User Datagram Protocol** (UDP) is less reliable than TCP but it is faster. **Raw Ethernet** does not use upper layer protocols. It provides the least features, but the fastest data throughput.

1.1.2.1 TCP Protocol

The **Transmission Control Protocol** (TCP) provides a reliable communication channel (also called a virtual circuit) between two tasks, allowing bidirectional data streams. TCP handles making, controlling, and closing virtual connections between remote application tasks. It guarantees that data is ordered correctly, detects missing data and directs its retransmission, and provides flow control to ensure that the AutoMax processor receives no more data than it can process.

The maximum size of a data packet in the TCP protocol is 1460 bytes. When continuously sending a message of this length from one AutoMax rack to another, the average rate at which data is moved is 586 KBits/sec. The CPU utilization of the AutoMax Processor (M/N 57C430A) is 31% on the sending end and 10% on the receiving end.

The TCP protocol provides a means of slowing down the sending Processor if the receiving Processor cannot keep up. Therefore, it is possible to do a SEND% with a length greater than 1460 bytes. The data to be sent is broken down into multiple packets by the ENI module. This requires less overhead in the AutoMax processor per SEND%. If an array of 14600 bytes is sent, the average rate at which data is moved is 1.09 MBits/sec. The CPU utilization of the AutoMax Processor (M/N 57C430A) is 32% on the sending end and 25% on the receiving end.

1.1.2.2 UDP Protocol

The UDP service is based on the **User Datagram Protocol** (UDP). This is a simple Internet Protocol-based datagram protocol whose reliability depends on the network integrity. The UDP service is much less reliable than the TCP and can be used when speed rather than accuracy is paramount.

The maximum size of a packet in the UDP protocol is 1472 data bytes. When continuously sending a message of this length from one AutoMax rack to another, the average rate at which data is moved is 878 KBits/sec. The CPU utilization of the AutoMax Processor (M/N 57C430A) is 54% on the sending end and 21% on the receiving end.

1.1.2.3 Raw Ethernet Protocol

The **Raw Ethernet** service provides communication over an Ethernet or IEEE 802.3 network without any use of the upper layer protocols. It can be used when maximum throughput and minimum reliability are required. When raw Ethernet is used, the ENI can transmit broadcast messages to other stations as well as receive messages that were broadcast from other stations.

The maximum size of a packet in the raw Ethernet protocol is 1500 data bytes. When continuously sending a message of this length from one AutoMax rack to another, the average rate at which data is moved is 1.06 MBits/sec. The CPU utilization of the AutoMax Processor (M/N 57C430A) is 69% on the sending end and 28% on the receiving end.

1.1.3 IEEE 802.3/Ethernet Protocol

The **IEEE 802.3/Ethernet Protocol** controls the access to the communication medium. The protocol supports the media access method called CSMA/CD, which stands for Carrier Sense Multiple Access with Collision Detection.

1.1.4 IEEE 802.3/Ethernet Physical Layer

The **IEEE 802.3/Ethernet Physical Layer** supports a data transmission rate of 10 Mbps. The ENI module supports what is called the Medium Attachment Unit interface specified in the IEEE 802.3/Ethernet standard. (The ANSI/IEEE 802.3 standard is the same as the international standard ISO 8802-3.)

The ENI module will work with various IEEE 802.3 and Ethernet compatible medium attachment units including thick and thin wire Ethernet transceivers and fiber optic and broadband modems. Selection of medium attachment units and the corresponding communication medium is left to the system integrator.

1.2 Additional Information

You must be familiar with all the instruction manuals that describe your system configuration. This may include, but is not limited to, the following:

- J-3618 NORTON EDITOR INSTRUCTION MANUAL
- J-3630 ReSource AutoMax PROGRAMMING EXECUTIVE INSTRUCTION MANUAL
- J-3649 AutoMax CONFIGURATION TASK MANUAL
- J-3650 AutoMax PROCESSOR MODULE INSTRUCTION MANUAL
- J-3670 AutoMax POWER SUPPLY MODULE and RACKS INSTRUCTION MANUAL
- J-3675 AutoMax ENHANCED BASIC LANGUAGE INSTRUCTION MANUAL
- J-3682 ReSource AutoMax SOFTWARE LOADING INSTRUCTIONS VERSION 2.0

- J-3683 ReSource AutoMax UPDATE LOADING INSTRUCTIONS VERSION 2.0
- J-3684 ReSource AutoMax PROGRAMMING EXECUTIVE INSTRUCTION MANUAL VERSION 2.0
- J-3750 ReSource AutoMax PROGRAMMING EXECUTIVE INSTRUCTION MANUAL VERSION 3.0

For a detailed discussion of 4.2/4.3BSD UNIX interprocess communications, refer to the following documents:

- *An Introductory 4.3BSD Interprocess Communication Tutorial*, Stuart Sechrest, Department of Electrical Engineering and Computer Science, University of California, Berkeley.
- *An Advanced 4.3BSD Interprocess Communication Tutorial*, Leffler, Fabry, Joy, and Lapsley, Department of Electrical Engineering and Computer Science, University of California, Berkeley.
- *ANSI/IEEE Std 802.3*, The Institute of Electrical and Electronics Engineers, Inc, New York, 1985.

1.3 Related Hardware and Software

M/N 57C440A contains one Ethernet Network Interface Module. The module is used with the following hardware and software.

The following equipment, purchased separately, can be used with the Ethernet module:

1. M/N 57C385 - DCS5000/AutoMax Power Supply Module
2. M/N 57C430A, 57C431, 57C435 - AutoMax Processor Module
3. Various Model Numbers - ReSource AutoMax Programming Executive Version 2.1 or later

2.0 MECHANICAL/ELECTRICAL DESCRIPTION

The following is a description of the mechanical and electrical characteristics of the Ethernet Network Interface module.

2.1 Mechanical Description

The Ethernet Network Interface (ENI) module is a printed circuit board assembly that plugs into the backplane of the DCS/AutoMax rack. It consists of the printed circuit board, a faceplate, and a protective enclosure. The faceplate contains tabs at the top and bottom to simplify removing the module from the rack. The enclosure has an opening through which a jumper can be set during installation. On the back of the module are two edge connectors that connect to the system backplane. Module dimensions are listed in Appendix A.

The faceplate of the module contains a 15-pin D-type connector that is used to connect the ENI to the transceiver cable. The connector is female, with a slide latch assembly. It conforms to the IEEE Standard 802.3 electrical interface requirements. Refer to section 2.3 and Appendix B for additional information. A green status LED is located just below the connector. Upon power-up or system reset, a series of ROM-based tests are performed to verify proper function of the printed circuit board. When the tests are completed, the LED should light, indicating that the board is operational.

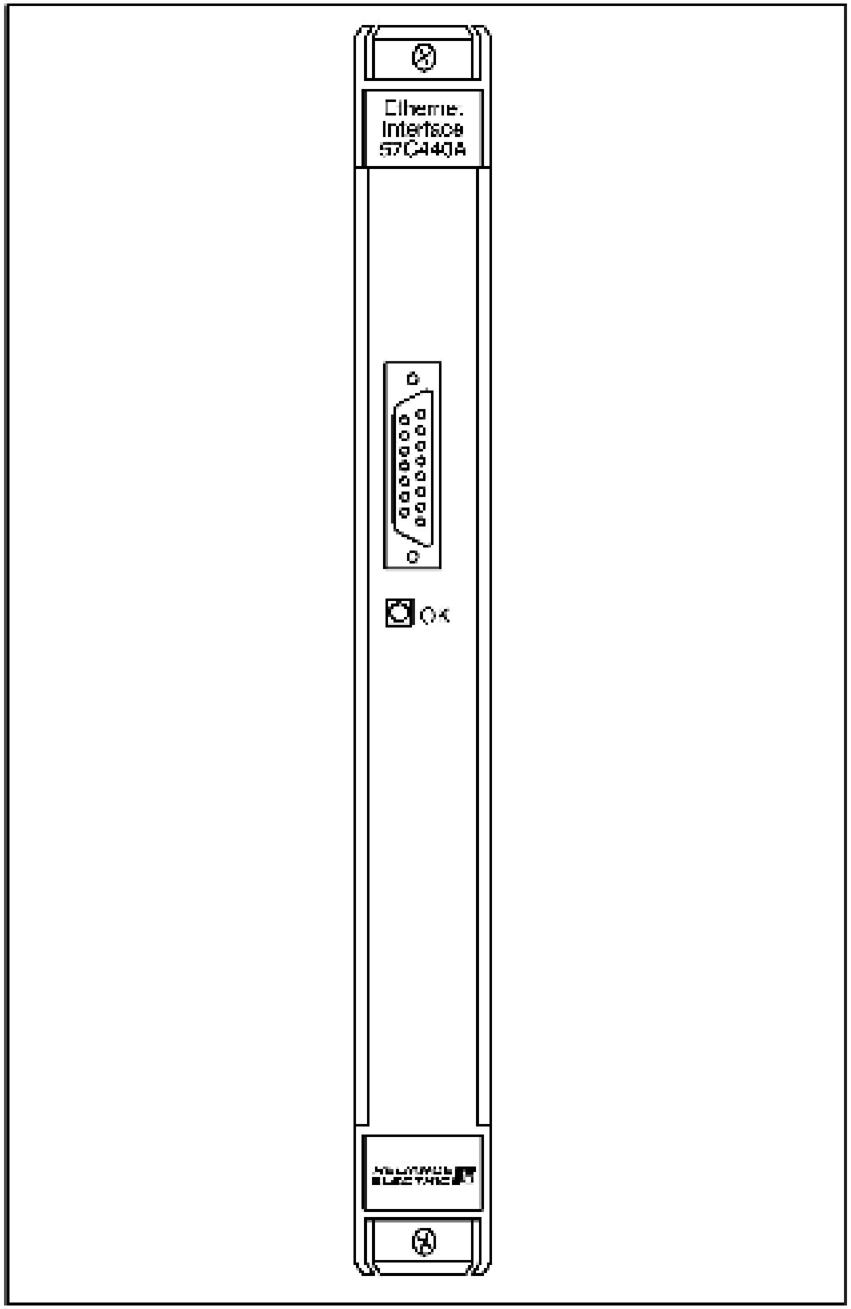


Figure 2.1 - Module Faceplate

2.2 Electrical Description

The ENI module contains a 10 Mhz MC68010 microprocessor that performs supervisory functions using a VLSI local area network controller for Ethernet. Memory consists of a 512 X 4-bit PROM which contains a unique 48 bit Ethernet address, two 64K X 8 EPROMs which control and monitor the hardware features of the ENI, and 512K bytes of Dynamic Random Access Memory (DRAM). 128K bytes of this memory is accessible from Multibus. The module has a Multibus interface and the signaling and timing utilities required to maintain communications. If loss of power occurs, communications will be lost. The ENI must be re-initialized to restore communications.

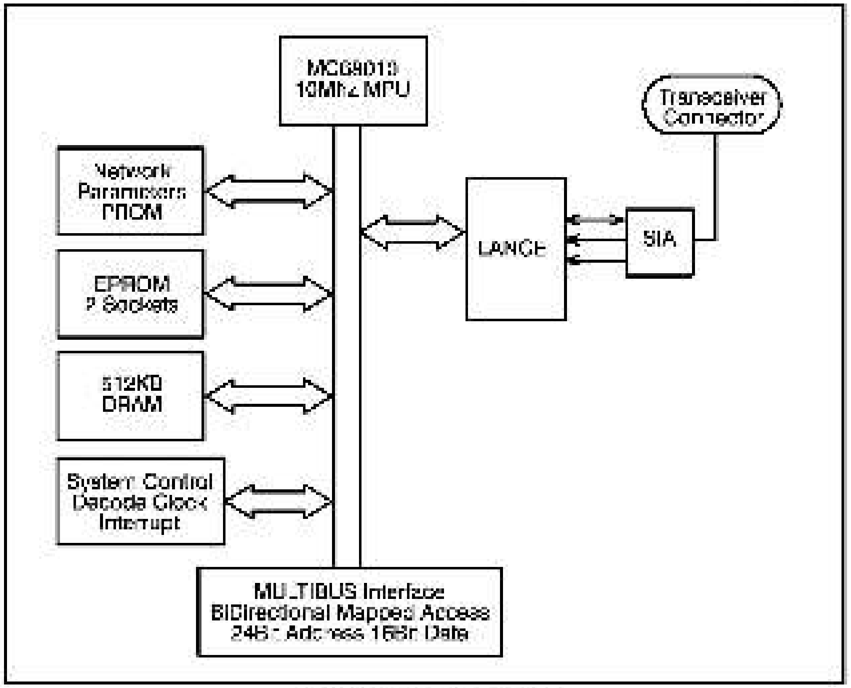


Figure 2.2- Block Diagram

2.3 Transceiver Interface

A transceiver is an interface device used for attaching the ENI module to the Ethernet cable. A transceiver cable is used to connect the ENI module and the transceiver. The transceiver cable consists of four shielded twisted-pair wires and two 15-pin D-connectors (see figure 2.3). The maximum cable length is 50 meters (164 feet). See Appendix B for additional information.

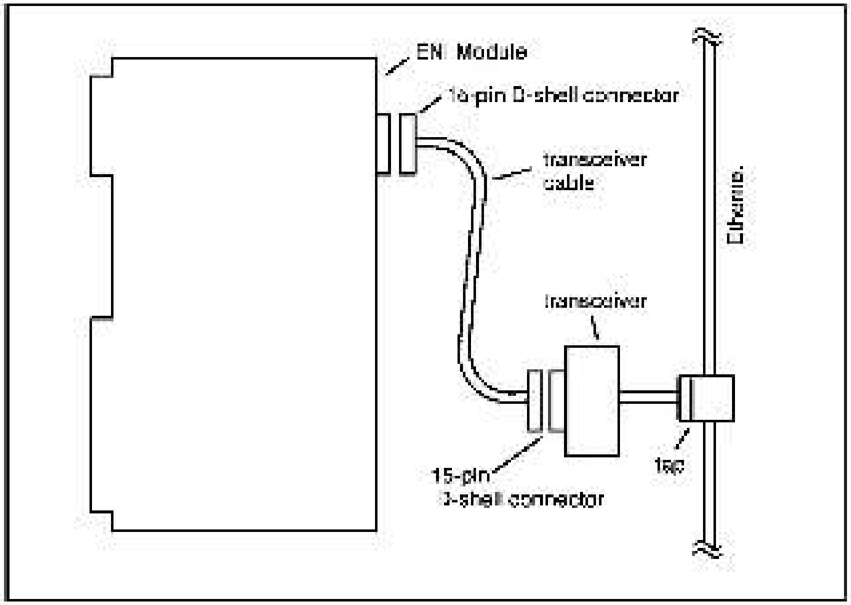


Figure 2.3- Transceiver Connections

3.0 INSTALLATION

This section describes how to install and replace the ENI module. See Appendix B for instructions on connecting the ENI to the transceiver. Consult your Ethernet supplier for specific information regarding installation of cables, transceivers, and other network equipment.

DANGER

THE USER IS RESPONSIBLE FOR CONFORMING TO APPLICABLE LOCAL, NATIONAL AND INTERNATIONAL CODES. WIRING GROUNDING, DISCONNECTS, AND OVER-CURRENT PROTECTION ARE PARTICULARLY IMPORTANT. FAILURE TO OBSERVE THIS PRECAUTION COULD RESULT IN SEVERE BODILY HARM OR LOSS OF LIFE.

3.1 Hardware Configuration

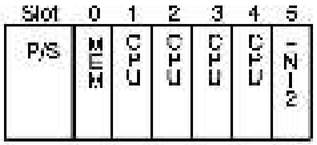
The ENI module is factory-configured for IEEE 802.3/Ethernet 2.0. If you are connecting to an existing network that uses Ethernet 1.0, consult your authorized Reliance representative.

3.2 Rack Configuration

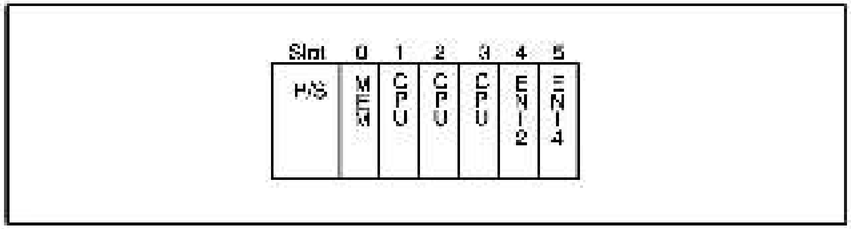
All application tasks that access an ENI module must reside in the left-most processor in the rack.

There can be a maximum of two ENI modules in a rack. Each ENI module uses two slots of address space (128K). An ENI module may be installed in any physical slot of the rack. However, the slot address range that the module responds to is selected with a jumper. This jumper must select either logical slot 2 or logical slot 4. If logical slot 2 is selected, then no other card, with the exception of a Processor (M/N 57C430A), may be in slot 2 or slot 3. If logical slot 4 is selected then no other card, with the exception of a Processor, may be in slots 4 and 5. Processors are allowed because they have no Multibus-addressable memory.

In the following example the ENI is in physical slot 5, but it is jumpered to respond to logical slots 2 and 3. The important rule is that two cards can not be in the same logical slot. Because the Processor does not have MultiBus memory, there is no conflict.



The second example shows two ENI modules jumpered to respond to logical slots 2 and 4. ENI2 takes up logical slots 2 and 3, ENI4 takes up logical slots 4 and 5.



3.3 ENI Installation

Use the following procedure to install the module:

Step 1. Turn off power to the rack and all connections.

Step 2. Take the module out of its shipping container. Take it out of the anti-static bag, being careful not to touch the connectors on the back of the module.

Step 3. Set the jumper (visible through the cutout in the enclosure) for the appropriate logical slot. Refer to figure 3.1 and section 3.2.

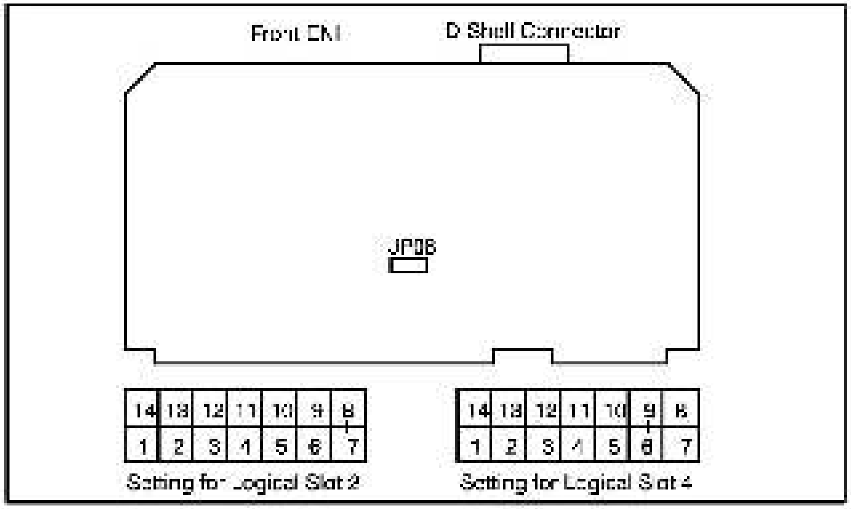


Figure 3.1- Setting Jumper for Logical Slot Location

Step 4. Insert the module into the desired slot in the rack. Use a screwdriver to secure the module into the slot.

Step 5. Connect the transceiver cable to the ENI according to the manufacturer's instructions.

Step 6. Turn on power to the rack.

Step 7. Verify the installation. After the power-up diagnostics are completed, the green status LED will go on.

3.4 Module Replacement

Step 1. Stop all tasks that are running.

Step 2. Turn off power to the rack.

Step 3. Disconnect the transceiver cable from the module.

Step 4. Use a screwdriver to loosen the screws that hold the module in the rack. Remove the module from the slot in the rack.

Step 5. Place the module in the anti-static bag it came in, being careful not to touch the connectors on the back of the module. Place the module in the cardboard shipping container.

Step 6. Take the replacement module out of its shipping container. Take it out of the anti-static bag, being careful not to touch the connectors on the back of the module.

Step 7. Set the jumper for the appropriate slot.

Step 8. Insert the module into the desired slot in the rack. Use a screwdriver to secure the module into the slot.

Step 9. Connect the transceiver cable to the module.

Step 10. Turn on power to the rack.

Step 11. Verify the installation. After the power-up diagnostics are completed, the green status LED will go on.

4.0 PROGRAMMING

This section provides an overview of the BASIC functions that are used to access the ENI module. Programming examples for TCP, UDP, and raw Ethernet communications are provided in section 4.2.7. A detailed listing of the BASIC functions used in ENI application software is in section 4.6. For more detailed information on the BASIC language refer to the AutoMax Enhanced BASIC Language Instruction Manual (J-3675).

4.1 Introduction

Establishing communication between two points on a network is analogous to establishing a telephone connection between two points, A and B.

The first step is for both A and B to initialize their respective ENI modules by executing the ENI_INIT% function. This is like asking the phone company to install phone service in an office. The ENI_INIT% function assigns a drop number to the card, referred to as the InterNet address. This is like assigning a main phone number for an office.

The next step would be to assign an extension number to every phone in the office. For the ENI this is done with two functions, SOCKET% and BIND%. The SOCKET% function selects the type of service the connection is to provide. The choices are TCP, UDP, or Raw EtherNet. This is somewhat like choosing between touch tone and pulse dial phone service. The value that is returned from the SOCKET% function is called the socket number. This is used in all subsequent function calls to specify where to communicate. The BIND% function then assigns a port number to the socket. The port number is like the phone extension number.

After the phones are installed, the next step is to place a call. This is done with the CONNECT% function. This function specifies the address to connect to, like a phone number to dial. For TCP sockets only, if point A is initiating the message (placing the call), it must execute a CONNECT% function and point B (waiting for messages) must execute an ACCEPT% function.

Messages can then be sent or received by either end by executing a SEND% or RECV% function. For example, in a phone conversation, if you wanted some information, you would start by telling the other person what you wanted. The other person would need to hear your request, understand it, and then respond with the answer. To do this with an ENI you would start with a SEND% that tells the other station what you wanted. The other station would need to do a RECV% to hear the message and then a SEND% to respond with the answer. In the meantime, you would be doing a RECV% to hear the response. When you are done talking on the phone, you say good bye and hang up the phone. In the ENI, when you want to end a session, you would execute the SHUTDOWN% function.

4.2 Programming Overview

This section gives an overview of the functions in BASIC that are used to access the ENI module. The functions are broken down into six categories:

- Card initialization
- Creation and binding of sockets
- Establishing a connection
- Transfer of data
- Support functions
- Shutting down sockets

The individual functions are described in detail in section 4.6.

4.2.1 Card Initialization

The ENI module is initialized with the ENI_INIT% function. This tells the AutoMax operating system what slot the card is in. It also assigns an InterNet address to the module and it selects how many sockets to allow for each of the three protocols supported. The value returned tells if the operation was successful or not.

STATUS% = ENI_INIT%(SLOT%, ADDR$, TCP%, UDP%, ETHER%)

4.2.2 Creation and Binding of Sockets

A socket (channel of communication) is created with the SOCKET% function. A socket is bidirectional, i.e., it can be used to send and receive. The parameters of the SOCKET function select on which ENI module to create the socket and what protocol the socket is to use. The value returned from the SOCKET function is called the socket number. This number is used in all subsequent functions to select which socket is being worked with.

SOCKET_NUM% = SOCKET%(SLOT%, TYPE%)

After a socket is created, the application program must do a BIND% to assign a port number to the socket. The parameters of the function select which socket to bind and the value for the port number. An ENI module can have a total of up to 64 sockets open at the same time. The port number acts as an extension to the Internet address, in assigning a unique address to each socket. The value returned in STATUS tells if the operation was successful or not.

STATUS% = BIND%(SN%, PORT%)

If the BIND% is not successful (STATUS% = -4), you must shut down the socket, wait 10 seconds, then recreate the socket and bind it with a different port number.

4.3.3 Establishing a Connection

For TCP sockets only, a connection must be established before communication can begin. There are two sides to each connection, the active side and the passive side. The active station does a CONNECT% function and the passive station does an ACCEPT% function. The ACCEPT% function must be executed by the passive side prior to the active side executing a CONNECT% function. The parameters of the CONNECT% select which socket to connect, as well as the destination address and port number. The value returned in STATUS indicates whether the operation was successful or not.

```
STATUS% = CONNECT%( SOCKET_NUM%, DEST_ADDR$,
DEST_PORT% )
```

The first parameter of the ACCEPT% function selects which socket should begin waiting for a connection to come. The second parameter is the name of the variable where this function will return the value of a new socket number that is created. The value returned in STATUS indicates whether the operation was successful or not. The original socket that was waiting for a connection remains open. The application program may loop back to the ACCEPT% function to wait for another client to connect, or the socket may be shut down if nothing else is expected. The new socket is accessed through the value in NEW_SOCKET_NUM. This is the socket through which the passive station will send and receive.

```
STATUS% = ACCEPT%( SOCKET_NUM%, NEW_SOCKET_NUM% )
```

If the TCP protocol is selected, a connection must be established before data can be transferred. If the UDP or raw Ethernet protocols are used, a connection is not established. However, a CONNECT% must be executed by the station that will be sending the message to select where to send it to. The ACCEPT% function is not used for UDP or raw Ethernet sockets.

4.2.4 Data Transfer

Data transfers can begin once a socket is created and connected. The SEND%, SENDL%, RECV% and RECVL% functions are used to send and receive messages.

4.2.4.1 Sending Data

To send data to another station, use the SEND% or SENDL% function. The parameters of the SEND% select which socket to work with, the variable to send, and the number of bytes to send. The variable to send may be any data type. The data can be contained within an array. Both local and common variables can be sent, as well as I/O variables. The value returned is either the total number of bytes sent successfully, or an error code.

```
BYTES_SENT% = SEND%( SOCKET_NUM%, DATA%, LENGTH% )
```

The parameters of the SENDL select which socket to work with and select a list of pointers and byte counts. This allows for building a message from various places in memory. The value returned is either the total number of bytes sent successfully, or an error code.

```
BYTES_SENT% = SENDL%( SOCKET_NUM%, LIST! )
```

4.2.4.2 Receiving Data

To receive data from another station, use the RECV% or RECVL% function. The parameters of the RECV% function select which socket to work with, the variable to receive into, and the number of bytes to receive. The variable to receive into may be any data type. The data can be contained within an array. It can also be scalar. Both local and common variables can be sent, as well as I/O variables. The value returned is either the number of bytes received successfully, or an error code.

BYTES_RECVD% = RECV%(SOCKET_NUM%, DATA%, LENGTH%)

The parameters of the RECVL% select which socket to work with, and include a list of pointers and byte counts. This allows for receiving a message into various places in memory. The value returned is either the total number of bytes received successfully, or an error code.

BYTES_RECVD% = RECVL%(SOCKET_NUM%, LIST!)

4.2.5 Support Functions

There are six functions that provide support for communications:

SETSOCKOPT%	is used to set options for a socket.
GETSOCKOPT%	is used to read the status and options selected for a socket.
READVAR%	is used to read the value of a variable expressed as a string.
WRITEVAR%	is used to write a value into a variable expressed as a string.
FINDVAR!	is used to find a pointer to variable expressed as a string. This is used in conjunction with the SENDL% and RECVL% functions.
CONVERT%	is used to convert between Motorola and IEEE floating point formats. It also takes care of byte swapping when needed.

4.2.6 Closing Sockets

The SHUTDOWN% function closes a socket's connection and releases all of its associated resources. TCP sockets only need to be shut down at one end. Either the active or passive side may close the connection. The other side will automatically shut down. UDP and raw Ethernet sockets need to be shut down at both ends.

STATUS% = SHUTDOWN%(socket_num%)

4.2.7 Sample Programs

The BASIC programs that follow provide examples of sending and receiving data using TCP, UDP, and raw Ethernet sockets.

4.2.7.1 TCP Sample Program

In TCP communication, one station is active and the other station is passive. The first step on both ends is to initialize the ENI and assign an Internet address to the module. This is followed by creating a socket and binding a port number to the socket. The active side does a CONNECT% and the passive side does an ACCEPT%. The parameters of the CONNECT% specify the Internet address and the port number of the destination. These are the same values used in the ENI_INIT% and BIND% on the passive side. The parameters of the ACCEPT% specify where to write the value of a new socket that will be created when a connection is made. The example shows that if the CONNECT% is not successful, it goes back to create a new socket to try again. After the connection is established, both sides can send and receive data. For this example, the active side is sending to the passive side. It could have been the other way, or they could take turns sending and receiving. In TCP, for every message sent, there is an acknowledgement returned to control the flow of information. After the data has been sent, doing a SHUTDOWN% on either side closes the sockets on both sides.

The following are examples of tasks that perform an active connection and a passive connection for a TCP socket. The example tasks show a STOP being executed when an error is returned. This is done only to show that some action should be taken when an error is detected. It is up to the application programmer to decide the appropriate response to an error.

```
100     REM Sample program to perform an active connection and
101     REM send data over a TCP socket

200     REM Local symbolic constants
210     LOCAL MY_ADDR$, MY_PORT%, DEST_ADDR$, DEST_PORT%
220     MY_ADDR$ = "128.10.3.29"
230     MY_PORT% = 5000
240     DEST_ADDR$ = "128.10.4.17"
250     DEST_PORT% = 5100

300     REM Local variables
310     LOCAL STATUS%, SOCKET_NUM%
320     LOCAL BYTES_SENT%
330     LOCAL MESSAGE%(99), I%, J%

1000    REM Initialize the ENI
1010    STATUS% = ENI_INIT%( 4, MY_ADDR$ , 10, 1, 8)
1020    IF ( STATUS% < 0 ) THEN STOP

1030    REM Create a socket
1040    SOCKET_NUM% = SOCKET%( 4, 1 )
1050    IF ( SOCKET_NUM% < 0 ) THEN STOP

1060    REM Bind a port number to the socket
1070    STATUS% = BIND%( SOCKET_NUM%, MY_PORT% )
1080    IF ( STATUS% < 0 ) THEN STOP

1090    REM Try to connect. If didn't connect try again
1100    STATUS% = CONNECT%( SOCKET_NUM%, DEST_ADDR$, DEST_PORT% )
1110    IF ( STATUS% = -102 ) THEN DELAY 2 SECONDS \ GOTO 1040
1120    IF ( STATUS% < 0 ) THEN STOP

1130    REM Send data 100 times, if connection lost, try to
1131    REM connect again
1140    FOR I% = 0 TO 99
1150        FOR J% = 0 TO 99
1160            MESSAGE%(J%) = J% + I%
1170        NEXT J%
1180        BYTES_SENT% = SEND%( SOCKET_NUM%, MESSAGE%, 0 )
1190        IF ( BYTES_SENT% = -102 ) THEN DELAY 10 SECONDS \           &
                                          GOTO 1040

1200    NEXT I%

1210    REM Shut down the connection
1220    STATUS% = SHUTDOWN%( SOCKET_NUM% )
1230    IF ( STATUS% < 0 ) THEN STOP
32767   END
```

```
100     REM Sample program to perform a passive connection and
101     REM receive data over a TCP socket

200     REM Local symbolic constants
210     LOCAL MY_ADDR$, MY_PORT%
220     MY_ADDR$ = "128.10.4.17"
230     MY_PORT% = 5100

300     REM Local variables
310     LOCAL STATUS%, SOCKET_NUM%
320     LOCAL NEW_SOCKET_NUM%, BYTES_RECVD%
330     LOCAL MESSAGE%(99), I%, J%, ERROR%

400     REM Initialize to no error found
410     ERROR% = -1

1000    REM Initialize the ENI
1010    STATUS% = ENI_INIT%( 4, MY_ADDR$, 14, 3, 2 )
1020    IF ( STATUS% < 0 ) THEN STOP

1030    REM Create a socket
1040    SOCKET_NUM% = SOCKET%( 4, 1 )
1050    IF ( SOCKET_NUM% < 0 ) THEN STOP

1060    REM Bind a port number to the socket
1070    STATUS% = BIND%( SOCKET_NUM%, MY_PORT% )
1080    IF ( BIND_STATUS% < 0 ) THEN STOP

1090    REM Wait for connection to come in
1100    STATUS% = ACCEPT%( SOCKET_NUM%, NEW_SOCKET_NUM% )
1110    IF ( STATUS% < 0 ) THEN STOP

1150    REM Recv data 100 times. If connection lost, try to
1151    REM connect again
1160    FOR I% = 0 TO 99
1170        BYTES_RECVD% = RECV%( NEW_SOCKET_NUM%, MESSAGE%, 0 )
1180        IF ( BYTES_RECVD% = -102 ) THEN DELAY 10 SECONDS \          &
                                  GOTO 1100
1190        IF ( BYTES_RECVD < 0 ) THEN STOP
1200        FOR J% = 0 TO 99
1210            IF ( MESSAGE%(J%) <> J% - I% )                           &
                THEN ERROR% = I% * 100 + J% \ STOP
1220        NEXT J%
1230    NEXT I%
32767   END
```

4.2.7.2 UDP Sample Program

In UDP communication, no connection is made. Like TCP, the destination address of a message is an Internet address and port number. However, unlike TCP, the receiving station does not send an acknowledgement to the sender. Both the sending and receiving station start by assigning an Internet address to the module and, after creating a socket, bind a port number to the socket. The station that will be sending the data then does a CONNECT% to specify the destination Internet address and port number. The station that will be receiving the data does not do an ACCEPT%; it does a RECV%. After the data has been sent, both sides must do a SHUTDOWN% to close the sockets.

```
100     REM Sample program to send data over a UDP socket
200     REM Local symbolic constants
210     LOCAL MY_ADDR$, MY_PORT%, DEST_ADDR$, DEST_PORT%
220     MY_ADDR$ = "128.10.3.40"
230     MY_PORT% = 5000
240     DEST_ADDR$ = "128.10.4.17"
250     DEST_PORT% = 5100

300     REM Local variables
310     LOCAL STATUS%, SOCKET_NUM%
320     LOCAL BYTES_SENT%
330     LOCAL MESSAGE%(20), I%, J%

1000    REM Initialize the ENI
1010    STATUS% = ENI_INIT%( 4, MY_ADDR$ , 2, 12, 1)
1020    IF ( STATUS% < 0 ) THEN STOP

1030    REM Create a UDP socket
1040    SOCKET_NUM% = SOCKET%( 4, 2 )
1050    IF ( SOCKET_NUM% < 0 ) THEN STOP

1060    REM Bind a port number to the socket
1070    STATUS% = BIND%( SOCKET_NUM%, MY_PORT% )
1080    IF ( STATUS% < 0 ) THEN STOP

1090    REM Fill in destination parameters
1100    STATUS% = CONNECT%( SOCKET_NUM%, DEST_ADDR$, DEST_PORT% )
1110    IF ( STATUS% < 0 ) THEN STOP

1120    REM Send data 100 times
1130    FOR I% = 0 TO 99
1140        FOR J% = 0 TO 20
1150            MESSAGE%(J%) = J% + I%
1160        NEXT J%
1170        BYTES_SENT% = SEND%( SOCKET_NUM%, MESSAGE%, 0 )
1180    NEXT I%

1190    REM Shut down the socket
1200    STATUS% = SHUTDOWN%( SOCKET_NUM% )
1210    IF ( STATUS% < 0 ) THEN STOP
32767   END
```

```
100     REM Sample program to receive data over a UDP socket
200     REM Local symbolic constants
210     LOCAL MY_ADDR$, MY_PORT%
220     MY_ADDR$ = "128.104.17"
230     MY_PORT% = 5100

300     REM Local variables
310     LOCAL STATUS%, SOCKET_NUM%
320     LOCAL BYTES_RECVD%
330     LOCAL MESSAGE%(99), I%, J%, ERROR_CNT%

400     REM Initialize error counter
410     ERROR_CNT% = 0

1000    REM Initialize the ENI
1010    STATUS% = ENI_INIT%( 1, MY_ADDR$, 4, 16, 2)
1020    IF ( STATUS% < 0 ) THEN STOP

1030    REM Create a UDP socket
1040    SOCKET_NUM% = SOCKET%( 2, 2 )
1050    IF ( SOCKET_NUM% < 0 ) THEN STOP

1060    REM Bind a port number to the socket
1070    STATUS% = BIND%( SOCKET_NUM%, MY_PORT% )
1080    IF ( STATUS% < 0 ) THEN STOP

1090    REM Recv data 100 times
1100    FOR I% = 0 TO 99
1110        BYTES_RECVD% = RECV%( SOCKET_NUM%, MESSAGE%, 0 )
1120        FOR J% = 0 TO 99
1130            IF ( MESSAGE%(J%) <> (J% + I%) )                           &
                THEN ERROR_CNT% = ERROR_CNT% + 1
1140        NEXT J%
1150    NEXT I%

1160    REM Shut down the socket
1170    STATUS% = SHUTDOWN%( SOCKET_NUM% )
1180    IF ( STATUS% < 0 ) THEN STOP
32767   END
```

4.2.7.3 Raw Ethernet Sample Program

In raw Ethernet communication, no connection is made. When a message is sent, every station on the network listens to see if the message is intended for it. Every Ethernet module has a unique address. To send a message to a particular station, you need to know the Ethernet address of that module. It is also possible to send or receive broadcast data. This is done in the following example. In TCP and UDP, all you need to know is the Internet address and port number of a station over which you have control.

As in TCP and UDP, the first step on both ends is to initialize the module with ENI_INIT%. An Internet address must still be given to the module, even though this protocol does not use it. After a socket is created and a bind is done, the sending station does a CONNECT% to specify where to send the message. The value for addr$ in the connect must be the Ethernet address or a broadcast address that the destination will recognize. The only broadcast address the ENI recognizes is FFFFFFFFFFFF. The value for port% in the connect must be the same value used in the bind on the receiving station. The value for port% is also used to select the 'packet type' in the message. See section 4.3 for more information on frame format. A station can have more than one Ethernet socket open; the packet type is used to select which socket an incoming message will be given to.

```
100     REM Sample program to send data over a raw Ethernet
101     REM socket
200     REM Local symbolic constants
210     LOCAL MY_ADDR$, MY_TYPE%, DEST_ADDR$, FRAME_TYPE%
220     MY_ADDR$ = "[illegible]"
230     MY_TYPE% = 5100
240     DEST_ADDR$ = "FF:FF:FF:FF:FF:FF"
250     FRAME_TYPE% = 5200

300     REM Local variables
310     LOCAL STATUS%, SOCKET_NUM%
320     LOCAL BYTES_SENT%
330     LOCAL MESSAGE%(99), I%, J%

1000    REM Initialize the ENI
1010    STATUS% = ENI_INIT%( 4, MY_ADDR$, 3, 1, 10 )
1020    IF ( STATUS% < 0 ) THEN STOP

1030    REM Create a raw Ethernet socket
1040    SOCKET_NUM% = SOCKET%( 4, 3 )
1050    IF ( SOCKET_NUM% < 0 ) THEN STOP

1060    REM Bind a port number to the socket
1070    STATUS% = BIND%( SOCKET_NUM%, MY_TYPE% )
1080    IF ( STATUS% < 0 ) THEN STOP

1090    REM Fill in destination parameters
1100    STATUS% = CONNECT%( SOCKET_NUM%, DEST_ADDR$, FRAME_TYPE%)
1110    IF ( STATUS% < 0 ) THEN STOP

1120    REM Send data 100 times
1130    FOR I% = 0 TO 99
1140        FOR J% = 0 TO 99
1150            MESSAGE%(J%) = J% + I%
1160        NEXT J%
1170            BYTES_SENT% = SEND%( SOCKET_NUM%, MESSAGE%, 0 )
1180    NEXT I%

1190    REM Shut down the socket
1200    STATUS% = SHUTDOWN%( SOCKET_NUM% )
1210    IF ( STATUS% < 0 ) THEN STOP
32767   END
```

```
100    REM Sample program to receive data over a raw Ethernet
101    REM socket
200    REM Local symbolic constants
210    LOCAL MY_ADDR$, FRAME_TYPE%
220    MY_ADDR$ = "128.10.4.17"
230    FRAME_TYPE% = 0800

300    REM Local variables
310    LOCAL STATUS%, SOCKET_NUM%
320    LOCAL BYTES_RECVD%
330    LOCAL MESSAGE%(99), I%, J%, ERROR_CNT%

400    REM Initialize error counter
410    ERROR_CNT% = 0

1000   REM Initialize the ENI
1010   STATUS% = ENI_INIT%( 4, MY_ADDR$ , 4, 1, 12)
1020   IF ( STATUS% < 0 ) THEN STOP

1030   REM Create a raw Ethernet socket
1040   SOCKET_NUM% = SOCKET%( 4, 3 )
1050   IF ( SOCKET_NUM% < 0 ) THEN STOP

1060   REM Bind a port number to the socket
1070   STATUS% = BIND%( SOCKET_NUM%, FRAME_TYPE% )
1080   IF ( STATUS% < 0 ) THEN STOP

1090   REM Receive data 100 times
1100   FOR I% = 0 TO 99
1110       BYTES_RECVD% = RECV%( SOCKET_NUM%, MESSAGE%, 0 )
1120       FOR J% = 0 TO 99
1130           IF ( MESSAGE%(J%) <> (J% + I%) )                              &
               THEN ERROR_CNT% = ERROR_CNT% + 1
1140       NEXT J%
1150   NEXT I%

1210   REM Shut down the socket
1220   STATUS% = SHUTDOWN%( SOCKET_NUM% )
1230   IF ( STATUS% < 0 ) THEN STOP
32767  END
```

4.3 Raw Ethernet Notes

Every ENI module has a unique factory-assigned Ethernet address stored in ROM memory. After the ENI has been installed in the rack and initialized by the ENI_INIT% function, the value of this 6-byte number can be read by the programming terminal using the Monitor I/O utility in the AutoMax Programming Executive software. This utility is used to read and write selected addresses across Multibus. See J-3750 for more information on Monitor I/O. The Ethernet address is required only if you want to communicate using raw Ethernet. For TCP or UDP communciation, the Internet address is user defined via the ENI_INIT function.

In a network which utilizes raw Ethernet communication, replacing a faulty ENI module will change the address of that Ethernet node. Application programs which communicate with that node will require changes to specify the new Ethernet node address.

To read the Ethernet address, display the following three registers in the logical slot selected for the card: 2316, 2317, 2318. Display these registers in hexadecimal format. The address is composed of the contents of each of the three registers strung together. For example, the sample display values shown below indicate the Ethernet address 02CF1F305599.

Slot	Register	Value
2	2316	02CF
2	2317	1F30
2	2318	5599

4.3.1 Ethernet Frame Format

A raw Ethernet frame consists of a 6-byte destination address, a 6-byte source address, a 2-byte type field, 46 to 1500 data bytes, and a CRC (Cyclical Redundancy Check) as shown below.

6 Bytes	6 Bytes	2 Bytes	46 to 1500 Bytes	
Des. Addr	Src Addr	Type	Data . . .	CRC

The destination address is the number assigned to the socket with the CONNECT% function. For raw Ethernet sockets, the CONNECT% function doesn't send a message to the destination as it does in TCP; instead, it only records the destination address for later use by the SEND% function. The source address is the raw Ethernet address of the stations sending the message. The type is used to determine what protocol is used. For raw Ethernet messages, Type is the port number that was assigned to the socket with the BIND% function. There are two reserved numbers for Type that may not be used by raw Ethernet messages: decimal 2048 and 2054. It is recommended that port numbers begin at 5000.

4.4 Data Formats

The following section describes the internal representation of data types used in AutoMax. See the AutoMax Enhanced BASIC Language Instruction Manual (J-3675) for more information.

4.4.1 Booleans

A boolean is a bit in a 16-bit word. Individual bits can not be sent or received. The smallest amount of data that can be sent is 1 byte (8 bits). If a single boolean variable is sent, 7 other bits are sent with it. If CONVERT% is not used, a boolean array is transmitted bits 7..0 first followed by bits 15..8, and so on.

4.4.2 Integers

Integers are stored in 2 bytes, high byte first. If CONVERT% is not used, the high byte is transmitted first.

4.4.3 Double Integers

Double integers are stored in 4 bytes, high byte first. If CONVERT% is not used, the high byte is transmitted first.

4.4.4 Reals

Real numbers are stored in 4 bytes. The format of the number is optimized for performance on the processor. It consists of a 24-bit mantissa, a 1-bit sign, and a 7-bit exponent in excess 64. This may be converted to IEEE standard with the CONVERT% function.

4.4.5 Strings

The default length of a string is 32 characters. This takes 34 bytes in memory. The first byte of a string contains the number of bytes available for string storage and the second byte indicates the actual length of the string variable. This is followed by the string itself.

4.5 Use of Hardware Interrupts in Racks Containing Ethernet or Network Modules

This section is applicable only to racks that contain Current Minor Loop (CML) tasks or hardware EVENT statements in BASIC or Control Block tasks. These two kinds of tasks require Processors to allocate hardware interrupt lines on the rack backplane because some portion of task execution depends upon receiving a user-defined hardware interrupt from another module in the rack, e.g., a Resolver Input module. The remainder of this section will first describe the basic method by which interrupt lines are allocated and then how Ethernet modules affect the allocation process. See the Enhanced BASIC Language Instruction manual (J-3675) for more information on hardware EVENT statements and the Control Block Language Instruction manual (J-3676) for more information on CML tasks.

Because the number of interrupt lines is limited to four, it is necessary to take into account the rules by which they are allocated in order to prevent errors when application tasks are put into run. Each of the four interrupt lines can "service" one of the following:

a) up to four BASIC language hardware EVENT statements that are found in BASIC or Control Block tasks

b) one CML task (used in racks containing drive modules only)

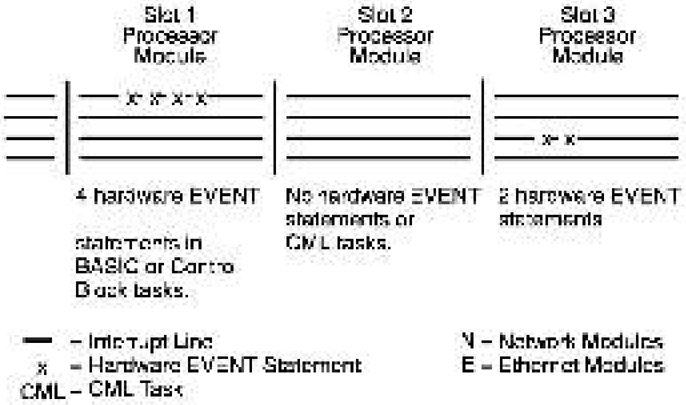
Any one Processor module can allocate up to one of the four interrupt lines for 0-4 hardware EVENT statements and one line for a CML task. (CML tasks are limited to to 2 per rack because of drive module configuration restrictions.) Note that a minimum of one hardware interrupt line will be allocated for a Processor module regardless of whether one or four hardware EVENT statements are used in application tasks loaded on that Processor.

The following examples of interrupt line allocation assume that there are three Processor modules in the rack. Note that these examples do not take into account the efficiency of distributing application tasks between Processor modules in this manner (in terms of system performance) and do not include Ethernet Network Interface modules (M/N 57C440A) or Network modules (57C404A or later only). These two modules will be added in later examples.

4.5.1 Examples of Interrupt Line Allocation

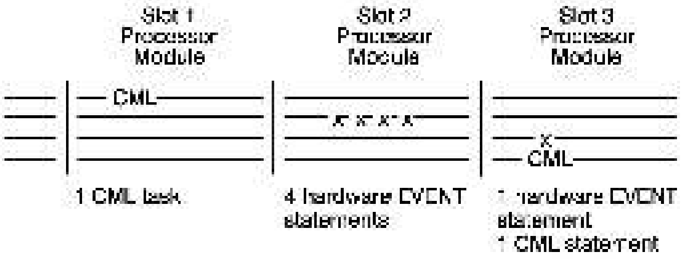
The following are examples of interrupt line allocation.

Example #1



Slot 1 Processor Module
Slot 2 Processor Module
Slot 3 Processor Module
x x x x
x x
4 hardware EVENT statements in BASIC or Control Block tasks.
No hardware EVENT statements or CML tasks.
2 hardware EVENT statements
— – Interrupt Line
x – Hardware EVENT Statement
CML – CML Task
N – Network Modules
E – Ethernet Modules

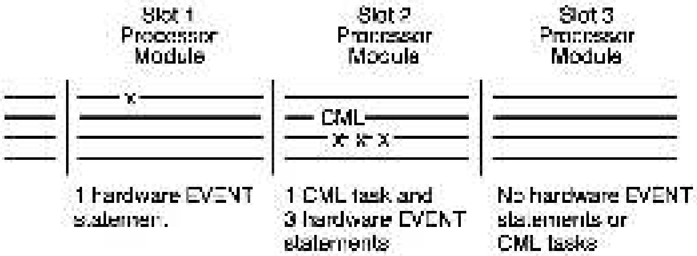
Example #2



— – Interrupt Line
x – Hardware EVENT Statement
CML – CML Task

N – Network Modules
E – Ethernet Modules

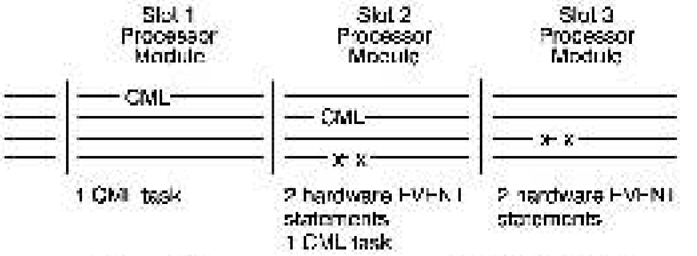
Example #3


Slot 1 Processor Module
Slot 2 Processor Module
Slot 3 Processor Module
x
CML
x x x
1 hardware EVENT statement
1 CML task and 3 hardware EVENT statements
No hardware EVENT statements or CML tasks

— – Interrupt Line
x – Hardware EVENT Statement
CML – CML Task

N – Network Modules
E – Ethernet Modules

Example #4



— – Interrupt Line
x – Hardware EVENT Statement
CML = CML Task

N – Network Modules
E – Ethernet Modules

4.5.2 Examples of Interrupt Allocation with Ethernet or Network Modules in the Rack

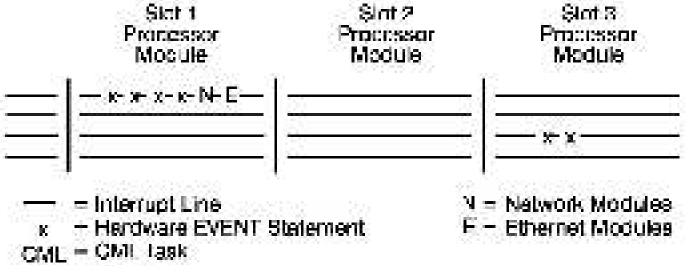
With the addition of Ethernet Network Interface modules or Network modules (M/N 57C404A and later only) to the rack, examples #2 and #4 in section 4.5.1 would cause an error (code 44 displayed on the Processor LEDs) when tasks were put into run and would not allow them to go into run. The following section explains the allocation of interrupts when Ethernet and Network modules are added to the examples in 4.5.1.

Ethernet and Network modules require the allocation of an interrupt line by the leftmost Processor module in the rack. The presence of either or both of these two modules in any quantity will require a single interrupt line on the leftmost Processor. The interrupt line required by these modules can, however, be shared with four hardware EVENT statements, but cannot be shared with the interrupt line required by a CML task.

If two Ethernet modules and two Network modules were added to the rack in the above examples, the following would occur. Note that when either of these modules are added to the rack, the leftmost Processor module will show an increase in CPU utilization (processing capacity used). The CPU utilization statistic is available through the Programming Executive software.

Example #1

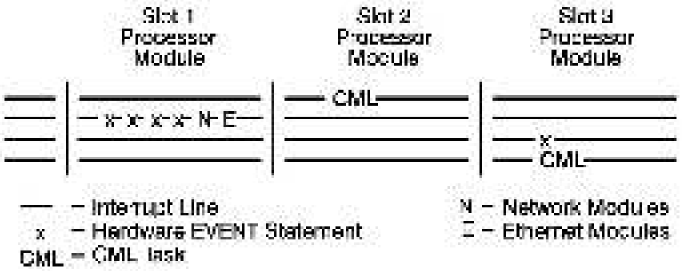
The Ethernet and Network modules would share the interrupt line with the four hardware EVENT statements in the left-most Processor module.



Example #2

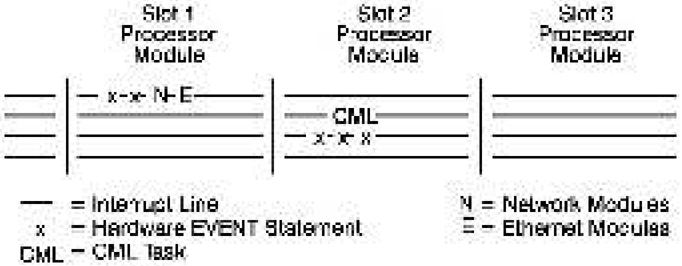
This example would cause an error when application tasks were put into run. The CML task in the leftmost Processor module cannot share its interrupt line, and the remaining three lines are already allocated (one on Processor in slot 2, two on Processor in slot 3).

One solution to this problem would be to move the CML task from the Processor in slot 1 to the Processor in slot 2 and the task(s) containing the four hardware events from the Processor in slot 2 to the Processor in slot 1. The Ethernet and Network modules could share the interrupt line required for the EVENT statements in the left-most Processor.



Example #3

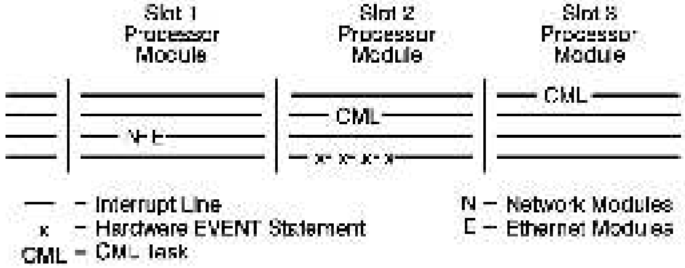
The Ethernet and Network modules would share the line required for the hardware EVENT statement in the leftmost Processor. Note that this line can be shared whether it is used for 1, 2, 3, or 4 EVENT statements.



Example #4

This example would cause an error when application tasks were put into run. Four interrupt lines have already been allocated. The leftmost Processor module has allocated one for its CML application task. The Processor module in slot 2 has allocated one for two hardware EVENT statements and one for its CML task. The Processor module in slot 3 has allocated one for two hardware EVENT statements. There are no lines left for the left-most Processor to allocate for the Ethernet and Network modules, and the interrupt line required for the CML task cannot be shared.

One solution is to move the CML task from the Processor in slot 1 to the Processor in slot 3 and to move the task or tasks containing two hardware EVENT statements to the Processor in slot 2. In this case, the Processor in slot 2 still requires one interrupt line. The Processor module in slot 3 requires one interrupt line. The left-most Processor will allocate an interrupt line for the Ethernet and Network modules. Note that this line could be shared with up to four EVENT statements.



4.6 Functions

The BASIC functions that follow are used in application software written for use with the Ethernet Network Interface.

When a function is executed, a value is returned in the variable specified. A negative value indicates an error. The error codes for each function are listed in the function descriptions. Appendix C contains a list of all the ENI error codes. Application software must check for these error codes.

4.6.1 ENI_INIT% Function

Format:

ENI_INIT%(slot%, addr$, tcp%, udp%, ether%)

where:

slot%	is the logical slot the ENI is to be in. This can be a variable or a constant. The only legal values are 2 or 4. See section 3.2 for information on rack configuration.
addr$	is the Internet address you assign to the ENI. This is a string of four decimal numbers separated by decimal points, each ranging from 0 to 255. A typical address is 128.0.0.10. (Note that 128.0.0.0 is an illegal address.)
tcp%	defines the number of sockets to use for the TCP protocol.
udp%	defines the number of sockets to use for the UDP protocol.
ether%	defines the number of sockets to use for raw Ethernet.

The ENI_INIT% function commands the Ethernet Network Interface to go through its initialization. The ENI supports three types of protocols: TCP, UDP and raw Ethernet. Up to 64 channels (sockets) can be assigned to each ENI. Part of the initialization selects how may sockets to allow for each protocol. At least one socket must be defined for each protocol. The green LED on the front of the ENI will turn off for approximately 10 seconds while the initialization is performed.

Values Returned:

1	Success
-7	ENI failed self test
-8	Bus error
-10	Error allocating interrupts
-11	Bad slot number
-12	Bad Internet address
-13	Total number of sockets > 64

For example:

STATUS% = ENI_INIT%(4, "128.0.0.10", 32, 10, 3)

4.6.2 SOCKET% Function

Format:

SOCKET%(slot%, type%)

where:

slot% is the logical slot the ENI is to be in. This can be a variable or a constant. The only legal values are 2 or 4. See section 3.2 for information on rack configuration.

type% is used to select the protocol for this socket.

Legal values for type are:

1 for a TCP socket
2 for a UDP socket
3 for a Raw EtherNet socket

This function will find an available socket of the requested type. If successful, the value returned is the number of the socket allocated. The socket number is a 16-bit word (e.g.. 022FH). The first byte is the logical slot the ENI is jumpered to (02 or 04) and the second byte is the socket (00-3F). All subsequent function calls to communicate with the ENI use the socket number to select the socket to talk through.

Values Returned:

>0	The socket number allocated
-2	ENI not initialized
-3	Did not create socket
-9	No buffer space
-11	Bad slot number
-14	Bad socket type
-40	No available buffer

For example:

SOCKET_NUM% = SOCKET%(4, 1)

4.6.3 BIND% Function

Format:

BIND%(sn%, port%)

where:

sn% is the number of the socket you want to bind. This is the value that was returned from the SOCKET% function. This can be specified as a simple variable or as an element of an array.

port% is the local port number you want to give to the socket. Begin assigning port numbers at 5000. Port numbers must be unique. A port number cannot be reassigned unless the socket using that number has been closed. For raw Ethernet sockets, this value is used to select the value of the 16 bit packet type in the message header.

This function assigns a local port number or Ethernet packet type to a socket.

Values Returned:

1	Success
-2	ENI not initialized
4	Did not bind socket
-9	No buffer space
-15	Bad socket number
-10	No available buffer

For example:

STATUS% = BIND%(SN%, 5000)

4.6.4 CONNECT% Function

Format:

CONNECT%(sn%, addr$, port%)

where:

sn% is the number of the socket you want to connect to a destination. This is the value returned from the SOCKET% function. This can be specified as a simple variable or as an element of an array.

addr$ is the destination Internet (used with TCP and UDP) or Ethernet (used with raw Ethernet) address you want to connect to. See ENI_INIT for applicable rules for Internet addresses. Ethernet addresses are 12 digit Hex number strings.

port% is the destination port number you want to connect to.

This function assigns a permanent destination for a socket. It must be executed before any messages can be sent using any of the three protocols. For raw Ethernet or UDP sockets, this function is used only to specify the destination address. For TCP sockets, it directs the ENI to do an active open. A passive open (ACCEPT%), done by the destination TCP socket, must occur prior to this function being executed to establish a connection. After the connection is made, messages can be exchanged.

If connecting a TCP socket and the other end is not ready to accept the connection, the socket will be closed. To try to connect again, the application must create a new socket and bind it again.

For raw Ethernet sockets, the port number defines the packet type of all messages that will be sent. The receiving end must do a BIND% with the same value for the port number.

Values Returned:

1	Success
-2	ENI not initialized
-9	No buffer space
-12	Bad InterNet address
-15	Bad socket number
-40	No available buffer
-102	Socket not connected

For example:

STATUS% = CONNECT%(SN%, DEST_INET_ADDR$, 5001)

4.6.5 ACCEPT% Function

Format:

ACCEPT%(sn%, nsn%)

where:

sn% is the number of the TCP socket that should begin waiting for a connection to be made. This can be specified as a simple variable or as an element of an array.

nsn% is filled in by this function with a new socket number created when a connection has been established. This must be a simple variable: array elements are not allowed.

This function is used to direct the ENI to do a passive open. This is valid only on TCP sockets. This function suspends execution of the task and waits until a connection is established. When a connection arrives, it creates a new socket with the attributes of the given socket to service the connection. The application program may then shut down the original socket sn%, or it may loop back to the ACCEPT% to wait for another connection to come in. In this way a given service may have more than one client at a time. Communication will take place through the new socket.

Values Returned:

1	Success
-2	ENI not initialized
-7	Did not accept
-8	No buffer space
-15	Bad socket number
-16	Not a TCP socket
-40	No available buffer

For example:

```
STATUS% = ACCEPT%( SN%, NSN% )
```

4.6.6 SEND% Function

Format:

SEND%(sn%, var, len%)

where:

sn% is the number of the socket through which the message is to be sent. This is the value that was returned from the SOCKET% or ACCEPT% function. This can be specified as a simple variable or as an element of an array.

var is the variable that has the data to send. It can be a boolean, integer, double integer, real, string, or an array of these types. It may be local or common. If an array is specified, no subscript may be given. It will always start with the zeroth element of the array.

len% is the number of bytes to send beginning at var. This parameter can be a constant, an integer, or a double integer.

If var is an array, and len% is zero, the length to send is the size of the array. An error is generated if len% is greater than the size of the array.

This function causes a message to be sent to the destination as defined by the socket number.

If a TCP socket is specified, it must be connected first (receiving side executes an ACCEPT function, then sending side executes a CONNECT function).

Values Returned:

>0	Number of bytes transferred
-2	ENI not initialized
-9	No buffer space
-15	Bad socket number
-17	Message too long, UDP > 1472, ETH > 1500
-18	Zero length for non-array
-26	Array is not single dimension
-32	Beyond end of array
-40	No available buffer
-102	Socket not connected

For example:

XMIT_LEN% = SEND%(SN%, SET_POINTS%, MSG_LEN%)

where SET_POINTS% is the name of an array.

4.6.7 SENDL% Function

Format:

SENDL%(sn%, lst!)

where:

sn% is the number of the socket through which the message is to be sent. This is the value that was returned from the SOCKET% or ACCEPT% function. This can be specified as a simple variable or as an element of an array.

lst! is a one-dimensional double integer array whose size is limited only by memory capacity. The values in the array define where to get the data to send. No subscript is given on this parameter.

Beginning at list! (0), the values in the array are structured so that an entry consists of two double integers.

Data Pointer	
Convert Mode	Byte Count

The even numbered elements of the array contain a pointer indicating where to put data received. These pointers are found with the VARPTR! or FINDVAR! functions.

LIST! (0) = FINDV! (VAR_NAME$)

The odd numbered elements contain the number of bytes to receive in the low word and a convert mode in the high word. The value for convert mode is the same as used in the CONVERT% function to change data formats. The following example converts from IEEE to Motorola floating point format.

LIST! (1) = BYTE_COUNT% + 00020000H

Such pairs of elements may be repeated as often as necessary with the only limitation being that UDP messages may not exceeds 1472 bytes and raw Ethernet messages may not exceede 1500 bytes.

The list is terminated by a data pointer with a value of zero.

The function causes a message to be sent to the destination as defined by the socket number.

If a TCP socket is specified, it must be connected first (passive side executes an ACCEPT function, then active side executes a CONNECT function).

Values Returned:

>0	Number of bytes transferred
-2	ENI not initialized
-9	No buffer space
-15	Bad socket number
-17	Message too long
-18	Zero length
-19	Illegal Pointer
-25	Not a double integer array
-26	Not a single dimension array
-27	Bad array format
-30	Odd number of bytes in list parameter
-40	No available buffer
-102	Socket not connected

For example:

```
XMIT_LEN% = SENDL%( SN%, NETWORK_LIST! )
```

4.6.8 RECV% Function

Format:

RECV%(sn%, var, len%)

where:

sn% is the number of the socket through which the message is to be received. This is the value that was returned from the SOCKET% or ACCEPT% function. This can be specified as a simple variable or as an element of an array.

var is the variable where the data received is written. It can be a boolean, integer, double integer, real, string, or an array of those types. If an array is specified, no subscript may be given.

len% is the number of bytes to receive. This parameter can be a constant, an integer or a double integer.

If var is a simple variable and len% is greater than the size of the simple variable, then var must be defined as I/O to avoid overwriting AutoMax memory.

If var is an array, and len% is zero, the length to receive is the size of the array. An error is generated if len% is greater than the size of the array.

For TCP only, if len% is -1, the number of bytes received will be returned to the sender.

This function writes up to LEN% bytes of data from socket SN% into the variable VAR. If a TCP socket is specified, it must be connected first.

A socket can be selected as blocking or non-blocking. If the socket is designated as blocking and no data has come in, the task will be suspended until data arrives. If the socket is designated as non-blocking and no data has come in, the RECV% command will return with the error "No message waiting". The default mode is blocking.

Values Returned:

>0	Length of message received
-2	ENI not initialized
-9	No buffer space
-15	Bad socket number
-17	Message too long
-18	Zero length for non-array
-28	Array is not single dimension
-29	Max size of strings are not equal
-31	Max size of string < recv size of string
-32	Beyond end of array
-101	No message waiting
-102	Socket not connected

For example:

```
RECV_LEN% = RECV%( SN%, SET_POINTS%, LEN%)
```

4.6.9 RECVL% Function

Format:

RECVL%(sn%, list!)

where:

sn% is the number of the socket through which the message is to be received. This is the value that was returned from the SOCKET% or ACCEPT% function. This can be specified as a simple variable or as an element of an array.

list! is a one-dimensional double integer array whose size is limited only by memory capacity. The values in this array define where to put the data received. No subscript is given on this parameter.

Beginning at list! (0), the values in the array are structured so that an entry consists of two double integers.

Data Pointer	
Convert Mode	Byte Count

The even numbered elements of the array contain a pointer indicating where to put data received. These pointers are found with the VARPTR! or FINDVAR! functions.

LIST! (0) = FINDV! (VAR_NAME$)

The odd numbered elements contain the number of bytes to receive in the low word and a convert mode in the high word. The value for convert mode is the same as used in the CONVERT% function to change data formats. The following example converts from IEEE to Motorola floating point format.

LIST! (1) = BYTE_COUNT% + 00020000H

Such pairs of elements may be repeated as often as necessary with the only limitation being that UDP messages may not exceed 1472 bytes and raw Ethernet messages may not exceed 1500 bytes.

The list is terminated by a data pointer with a value of zero.

This function receives data from socket SN% into memory pointed to by the list. All pointers must reference variables defined as I/O. Pointers may not reference variables defined in the Common Memory Module or AutoMax Processor. If a TCP socket is specified, it must be connected first.

A socket can be selected as blocking or non-blocking. If the socket is designated as blocking and no data has come in, the task will be suspended until data arrives. If the socket is designated as non-blocking and no data has come in, the RECVL% command will return with the error No message waiting. The default mode is blocking.

Values Returned:

>0	Number of bytes transferred
-2	ENI not initialized
-9	No buffer space
-15	Bad socket number
-17	Message too long
-18	Zero length
-19	Illegal pointer
-25	Not a double integer array
-26	Not a single dimension array
-27	Bad array format
-30	Odd number of bytes in 1st parameter
-37	Pointing to on-board memory
-101	No message waiting
-102	Socket not connected

For example:

```
RECV_LEN% = RECVL%( SN%, NETWORK_LIST! )
```

4.6.10 SETSOCKOPT% Function

Format:

SETSOCKOPT%(sn%, opnum%, opval%)

where:

sn%	is the number of socket whose option you want to set.
opnum%	is the number of the option to set.
opval%	is the value to write into the ENI.

This function is used to select different modes of operation. OPNUM% selects which option to change, and OPVAL% selects the mode of operation.

Options	OPNUM%	OPVAL%	Description
"Keep Alive"	0008h	0	Keep alive is disabled (Default)
		1	Keep alive is enabled

This option is only used on TCP sockets. When enabled, the ENI will periodically send an empty message to maintain the connection. If this option is not used and a frame is not received within 8 minutes, the ENI will assume the connection has been broken and it will close this socket.

"Linger"	0080h	0	Linger is disabled (Default)
		1	Linger is enabled

This option is only used on TCP sockets to select how the SHUTDOWN function will operate. When linger is enabled, the socket will wait until remote SHUTDOWN% is completed before shutting down.

"Non Blocking" (Default)	0200h	0	Non Blocking is disabled
		1	Non Blocking is enabled

This option is used to select how the RECV% and RECVL% function will operate. If Non blocking is enabled and no message has arrived for the RECV% or RECVL%, control is returned to the application program and an error code -101 is returned by the RECV% or RECVL%.

Values Returned:

1	Success
-2	ENI not initialized
-6	Did not set option
-9	No buffer space
-15	Bad socket number
-20	Bad option number
-21	Bad option value
-40	No available buffer

For example, to set the socket to nonblocking:

```
STATUS% = SETSOCKOPT%( SN%, 0200h, 1 )
```

4.6.11 GETSOCKOPT% Function

Format:

```
GETSOCKOPT%( sn%, opnum%, opval% )
```

where:

sn% is the number of socket whose option you want to read.

opnum% is the number of the option to read.

opval% is the name of the option variable where the current value is written.

This function is used to examine what modes of operation are selected. OPNUM% selects which option to look at, and OPVAL% displays the current status.

Options	OPNUM%	OPVAL%	Description
"Keep Alive"	0008h	0	Keep alive is disabled (Default)
		1	Keep alive is enabled

This option is only used on TCP sockets. When enabled, the ENI will periodically send an empty message to maintain the connection. If this option is not used and a frame is not received within 8 minutes, the ENI will assume it has been broken.

"Linger"	0080h	0	Linger is disabled (Default)
		1	Linger is enabled

This option is only used on TCP sockets to select how the SHUTDOWN function will operate. When linger is enabled and there are messages in any transmit or receive queues the ENI will process those messages before doing the shutdown.

"Non Blocking"	0200h	0	Non Blocking is disabled (Default)
		1	Non Blocking is enabled

This option is used to select how the RECV% and RECVL% function will operate. If Non Blocking is enabled and no message has arrived for the RECV% or RECVL%, control is returned to the application program and an error code -101 is returned by the RECV% or RECVL%.

"Connected"	0800h	-102	Socket not connected
		1	Socket connected

This option is only used on TCP sockets. It allows the application program to test if a connection is established without doing a SEND% or RECV%.

Values Returned:

1	Success
-2	ENI not initialized
-6	Bad no. get option
-9	No buffer space
-15	Bad socket number
-20	Bad option number
-40	No available buffer
-103	No buffer space

For example, to test if the socket is connected:

```
STATUS% = GETSOCKOPT%( SN%, 0800h, OPTION_VALUE% )
```

4.6.12 SHUTDOWN% Function

Format:

SHUTDOWN%(sn%)

where:

sn% is the number of the socket for which the connection should be terminated.

This function closes the socket to allow it to be re-used at a later time.

TCP sockets need to be shut down at only one end. Either the active or passive side may close the connection. The other side will automatically shut down. UDP and raw Ethernet sockets need to be shut down at both ends.

Values Returned:

1	Success
-2	ENI not Initialized
-7	No free channel
-15	Bad socket number
28	Socket closed by destination
-40	No available buffer

For example:

STATUS% = SHUTDOWN%(SOCKET_NUM%)

4.6.13 READVAR% Function

Format:

READVAR%(vn$, value)

where:

vn$ is a string expression for the name of the variable to read. It can be a boolean, integer, double integer, real or string, or an array of these types. Only one-dimensional arrays are allowed.

value is the variable where the value read is written.

This function accepts a variable name as a string expression and returns the value in variable VALUE. The string that defines the variable name must have a suffix as follows:

@	Booleans
%	Integers
!	Double Integer
$	Strings
No suffix	Reals

If specifying an array element, the subscript must be after the data type character if there is one. Only common variables can be accessed.

Values Returned:

1	Success
-22	Variable not found
-23	Data type mismatch

For example:

```
VARIABLE_NAME$ = "SET_POINTS(17)"
STATUS% = READVAR%( VARIABLE_NAME$, VALUE )
```

4.6.14 WRITEVAR% Function

Format:

WRITEVAR%(vn$, value)

where:

vn$ is a string expression for the name of the variable to write to. It can be a boolean, integer, double integer, real or string, or an array of these types. Only one-dimensional arrays are allowed.

value is the variable that has the value to write.

This function accepts a variable name as a string expression and a value to write into the variable. The string that defines the variable name must have a suffix as follows:

@	Booleans
%	Integers
!	Double integers
$	Strings
No suffix	Reals

If specifying an array element, the subscript must be after the data type character if there is one.

If the data type of the variable, as defined in the string vn$, is different than that of VALUE, an error is generated. Only common variables can be accessed.

Values Returned:

1	Success
-22	Variable not found
-23	Data type mismatch
-24	Variable forced

For example:

```
VARIABLE_NAME$ = "SET_POINTS(17)"
VALUE = 12.345
STATUS% = WRITEVAR%( VARIABLE_NAME$, VALUE )
```

4.8.15 FINDVAR! Function

Format:

FINDVAR!(varname$)

where:

varname$ is a string expression for the name of the variable to find.

This function accepts a variable name as a string expression and returns a pointer to that variable. This may then be used in the SENDL% and RECVL% functions.

@	Booleans
%	Integers
!	Double integers
$	Strings
No suffix	Reals

If specifying an array element, the subscript must be after the data type character if there is one.

Values Returned:

>0	Pointer to Variable
-22	Variable not found

For example, to find a pointer to XYZ%(10):

```
VARIABLE_NAME$ = "XYZ%(10)"
POINTER! = FINDVAR!( VARIABLE_NAME$ )
```

4.6.16 CONVERT% Function

Format:

CONVERT% (src_variable, src_subscript, dest_variable, &
dest_subscript, num_of_words, mode)

where:

src_variable is the variable that selects where to get data from. This parameter may be a scalar or an array of any data type. If src_variable is an array, it should be the base name and any data type character only.

src_subscript is only used if the src_variable is an array. It determines where in the array to begin reading. If not an array, the value should be 0.

dest_variable is the variable that selects were to move the data. This parameter may be a scalar or an array of any data type. If dest_variable is an array, it should only be the base name and any data type character.

dest_subscript is only used if destination_variable is an array. It determines where in the array to begin writing. If not an array, the value should be 0.

num_of_words selects the number of words to move.

mode determines the mode of operation.

VALUE	FUNCTION
0	Move data with no change in format
1	Convert from Motorola Floating Point to IEEE format
2	Convert from IEEE Floating Point to Motorola format
4	Word swap (0102H to 0201H)
8	Long word swap (01020304H to 04030201H)
9	Motorola to IEEE followed by long word swap
10	Long word swap followed by IEEE to Motorola

All other values are illegal

This function is used to convert between data formats used by AutoMax and data formats used by other computers.

Values Returned:

1	Success
-28	Array is not single dimension
-32	Beyond end of array
-33	Illegal mode value
-34	Zero number of words
-35	Odd number of words on long word swap
-36	Number of words > dest data type when dest memory is on CPU

For example, to move 30 real numbers beginning at SRC_ARRAY(10) to DST_ARRAY(20) converting from Motorola to IEEE and inverting the byte order:

STATUS% = CONVERT%(SRC_ARRAY, 10, DST_ARRAY, 20, 60 ,9

5.0 DIAGNOSTICS AND TROUBLESHOOTING

Upon power-up, the ENI module will automatically run its on-board diagnostics. After approximately 10 seconds, the "OK" LED should turn on. The "OK" LED will turn off while the initialization procedure is run, and will turn on at its completion. It will also turn off if a STOP ALL command is executed, and will remain off until the ENI is re-initialized.

Software errors are indicated by error codes returned by BASIC functions. Your application software must check for these error codes.

Hardware errors are indicated by the LED on the faceplate turning off. Follow the procedures below in the order listed to isolate a hardware problem. If none of the procedures listed below isolates the problem, the module is not user-serviceable.

Step 1. Check the LEDs on the Power Supply module faceplate. Any problems with the Power Supply module or the rack can usually be isolated by observing the condition of the LEDs on the Power Supply module faceplate. Refer to the AutoMax Power Supply Module and Racks Instruction Manual (J-3670) for detailed procedures for troubleshooting the Power Supply.

Step 2. Turn off power to the rack. Check the seating of the ENI. Use a screwdriver to loosen the screws that hold the module in the rack. Remove the module from the slot in the rack, and then reinsert it. Turn on power to the rack.

Step 3. Check all cable connections of the ENI to the Ethernet network.

Appendix A

Technical Specifications

Ambient Conditions

- Storage temperature: −40°C – 85°C
- Operating temperature: 5°C – 50°C
- Humidity: 5–90% non-condensing

Dimensions

- Height: 11.75 inches
- Width: 1.25 inches
- Depth: 7.375 inches
- Weight: 2 lbs.

System Power Requirements

- Input Voltage
- +5 VDC: 5000mA
- +12 VDC: 500mA
- −12 VDC: 100mA

Maximum Transceiver Cable Length

- 50 meters (164 feet)

Appendix B

Connecting the ENI to the Transceiver

Ethernet Version 1.0, Version 2.0, and IEEE 802.3 standards all require different style transceiver cables. Since cable grounding is done at the ENI end of the cable, proper matching is critical.

If you wish to fabricate your own cable, you can do so following the directions below.

WARNING

THE FOLLOWING INSTRUCTIONS ARE INTENDED ONLY TO ALLOW FABRICATION OF PROPER CONNECTIONS BETWEEN RELIANCE EQUIPMENT AND USER-PROVIDED DEVICES. THE USER MUST READ AND UNDERSTAND ALL APPLICABLE INSTRUCTION MANUALS PRIOR TO FABRICATING THE CABLE. FAILURE TO OBSERVE THIS PRECAUTION COULD RESULT IN BODILY HARM.

1. Cut a suitable length of Ethernet/IEEE 802.3 Transceiver cable. Maximum cable length is 50 meters (164 feet).
2. Follow the connector manufacturer's instructions to make cable connections using the figure below.
3. Check for grounds, shorts, and continuity using an Ohm meter.

ENI and Pin Number	Signal Name
1	Shield (Ethernet 1.0, 2.0)(Ground)
2	Collision Presence –
3	Transmit –
4	Ground
5	Receive +
6	Power Return (Ground)
7	Reserved
8	Ground
9	Collision Presence –
10	Transmit –
11	Reserved
12	Receive –
13	Power (+12VDC Fused)
14	Ground
15	Reserved

Figure B-1 - Transceiver Cable Pin Connections

Appendix C

Error Code Summary

Codes for Errors found by ENI		Returned by
−1	ENI failed self test	ENI_INIT
−2	ENI not initialized	SOCKET, BIND, CONNECT, ACCEPT SEND, SENDL, RECV, RECVL SETSOCKOPT, GETSOCKOPT SHUTDOWN
−3	Did not create socket	SOCKET
−4	Did not bind socket	BIND
−5	Did not set option	SETSOCKOPT
−6	Did not get option	GETSOCKOPT
−7	Did not accept	ACCEPT
−8	Bus error	ENI_INIT
−9	No buffer space	SOCKET, BIND, CONNECT, ACCEPT SEND, SENDL, RECV, RECVL SETSOCKOPT, GETSOCKOPT
Codes for errors found by AutoMax		
−10	Error locating interrupts	ENI_INIT
−11	Bad slot number	ENI_INIT, SOCKET
−12	Bad Internet address	ENI_INIT, CONNECT
−13	Total number of sockets > 64	ENI_INIT
−14	Bad socket type	SOCKET
−15	Bad socket number	BIND, CONNECT, ACCEPT SEND, SENDL, RECV, RECVL SETSOCKOPT, GETSOCKOPT SHUTDOWN
16	Not a TCP socket	ACCEPT
−17	Message too long	SEND, SENDL, RECV, RECVL
−18	Zero length for non array	SEND, SENDL, RECV, RECVL
−19	Illegal pointer	SENDL, RECVL
−20	Bad option number	SETSOCKOPT, GETSOCKOPT
21	Bad option value	SETSOCKOPT
−22	Variable not found	READVAR, WRITEVAR, FINDVAR
−23	Data Type Mismatch	READVAR, WRITEVAR
24	Variable Forced	WRITEVAR
−25	Not a double integer array	RECVL, SENDL
−26	Not a single dimension array	SEND, SENDL, RECV, RECVL, CONVERT
−27	Bad array format	RECVL, SENDL
28	Socket closed by destination	SHUTDOWN
−29	Max size of strings are not equal	RECV
−30	Odd number of bytes in list parameter	SENDL, RECVL
−31	Max size of string < recv size of string	RECV
−32	Beyond end of array	CONVERT, SEND, RECV
−33	Illegal mode value	CONVERT
−34	Zero number of words	CONVERT
−35	Odd number of words on long word swap	CONVERT
−36	Number of words > dest data type when dest is on CPU	CONVERT
−37	Pointing to on-board memory	RECVL
Warning Status Codes (from ENI — not critical)		
−101	No Message Waiting	RECV, RECVL
−102	Socket Not Connected	CONNECT, SEND, SENDL, RECV, RECVL

Appendix D

Glossary

connection

The path between two protocol modules. In Internet, a connection extends from a TCP module on one machine to a TCP module on another.

CSMA/CD

Carrier Sense Multiple Access with Collision Detection. A characteristic of network hardware that allows stations to contend for access to a transmission medium by listening to see if it is idle.

Ethernet

The name given to a popular local area packet-switched network technology invented by Xerox PARC in the early 1970s.

port

The abstraction that transport protocols use to distinguish among multiple destinations within a given host computer. Internet protocols identify ports using small positive integers. Usually, the operating system allows an application program to specify which port it wants to use.

protocol

A formal description of message formats and the rules two or more machines must follow to exchange those messages.

Raw Ethernet

A transmission protocol that allows a message to be broadcast only, using the Ethernet address. There is no acknowledgment of the message being received. Cyclical Redundancy Check (CRC) is used for transmission error detection.

socket

The abstraction provided by Berkeley 4.3 BSD UNIX that allows a process to access the Internet. A process opens a socket, specifies the service desired, binds the socket to a specific destination, and then sends or receives data.

TCP

Transmission Control Protocol. TCP allows you to send a message to a specific internet address and socket. There is an acknowledgment sent back to the source that the message was received. Cyclical Redundancy Check (CRC) is used for transmission error detection.

TCP/IP

(Transmission Control Protocol/Internet Protocol) The Internet standard transport level protocol that provides the reliable, full duplex, stream service on which many application protocols depend. It allows a process on one machine to send a stream of data to a process on another. It is connection-oriented in that before transmitting data, a connection must be established. Software implementing TCP usually resides in the operating system and uses IP protocol to transmit information across the Internet.

Appendix D

Glossary (Continued)

Transceiver

A device that connects a host interface to a local area network (e.g., Ethernet).

UDP

User Datagram Protocol. The Internet standard protocol that allows an application program on one machine send a message to an application program on another machine. UDP messages include a protocol port number, allowing the sender to distinguish among multiple destinations on the remote machine. It also includes a checksum over the data being sent.

RELIANCE CONTROLS
DOCUMENTATION IMPROVEMENT FORM

Document Number: ____________________

Page Number(s): ____________________

Comments: *(Please give chapters, page numbers or specific paragraphs that the change will affect. Include markups from the document or attach additional pages if necessary.)*

____

____

____

____

____

____

____

____

____

What will this improvement suggestion provide?

____

Originator: ____________________ City: ____________________ State: ______ ZIP: ________

Company: Phone: ( )

Address: ____ Date: ____________

Technical Writing Internal Use: **Follow-Up Action:**

Writer: ____________________ Date: ____________________

Thank you for your comments . . .

RELIANCE ELECTRIC

For additional Information
1 Allen-Bradley Drive
Mayfield Heights, Ohio 44124 USA
Tel: (800) 241-2886 or (440) 646-3599
http://www.reliance.com/automax

www.rockwellautomation.com

Corporate Headquarters
Rockwell Automation, 777 East Wisconsin Avenue, Suite 1400, Milwaukee, WI, 53202-5302 USA, Tel: (1) 414.212.5200, Fax: (1) 414.212.5201

Headquarters for Allen-Bradley Products, Rockwell Software Products and Global Manufacturing Solutions
Americas: Rockwell Automation, 1201 South Second Street, Milwaukee, WI 53204-2496 USA, Tel: (1) 414.382.2000, Fax: (1) 414.382.4444
Europe/Middle East/Africa: Rockwell Automation SA/NV, Vorstlaan/Boulevard du Souverain 36, 1170 Brussels, Belgium, Tel: (32) 2 663 0600, Fax: (32) 2 663 0640
Asia Pacific: Rockwell Automation, 27/F Citicorp Centre, 18 Whitfield Road, Causeway Bay, Hong Kong, Tel: (852) 2887 4788, Fax: (852) 2508 1846

Headquarters for Dodge and Reliance Electric Products
Americas: Rockwell Automation, 6040 Ponders Court, Greenville, SC 29615-4617 USA, Tel: (1) 864.297.4800, Fax: (1) 864.281.2433
Europe/Middle East/Africa: Rockwell Automation, Brühlstraße 22, D-74834 Elztal-Dallau, Germany, Tel: (49) 6261 9410, Fax: (49) 6261 17741
Asia Pacific: Rockwell Automation, 55 Newton Road, #11-01/02 Revenue House, Singapore 307987, Tel: (65) 6356-9077, Fax: (65) 6356-9011

Publication J-[illegible] – February 1995