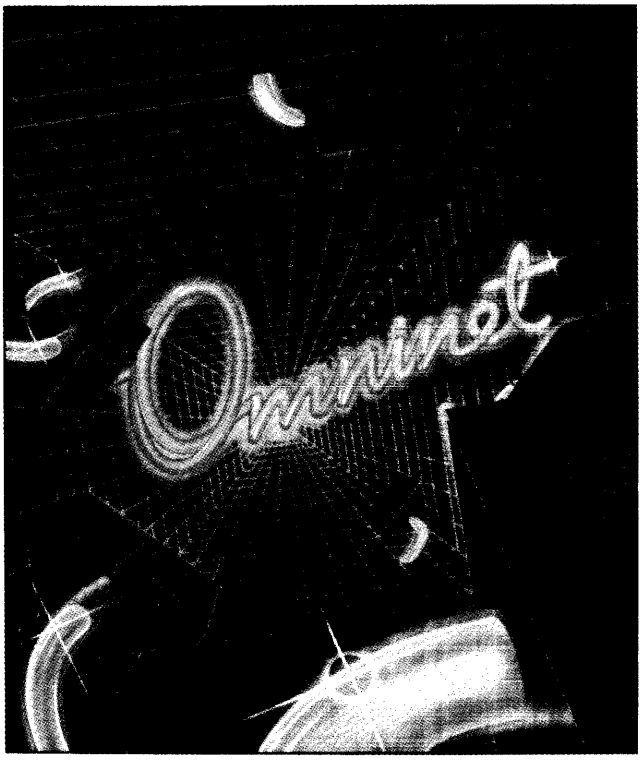
General Technical Information

OMNINET Local Area Network



CORVUS

LIMITED WARRANTY

Corvus warrants its hardware products against defects in materials and workmanship for a period of 180 days from the date of purchase from any authorized Corvus Systems dealer. If Corvus receives notice of such defects during the warranty period, Corvus will, at its option, either repair or replace the hardware products which prove to be defective. Repairs will be performed and defective parts replaced with either new or reconditioned parts.

Corvus software and firmware products which are designed by Corvus for use with a hardware product, when properly installed on that hardware product, are warranted not to fail to execute their programming instructions due to defects in materials and workmanship for a period of 180 days. If Corvus receives notice of such defects during the warranty period, Corvus does not warrant that the operation of the software, firmware or hardware shall be uninterrupted or error free.

Limited Warranty service may be obtained by delivering the product during the 180 day warranty period to Corvus Systems with proof of purchase date. YOU MUST CONTACT CORVUS CUSTOMER SERVICE TO OBTAIN A "RETURN AUTHORIZATION CODE" PRIOR TO RETURNING THE PRODUCT. THE RAC (RETURN AUTHORIZATION CODE) NUMBER ISSUED BY CORVUS CUSTOMER SERVICE MUST APPEAR ON THE EXTERIOR OF THE SHIPPING CONTAINER. ONLY ORIGINAL OR EQUIVALENT SHIPPING MATERIALS MUST BE USED. If this product is delivered by mail, you agree to insure the product or assume the risk of loss or damage in transit, to prepay shipping charges to the warranty service location and to use the original shipping container. Contact Corvus Systems or write to Corvus Customer Service, 2100 Corvus Drive, San Jose, CA, 95124 prior to shipping equipment.

ALL EXPRESS AND IMPLIED WARRANTIES FOR THIS PRODUCT, INCLUDING THE WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, ARE LIMITED IN DURATION TO A PERIOD OF 180 DAYS FROM DATE OF PURCHASE, AND NO WARRANTIES, WHETHER EXPRESS OR IMPLIED, WILL APPLY AFTER THIS PERIOD. SOME STATES DO NOT ALLOW LIMITATIONS ON HOW LONG AN IMPLIED WARRANTY LASTS, SO THE ABOVE LIMITATIONS MAY NOT APPLY TO YOU.

IF THIS PRODUCT IS NOT IN GOOD WORKING ORDER AS WARRANTED ABOVE, YOUR SOLE REMEDY SHALL BE REPAIR OR REPLACEMENT AS PROVIDED ABOVE. IN NO EVENT WILL CORVUS SYSTEMS BE LIABLE TO YOU FOR ANY DAMAGES, INCLUDING ANY LOST PROFITS, LOST SAVINGS OR OTHER INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF OR INABILITY TO USE SUCH PRODUCT, EVEN IF CORVUS SYSTEMS OR AN AUTHORIZED CORVUS SYSTEMS DEALER HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, OR FOR ANY CLAIM BY ANY OTHER PARTY.

SOME STATES DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES FOR CONSUMER PRODUCTS, SO THE ABOVE LIMITATIONS OR EXCLUSIONS MAY NOT APPLY TO YOU.

THIS WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS, AND YOU MAY ALSO HAVE OTHER RIGHTS WHICH MAY VARY FROM STATE TO STATE.

CORVUS SYSTEMS

GENERAL TECHNICAL INFORMATION

Omninet Local Area Network

GENERAL TECHNICAL INFORMATION

Omninet
Local Area Network

Part Number: 7100-06614-01
Release Date: October 1984
Revision: A

FCC WARNING

This equipment has been tested with a Class A computing device and has been found to comply with Part 15 of FCC Rules. Operation in a residential area may cause unacceptable interference to radio and TV reception requiring the operator to take whatever steps are necessary to correct the interference.

NOTICE

Corvus Systems, Inc. reserves the right to make changes in the product described in this manual at any time without notice. Revised manuals will be published as needed and may be purchased from authorized Corvus Systems dealers.

Corvus Systems, Inc.
2100 Corvus Drive
San Jose, CA. 95124

Telephone: (408) 559-7000
TELEX: 278976

TABLE OF CONTENTS

SCOPE

This manual contains an introduction to the Omninet (TM) local area network as well as an in-depth explanation of its technical aspects. It describes both hardware and internal software but does not discuss high level networking software packages such as the Corvus Constellation (R) in any detail.

Readers who wish to design their own high level networking software for Omninet will benefit most from this manual. These users will find that the guide provides all the information that they need to program original Omninet protocols.

This page intentionally left blank.

CONVENTIONS

Throughout this manual, computer memory addresses and the contents of those addresses appear in two digit hexadecimal format. Thus, wherever a pair of numeric or alphanumeric digits appears without other explanation, they can be assumed to represent a hexadecimal value. Octal representations of numeric values also appear in the manual but are so noted wherever they occur.

This page intentionally left blank.

INTRODUCTION 1

As its name implies, Omninet is a system for interconnecting any and all personal computers within a certain area into an efficient communications network. Once this network is established, users at different workstations working with personal computers of various manufacturers can communicate not only with each other but also with shared peripheral equipment such as disk drives and printers. The network itself supports up to 64 host computers but links can be established through the network to other Omninet networks or to the larger computer networks of other manufacturers.

Corvus Systems provides its own high level networking software for computers on Omninet. This software, called Constellation, allows single user operating systems such as UCSD Pascal (TM), CP/M (R), MS(TM)-DOS and others to share mass storage, printers and other peripherals on the Omninet network. However, since Constellation is only one of many possible ways in which to use Omninet, this manual will not discuss it in any detail. This manual will instead concentrate on Omninet hardware and the internal software of Omninet's intelligent controllers.

Given below is a brief summary of the functions and considerations of each of the seven layers of the ISO Seven Layer Network Model. The Omninet package provides the services of the bottom four layers of the ISO seven layer network model while Corvus Constellation software provides the services of the top three layers.

Layer 1, Physical Layer -- the unit of exchange is the bit; considerations are voltage or current levels, signal timing, connector pin assignments, etc.

Layer 2, Data Link Layer -- the unit of exchange is the frame, independent of any data content; considerations are error detection, link access, contention, and collision avoidance.

Layer 3, Network Layer -- the unit of exchange is the packet; considerations are message/packet conversion, verification of receipt, routing within the local network.

Layer 4, Transport Layer -- the unit of exchange is the message; considerations are message ordering, host to host level communication, frame acknowledgement, retransmission on errors, duplicate frame detection, etc.

Layers 5-7, Session, Presentation and Application Layers -- considerations are applications oriented, such as billing, encryption, code conversion, terminal compatibility, data compression, resource allocation, etc.

This manual provides an in-depth look at the structure and functions of the Omninet network. Chapter 2 discusses the details of message transmission and reception, Chapter 3 examines the network command mechanisms, and Chapter 4 presents a technical overview of the Transporter (TM), the intelligent network interface device.

A number of appendices follow the main text of the document. The first appendix on the anatomy of a packet is of general interest and will be helpful to any Omninet user. Appendices B through M provide details on specific Transporters. These will be of interest only to users of the particular computer involved.

OMNINET NETWORK STRUCTURE

TRANSFER MEDIUM

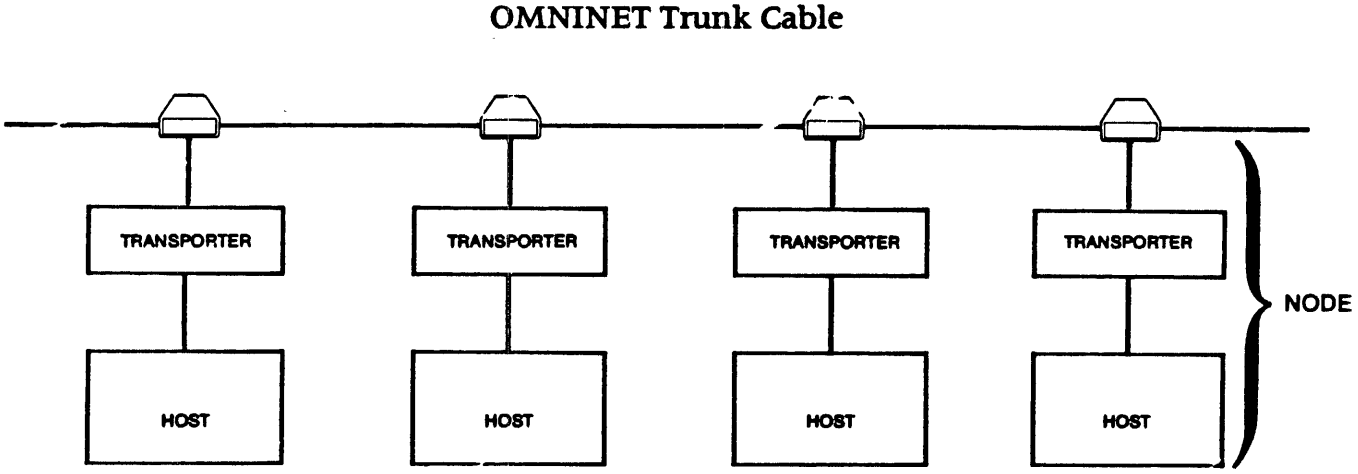
Omninet is implemented as a shared-bus system, where the bus is a twisted pair of wires. What this means is that every device on the Omninet network is attached to a cable consisting of two wires twisted together over their entire lengths. All information exchanges between devices take place over this cable.

There are three main reasons for choosing twisted pair as the communications medium for Omninet. First, the cost per foot of cable is low. Second, a nontechnical person can install an Omninet interface with a few simple tools; the task is similar to hooking up a hi-fi speaker. Third, the twisted pair wire neither absorbs nor radiates more radio frequency interference (RFI) than does coaxial cable. RS-422 twisted pair cable was chosen in

particular because it allows for high speed, long cable runs, superior noise rejection, low RFI and high reliability.

BASIC CONFIGURATION

Connections to the Omninet trunk cable are made through devices called Transporters. Every device on the network, whether it is a computer, a disk drive, or a printer, has its own Transporter to handle communication over Omninet. The combination of a host device and its Transporter connected to the cable is known as a node. There may be as many as 64 nodes attached to a network.

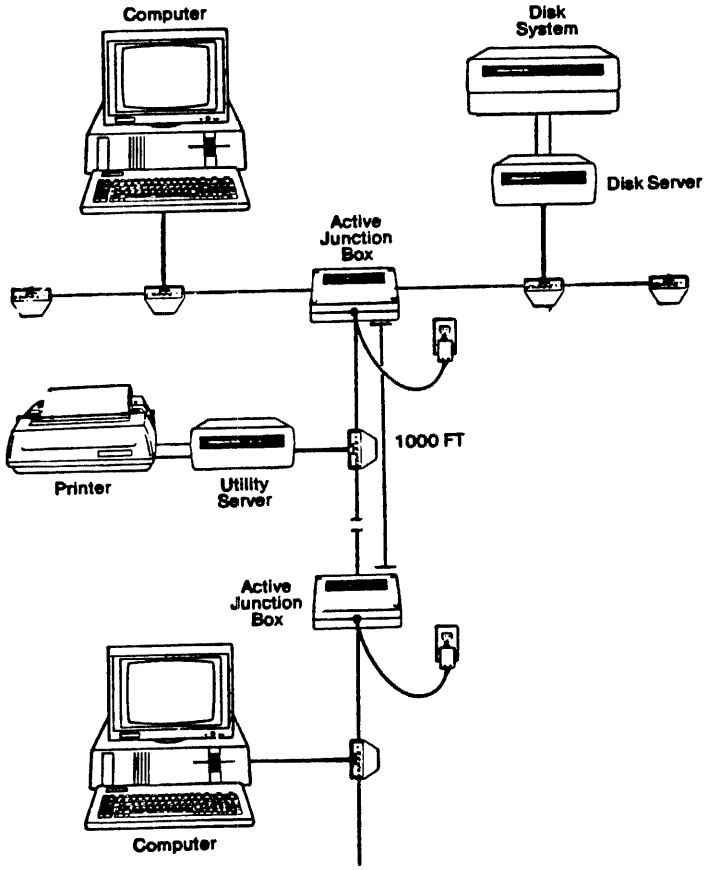


Basic Network Configuration

The diagram above shows all the network nodes connected to the single Omninet trunk cable in sequence. While this is the most common configuration for Omninet, it is also possible to create T-connections splitting the Omninet trunk into several branches. This is done using the Omninet Active Junction Box at the site of the splice.

THE ACTIVE JUNCTION BOX

The Active Junction Box is a signal repeater whose presence on the line allows greater network cable lengths. Using unshielded twisted pair cable, Omninet Transporters may be up to 2000 feet apart. However, if the length of the Omninet trunk is to exceed 2000 feet, Active Junction Boxes should be installed at 1000 foot intervals along the cable. The network length limit expands to 9000 feet when Active Junction Boxes are used to create branches in the cable, but no two nodes should ever be farther than 4000 feet apart. The diagram below shows a network in which Active Junction Boxes are used both to increase the length of trunk cable runs and to create a branch in the trunk.



Active Junction Box Application

For more information on the Active Junction Box, see the *Active Junction Box Installation Guide.*

TRANSPORTERS

Transporters are the intelligent network interface cards which handle the mechanics of communication between network nodes. Every device on Omninet is attached to the trunk cable through a Transporter. These Transporters accept communications commands from their hosts and perform the necessary operations in order to actually send and receive messages over the network trunk.

All Transporters on the network operate in exactly the same way so that there is no master network controller. Network management is handled by the individual Transporters using a distributed control mechanism called Carrier-Sense Multiple Access (CSMA). When a Transporter has a message to send it waits until it detects the Omninet bus idle state and then assumes control of the bus for the time it takes to transmit its message. Since every Transporter is connected to the bus at all times, all Transporters see all communications but accept only those which are addressed to them. Unique network addresses are set with switches on each Transporter at the time of installation.

In order to reduce the software burden placed on the host computer, the Transporter performs many of the high level network tasks that are usually the host's responsibility. Generation and reception of message acknowledgements, message retransmission when messages are not acknowledged, and detection of duplicate messages are all handled automatically by the Omninet Transporter. Host computers invoke network communication by issuing high level commands to their Transporters. They are not disturbed during command execution but are only notified when a command is completed.

MESSAGES

Messages are the units of information which Omninet host devices send and receive. They may contain information such as electronic mail memos or portions of disk reads and writes and they may be sent by any node on the network to any other node on the network. Because hosts consider messages the unit of information transfer, host commands to their Transporters deal only with messages as

the objects of concern. It is the responsibility of the Transporters and not of the hosts to format and transmit the individual message data bytes.

A single message is transmitted directly from an originating host to a receiving host. In fact, at the same time as message data is being transferred out of a sending host's memory, it is being transferred into a receiving host's memory. Both of these transfers take place through the process of Direct Memory Access (DMA) so the overall process takes place through simultaneous DMA of both the sending and receiving hosts.

Individual message data bytes are transferred one at a time from a transmitting host to its Transporter utilizing the process of DMA. Each byte is converted into a serial bit stream as it is received from host memory and sent out onto the network as part of the current "packet" of information. Packet transfers begin when the transmitting Transporter makes a send request to its cable interface device. The Transporter then begins to transmit the packet by first sending out several flag bytes, then continuing to send out message bytes as it receives them from its host. After it has received and sent the last message byte, it closes the packet with additional flag bytes.

Data sent out on the network "travels" to, and is seen by, all network nodes. However, addressing information is contained in packet headers, and this information is read by all the active Transporters on the network. Transporters that are not intended to receive the packet take no action. The Transporter intended to receive the packet strips off the control information, converts the serial data stream back to bytes, and transfers the message data bytes to the receiving host one byte at a time as it receives them, again utilizing DMA. Network collision avoidance, error detection, error recovery, and duplicate packet detection are all handled by the Transporters without disturbing the host.

A message consists of two portions, both of which can be variable length and may be null. The portions, the user data portion and the user control portion, are usually located in different parts of the transmitting host's memory. Each host specifies to its tranporter where these two portions are to be found or, in the case of the receiving host, where each portion is to be put. Both portions are included in the same message packet but are kept separate within the packet.

SOCKETS

A socket is a location in Transporter memory that contains information used by the Transporter when it receives a message. Sockets consist primarily of pointers, which tell where in host memory to place separate message portions and information about received messages such as source host number and length. Since each host may have up to four sockets active for receiving, four separate messages may be received by a host and separately routed into memory before a new socket need be activated.

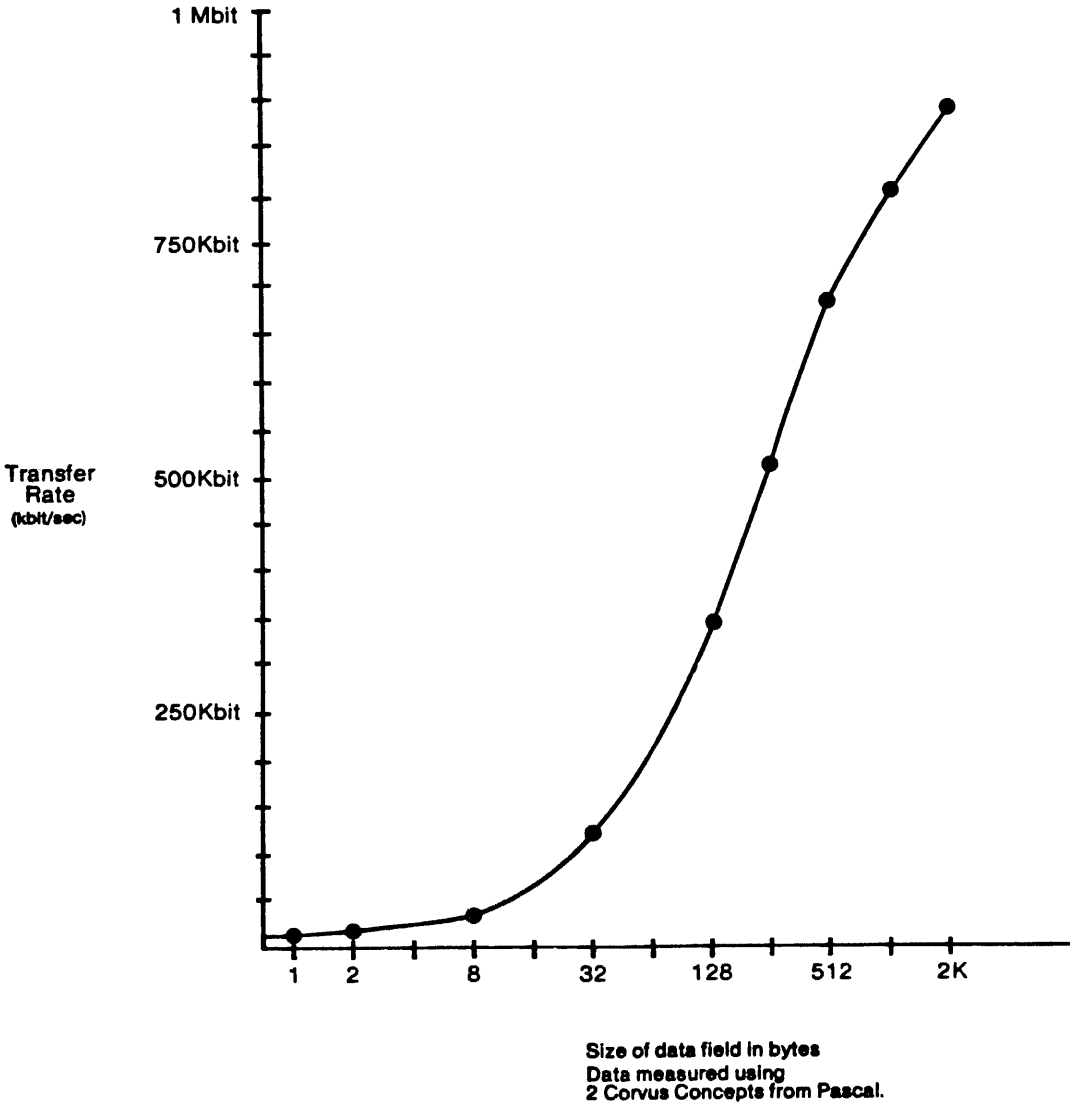
Sockets are separately addressable by transmitting hosts. This means that a transmitting host must send its message not only to a specific receiving node but also to a specific socket of that node as well. Since the receiving host must explicitly activate any socket through which it is willing to receive data, it is possible that a transmitting host may send a message to an inactive socket of the intended receiver. When this happens, the receiving Transporter acknowledges that it received a message but indicates that it was unable to use the message because it was addressed to an inactive socket.

Sockets provide Omninet with one great advantage over other network systems. Because separate message portions can be routed through a socket to different areas of host memory, it is possible to place some message information into the host system space while the remainder of the message, the user data portion, is placed in the host memory space. This eliminates any need for the host to copy message data from one part of memory to another each time a message is received.

PERFORMANCE

One measure of Omninet's performance is the number of data bytes per second which can be transmitted over the network. Omninet transmits bits at an instantaneous rate of 1,000,000 per second, and bytes are therefore transmitted at a rate of 125,000 per second. Not all of these bytes are user data bytes, however. Each message packet must contain a certain amount of control information, and transmission of this information takes time. Nevertheless, the proportion of packet data which consists of control information is usually small. In

fact, since the amount of control information within a packet is constant for each type of packet, control information becomes an increasingly smaller proportion of packets as packet size increases. As a result, the rate of user data transmission increases with increasing average packet size, and at large average packet sizes, the effect of the control information on the data transmission rate is negligible. The graph below presents the empirical relationship between average packet size and user data transmission rates on Omninet.



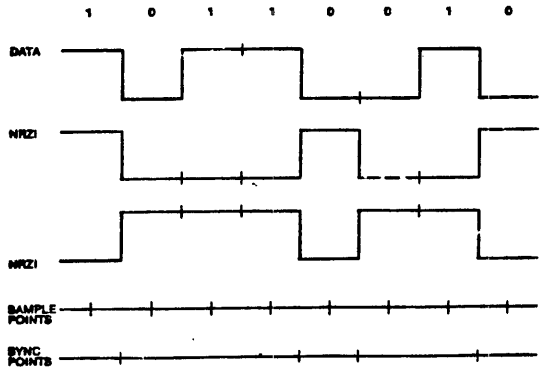
Omninet Station-to-Station Throughput vs. Packet Size

NETWORK COMMUNICATION 2

DATA TRANSMISSION

Omninet transmits data bits along the trunk cable from one Transporter to another by a process known as NRZI. Under NRZI, line transceivers generate data one bits by maintaining an unchanged voltage differential of approximately two volts across the two wires of the Omninet bus. Receiving Transporters sample the Omninet bus at regular intervals and interpret an unchanged differential as a data one bit. When a sending transceiver wishes to generate a data zero bit, it reverses the direction of the voltage differential. The receivers interpret a reversal as a data zero bit.

A sample data bit stream and the resultant NRZI output generated is shown below. Note that either of the NRZI signals shown may be generated, depending upon the state of the lines prior to the generation of the first data bit. It is the line transitions and not the polarity that convey all the information.



Data Bit Stream and NRZI Output

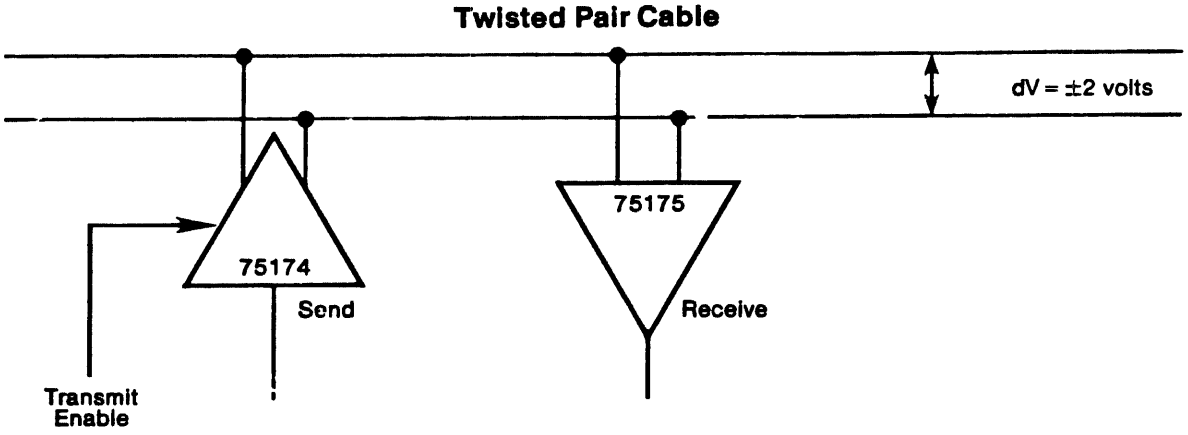
All Transporters are assumed to operate at approximately the same bit rate, so that when the line is idle the correct number of one bits are received. To ensure that

this occurs, the clocks of the receiving and sending Transporters are synchronized at every zero bit (line transition). When a transition is sensed the receiving clock is reset to sample the line every microsecond, starting half a microsecond from the zero bit transition. Synchronization points are indicated in the diagram above.

In cases where the data contains a long series of one bits, the sending Transporter will insert a zero bit after every five consecutive ones. The resulting line transition allows the receiving Transporter to resynchronize its clock so that it does not get badly out of phase with the sending clock. The inserted zero bit is filtered out by the receiving Transporter.

Data is transmitted along the network at an instantaneous rate of 1 megabit per second. The actual user data rate is somewhat lower, though, because of overhead within a packet, line acquisition delays, zero-insertion, and acknowledgement time.

The diagram below illustrates the Transporter interface to the trunk cable. The transceiver circuits chosen, the Texas Instruments SN75174 and SN75175, have built in current limiting and overload protection.



Transporter Interface to Omninet Trunk

PACKETS AND PACKET FORMATS

All information which is exchanged over Omninet travels from one Transporter to another in the form of data packets. There are several different kinds of packets, including the message packet, the acknowledgement packet,

the sync packet, and the echo packet. This section describes the purpose and format of each of these packet types.

The Advanced Data Link Controller (ADLC), an MC6854 chip that communicates directly with the line drivers, ensures that all packets transmitted have a certain amount of control information in common. This information is generated by the transmitting ADLC and is interpreted and stripped by the receiving ADLC. A packet has the general format shown below:

General ADLC Frame Format

Leading Flags. The leading flag field contains a number of flag bytes, which allow the receiving Transporter to synchronize its clock with the incoming data and obtain bit and byte alignment. A flag consists of the bit pattern 01111110. This pattern is unique because no zeros are inserted into the string of ones. Data bytes elsewhere in the packet are all subject to zero-insertion and so cannot be mistaken for flag bytes. The actual number of flags in the leading flag field is variable.

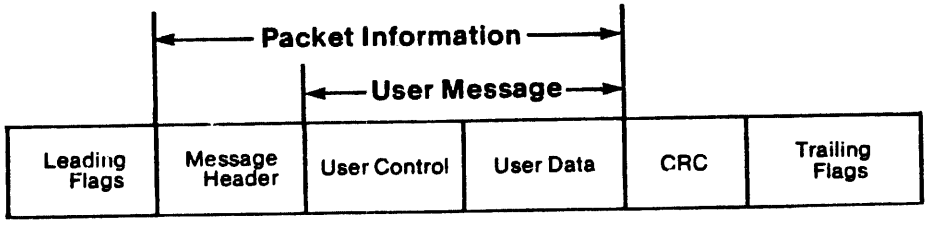
Packet Information. This field is described in the sections "Message Packet Format," "Acknowledgement Packet Format," "Sync Packet Format," and "Echo Packet Format," below.

CRC. The CRC is a 16-bit Cyclic Redundancy Code which is generated by the transmitting ADLC and validated by the receiving ADLC. This validation is just one of several error checks performed by the receiving Transporter.

Trailing Flags. Trailing flags indicate the end of an incoming packet and allow the receiving Transporter to validate the data and to start sending the message acknowledgement. As before, a flag consists of the bit pattern 01111110 and the number of flags is variable.

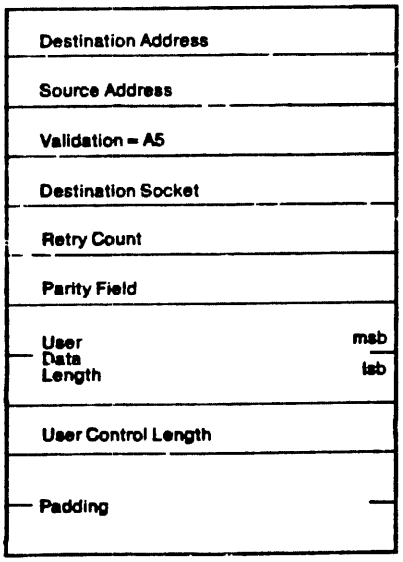
MESSAGE PACKET FORMAT

The data or message packet carries data from one host to another. In the case of Omninet, a host message corresponds one to one with a Transporter packet. The diagram below shows the layout of a message packet.



Message Packet Format

Message Header. The Message Header field contains packet addressing and control information as shown in the diagram below. Divisions into bytes are marked with by horizontal lines or dashes. The abbreviations *msb* and *lsb* stand for most significant byte and least significant byte, respectively.



Message Header Contents

Destination Address -- The Omninet node number of the intended receiver. 0-63 are legal node numbers, with

a value of 255 (FF hex) indicating a "broadcast" message. All Transporters receive messages addressed to destination 255 but none acknowledges such a message.

Source Address -- The Omninet node number of the message originator. 0-63 are legal node numbers.

Validation Byte -- A constant A5 hexadecimal (165 decimal).

Destination Socket Number -- The Transporter socket number within the receiver. Hexadecimal 80, 90, A0 or B0 are the legal values.

Retry Count -- Contains the number of times this message has been retransmitted. In the very first transmission of a message, the value is zero.

Parity Bit -- Contains 00 or 01. This is the parity bit used for duplicate packet detection.

User Data Length -- The length, in 8-bit bytes, of the user data portion of the message. Possible values range from 0 to 2047.

User Control Length -- The length, in 8-bit bytes, of the user control portion of the message. Possible values range from 0 to 255.

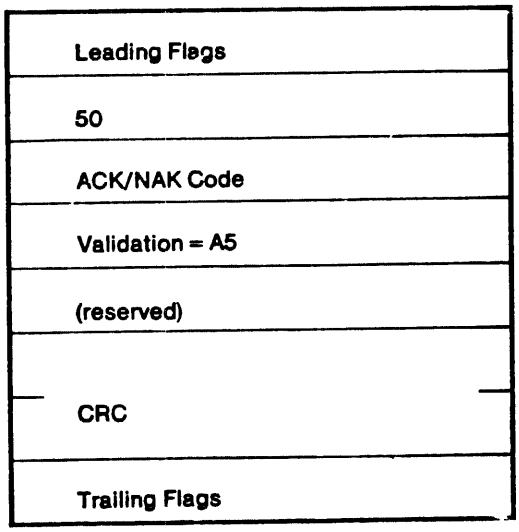
The rest of the packet information is the user message. The user message can be up to 2302 bytes long and consists of two portions, the user control and user data portions.

User Control Portion. 0 to 255 bytes as specified by the user control length byte in the packet header. The data in this portion of the message are not examined by the Transporter.

User Data Portion. 0 to 2047 bytes as specified by the user data length bytes in the packet header. The data in this portion are not examined by the Transporter.

ACKNOWLEDGEMENT PACKET FORMAT

The acknowledgement packet is sent by a receiving Transporter to acknowledge the reception of a message or echo packet from a transmitting Transporter. If a message has been received without difficulty and without detectable errors, a positive acknowledgement (ACK) is sent. If errors were detected or if for some other reason the packet could not be received, a negative acknowledgement (NAK) is sent.



Acknowledgement Packet Format

ACK/NAK Code -- Contains one of the five values shown below.

00 Message was received successfully (ACK).
81 Message user data portion was too long for buffer.
82 Message was sent to an uninitialized socket.
83 Message user control portion length did not match that setup for socket.
C0 Echo packet was received successfully. The Echo packet is described later in this chapter.

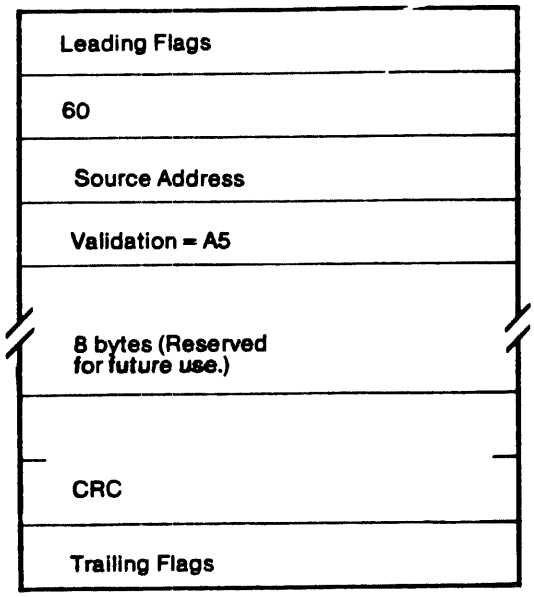
Validation Byte -- A constant A5 hexadecimal (165 decimal).

Destination field -- In Transporter code versions 6.4 and 8.A, this field is not used. In version 9.B this field holds the Omninet node address of the intended receiver of the acknowledgement (the source of the original message) plus 128 decimal. The addition of 128 turns on bit 7 of the field. Since bit 7 is always off in versions 6.4 and 8.A, this makes the versions distinguishable.

Note that in Transporter code versions 6.4 and 8.A, the acknowledgement packet contains no valid destination address yet is intended for only one host. This is possible because the acknowledgement packet is sent immediately after the message packet is received. No bus idle time is allowed between a message and its acknowledgement, so there is no chance that another Transporter could have sent a message and also be expecting an acknowledgement.

SYNC PACKET FORMAT

The sync packet is the way a Transporter makes its presence known to all of the other Transporters on the network. It causes all active Transporters to initialize their transaction parity tables, as explained later in this chapter.



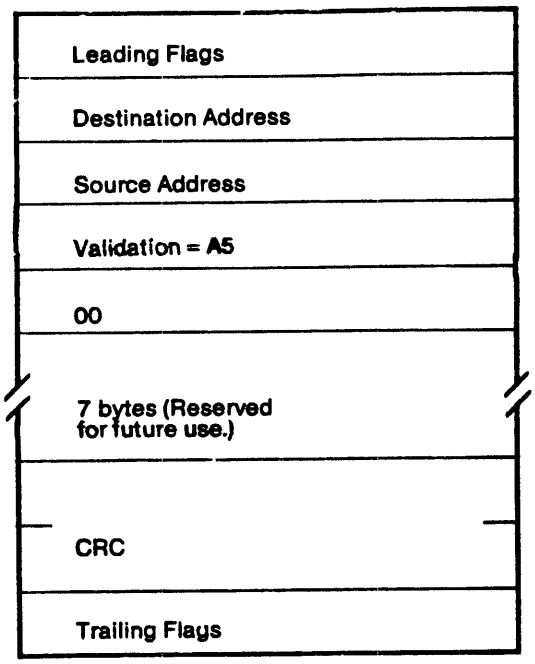
Sync Packet Format

Source Address -- The node address of the Transporter that sent the sync packet.

Validation byte -- A constant A5 hexadecimal (165 decimal).

ECHO PACKET FORMAT

The echo packet is the way one host may verify the presence of another node without disturbing the host attached to that node. If the receiving Transporter is functioning properly, it will acknowledge reception of the packet but will not inform its host that anything happened.



Echo Packet Format

Destination Address -- The node number of the Transporter which is to echo.

Source Address -- The node number of the Transporter that sent the echo packet.

Validation Byte -- A constant A5 hexadecimal (165 decimal).

PACKET TRANSMISSION: THEORY OF OPERATION

This section describes the specific details of operation of the Transporter (viewed as a black box) in its handling of packet transmission and reception.

LINE ACQUISITION

Line acquisition, or collision avoidance, involves checking the line for traffic. The transmission line is considered to be idle if there are no zeroes (transitions) for 15 bit times. In other words, 15 or

more consecutive ones (lack of transitions) defines an idle line. The Transporter ADLC chip checks the line continuously for the idle state. So long as there have been no transitions within the last 15 usec, the ADLC considers the line idle.

When a Transporter has a packet to transmit, it checks its ADLC chip to see if the line is idle. If the line is idle, the Transporter prepares to transmit its packet. This preparation takes about 10 usec. If a line transition occurs during this time, packet transmission is halted. If no transition occurs, the ADLC forces a transition on the line, the first zero of the first flag byte, and continues to transmit the packet.

If a Transporter cannot transmit either because the line is not idle or because another Transporter starts transmitting during the preparation period, the Transporter which is waiting to transmit calculates a random delay count. This count falls during line idle periods. When the count reaches zero the Transporter again attempts to acquire the line. The process is repeated until the Transporter acquires the line or until the number of retries exceeds an internal retry limit.

The random delay mechanism eliminates the problem of several Transporters all attempting to acquire the line at once as soon as it becomes available. Such an occurence would result either in collisions and garbled messages or in no Transporter managing to acquire the line at all.

PACKET TRANSMISSION

Once the transmitting Transporter has acquired the line, the packet is sent with no further attempt to detect collision with another Transporter. Nevertheless, collisions will rarely occur, as other Transporters waiting to transmit will sense that the line is busy. Zero-insertion plays an important role here because it prevents Transporters elsewhere on the line from mistaking a long string of ones in the packet data for a line idle state.

COLLISIONS

It might seem from our discussion so far that packet collisions on the line are completely impossible. Unfortunately, this is not the case. Voltage differentials can propagate over the twisted pair trunk cable at a rate no faster than the speed of light, and, as a result, there will always be a small delay between the time that an individual Transporter begins transmitting and the time that each additional Transporter on the line becomes aware of it. During this time, the inactive Transporters will continue to believe that the line is idle and may decide to become active and transmit their own messages. The result will be a collision between the packet being transmitted by the original Transporter and the packet now being transmitted by an initially inactive Transporter. Both messages will be destroyed and will have to be transmitted again.

PACKET RECEPTION

The Transporter designated as the recipient of a particular packet makes a number of checks as the packet is received. If no errors are discovered, the Transporter accepts the packet, sends an ACK packet to the sending Transporter, and DMAs the user portion(s) of the received packet to its host. If an error is discovered, the action taken depends upon the type of error. Listed below are all the error checks performed by the receiving Transporter in the order in which they are made. For each error the Transporter response is also described.

If there is no socket set up for the packet and no CRC error, the receiver waits until the packet is complete and then returns a negative acknowledgement packet (NAK code = 82 hex) to the transmitting Transporter.

If there is no buffer set up to receive the incoming packet (i.e. the destination socket has just received a message and its data buffer is full), the receiver ignores the packet and takes no other action. By not sending any kind of acknowledgement packet to the transmitter, the receiver in effect asks that the packet be sent again. This gives the host a chance to clear the socket buffer for the receipt of the retransmission.

If the user data portion of the message is too long for the buffer and there is no CRC error, the receiver waits until the packet is complete and then returns a NAK packet (NAK code = 81 hex) to the transmitting Transporter.

If the user control portion of the message is not exactly the same size as that expected by the receiving socket and there is no CRC error, the receiver waits until the packet is complete and then returns a NAK packet (NAK code = 83) to the transmitting Transporter.

If the packet generates a CRC error (possibly because of a collision), the receiver throws the packet away and takes no other action.

PACKET RETRANSMISSION

Once the packet has been sent, the transmitting Transporter waits for an ACK or NAK packet to be returned by the receiving Transporter. If no such packet is received within 20 usec., the transmitting Transporter makes an attempt to reacquire the line and retransmit the message.

As soon as an ACK or NAK packet is received, the transaction is considered to have been completed. The Transporter informs its host of the transaction status by setting the value of the return code. Return codes are discussed in Chapter 3.

DUPLICATE MESSAGE PACKET AVOIDANCE

To guard against duplicate message packets being received (when an ACK is sent by the receiver but not seen by the transmitter), there is special avoidance logic which utilizes two packet header fields as well as long-term memory in both the transmitting and receiving Transporters.

The crucial fields in the packet header are the retry count field and the transaction parity field. The retry count is always zero on the first transmission attempt for any packet and is increased on each retry. The transaction parity is a single bit and basically alternates in value between zero and one for each new

(non-retry) packet being transmitted.

Each Transporter contains a table of parity bits, one bit for each possible Transporter in the system (itself plus 63 others). During normal operation, taking two Transporters A and B as an example, the A bit in B's table and the B bit in A's table will be equal in value. This fact is used in the manner explained below to prevent duplicate packets from being received.

The transmitting Transporter (A transmitting to B) will include the complement of the B bit from its table in all packets. If the packet is positively acknowledged, Transporter A will complement the B bit in its table. Otherwise, the B bit is left unmodified.

The receiving Transporter (B receiving from A) updates the A entry in its table from the parity bit in the packet whenever it positively acknowledges a packet from A. The receiving Transporter also examines the parity bit in a packet whenever the retry count is not equal to zero. In that case, if the parity bit is equal to the A bit of the table, then a duplicate has just been received (Transporter A missed the ACK), and the receiver sends another ACK and throws the packet away.

Note that in a NAK situation neither the transmitting nor receiving Transporter updates its internal table, thus retaining mutual synchronization.

The duplicate avoidance scheme requires that all parity tables be synchronized prior to the beginning of message transfers. This is accomplished by having each Transporter send a synchronizing packet out onto the network whenever it receives an initialize command from its host (initialization commands are discussed in Chapter 3). A synchronizing packet from Transporter A tells all other Transporters in the network to set the A bit of their parity tables to zero. This packet is sent many times on initialization to assure synchronization. Initialization occurs on power-up and on hardware resets.

All active Transporters have the ability at any time during normal network operation to receive synchronizing packets from Transporters being initialized. Transporters update their parity tables on the basis of the information they receive through synchronization packets, but they do not respond with ACK packets.

This page intentionally left blank.

TRANSPORTER AND COMPUTER INTERFACE 3

This chapter describes the communications that occur between a host and its Transporter in order to perform the various functions provided by the Omninet Transporter.

The interface between host computer and Transporter allows two kinds of information exchange to take place. First, the host computer can command its Transporter to perform one of several different tasks. Second, as part of a particular command, the host can provide message data to the Transporter, or the Transporter can hand over data that it has received to its host. Both types of information exchange take place through the process of Direct Memory Access.

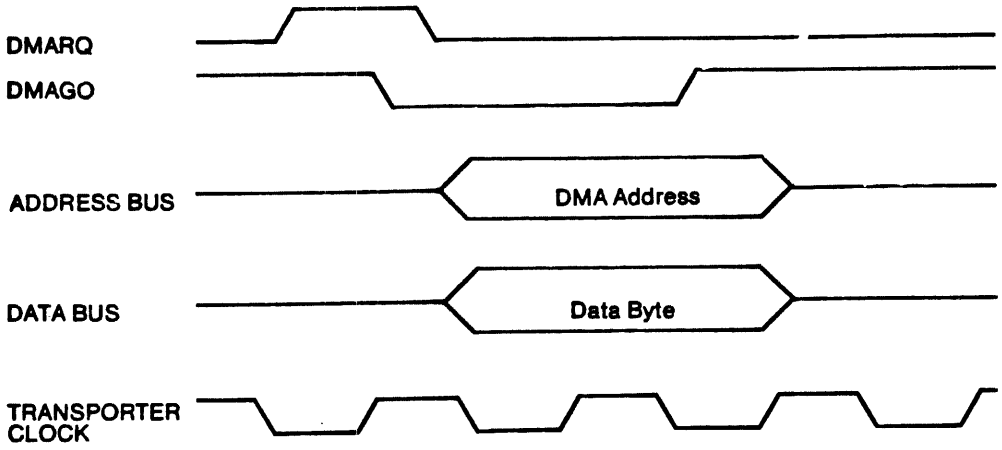
DIRECT MEMORY ACCESS

Direct Memory Access (DMA) is the process by which a Transporter receives data from or sends data to its host. Under DMA a Transporter may read data directly from host memory and write data directly to host memory without interfering with the host CPU. The host simply allows the Transporter to take control of its memory address and data buses for the length of one host memory cycle. During this time the Transporter drives the address lines and reads or writes data at its chosen address.

DMA cycles actually take place in two stages. During the first stage data is transferred between the Transporter and a data latch. In the second stage, data transfer between the data latch and the host takes place. It should be clear that on DMA transfers to the host, this ordering of the stages will work. However, on DMA transfers out of host memory, an initial byte transfer to the data latch must occur before the first full DMA cycle begins.

The host interface portion of the DMA cycle begins with a DMA request from the Transporter. The Transporter drives

its DMA request line (DMARQ) high and waits for the host to respond by driving DMAGO low. This indicates the host's willingness to allow a DMA cycle. When DMAGO goes low, the host relinquishes control of its memory address and data lines to the Transporter and the Transporter simply reads or writes data at its chosen address. If the Transporter is performing a read operation, the data is clocked in to the data latch when DMAGO goes high again. This marks the end of the DMA cycle.



DMA Timing Diagram

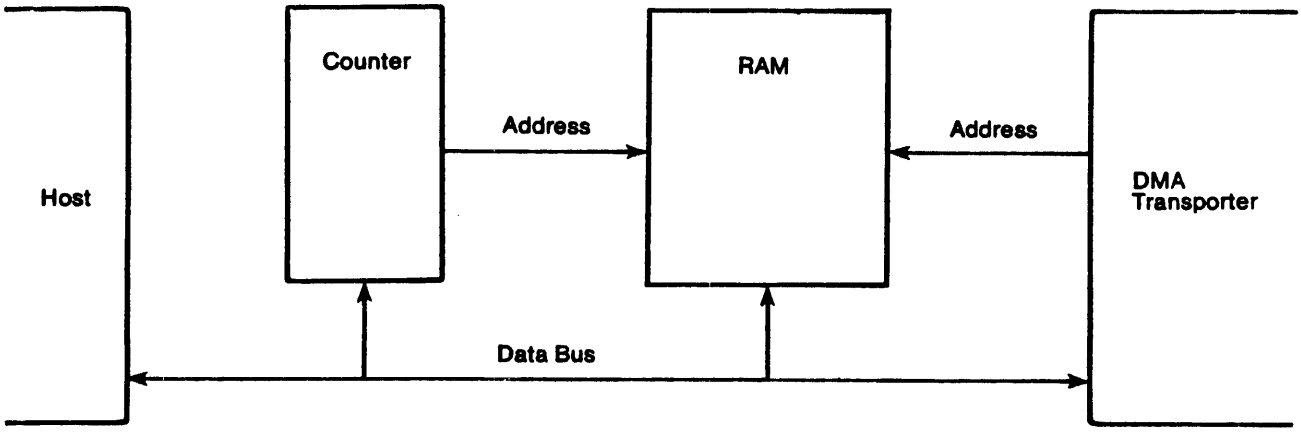
BUFFERED AND UNBUFFERED TRANSPORTERS

In most cases, the transmission of a message by a Transporter requires that the Transporter retrieve the message data from a location in host memory specified in the Send Message command vector. By the same token the reception of a message usually requires that the Transporter write the received data to the location in host memory specified in the Setup Receive command. Both of these operations take place through DMA. However, there are some computers which, for one of a number of reasons, may not be able to support the standard DMA Transporter.

The DMA Transporter cannot be supported by microprocessors which are unable to turn control of their memory address buses over to other processors. This,

quite simply, makes DMA impossible. More commonly, though, a microprocessor is able to support DMA but its memory is too slow to accommodate the DMA Transporter. During message sends and receives, the DMA Transporter must initiate a new DMA cycle every 7-8 microseconds, and only 5.9 microseconds of this is available for the second stage of the DMA operation involving the host memory. This means that if a host cannot complete its responses to a DMA request within 5.9 usec of the initiation of that request, a new cycle will begin and data will be lost. This condition is unacceptable if it happens frequently, and in such cases, or when DMA is impossible, the buffered Transporter provides a solution.

The buffered Transporter adds a certain amount of fast random access memory (RAM), usually about 4k, to the DMA Transporter. The host is allowed access to this memory and places all message and command data in it. The Transporter, meanwhile, treats this memory as if it were host memory and transfers data in and out of it utilizing normal DMA procedures. Host addressing of the buffer is performed by a counter, which the host presets to an initial value before beginning to transfer a block of data. The counter is then automatically increased by one after each byte of the block is read or written.



Buffered Transporter and Memory Access

INTRODUCTION TO COMMANDS

The second type of information exchange that takes place between a host and its Transporter is the issuance of commands. While the process of issuing a command varies slightly between DMA and buffered Transporters, the commands themselves are the same. The only difference in command execution is that with the DMA Transporter, data exchanges involve host memory, while with the buffered Transporters, these exchanges involve buffer memory.

A host computer issues a command to its Transporter by first formatting a command vector in its own memory or in the buffer and then sending the address of that vector to the Transporter. The command vector tells the Transporter what command to execute and how to execute it.

One element that is always contained within a command vector is the memory address of a result record. A result record is an area of buffer or host memory which contains command status and other types of information. This information is updated by the Transporter upon completion of the command.

One field is common to all result records no matter what their associated command. This field is a one byte return code whose value indicates current command status. When a command completes, the Transporter first updates the return code to indicate successful or unsuccessful command completion and then generates an interrupt to the host. When the host receives this interrupt it should check the return code to discover whether or not the command was successfully executed. Hosts that do not support interrupts should simply check the return code repeatedly for any change in value.

Normally a Transporter interrupts its host only when it changes the return code. However, an interrupt will also be generated if a command is issued which is unintelligible to the Transporter. In this case, the very location of the return code will probably be unclear.

There is only one instance in which more than one interrupt is generated during execution of a single command. This happens during execution of the Setup Receive command, discussed later in this chapter.

There are seven commands currently implemented on Omninet. The two most common are the Send Message command and the Setup Receive command. These two commands deal with single messages to or from another host. Transporters transmit one message per Send Message command and accept one message per Setup Receive command.

Although Send Message commands can only be issued after previous Send Message commands have been completed, multiple Setup Receive commands can be executed. Transporters may be activated via a succession of Setup Receive commands, and they can receive up to four messages in a row without an intervening command. Nevertheless, there is no command chaining capability. Hosts must issue one command at a time, and the Transporter will only execute each command once.

SOCKET USAGE

There are four sockets which may be activated for receiving a message. The four socket numbers are 80, 90, A0, and B0. Each socket accepts messages destined for its host and its socket number only. Each socket is activated by a unique setup receive command and as a result operates completely independently of all other sockets.

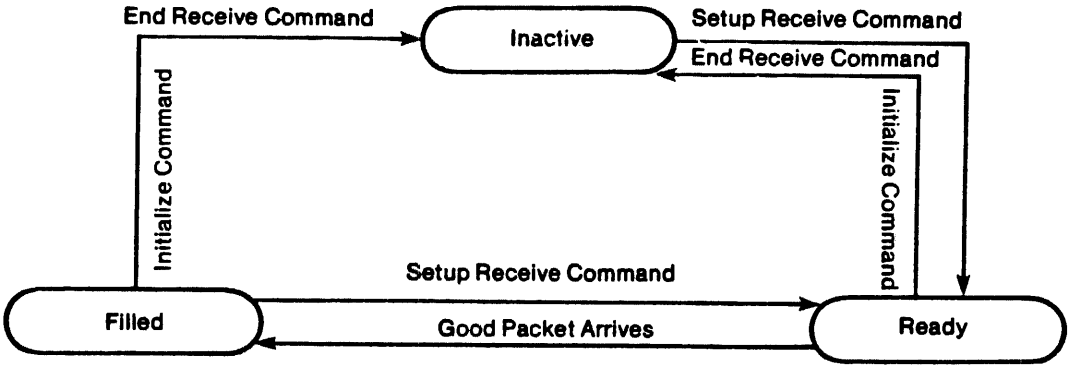
In addition, all messages for a socket are split into a user data portion and a user control portion, and each is given its own host or buffer memory region by the Setup Receive command. Thus, the user data and user control portions of an incoming message can be separately routed by the host to any locations in memory. Specifically, it is possible to route message data to the host memory space and message control information to the system space.

Sockets exist at all times in one of three possible states: inactive, ready, and filled. When a socket is inactive, it cannot receive incoming messages and any messages which do arrive for the socket will be negatively acknowledged.

Issuing a Setup Receive command for a socket brings the socket into its ready state. At this point the socket is ready to receive a single message from a distant node, and the message will be positively acknowledged after it is received.

The successful receipt of a message puts the socket into its filled state, and once again it is unable to receive messages until it is made ready via a Setup Receive command. While a socket is in the filled state, messages that arrive for the socket are neither positively nor negatively acknowledged. So long as the transmitter's maximum number of retries has not been exceeded, no acknowledgement causes the message to be retransmitted. The receiving host thus has a chance to prepare the filled socket in time for the retransmission.

To get from either the filled or ready state to the inactive state, the socket must receive an End Receive command from its host. The diagram below shows the three possible socket states and the means by which sockets get from one state to another.



Socket States

One important point should be made here. It is not possible to reactivate a socket which is already in the ready state simply by issuing another Setup Receive command. In order to change the parameters of an already active socket, the host must first issue an End Receive command to the socket and then issue the new Setup Receive command. This prevents different programs that utilize the same sockets from unwittingly altering the parameters of a socket already in use by another program.

COMMAND MECHANISM

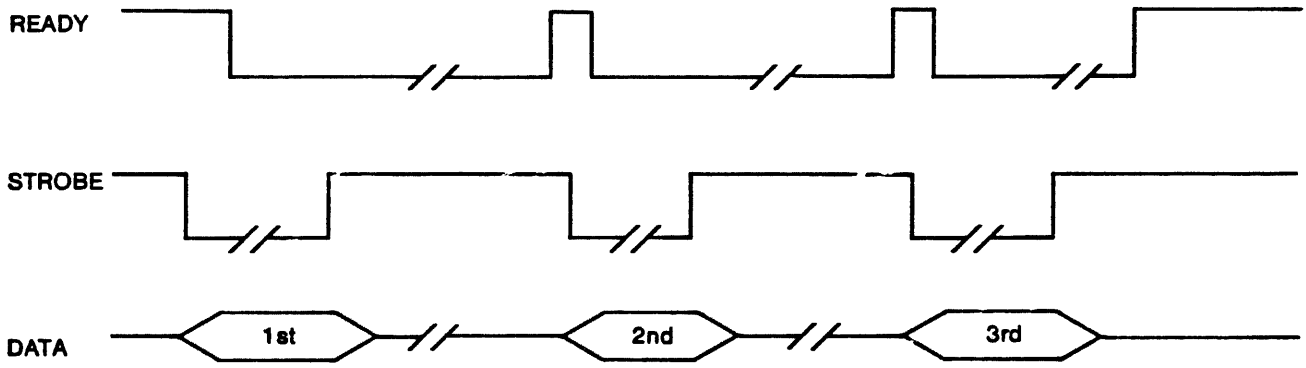
Commands are initiated from the host to the Transporter by the host sending a 24 bit address to the Transporter in the form of 3 bytes (most significant byte first). The address identifies the command vector, which contains a command code, a result record address, and other command-dependent information.

Before issuing a command, the host must write the value FF hexadecimal to the first byte (called the status byte or return code) of the result record. When the command is completed, the Transporter signals the host by setting that same byte to some other value. On host computers that support interrupts, an interrupt occurs after the result record has been modified.

Command vector addresses are sent to the Transporter, one byte at a time, using a polling procedure to determine when the Transporter is ready to accept each byte. The status bit of interest is the Transporter READY line, which is indicates the Transporter's ability to accept the next byte in the sequence, but does not reflect the Transporter's ability to accept the entire sequence immediately.

If the Transporter is transferring a message to the host at the beginning of a command sequence (or starts to transfer a message to the host in the middle of a command sequence) the READY line will go low, indicating the Transporter's inability to accept the next byte of the command vector address. The READY line will remain low until the Transporter has finished writing its message into host memory.

The above process is implemented utilizing the two wire handshake illustrated below. When the Transporter is ready to accept an address byte from the host it sets the READY line high. When the host is ready to send a byte to the Transporter it sets the STROBE line low and presents the data byte on the DATA lines. As soon as the host sets the STROBE line low, the Transporter sets the READY line low to insure that the second address byte is not sent before the first has been received. The address byte on the data lines is latched in and the transfer is complete when the host brings the STROBE line high again. The host then waits for the READY line to go high before sending the next address byte.



Sending Command Vector Addresses

There is only one potential problem with the READY/STROBE handshake shown above. After receiving the third byte of a command vector address, the Transporter sets the READY line high. It is not, however, ready to receive the first byte of a new command vector address. The host must wait until the command just issued has been completed before issuing a new command. Command completion is indicated by the return code and interrupt mechanism, as described later in this chapter.

If the host does begin to issue a new command before the previous command has completed, the first byte of the new command vector address will be lost. The host will set its STROBE line low and then high but the Transporter will be busy with the previous command and will not latch in the new address byte when STROBE returns high. This can cause serious problems for both the host and the Transporter because when the host believes it has finished issuing a command, the Transporter will still be waiting for the last address byte. Thus the host will expect command completion notification, which will never come, and the Transporter will expect a third address byte, which may only come in the form of the first address byte of a subsequent command.

COMMANDS

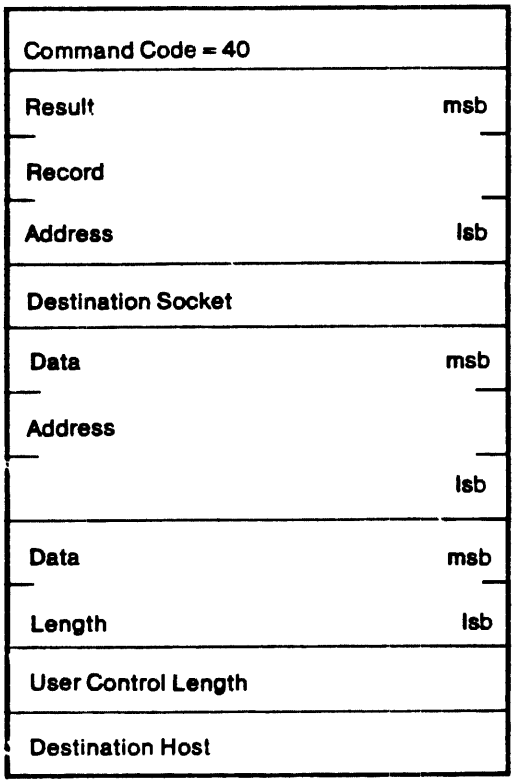
The following commands are supported by the Transporter:

Send Message
Setup Receive
End Receive
Initialize
Who Am I
Echo
Peek/Poke

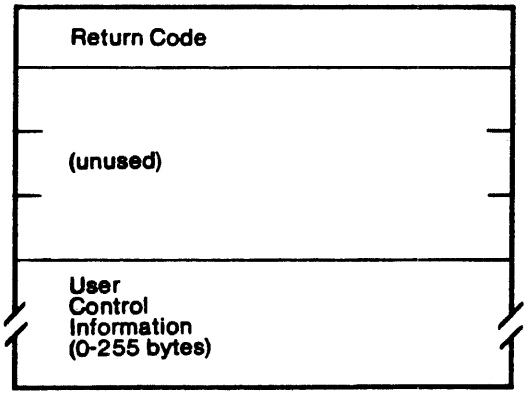
A brief description of each command as well as the command vector and result record formats for these commands are given below. In the command vector and result record diagrams, except where indicated otherwise, the vertical divisions marked by horizontal lines or dashes denote single bytes. The abbreviations *msb* and *lsb* stand for most significant byte and least significant byte, respectively.

SEND MESSAGE

Send Message commands the Transporter to send one message to the indicated host and socket. In the case of an error, the message will be retransmitted until it has been acknowledged or until the retry count has been exceeded (the retry count has a default value of 10, but may be altered by the Peek/Poke command). The user must not modify the message buffer or attempt to issue any command to the Transporter until this command is finished.



Command Vector



Result Record

In addition to sending direct messages to specific nodes, the Send Message command can be used to "broadcast" a message over the network. A host computer broadcasts a message by specifying the value FF hex (255 decimal) in the Destination Host field of the Send Message command vector. All Transporters receive messages addressed to destination 255 but none acknowledges such a message.

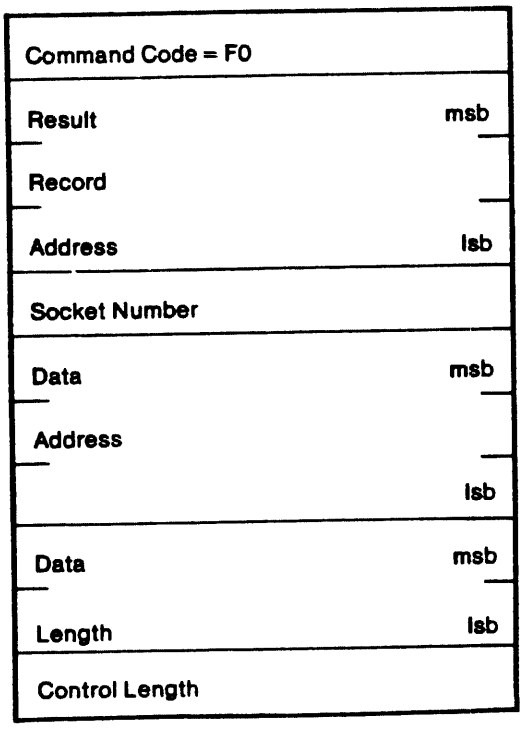
One of the following return codes will be placed in the first byte of the result record upon completion of the Send Message command.

00 -- Message was sent successfully with no retries.
nn -- Message was sent successfully after nn retries (00< nn< 80).
80 -- Message was not acknowledged (retry count exceeded).
81 -- Message data portion was too long for the receiver's buffer.
82 -- Message was sent to an uninitialized socket.
83 -- Message control portion length did not match receive socket's control buffer length.
84 -- Invalid socket number in command vector.
86 -- Invalid destination node number in command vector.

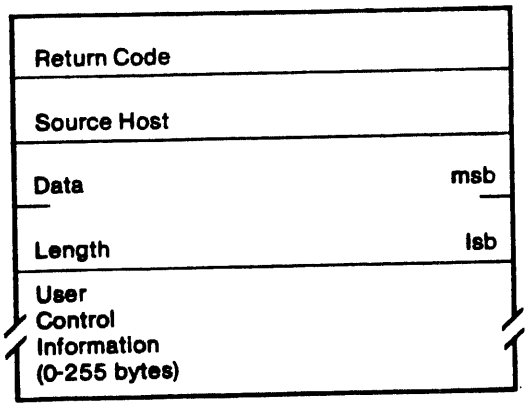
SETUP RECEIVE

Setup Receive prepares a socket (80, 90, A0, or B0) to receive a single message. The return code in the result record will be modified twice if the command is successful. Those hosts which support interrupts will receive an interrupt with each return code modification.

The first return code modification indicates whether or not the command has been successful in setting up a receive socket. If the setup has been unsuccessful, the command is complete and no further return code modification will take place. If the setup has been successful, the Transporter is now ready to receive a message through the socket, and a second return code modification will take place when a valid message is received. In either case, a host that is waiting to issue a second command to its Transporter may do so after the first return code modification.



Command Vector



Result Record

The first return code modification will yield one of the following values:

84 -- Invalid socket number in command vector.
85 -- Receive socket in use.
FE -- Socket set up successfully.

The second return code modification will occur only if the setup was successful. It will take place after a valid message has arrived and after the result record has been altered to reflect this fact. The contents of the result record will be as follows:

Return Code -- 00. Message received successfully.
Source Host Number -- Node number of message originator.
Data Length -- Length in bytes of user data portion of message.
User Control Information -- User control portion of the message

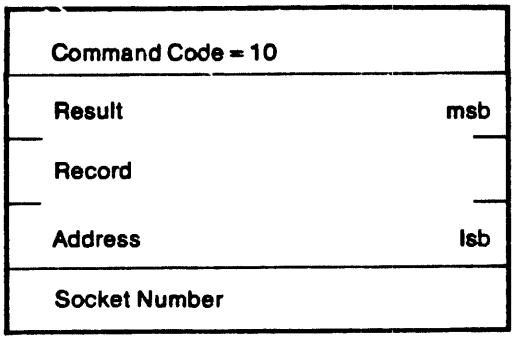
The length of the user control portion must exactly equal the length specified in the Control Length field of the command vector.

The user data portion of the message will be found in the data buffer whose location and length were specified by the Setup Receive command vector.

Note that if a message is received in which the user data portion is larger than the receiver's buffer, or the user control portion is not identical in size with the receiver's specified length, the Transporter rejects the message without informing its host that any message was received. It does, however, send a NAK packet to the transmitter to indicate unsuccessful message reception.

END RECEIVE

End Receive commands the Transporter to release the indicated socket, thus disabling the reception of further messages for that socket until the next Setup Receive command is sent.



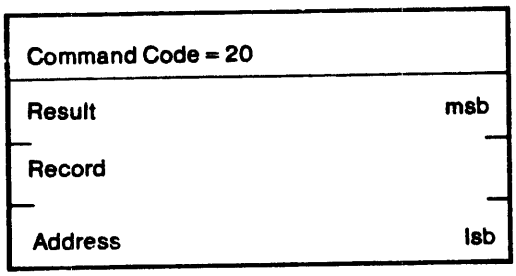
Command Vector *Result Record*

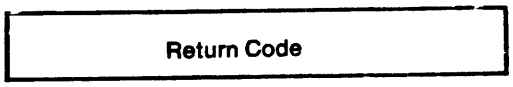
One of the following return codes will be written to the Result Record upon completion:

00 -- Operation complete.
84 -- Invalid socket number.

INITIALIZE

Initialize commands the Transporter to initialize as in a hardware reset (or power-up). All parameters are set to their default values, the event counters are reset to zero, all four sockets are put into the inactive state, and a group of synchronizing packets is sent out onto the network.



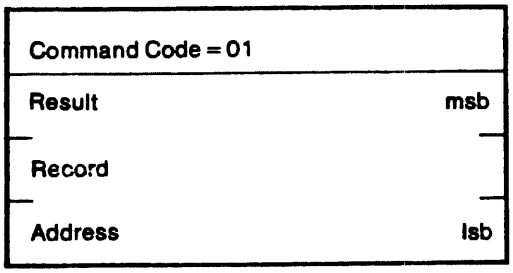


Command Vector *Result Record*

A return code value equal to the host node number will be returned upon completion of the Transporter initialization.

WHO AM I

Who am I returns the network node number of requesting Transporter (00-3F) as the return code.

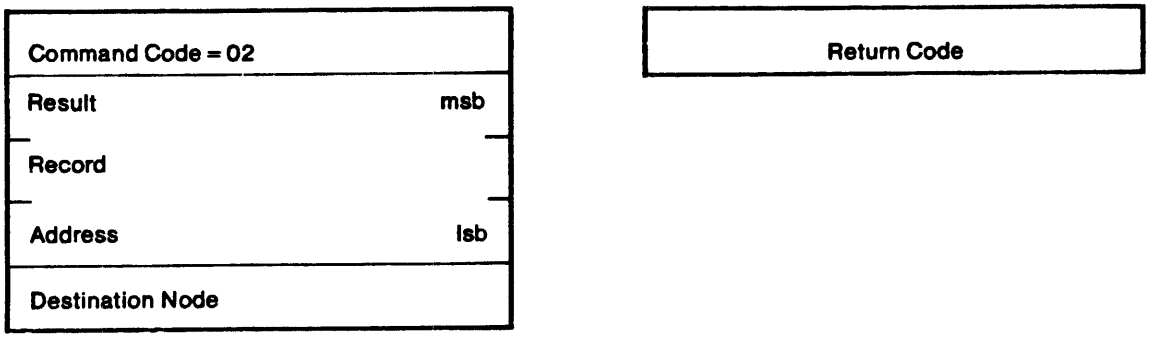


Command Vector

Result Record

ECHO

Echo commands the Transporter to send an echo packet to another node and expect an acknowledgement from that node. The Echo command is used to verify the presence of another active Transporter on the line.



Command Vector *Result Record*

One of the following codes will be returned upon completion of this command:

CO -- Destination node acknowledged successfully.
80 -- Packet was not acknowledged (retry count exceeded).
86 -- Invalid destination node number.

PEEK/POKE

Peek/Poke commands the Transporter to examine or alter its own internal memory as directed. Upon command completion the return code contains 00 for a Poke command and contains the value of the byte at the address specified for a Peek command.

The table below lists Transporter operational parameters and event counters and their addresses in Transporter memory. Addresses are given for all standard versions of the Transporter code. To check the version, execute a Peek command at location F800.

Only parameters should be altered with the Poke command, but both parameters and event counters may be viewed using the Peek command.

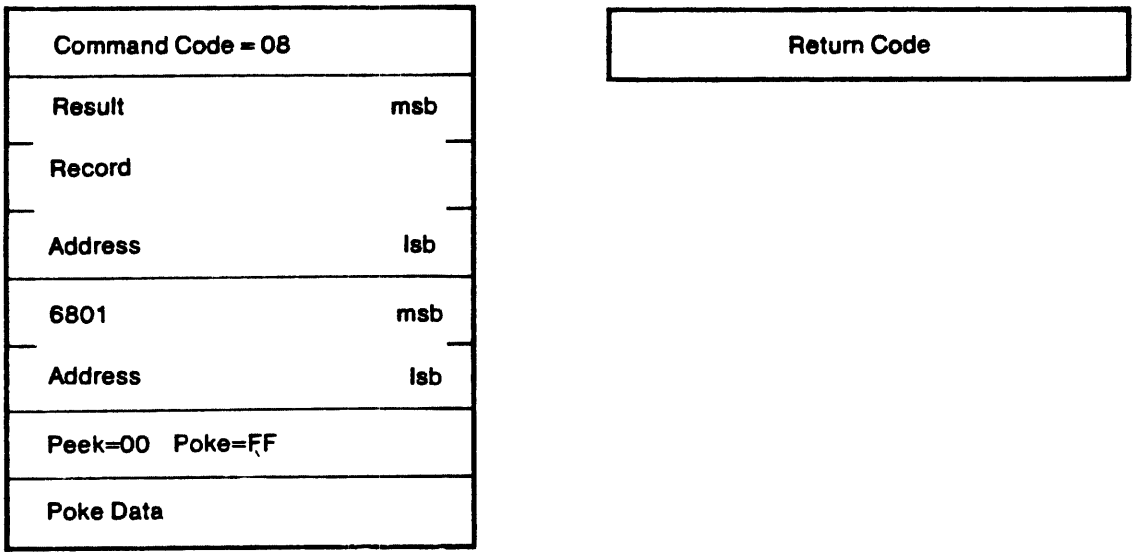
Parameters

Addresses by Version 6.4	8.A	9.B		
00E1	00E1	00E1	--	Maximum number of retries on transmit message failure. The default value is 0A (10 decimal).
00E2	00E2	00E2	--	Number of trailing flags to be sent after each packet. The default value is 10, which generates 12 trailing flags. The user should not modify this value.
00E5	00E4	00E4	--	Number of leading flags to be sent in an Acknowledgement packet. The default value is 18, which generates 13 leading flags. The user should not modify this value.
00E3	None	00E5	--	Scale factor. This value is used to scale the random number which determines wait times between packet retransmission attempts. Possible values are 1, 3, 7, 15, 31, 63, 127, and 255. The default is 7. The user should not normally modify this value.
F800	F800	F800	--	Version number of Transporter code.

Event Counters

Addresses by Version 6.4	8.A	9.B		
00E6	00E5	None	--	Missed packet counter. This counter is increased by 1 whenever the Transporter sees a packet destination address on the line but has missed the leading flags of the packet.
00E7	00E6	00E6	--	Collision avoidance interrupt counter. This counter is increased by 1 whenever the Transporter senses a line transition during the 10 usec preparation period prior to packet transmission.
00E9	00E7	00E7	--	ADLC receive status. Contains the value of the ADLC receive status register. For the possible values of this register see the Motorola MC6854 data sheet.

Note that one or more of the event counters may be non-zero even in a network that is functioning properly.



Command Vector *Result Record*

When issuing a Peek command to a Transporter, the contents of the Poke data field of the command vector are unimportant. The Transporter will not read this field.

RETURN CODES

The values for the return codes mentioned above in the context of the seven commands are summarized in the table below:

00 -- Command was successfully completed.
00-3F -- Node identification number resulting from an Initialize or Who Am I command.
01-7F -- Transmit retry count.
80 -- Transmit failure (no acknowledgement after maximum number of retries).
81 -- Transmitted message user data portion was too long for receiver's buffer.
82 -- Message was sent to an uninitialized socket.
83 -- Transmitted message user control portion size did not equal receiver's control buffer size.
84 -- Bad socket number in command (must be 80, 90, A0 or B0).
85 -- Receive socket in use; a valid buffer is attached.
86 -- Bad node number in command (must be 0-7F or FF).
C0 -- Received an ACK for an Echo command.
FE -- Socket set up successfully.

RESTRICTIONS

Restriction #1: Once a host has initiated a Transporter command, the host should not attempt to issue a second command until the previous command has been completed. Completion is indicated by theCreturn code in the result vector.

Restriction #2: When the Transporter is receiving a packet from the network, it will not process command address bytes from the host. The Transporter byte buffer will take one byte and the READY line will remain low until the Transporter is able to process that byte. This leads to long delays as seen by the host.

INTERFACING TO THE DMA TRANSPORTER

There are only two facts which a host must know in order to interact with a DMA Transporter. First, the host must know the location of the Transporter's command address register (CAR) so that it can write command vector address bytes to the correct place. Second, the host must know where to find the Transporter's READY signal so that it can determine when the CAR is ready to receive data. In almost all cases, both the CAR and the Transporter status port (which contains the READY signal) are located on specific host I/O ports. The particular

port numbers and the bit of the status port which is the READY signal are given for specific computers in the appendices to this manual.

When the location of the CAR and Transporter READY bit are known, the actual process of issuing a command to the Transporter is simple. Following one of the formats outined in this chapter for specific commands, the host builds a command vector in memory and allocates space for the result record and data buffer. If the command is a Send Message command, the data buffer must be filled with the user data portion of the message and the final bytes of the result record must be filled with the user control portion of the message. Finally the host must write the code FF to the return code.

Once the command vector, result record, and data buffer are properly initialized, the host may issue the command. The host should follow the procedure of checking the Transporter READY bit before writing each command vector address byte to the CAR. After all three bytes have been written, the host may turn to other concerns until an interrupt arrives, indicating a change in command status. If interrupts are not supported, the host should check the return code repeatedly until a change from code FF occurs.

INTERFACING TO THE BUFFERED TRANSPORTER

Interfacing to the buffered transporter is somewhat more complicated than interfacing to a DMA Transporter. Almost all data exchanges between a buffered Transporter and its host utilize the buffer RAM. Thus the host must create all command vectors, result records, and data areas in the buffer.

To interface with a buffered Transporter, a host must be able to perform four write and three read functions:

Write Functions

Port	Function
A	Write high nibble of the counter.
B	Write low byte of the counter.
C	Write to the buffer location pointed to by the counter. Add 1 to the counter following the write.
D	Write to the CAR.

Read Functions

Port	Function
A	Read buffer location pointed to by the counter.
B	Read buffer location pointed to by the counter. Add 1 to the counter following the read.
C	Read Transporter status port.

Ports A, B, C, and D are the I/O ports at which the operations must be performed. For example, if a host wishes to read the buffer location pointed to by the counter, it should read I/O port A. If it wishes to read the buffer and add 1 to the counter, it should read port B. The choice of operation is affected by the direction of data transfer: a read at port A reads the buffer memory while a write at port A writes to the counter.

The buffered transporter takes time to execute its host's commands. After issuing a command, a host must wait to issue the next. The following table shows minimum allowable times between commands:

Between two Write-RAM-Add commands	3.2 usec
Between two Read-RAM-Add commands	1.6 usec
Between a Load Counter and any Read-RAM command	1.6 usec
Between a Write-CAR and a Read-Status command	2.2 usec

To communicate with a buffered Transporter, a host must know the actual port numbers for I/O ports A, B, C, and D and the location of the Transporter READY bit within the Transporter status port. The appendices contain this information for specific Transporters.

Writing to the buffer requires some thought if it is to be done efficiently. It is easier to write out an entire block of data at once, using the Write-RAM-Inc command, than to write out portions of a block and then come back later to fill in details. The host should, therefore, know the values of all fields in a data block before beginning to write the block to RAM. If a command vector is being written, the host should already know the addresses in RAM where it will place the result record and data buffer so that it can correctly enter these addresses in the appropriate fields of the command vector.

If all values for a data block are known at the start, a write can proceed smoothly and quickly. The host's first action will be to set the counter by writing to ports A and B. The address written will be the starting address in the buffer of the data block. The host may now proceed to write successive bytes of the block to port C. This adds 1 to the counter after each byte is written so the host need not be concerned with the counter until it is ready to begin writing a new data block.

After all necessary data blocks for a command have been written to the buffer, the host may issue the command by writing the three-byte buffer address of the command vector to the Transporter's CAR. As before, the host should check the Transporter's READY bit before writing each byte.

After a command has been issued, the host should follow the same procedure of waiting for command completion as with the DMA Transporter. If interrupts are supported, the host may simply wait for an interrupt to arrive while working on other tasks. If interrupts are not supported, the host should repeatedly read the location in the buffer where the return code is located.

There is one complication with repeated reads of the return code. It is possible for the Transporter to be modifying the return code at the same time as the host is attempting to read it. This will cause the host to see a garbled code. Nevertheless, the garbled value will tell the host that the code has changed from its previous value and, by performing an additional read at the return code location after the change is observed, the host can discover the correct value for the code.

Once a command is completed, the host will find all resulting information in the buffer. Received user

message data will be found at the buffer location designated in the Setup Receive command while user control information will be located at the end of the result record within the buffer. In order to use any of this data, the host will have to read it out of the buffer into its own memory.

One final issue must be addressed: the size of the buffer. On most buffered Transporters the buffer is a 4k RAM addressable with 12 address lines. Thus the counter requires only that a high nibble (4 bits) be written to port A instead of an entire high byte. This 4k buffer begins at address 000h and goes through address FFFh. All intermediate values are valid addresses for placing data.

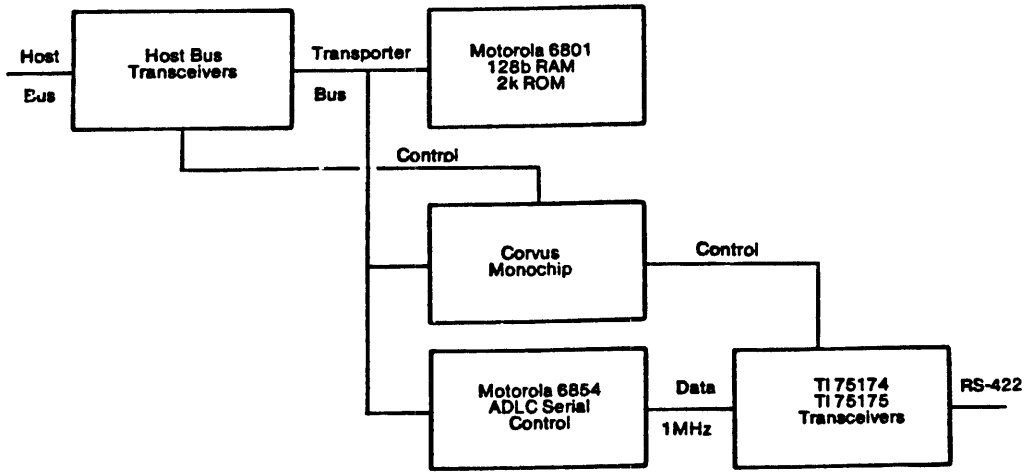
This page intentionally left blank.

TRANSPORTER OPERATION AND INTERNAL IMPLEMENTATION | 4

A transporter for a given host consists of two parts, the host independent portion and the host dependent portion. The host dependent portion provides the hardware interface to the actual memory bus of the host system. This chapter describes primarily the host independent portion of the Transporter, as the other portion varies from host to host.

HARDWARE

Below is a block diagram of a Transporter. The host bus on the left consists of the data, address, and control lines which interface the Transporter to its host. The Texas Instruments 75174 and 75175 transceivers interface the Transporter directly to the RS-422 twisted pair cable.



Transporter Block Diagram

The following sections describe the various components in the block diagram:

COMPONENT DESCRIPTIONS

The MC6801 -- The Motorola 6801 is a microprocessor which functions as the CPU for the Transporter. Within the 6801 are 2k bytes of ROM that hold the instructions for controlling the Transporter. The 6801 begins execution of this code on Transporter power-up.

The 6801 also contains 128 bytes of RAM. The Transporter code uses this RAM for its stack space and data structures. Command vectors are stored in here after they are read from the host.

The MC6854 -- The Motorola 6854, also known as the Advanced Data Link Controller (ADLC), provides the functions of bit serialization, zero insertion, CRC generation, packet framing and data byte buffering when transmitting a message. When receiving, the ADLC strips the packet header, eliminates the inserted zeroes, checks the CRC, and buffers the incoming data for DMA to the host. The ADLC also interprets the line data to determine whether or not the line is idle.

Corvus Monochip -- The Corvus Monochip is a custom gate array that provides the synchronization for all data transfers occurring outside of the 6801 microprocessor. In particular, the monochip performs the functions of clock separation, last-minute line idle detect, and intelligent DMA.

The RS-422 Transceivers -- The Texas Instruments 75174 and 75175 are the RS-422 twisted pair transceivers. The 75174 takes the series of bits sent to it by the ADLC and converts it to differential voltages on the twisted pair. The 75175 provides the inverse functions of the 75174.

Host Bus Transceivers -- The host bus transceivers consist of a series of data latches and buffers for interfacing the Transporter to the host. The structure of this logic is dependent upon the peculiarities of the particular host.

HOST INTERFACE LINES

The host-Transporter interface lines are TTL compatible and have the characteristics described below.

Address Lines -- The Transporter selects the byte of host memory to write to or read from during a DMA cycle through 24 address lines. When the host computer is driving the address lines (no DMA cycle in progress), the Transporter ignores them.

Data Lines -- Eight bidirectional DATA lines are used for reading data from host memory and writing data to host memory.

DMA Control -- Three lines are provided for control of DMA transfers to and from host memory:

DMA REQUEST line (DMARQ) from the Transporter.

DMA GRANT line (DMAGO) from the host.

DMA DIRECTION line (IN/OUT) from the Transporter, driven high or low depending on whether it is writing or reading.

Command Control -- The two-wire handshake that formalizes the transfer of each command vector address byte from host to Transporter uses two interface lines:

READY line from the Transporter.

STROBE line from the host.

Interrupt -- the Transporter may notify the host of a change in command status through one interrupt line. The Transporter generates a low going pulse on the interrupt line whenever it modifies the return code.

For information on the electrical and timing characteristics of the host/ Transporter interface lines, see the *Omninet Circuit Description* manual.

THEORY OF OPERATION

This section describes three aspects of Transporter operation: host command reception, DMA overrun detection, and the sequence of events when a Transporter is powered up.

COMMANDS FROM THE HOST

The host initiates commands by sending a three byte vector address to the Transporter. The transmission of each byte occurs through a dummy DMA cycle. There is also a two wire handshake associated with each address byte (see the section "Command Mechanism" in Chapter 3 for more details). The handshake control logic is contained in the monochip. A one byte buffer that handles each address byte in succession is contained in random logic.

DMA OVERRUN DETECTION

In unbuffered Transporters, it sometimes happens that one DMA cycle is not finished before a second one must begin. This event is known as a DMA overrun. Where DMA overruns occur with sufficient frequency, a buffered Transporter is used. However, a certain number of DMA overruns can be tolerated by an unbuffered Transporter so long as the Transporter is equipped with overrun detection circuitry.

Two basic types of DMA overruns can occur in an unbuffered Transporter. In the first case, DMAGO, the signal from the host, does not return high to indicate the end of a DMA cycle soon enough. During message reception or transmission, new DMA cycles must begin every 7.25 microseconds in order to keep up with the 1 megabit per second data transmission rate. This leaves only 5.9 usec for execution of the half of the DMA cycle involving the host memory. Thus, if DMAGO does not return high to latch in a DMA'd data byte within 5.9 usec of the time that DMARQ went high to initiate the host DMA, the byte will be lost.

The second type of DMA overrun occurs when DMAGO fails to go low to allow a requested DMA operation within 5.9 usec, and the data byte to be transferred never gets onto the bus. As in the case above, the data byte is lost.

The possibility of DMA overruns of either type requires some kind of DMA overrun detection mechanism to let the 6801 processor know that an overrun has occurred. Once the processor is informed that an overrun has occurred and, therefore, that data has been lost, it makes a graceful recovery. If possible, the processor begins the transfer of the data block again. However, in the case of a message received from a distant Transporter, the processor must wait for the sending Transporter to repeat its transmission. The processor insures a retransmission by not acknowledging the first message attempt. Only if the sending Transporter's retry count has been exceeded will the message not be retransmitted. In any case, even if the message is not repeated, the host will never know that a breakdown occured.

Overrun detection circuitry is included on all Transporters that run the risk of occasional DMA overruns. The circuitry functions by generating an overrun signal on a special overrun line whenever DMAGO fails to return high within 5.9 usec of the initiation of a DMA request or when DMAGO fails to go low in the first place. In either case, the signal travels to the ADLC chip, where continued message transmission or reception is blocked. This interrupts the 6801 so that it can take the required action and prevent the generation of the next clock pulse, which would begin the next DMA cycle.

There is one potential problem with the inclusion of overrun detection circuitry on an unbuffered Transporter. When the overrun circuit is triggered, the configuration of the Transporter changes from that which exists during normal host DMA cycles. This means that if the DMA cycle is complete (DMAGO returns high) after the triggering of the circuit, the DMA'd data byte may be written to an unexpected location within the Transporter. This can cause the Transporter to freeze and the only solution will be to reboot the host.

Thus, overrun detection circuitry should only be included on a Transporter when it is certain that the host will be able to complete all DMA cycles before a change in Transporter configuration occurs. In practice, this means that all DMA cycles must complete within 12 usec of the DMA request. If it is possible that the host will take longer than 12 usec to complete a DMA cycle, a buffered Transporter must be used.

POWER-UP

When a network device is powered up, a number of initializations and error checks must be made to ready the device to accept user commands. The procedure is identical to that performed by a Transporter on reception of an Initialize command from its host with the exception that, on power-up, the host node number is not returned to the host, as no return code location has been designated. The Power-Up/Initialize routine is standard for every Omninet Transporter and is part of the Transporter operating program. The procedure that the program follows is outlined below.

1. Read address switches to discover own network address. Save the result in 6801 RAM.

2. Turn on Transporter card LED.

   During normal operation, the LED flickers whenever the Transporter is transmitting a packet.

3. Initialize ADLC chip.

   The ADLC chip must be put into the proper operating mode to handle packet transmission and reception.

4. Mark receive sockets as inactive.

   Since no Setup Receive command has yet been issued, the sockets must indicate that they are not prepared to receive messages.

5. Zero all internal counters and the parity table.

   Because this Transporter has not yet communicated, the event counters should all register zero. The parity table entries must also register zero to ensure synchronization with other parity tables. When this Transporter transmits its Sync packets, the appropriate entries in distant Transporters' parity tables will be zeroed.

6. Check integrity of the program in the 6801 ROM. This program is the one currently running.

   The ROM check determines if data in the ROM has been altered or destroyed. The check sums the bytes of the ROM and examines the low byte of the result. If the ROM is intact, this byte will be zero.

7. If ROM is intact, turn off Transporter card LED. Otherwise, stop executing code and keep the light on.

8. Initialize internal variables.

9. Enable packet reception by the ADLC.

10. Send out Sync packets onto the network.

    Sync packets tell receiving Transporters to zero the sending Transporter's entry in their parity tables. Zeroing insures synchronization of parity tables for use in duplicate packet detection.

    Eleven Sync packets will be sent on a power-up or initialization. The Transporter arrives at 11 by adding one to the maximum number of retries value, which will always be set to the default, ten, at power-up or initialization.

    During Sync packet transmission, and all other packet transmissions, the LED flickers.

11. Go to main program loop; wait for a host command or incoming packet.

At this point the initialization procedure is complete. The Transporter remains at step 11 until some external event requires its attention. When some such event does occur, the Transporter executes the appropriate code to enable it to deal with that event and then returns to the wait state.

This page intentionally left blank.

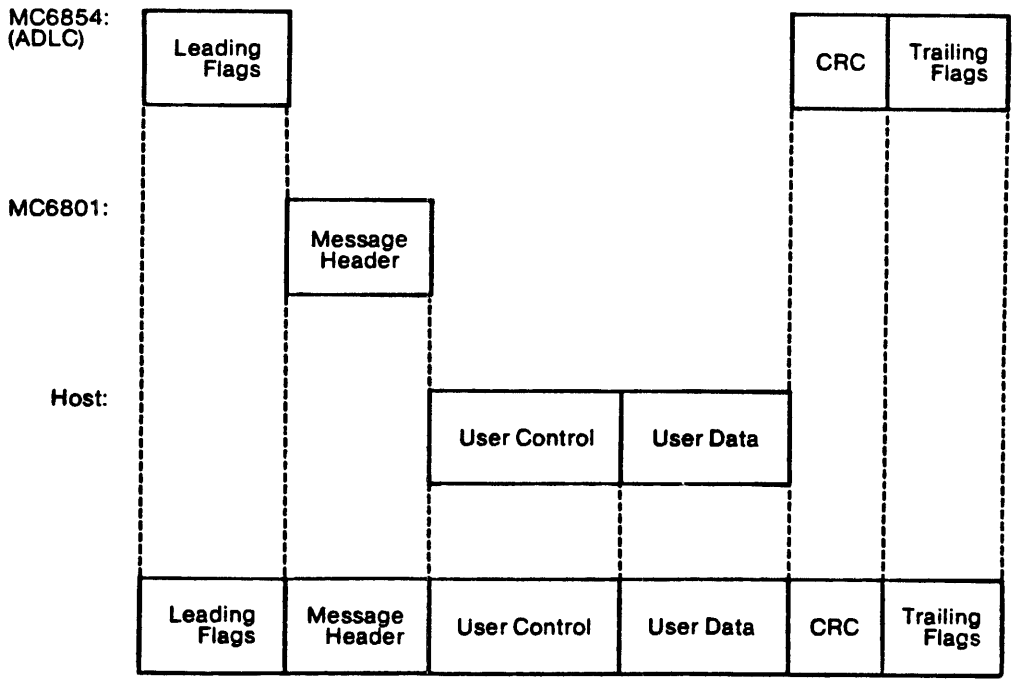
ANATOMY OF A PACKET | A

This appendix illustrates packet generation and packet reception, showing the source of packet components and the intended receivers.

PACKET GENERATION

During the transmission of a packet, the various portions of the packet originate from different sources within the transporter and host. The diagram below indicates the source of each packet element.

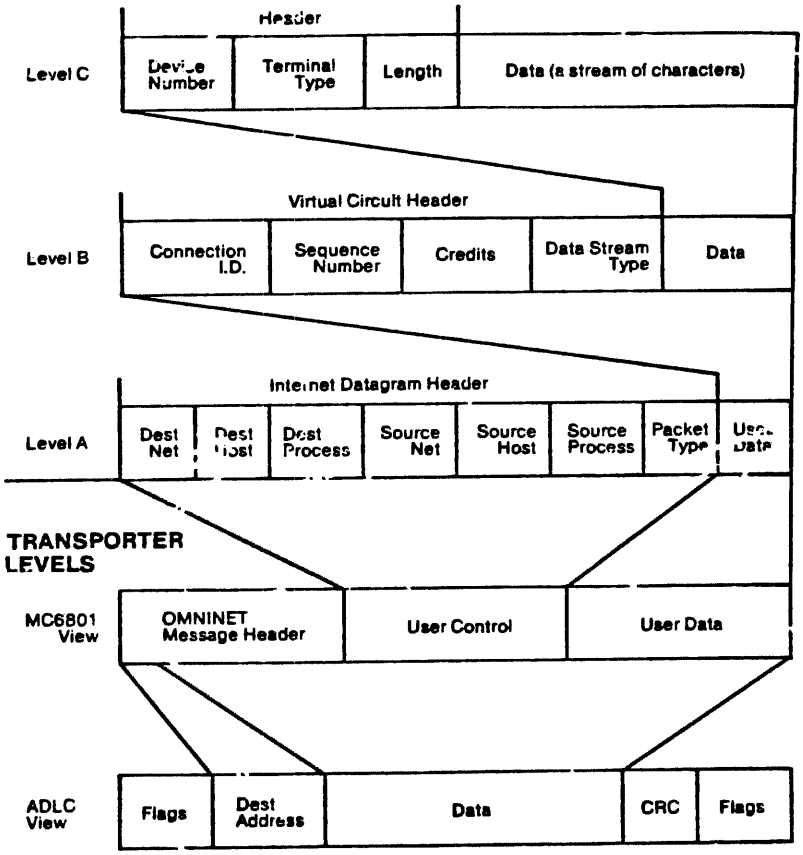
SOURCE:



Packet Generation

PACKET RECEPTION AND INTERPRETATION

During packet reception, the various parts of the transporter and the successive levels of user-defined communications protocols examine some packet fields and ignore others. Those fields that each level examines are usually stripped from the packet at that point and the remainder of the packet is passed on to the next higher level. The diagram below illustrates the view that each part of the transporter and each protocol level takes of a packet. The ADLC and MC6801 breakdowns are the same for all packets, but the higher level breakdowns are those of a fictional user protocol.



Packet Reception and Interpretation

THE APPLE II TRANSPORTER | B

The Apple II communicates with its Transporter by first formatting a command vector and then sending the command vector address to the Transporter through the use of one control register. This control register is referred to as the Command Address Register (CAR). When the command is completed, a return code is placed in the result record. The address of the result record is specified in the command vector.

The CAR is an 8-bit register. Its address is determined by the slot in which the Transporter is installed as shown in the chart below. Apple II I/O space is memory mapped so the addresses below are normal memory addresses and not I/O addresses.

SLOT NUMBER	CAR ADDRESS Hexadecimal	Decimal	Decimal
1	C090	49296	-16240
2	C0A0	49312	-16224
3	C0B0	49328	-16208
4	C0C0	49344	-16192
5	C0D0	49360	-16176
6	C0E0	49376	-16160
7	C0F0	49392	-16144

The sign bit of the CAR is the Transporter READY bit (RDY). When set, this bit indicates that the Transporter is ready to receive the next address byte of the three byte command vector address. To issue a command to the Transporter, this address must be given to the Transporter one byte at a time. Every time an address byte is placed into the CAR, the RDY bit of the CAR goes low and the next byte cannot be sent until the RDY bit returns high again.

The three byte address is sent with the most significant byte first. For the Apple II the first byte is always zero since the Apple II address space only requires two address bytes.

SOFTWARE NOTES

While the Transporter is receiving a packet from the network, it cannot process a byte moved into the CAR, so the RDY bit of the CAR remains low until the Transporter can process the next byte. This leads to a situation where a software I/O driver may have to wait up to several milliseconds before the RDY goes high again.

Since the Apple II processor does not support interrupts, the communication program should periodically check the return code for a change in value. As it is conceivable, though highly improbable, that the Transporter could be modifying the return code at the same moment that the processor is viewing it, the processor should check the code a second time after detecting a change. This will insure that the processor sees the correct code value rather than a mid-change garble.

Until the command has been completed, as indicated by the return code, no additional data should be placed into the CAR by the sending computer. Additional data should not be placed because the Transporter will process only one command at a time.

The Apple II Transporter is unbuffered. Data transfers with host memory take place through DMA and do not disturb the processor. There is no DMA overrun detection circuitry on the Apple II Transporter because host memory is sufficiently fast that it is not needed. A boot ROM is provided with the Transporter.

THE CONCEPT TRANSPORTER | C

The Concept Transporter is a normal DMA Transporter which supports interrupts. Interrupts arrive at priority three. After an interrupt arrives, the host must reset the interrupt mechanism before another interrupt can happen. Interrupts are reset when the processor performs a write operation to any address between 030FC1 and 030FDF. The contents of the write are unimportant.

A potential problem exists when several Transporter operations are pending concurrently. If two commands are completed within a short time of each other, it is possible that the processor will not have a chance to reset the interrupt mechanism between the two command completion interrupts. To avoid this eventuality, the processor should check the values of all the outstanding return codes before returning from the interrupt subroutine. If any of these return codes indicates that the associated command has also been completed, the processor can then take appropriate action.

Concept I/O space is memory mapped so all I/O addresses are simply standard memory addresses. This includes those given above for interrupt resets.

To issue a command to the Transporter, the processor must write the command vector address, byte to byte, to any address between 030FA1 and 030FBF. Between each byte write, the processor must check the Transporter READY bit. This is bit 0 of address 030F7F. Bit 0 high indicates that the Transporter is ready to accept the next byte of the command vector address into the CAR.

A boot ROM is included on the Concept Transporter.

This page intentionally left blank.

THE IBM PC TRANSPORTER | D

There are two versions of the IBM PC Transporter. The earlier version is a buffered Transporter that does not support interrupts. The new version does support interrupts and uses a buffered gate array. The new version is backwards compatible with the earlier version, and it can also be used as the TI Professional Transporter. The functions described below are the same for the TI, but the jumper at location E2 must connect B and C to map the PROM to the F6000-F6FFF space.

Both the new and the old version of the Transporter have a boot ROM that extends from host CPU address DF000 to address E0000. The ROM utilizes the first 1024 bytes of the 4K buffer RAM and must have exclusive use of this area. The host should not place command vectors or other command information in this section of this buffer.

All processor read and write operations to and from both versions of the IBM PC Transporter take place through the I/O ports. The following is a list of the possible processor actions and the I/O ports to which they should be directed:

Read and Write Operation	I/O Port
Read Transporter Status Byte	0248
Read RAM	0249
Read RAM; then increase the Counter by 1	024B
Read the Counter lower byte	024A
Write to the CAR	0249
Write the Counter High Byte	0248
Write the Counter Low Byte	024A
Write to RAM; then increase the Counter by 1	024B

All read and write operations directed at the RAM occur at the address to which the counter is currently pointing. The counter is subsequently increased only for those commands which specify a post-increment.

Bit 7 of the Transporter status byte is the READY signal. Bit 7 high indicates that the Transporter is ready to accept the next byte of the command vector address into the CAR.

For the new version, the following interrupt support and jumper options are also available:

Interrupt Operation	I/O Port
Disable Omninet interrupts	024C
Enable Omninet interrupts	024E
Clear current Omninet interrupt request (All three operations can be done by reading or writing with meaningless data)	024D
Interrupt status (bit 4 set=interrupt pending) (bit 5 set=interrupts enabled)	024F

Jumpers for interrupt selection (Location E1)

IRQ2 - A to B (default, must be cut to change)
IRQ3 - C to D
IRQ4 - E to F
IRQ5 - G to H
IRQ7 - J to K

Jumpers for PROM mapping (Location E2)

PROM at DF000-DFFFF - A to B (Required for Corvus IBM PC software)

PROM at F6000-F6FFF - B to C (Required for use with TI)

PROM not selected - Remove jumper

ROM SERVICES

There are four separately executable routines contained within the IBM Transporter ROM. Each routine is initiated by a standard 8086 intersegment long CALL to one of the four ROM entry points. The four routines and their entry points are as follows:

COLDSTART - DF000
WARMSTART - DF003
I/O - DF006
DUMMYRET - DF009

The COLDSTART routine initializes the Transporter card, locates a disk server on the network, loads the Constellation II boot program from the disk, and transfers control to that program.

The WARMSTART routine initializes the Transporter card.

The I/O routine performs one of a number of services depending on the contents of the AH register at the time the routine is entered. The I/O services are discussed in detail below.

The DUMMYRET routine performs a dummy interrupt return.

I/O SERVICES

There are six I/O services. The contents of the 8086 AH register at the time of entry to the I/O routine determine which service is performed. However, before any I/O service is requested, the host must call the WARMSTART routine. The I/O services will not function until the WARMSTART routine has been executed. COLDSTART calls WARMSTART, though, so the host need not make a separate WARMSTART call if the host used the ROM to boot. Each I/O service is described below.

Identify Interface: (AH) = 00

Contents of 8086 registers on entry:

(AH) = 00

Contents of 8086 registers on exit:

(AL) = 00
(AH) = Omninet node number of the Transporter (if the node number is unique)
= FF (if a second Transporter exists on the network with the same node number)

Transmit Data to the Drive and Accept a Response: (AH) = 01

Contents of 8086 registers on entry:

(AH)	= 01
DS: (SI)	= address of data to send to drive
ES: (DI)	= address of buffer to receive data from drive.
(CX)	= number of data bytes to send to the drive (maximum = 530)
(DX)	= number of bytes expected back from the drive excluding the return code (maximum = 530)
(AL)	= network address of disk server
(BL)	= number of timer units to wait for a reply from the disk server. 00 = do not abort; wait forever. (a timer is approximately .86 seconds)
(BH)	= number of transmit tries. 00 = 255 tries. FF = try until successful. Should be greater than 0.

Contents of 8086 registers on exit:

(AL)	= return code from the drive. FF = aborted.
(CX)	= number of bytes received from the drive including the return code.

Transmit Data to a Network Server: (AH) = 02

Contents of 8086 registers on entry:

(AH)	= 02
ES: (DI)	= address of data to transmit
(CX)	= number of data bytes to transmit
(AL)	= network address of server. FF = broadcast to all servers.
(BH)	= number of transmit tries. For broadcasts, (BH) = number of times to transmit the data.

Contents of 8086 registers on exit:

(AL)	= 00 (transmit successful)
	= FF (transmit aborted)

Transmit Data to a Network Server and Accept a Response: (AH) = 03

Contents of 8086 registers on entry:

(AH) = 03
DS: (SI) = address of data to transmit
ES: (DI) = address of buffer to receive data from server.
(CX) = number of data bytes to transmit (maximum = 530)
(DX) = number of data bytes expected from the server (maximum = 530)
(AL) = network address of server. FF = broadcast to all servers.
(BL) = number of timer units to wait for a reply from the server. 00 = do not abort; wait forever. (a timer unit is approximately .86 seconds)
(BH) = number of transmit tries. 00 = 255 tries. Should be greater than 0.

Contents of 8086 registers on exit:

(AL) = 00 (transmit successful)
= FF (transmit aborted)

Find any Disk Server on the Network: (AH) = 04

Contents of 8086 registers on entry:

(AH) = 04
(BL) = number of timer units to wait for a reply from a disk server. 00 = do not abort; wait forever. (a timer unit is approximately .86 seconds)
(BH) = number of tries. 00 = 255 tries. Should be greater than 0.

Contents of 8086 registers on exit:

(AL) = 00 (operation successful)
= FF (operation unsuccessful)
(AH) = network address of the disk server that responded.

Send a Write Command to the Drive: (AH) = 05

Contents of 8086 registers on entry:

(AH)	= 05
DS: (SI)	= address of command block to send to drive
ES: (DI)	= address of data to send to drive
(CX)	= length of command block in bytes (normally 4)
(DX)	= number of data bytes to send to the drive (normally 512)
(AL)	= network address of disk server
(BL)	= number of timer units to wait for a reply from the disk server. 00 = do not abort; wait forever. (a timer unit is approximately .86 seconds)
(BH)	= number of transmit tries. 00 = 255 tries. Should be greater than 0.

Contents of 8086 registers on exit:

(AL)	= return code from the drive. FF = aborted.
(CX)	= number of bytes received from the drive including the return code.

THE NC-TRANSPORTER | E

The NC-Transporter is a buffered Transporter which functions with both the 8001 and 8801 NEC microcomputers. When used with an 8001, it should be plugged into an 8031 expansion box.

The NC-Transporter has a 2K boot ROM that occupies addresses 00000h to 03FFFh and kills the microcomputer internal ROM when enabled. The ROM can be software enabled by setting bit 5 of I/O port 97 high, or, at reset time, by selecting the auto boot option with the jumpers. For more information see the NC-Transporter Installation Guide.

The NC-Transporter also supports interrupts. The interrupt level can be selected using the Transporter jumpers. Information on the jumpers is available in the NC-Transporter Installation Guide. To enable the interrupt facility, the processor must set bit 4 of I/O port 97 high. The processor can check interrupt status by examining bit 4 of I/O port 97.

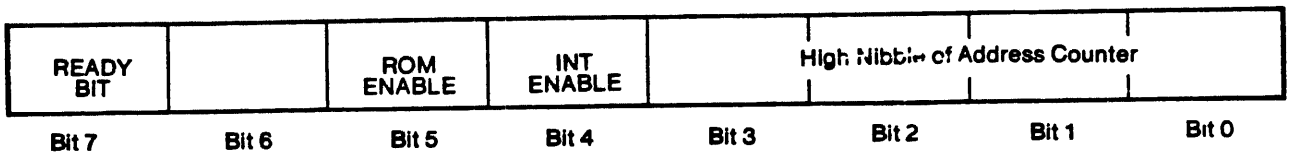
Because the NC-Transporter is buffered, DMA overrun detection circuitry is not needed.

All processor read and write operations from and to the NC-Transporter take place through the I/O ports. The following is a list of the possible processor actions and the I/O ports to which they should be directed.

Operation	I/O Port
Read Transporter Status Byte	97
Read RAM	96
Read RAM; then increase the Counter by 1	94
Write to the CAR	96
Write the Counter High Nibble, the Interrupt Enable Bit, and the Boot ROM Enable Bit	97
Write the Counter Low Byte	95
Write to RAM; then increase the Counter by 1	94

All read and write operations directed at the RAM occur at the address to which the counter is currently pointing. The counter is subsequently increased only for those commands which specify a post-increment.

As is clear from the list above, port 97 is used for a number of different operatons. A clearer understanding of the structure of port 97 may be gained from the diagram below.



Structure of Port 97

As shown above, bit 7 of the Transporter status byte (port 97) is the READY signal. When port 97 is read, bit 7 high indicates that the Transporter is ready to accept the next byte of the command vector address into the CAR. When writing to port 97, the value to which bit 7 is set is unimportant.

THE VT-180 TRANSPORTER | F

The VT-180 Transporter is a normal DMA Transporter that supports interrupts. Interrupts may be enabled by setting bit 0 of address EE high. The CPU must also be running in interrupt mode 0. After an interrupt occurs, the RESTART 8 command should be issued to the CPU.

The command address register on the VT-180 card lies at I/O address EF. Command vector address bytes must be written to this address.

The status port of the VT-180 card lies at I/O address EE. Bit 7 of this byte is the Transporter READY line. Bit 7 high indicates that the Transporter is ready to accept the next byte of the command vector address into the CAR.

This page intentionally left blank.

THE SONY TRANSPORTER | G

The Sony Transporter is a buffered Transporter which has an interrupt status bit that can be checked when a line time 60 HZ interrupt (or other interrupts) occur.

All processor read and write operations from and to the Sony Transporter take place through the I/O port 4CH-4FH. The following is a list of the possible processor actions and the I/O ports to which they should be directed.

Operation	I/O Port HEX
Read Transporter status byte	4F
Read RAM	4E
Read clears interrupt status bit	4D
Read buffer RAM; then increase the Counter by 1	4C
Write to the command address register (CAR)	4E
Write the counter high byte	4F
Write the counter low byte	4D
Write the RAM; then increase the Counter by 1	4C

All read and write operations directed at the RAM occur at the address to which the counter is currently pointing. The counter is subsequently increased by 1 only for those commands which specify a post-increment.

Bit 7 of the Transporter status byte is the READY signal. Bit 7 high indicates that the Transporter is ready to accept the next byte of the command vector address into CAR.

Bit 4 of the Transporter status byte is the INT STATUS (interrupt status) and is set upon completion of each Transporter command. This bit is cleared by reading port 4D.

This page intentionally left blank.

THE UNIVERSAL BUFFERED TRANSPORTER H

The Universal Buffered Transporter (UBT) is a basic buffered Transporter upon which many buffered Transporters for specific microcomputers are built.

There is no DMA overrun detection circuitry on the UBT and in fact no overrun detection circuitry on any buffered Transporter. Buffered Transporters perform their DMA operations on the buffer RAM which by design is sufficiently fast that no overruns can occur.

The UBT does not support interrupts, and it has no boot ROM.

All processor read and write operations from and to the UBT take place through I/O ports. The two least significant port address bits for each operation are determined by the UBT. The upper 6 port address bits are defined by host-dependent circuitry.

The following is a list of the possible processor actions and the I/O ports to which they should be directed. In the table, "n" represents the upper six bits of the port address.

Operation	I/O Port
Read Transporter Status Byte	n3
Read RAM	n2
Read RAM; then increase the Counter by 1	n0
Write to the CAR	n2
Write the Counter High Byte	n3
Write the Counter Low Byte	n1
Write to RAM; then increase the Counter by 1	n0

All read and write operations directed at the RAM occur at the address to which the counter is currently pointing. The counter is subsequently increased only for those commands which specify a post-increment.

Bit 7 of the Transporter status byte is the READY signal. Bit 7 high indicates that the Transporter is ready to accept the next byte of the command vector address into the CAR.

THE Z-80 TRANSPORTER | I

The Z-80 Transporter is a normal DMA Transporter which does not support interrupts. There is no boot ROM on the Z-80 Transporter, but there is limited DMA overrun detection circuitry.

The command address register on the Z-80 card lies at I/O address F8. Command vector address bytes must be written to this address.

The status port of the Z-80 card lies at I/O address F9. Bit 4 of this byte is the Transporter READY line. Bit 4 high indicates that the Transporter is ready to accept the next byte of the command vector address into the CAR.

This page intentionally left blank.

THE IBM PCjr TRANSPORTER | J

The IBM PCjr Transporter is a buffered Transporter which does not support interrupts. There is a boot ROM the Transporter that extends from host CPU address DF000 to address E0000. The ROM utilizes the first 1024 bytes of the 4K buffer RAM and must have exclusive use of this area. The host should not place command vectors or other command information in this section of this buffer.

All processor read and write operations from and to the PCjr Transporter take place through the I/O ports. The following is a list of the possible processor actions and the I/O ports to which they should be directed.

Operation	I/O Port HEX
Read Transporter Status Byte	3F8
Read RAM	3F9
Read RAM; then increase the Counter by 1	3FB
Write to the CAR	3F9
Write the Counter High Byte	3F8
Write the Counter Low Byte	3FA
Write to RAM; then increase the Counter by 1	3FB

All read and write operations directed at the RAM occur at the address to which the counter is currently pointing. The counter is subsequently increased only for those commands which specify a post-increment.

Bit 7 of the Transporter status byte is the READY signal. Bit 7 high indicates that the Transporter is ready to accept the next byte of the command vector address into the CAR.

This page intentionally left blank.

THE Z-100 TRANSPORTER | K

The Z-100 Transporter is a normal DMA Transporter that supports interrupts. It is possible, using the jumpers and exposed pins on the Z-100 card, to select the level at which interrupts will arrive. For details see the Z-100 Installation and Programming Guide.

Once an interrupt arrives it is necessary for the interrupt handler software to reset the interrupt mechanism before returning control to the interrupted program. The interrupt is reset by writing to the Reset Interrupt Register at I/O port FB. The contents of the write are unimportant. If the interrupt is not reset, it will be impossible for the Transporter to interrupt the processor again.

The Z-100 Transporter has the facility to support a boot ROM at IC location 7, but Corvus does not supply a ROM. A user-installed ROM is addressed using the phantom scheme to overlay an area of memory. The user selects this address space by using jumpers E2 through E5. The chart below shows how to select the phantom address.

Memory Address Bit	15	14	13	12	11-0
Jumper	E3	E4	E2	E5	No Jumper
Default (0XXX)	0	0	0	0	X

jumper A-B = bit low jumper A-C = bit high

For more information on ROM access, see the Z-100 Installation and Programming Guide.

The command address register on the Z-100 card lies at I/O address 5A. Command vector address bytes must be written to this address.

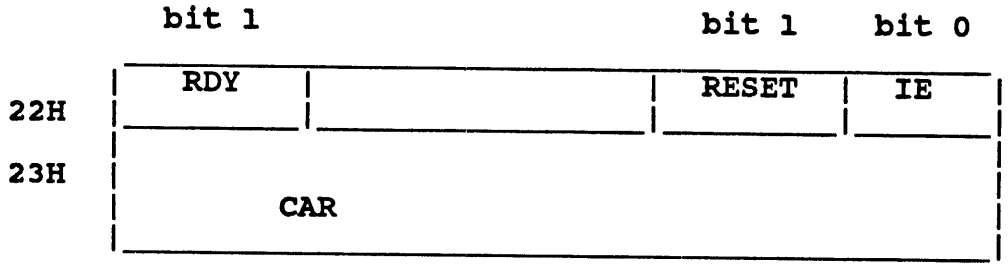
The status port of the Z-100 card also lies at I/O address 5A. The reason that this does not create confusion is that the host only writes to the CAR and only reads from the status port. The read/write line from the CPU determines which register is attached to the data lines.

Bit 0 of the status byte is the Transporter READY line. Bit 0 high indicates that the Transporter is ready to accept the next byte of the command vector address into the CAR.

The Z-100 Transporter card includes DMA overrun detection circuitry.

THE RAINBOW TRANSPORTER | L

The Rainbow Transporter is unbuffered with no underrun or overrun support. This means that most communication with the Transporter is done via DMA. A DMA cycle is guaranteed to start within 3.5 microseconds from a request so overrun or underrun will never happen. The host passes command addresses, controls interrupts and RESET with the help of 2 I/O registers (address 22H-23H).



RDY - Transporter ready to accept one byte of a command address. A write operation does not have any effect on this bit.

IE - Interrupt enable. When set (=1) the Transporter will interrupt the host. It is cleared by reading in the CAR register. This bit is cleared on power up.

CAR - Command address register. For each Omninet command a three byte address is passed in this register (MSB first). Reading this register will clear interrupt requests.

RESET - When set, the RESET line to the generic Transporter is held low and pending interrupts are cleared. This bit is cleared on power up. Interrupts must not be enabled until 50 micorseconds after reset cycle has been completed.

INTERRUPTS

The Transporter supports the DMA Controller Interrupt normally used by the extended communication option. The interrupt is of type 23H and uses interrupt vector 8CH. An interrupt request is cleared by reading the CAR (address 23H).

THE LSI-11 TRANSPORTER | M

The LSI-11 Omninet Transporter contains jumpers to select the LSI-11 control and status register (CSR) address, the interrupt vector address, and interrupt priority. There is also a jumper to enable/disable the bootstrap.

BOOTSTRAP

The Transporter board has a 256 word bootstrap area with a starting address of 773000. The bootstrap sockets accept two 256 x 8 proms compatible with MMI 6309-1J or TI 74S471. Location U23 contains the low order bytes and location U16 contains the high order bytes of the bootstrap code. When shipped, the bootstrap is enabled and contains the boot code for a DEC RL01 disk drive or the Corvus RL01 compatible disk system. The bootstrap can be disabled by removing the jumper between pins J8 and J13.

DEVICE ADDRESS

The Transporter hardware has support for a 20-bit address; However, an 18-bit address is normally used with Q-bus devices. The Transporter contains jumpers to select bit 3 to bit 12 of the device CSR address. Pins used to set the CSR address are J1-J6 and J9-J12. Pin J7 is used as a ground. A jumper installed from an address pin to the ground pin results in a zero for that bit of the device address. Since there is a single ground pin, the jumpers are installed in a daisy-chained fashion. The CSR device address is preset to 766000 as shown in the chart that follows:

Bit	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01
Pin	1	1	1	1	1	J1	J2	J3	J4	J5	J6	J9	J10	J11	J12	0	0
766000	1	1	1	1	1	0	1	1	0	0	0	0	0	0	0	0	0

Bit 17-13 are implied ones and bits 2-0 are implied zeroes. To create the preset device address of 766000, pins J1 and J4-J12 must be jumped to the ground pin J7. This can be performed with the following jumpers: J1-J4, J4-J5, J5-J6, J6-J7, J9-J10, J10-J11, J11-J12, and J12-J7.

TRANSPORTER COMMANDS

Chapter 3 of this manual describes the commands that can be used with the Transporter. The LSI-11 communicates with the Transporter by first formatting a command control block and then sending the command control block address to the Transporter through the use of two control registers. When the command is completed, a return code is placed in the result record address as specified in the command control block. An interrupt is generated when the operation is completed. For a detailed description of commands, control block formats, and return codes, see Chapter 3.

CSR - CONTROL AND STATUS REGISTER

The Control and Status Register (CSR) is a 16-bit register with a standard address of 766000. All bits can be read or written.

Bit 0-6 Not used

Bit 7 Interrupt Enable (IE)

This bit is set to 1 upon power up and hardware reset. If this bit is cleared, the Transporter cannot interrupt the processor.

Bit 8-14 Not used

Bit 15 Transporter Ready (RDY)

When set, this bit indicates the Transporter is ready to receive the next address byte of the three byte command control block address. This bit is cleared when a byte is moved into the Command Address Register (CAR).

CAR - COMMAND ADDRESS REGISTER

The Command Address Register (CAR) is a 16-bit write-only register with a standard address of 7660002.

Bit 0-7 Command Address Byte

To issue a command to the Transporter, the three byte address of the command control block must be given to the Transporter one byte at a time. Every time an address byte is placed into the CAR, the RDY bit of the CSR goes low and the next byte cannot be sent until the RDY bit returns high again. The three byte address is sent with the most significant byte first.

Bit 8-15 Not used

SOFTWARE NOTES

While the Transporter is receiving a packet from the network, it will not process a byte moved into the CAR, so the RDY bit of the CSR remains low until the Transporter can process the next byte. This leads to a situation where a software I/O driver may have to wait up to several milliseconds before the RDY goes high again. Since the Transporter processes one command at a time, the computer should not place any additonal data into the CAR after it has issued a command until the command has been completed, as indicated by the command return code.

INTERRUPTS

An operation complete interrupt is generated after the completion of each command issued to the Transporter. Before the interrupt is generated, a return code is placed in the address specified as the result record address in the command control block. Two interrupts are generated for a valid

setup receive command. The first interrupt indicates the command was accepted and the socket is set up to receive a message. The second interrupt occurs when a message is received. The program should initialize the return code byte in the result record to hex $FF before the command code block is sent to the Transporter. When a Transporter interrupt occurs, the program must check the return code value of each active Transporter command to determine which operation has just completed.

BYTE ORDER

All Omninet addresses and lengths must be specified with the most significant byte first and the least significant byte last. Additionally, some addresses and lengths are not on word boundaries.