To: Distribution

From: R. W. Franklin/E. Perry

Date: 06/12/78

Subject: Gateway Processor

The Very Distant Host protocol (VDH) project has been updated and replaced by the GateWay (GWP) project. MTB-295 described the VDH project and much of that information is still valid. That MTB should be read as background information before proceding with this MTB.

TABLE OF CONTENTS

1.0 GENERAL OVERVIEW

The VDH project was changed to the Gateway project during the BCA (Blue Cross Association) proposal bid. At that time Honeywell submitted a proposal to BCA for:

a. A Multics system
b. A packet-switched network using Level-6/06 minicomputers
c. approximately 111 Plan Node level-6 systems which would interface to the network.

The packet-switched network (b) was to be sub-contracted to BBN (Bolt, Beranek, & Newman), the Boston-based consulting firm that originally designed and developed ARPANET. Since 1969, BBN has developed five other packet-switched networks based on ARPANET technology. Their most recent offering uses the Honeywell Level-6 minicomputer with HDLC and X.25 Level-2 protocols in a packet-switched ARPANET-like network (see figure 7). This was the product which was bid in the BCA proposal and to which we would have to interface if we were awarded the contract.

The switchover to interface to a Level-6 BBN network instead of interfacing to ARPANET itself via the VDH protocol produced almost identical capabilities. In addition the design goals and basic decisions, as described in MTB-295, have changed very little.

The main differences between the old VDH Project and the new GWP project are:

a. If we connect a GWP to a network, then it is to BBN's new Level-6 packet-switched network and not to the real ARPANET. Multics may still interface to the real ARPANET via the local/distant host interface without using a Gateway Processor.
b. The protocol used to interface to the network from a Gateway Processor or to interface two Gateway Processors together is the new International X.25 Level 2 Link Access Protocol (LAP) enveloped in HDLC instead of the VDH binary synchronous protocol.
c. The packaging of the Gateway Processor has changed. The Gateway Processor is housed in a free-standing, off-the-shelf, commercial offering of Mostek with one (possibly two) special board(s) built by us. This unit can contain one or two Gateway Processors in one cabinet and uses the Z80 (Zilog-80) microprocessor instead of the Intel 8080 microprocessor.

Figures 1, 2, and 3 in MTB-295 are still valid with the "VDH" being replaced by "GWP" (for Gateway Processor). Note also that the GWP is a free-standing unit and does not connect directly to the IOM except via the ABSI cable.

1.1 Routing

When Process 1 (P1) in one host (figure 3) sends an ARPANET message to Process 2 (P2) in another host there is a destination address placed within each message by P1's host resident Network Control Program (NCP) prior to its transmission to the network. When we place a Gateway Processor next to a host (figures 1A,1B,2) the GWP receives the message containing this destination address from its adjacent host.

Note now that there are two possible ways to connect hosts together via GWPs; either with (figure 2) or without (figures 1A/1B) a network. When a network (BBN Level 6/06 IMPs) is configured (figure 2) then there is only one HDLC line on the GWP and it is connected directly to the network. In this case all messages from the host are routed through that HDLC line to the network with complete disregard of the destination address contained in the message. When there is not a network configured (figures 1A/1B), then there can be from one to three HDLC lines emanating from a GWP and now the destination address in the message must be taken into consideration for routing purposes. There is a destination table maintained in each GWP configured in a non-network environment. The destination address in each message from a host is matched with the up to three destination table entries to determine which HDLC line is to be used.

1.2 Justification

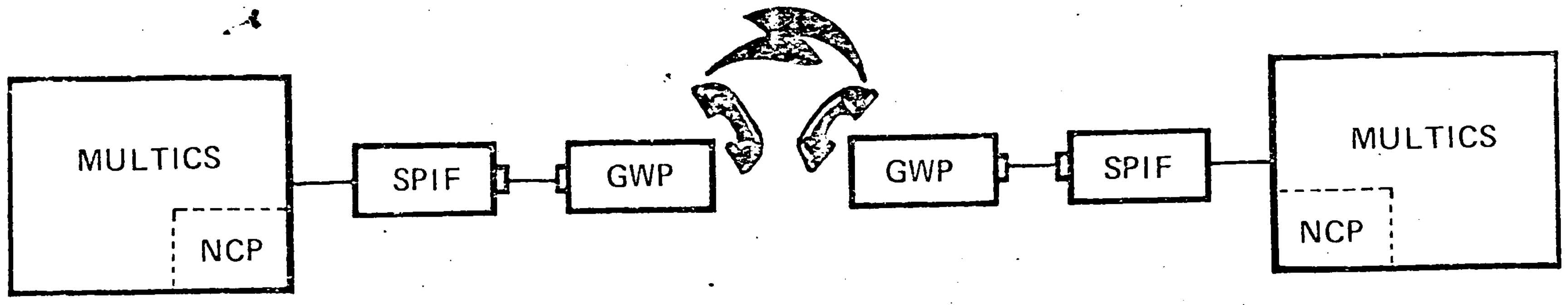
The main goal has always been to provide an interface which would allow us to connect foreign hosts together yet retain the ARPANET functionality. One way of accomplishing this was to interface a micro on one side to the SPecial InterFace (SPIF) cable (connecting a host to an IMP) using BBN specification 1822 and on the other side interface to an ARPANET-like IMP using an HDLC X.25 Level 2 protocol.

Serious consideration was given to the use of a Level 6 minicomputer and even to a H316 minicomputer. Arguments against their use were overwhelming in light of our goals and included;

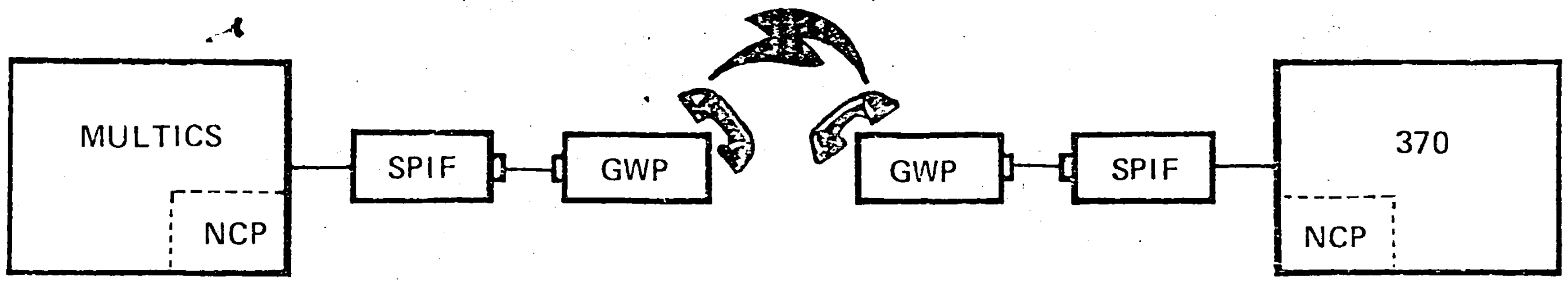
a. Much higher cost for each Gateway Processor (Minicomputers just cost more than microcomputers).
b. Requirement of a SPIF interface board to be developed for the Level 6 minicomputer. This would be a major item requiring engineering and factory (at a minimum) involvement with a high probability of an enormous cost (in excess of $300,000).

c. H316 is an obsolete piece of equipment. The development of an HDLC interface for it might prove an insurmountable task.
d. See MTB-295 for greater detail

▨ GWP WITHOUT A NETWORK (NO IMPS)



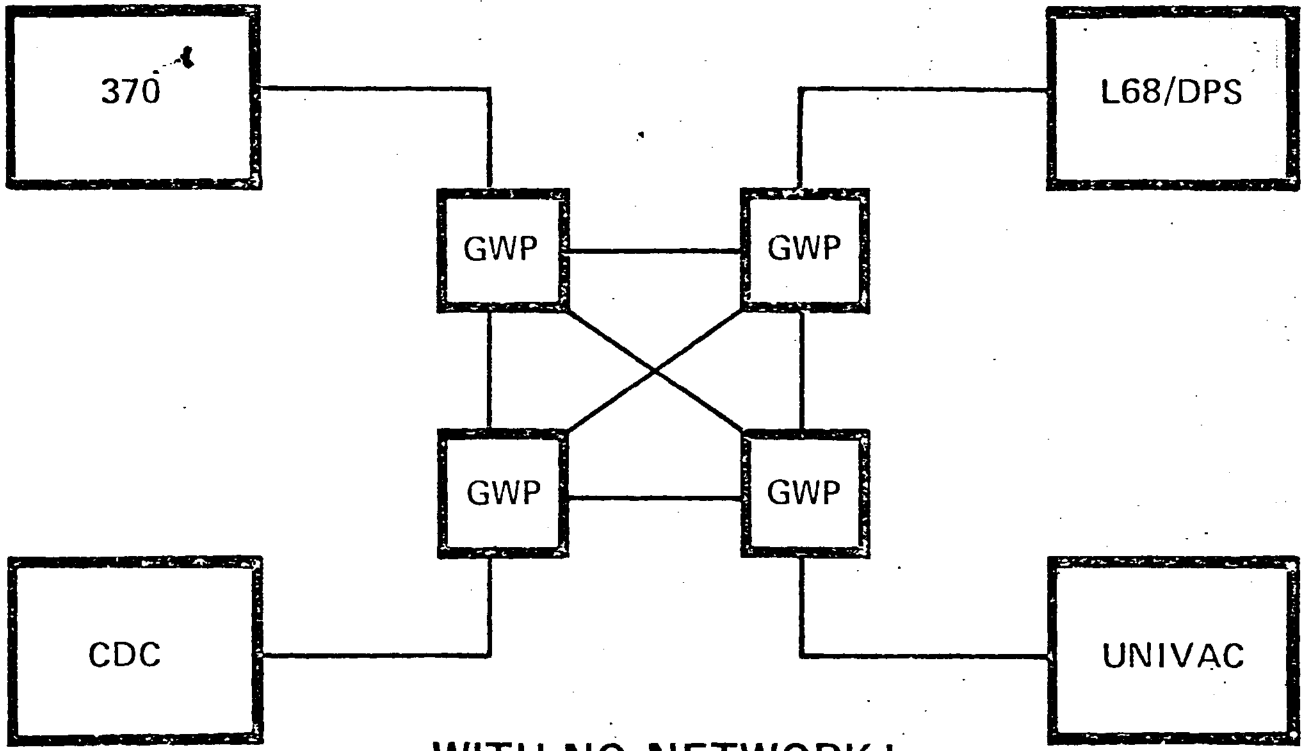
OR . . . . .



OR . . . . .

Figure 1A

▪ UP TO 4 SYSTEMS INTERCONNECTED



...WITH NO NETWORK!

Figure 1B

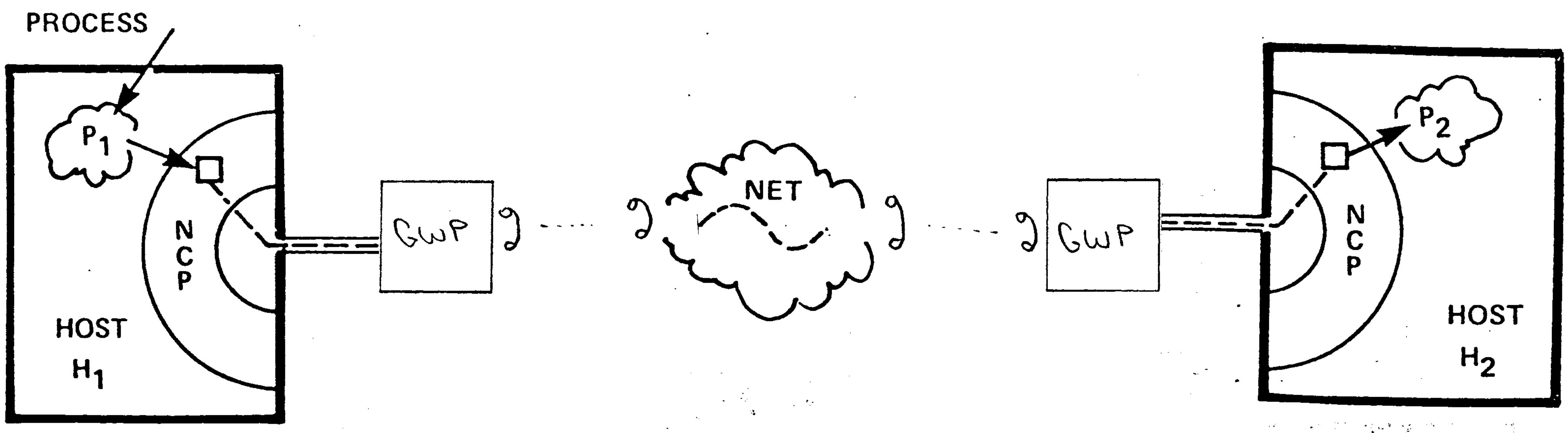


Figure 2

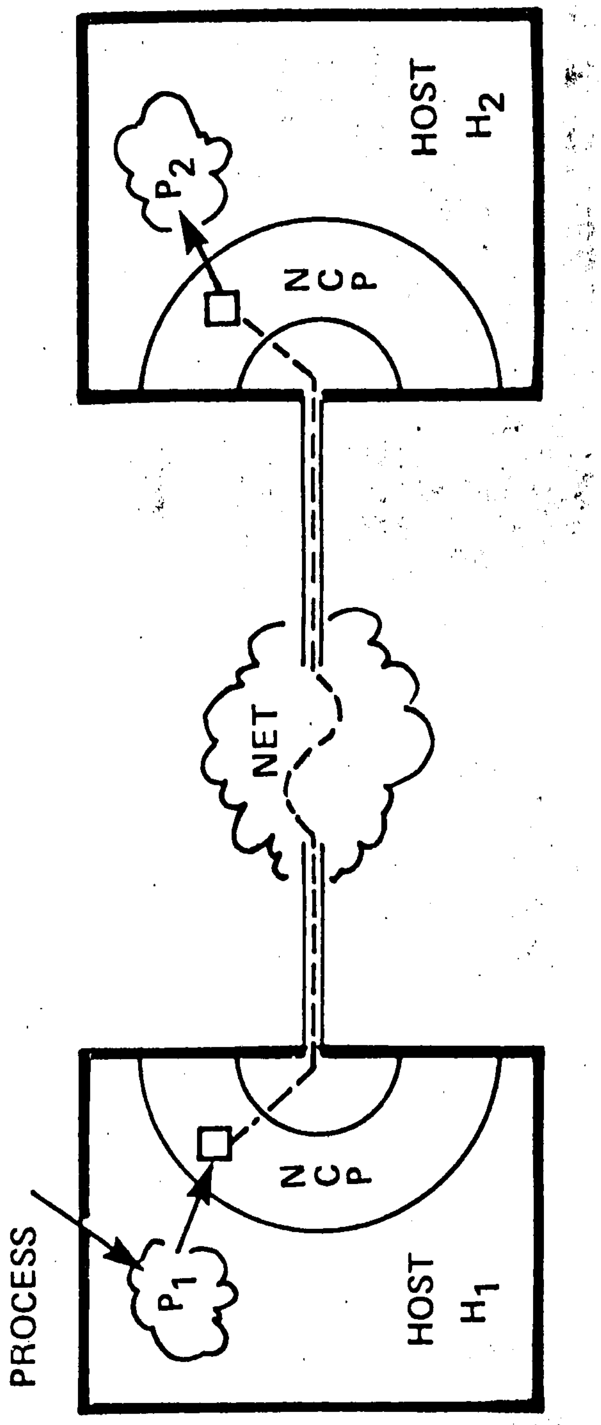


Figure 3

~~Hardware~~ Software Overview - Gateway Processor

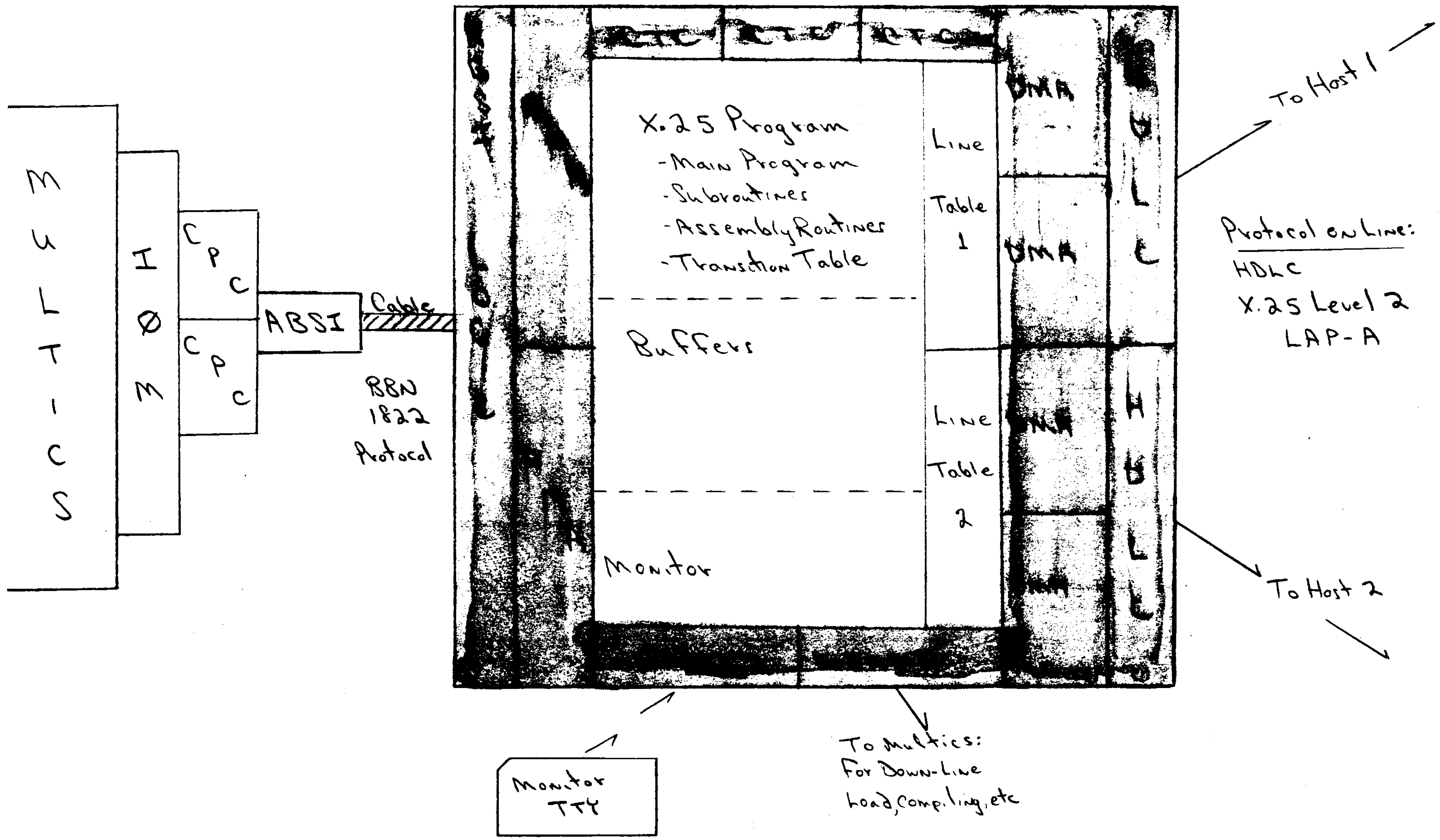


Figure 4

For discussion purposes, Figure 4 above functionally depicts the Gateway Processor.

2.0 SOFTWARE OVERVIEW

The software for the Gateway Processor has the very basic function of taking standard ARPANET messages from a local or distant host interface and routing them out the other side over one (of up to three) HDLC lines and vice versa; i.e., it performs a Gateway function of converting from one network protocol to another (ARPANET local/distant host protocol to X.25 Level 2 LAP-A protocol).

It allows a host (operating in an ARPANET network mode) to communicate with another host (also operating in an ARPANET network mode) without the ARPANET network itself being present. The gateway Processor can operate in one of two modes.

a. It can communicate directly with up to three other Gateway Processors (i.e. no network at all and up to three HDLC lines) (see figures 1A, 1B, 8) or
b. it can communicate directly to an ARPA-like network (CITINET) produced by Bolt, Beranek, and Newman (one HDLC line to the network/). See figure 2.

The software, resident in a GWP, consists of a finite state machine implementation of the Internationally approved X.25 Level 2 Link Access Protocol, LAP-A. It consists of:

a. The I/O routines required to service a standard ARPANET local/distant host interface
b. The I/O routines required to service HDLC lines
c. The finite state machine implementation itself consisting of a main program, subroutines, transition table, and line tables
d. A debugging monitor which also allows for connection to an external host for down-line loading, debugging, etc.

3.0 SOFTWARE CATEGORIES

The Software can be divided into five categories. The first four of these categories are the tools that have been assembled and/or developed which we require to produce the X.25 programs described in category 5.

3.1 Cross-Assemblers/Compilers and Simulators.

This software has been produced by the microprocessor vendors and is written in the FORTRAN language. They have been installed and are maintained on Multics System M as part of an unscheduled effort to build a microprocessor software factory. This includes a lot of compilers and assemblers for microprocessors that we are not currently using and in those cases the software will probably not work without more effort. This includes the Signetics-2650, the Motorola-6800, the Fairchild-F8, the RCA-1502, etc microprocessor software. Those that we are currently using are the Zilog Z80 and INTEL 8080 microprocessors. The pertinent software for these microprocessors include the;

3.1.1 INTEL PL/M cross-compiler

PL/M is a high-level, PL1-like programming language, especially designed to simplify the task of system programming for the INTEL 8-bit family of microcomputers; the 8008 and the 8080. It provides an effective software tool suited to the requirements of the microcomputer system designer and implementor. It gives the programmer control of the processor sufficient for the needs of system programming, but provides automatic control of many specific processor resources; e.g., registers, memory, and stack. In consequence, PL/M programs can enjoy a high degree of portability between systems. It has been designed to facilitate the use of modern techniques in structured programming. The cross-compiler version is the older version in that it is not being extended or updated by INTEL anymore. The X.25 main program and subroutines of our software are written in this language. A newer disk-resident version of the PL/M compiler (not a cross compiler) exists on INTEL's disk-resident operating system. It may prove advantageous to use that version in the future if it proves to generate better code and becomes available to us.

3.1.2 INTEL MAC80 cross-assembler

This FORTRAN-written assembly program translates a symbolic representation of 8080 microcomputer instructions and data into a form which can be executed by the 8080 microcomputer.

The Macro Assembler accepts 8080 assembly language, including macro definitions and references, as input, and produces 8080 object code as output. It is designed to run on any general-purpose digital computer with sufficient memory and an integer size of 30 bits or more. It is written in ANSI Standard FORTRAN (1966), and is designed to be compatible with most standard systems software with minimal modifications.

3.1.3 ZILOG PLZ cross-compiler

PLZ is Zilog's PL1-like cross compiler and is not yet in a running state on System M. Since INTEL 8080 code is a subset of Zilogs Z80 code we have been able to stay with PL/M while switching to the Z80 microprocessor from the INTEL 8080 microprocessor. We will continue to attempt to get PLZ running and then determine if it is generating better code or not.

3.1.4 ZILOG cross-assembler

This FORTRAN-written program accepts Zilogs Z80 assembly code and translates it to an object format which matches the INTEL 8080 object format.

3.1.5 INTERP/80 Simulator

INTERP/80 is a FORTRAN IV program which provides a software simulation of the INTEL 8080 CPU, along with execution monitoring commands to aid program development of an 8080 program. It accepts machine code produced by the INTEL 8080 Assembler or PL/M 8080 compiler, along with execution commands from a time-sharing terminal, card reader, or disk file. The execution commands allow manipulation of the simulated 8080 memory and CPU registers. In addition, operand and instruction breakpoints may be set to stop execution at crucial points in the program. Tracing features are also available which allow the CPU operation to be monitored. It provides symbolic reference to storage locations as well as numeric reference in various number bases.

3.1.6 ANALYZER Program

The Analyzer program is a Multics PL1 program which accepts as input a transition table and a set of control arguments and produces a matrix output for a specified target machine.

3.1.7. EPROM Burning Program

We have loaded a program from paper tape onto Multics which gives us the capability of down-line loading the EPROM burning program into the microcomputer so that we may produce our own EPROMs.

NOTE that the object format created by all four cross assemblers/compilers listed above are consistent in that the same loader can load all four.

3.2 Utilities

A hodgepodge of utilities exist on Multics to perform services such as

3.2.1 Indent/tabbing routines to indent assembler or compiler source files

3.2.2 Conversion routines to allow the source to be written in lower case while the compilers and assemblers require upper case

3.3 Mostek Monitor routines

The Gateway Processor uses a Mostek SDB-80 Software Development Board. A standard feature of the SDB-80 is a complete package of development software aids which are resident in the five MK-34000 2K*8 ROM memories located on the board. This firmware includes a sophisticated operating system, debug package, assembler, and text editor. Among the many features provided are execute and breakpoint commands, console routines for examining and/or modifying memory and port locations, object load capability for both absolute and relocatable object modules, I/O driver routines for a variety of standard peripheral devices, and channeled I/O for user defined peripheral drivers. The presence of this software in ROM provides instant access to these development aids, eliminating the time-consuming requirement of loading the software from some peripheral device into RAM. Another key feature of having the development aid software in ROM is that the entire RAM space is available for the user's programs. This set of software is currently being used in our debugging mode but the question as to whether it will remain in the released product has not been addressed.

3.4 Our Monitor Routines

We have developed a small monitor, tailored after Mosteks Monitor, to accomplish additional functions we need for a debugging environment. This monitor allows us to connect to Multics with the microcomputer between the terminal and the host (Multics e.g.); i.e., the micro is acting as a black box in a mode transparent to Multics and the user until certain control characters it is looking for are input. The monitor provides us with the following functions;

3.4.1 Down-Line Load

When a control-escape character is input, our monitor assumes that Multics (or whatever host it is connected to) is beginning to transmit an object file to the terminal; i.e. we have issued a print of the object file or we are in an editor and have requested a print. The command to the host is complete except the carriage return character has not yet been typed so the host has not yet begun to transmit the object file. The monitor will issue the carriage return character to the host, turn off the terminal, and begin looking for an object format record. Any line beginning with a colon in column 1 is considered an object format record and will be loaded into memory by the monitor. If a checksum error is encountered the load is aborted by sending a break to the host, turning on the terminal, and exitting from the down-line load into the transparent mode. Correct completion is assumed when a record with a zero count is encountered. At that time the terminal is placed back on-line and the monitor returns to the original transparent mode.

3.4.2 Switch Systems

When a control Q character is input, a jump to Mosteks monitor is executed making available the Mostek debugging commands. At this time the line connected to Multics is ignored; i.e., no characters typed in are transmitted to the host and any characters sent out from the host are ignored by the micro. It is imperative then to defer Multics messages prior to doing this so that they will not be lost. Resumption of the transparent mode is affected by transferring back to our monitor.

3.5 X.25 Software

We have developed programs which accept the ARPANET local/distant host interface on one side and the X.25 Level 2 Link Access Protocol on the other (see figure 4). This body of software is the heart of the Gateway Processor and the reason for its existence.

The X.25 Level 2 Link Access Protocol is the LAP-A version (2-way Simultaneous Asynchronous Response Mode - ARM) as defined in Appendix A of the Standard Network Access Protocol (SNAP) specification for Datapac. It has been implemented using a Finite State Machine concept with machine independent transition tables. This approach lends itself to an easy upgrade to LAP-B (Asynchronous Balanced Mode) or Level 3 (virtual call) when/if they are desired/required. In fact it may be possible to have both the LAP-A (ARM) and LAP-B (ABM) capabilities expressed within the same transition table. The X.25 program itself can be broken into four areas; the main program and subroutines, the transition table, the line tables, and the assembly language routines. There have been two prior developments similar to this outside of the Multics project (but in HIS) to generate an X.25 program. Most of our GWP implementation is based on their work. The following is background information on those efforts.

The Analyzer program, described in 3.1.6 above, was written by B.Chittenden to accept as input the Finite State Machine Transition Tables (as defined by R.Hay) and produce matrix tables as output dependant upon the specified target machine. An implementation effort on the Level-6 minicomputer was then undertaken by R.Hay, R.Crawford, and S.Ramsdell to prove the validity of the tables and the concept. They developed a Main Program, Subroutines, and a set of HDLC routines and debugged them using a transition table. That transition table had been run through the Analyzer program specifying the Level-6 as the target machine. After a minimum of testing a comparable effort was begun on a Multics PL1 version. It was written by K.Marietta and was checked out using TTY simulated I/O and a transition table that had been run through the Analyzer specifying Multics as the target machine. That program is currently being tested by having one line table drive another and vice-versa and is also being modelled (using SIMSCRIPT techniques) by an outside contractual consultant. The GWP was then begun using all of the above as its base. The GWP software has been produced by implementing a Finite State Machine, event-driven, set of programs. The components of the X.25 software can be placed into 4 categories which are;

3.5.1 The Transition Table

The Transition Table is where the definition of the X.25 Level 2 Link Access Protocol is expressed. The protocol is defined by a set of machine independent states, inputs, and outputs. All possible combinations are described for states, inputs, and outputs thus the term Finite State Machine. The Transition Table, together with parameter options, is run through the ANALYZER program on Multics to produce a two-dimensional array specific for the INTEL 8080 microcomputer. This array, indexed by state and input, yields a new state and action routine(s) as output. The Transition Table, together with the state diagrams, completely document the X.25 Level 2 Link Access Protocol. The Analyzer program accepts options such as:

a. Range of input parameters
b. Word size in bits of target machine (e.g. 8 bits).
c. Language of target machine (.e.g. PL/M).

3.5.2 Line Tables

Each possible (up to three in the initial GWP hardware implementation) HDLC X.25 line keeps its particular data in a line table dedicated to line dependent information. Items such as current state, flags, X.25 sequence numbers, timers, input/output queue pointers, supervisory frame buffers, etc are maintained in each table. Basically a line table is analagous to an MCS TIB for a particular line. There are approximately 100 bytes of information in a line table.

3.5.3 Main Program and Subroutines

The main program and subroutines are written in PL/M and comprise the dispatcher or executive. The Main Program initializes everything and, once started, is event-driven. It contains all of the interfaces to the assembly written routines for I/O, debug, get_events, etc., and manages the queues and buffer space. There are currently 95 possible actions which invoke one or more of 60 subroutines. Events which drive the main program are;

a. Frame received (Information, Supervisory, or unnumbered)
b. Output Complete - i.e. a line, in transmission, has just finished transmitting a complete message.
c. Output Available - The GWP has just received a complete message from the Host computer.
d. Timeouts - There is a timer associated with each HDLC line which is set and reset as defined in the X.25 protocol definition. The timer is continuously running; i.e. once it is stopped due to an expected event occurring it is immediately restarted. The single timer for each line can thus be used for both timeout and idle timer purposes. One Counter Timer Component (CTC) chip is used for each HDLC line. The chip itself has four 8-bit counters which can be used separately or in combination. One channel of the CTC is used to control the baud rate of the HDLC chip while the other three channels are tied together to provide a timer with a maximum time period of 6.83 * 256 seconds using a 2.458 Mhz system clock. The time period is variable depending on the baud rate of the HDLC chip, the protocol, and what exactly is being done at the time. Normal T1 timeout periods are set to slightly greater than three times the duration of time it takes to transmit the longest length message over that HDLC line.
e. Disconnects

Each event that is passed to the main program states; what type of event this is, the line table to which it applies, and other pertinent information.

3.5.4 Assembly Routines

The assembly routines are written in the Z80 microprocessor machine language and consist of the I/O, event-queue, and interrupt routines.

4.0 PROTOCOLS

The GWP takes standard ARPANET messages from a host computer and transmits them to another GWP or to a BBN network. The protocol used to transmit the standard ARPANET message is different on each side of the GWP.

An ARPANET message can vary in length from 32 up to 8095 bits; the first 32 of which are control bits called the leader. The leader is also used for sending control messages between the host and its IMP. The remainder of the message is the data, or the text.

4.1 Host to/from Gateway Processor

The protocol accepted by the GWP from the host local/distant cable is defined by BBN specification 1822. The BBN 1822 specification defines the asynchronous bit serial handshake protocol implemented by all ARPANET SPecial InterFace (SPIF) units. The "information" transmitted via this protocol consists of standard ARPANET local/distant host messages.

4.2 GWP to/from Network or another GWP

The protocol used to interface a GWP to a BBN Network or to another GWP is as shown in Figure 5.

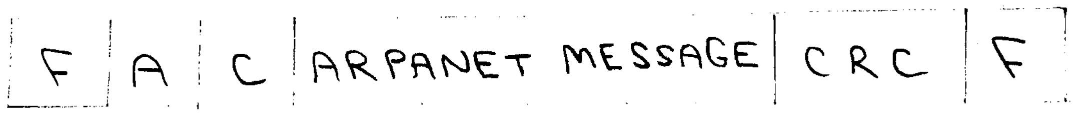


Figure 5

where:
- F = HDLC flag = 7E (hex)
- A = HDLC address byte (used in X.25 protocol)
- C = HDLC control byte. Used for X.25 Level-2 LAP information.
- ARPANET message = Regular Message (up to 8160 bits)
- CRC = 16 bit (2 byte) SDLC polynomial X(16) + X(12) + X(5) + 1.

This is the Internationally approved X.25 Level 2 Link Access Protocol (LAP-A) enveloped in HDLC for transparency at the bit level. X.25 Level 2 LAP-A is defined to be a 2-way simultaneous, asynchronous response mode (ARM) protocol.

5.0 HARDWARE OVERVIEW

Figure 6 is a top view of a dual Gateway Processor cabinet. It contains a four board Gateway processor in each half of the cabinet. The cabinet, power supply, SDB-80, and RIO boards are standard, off-the-shelf Mostek products. The HDLC Controller and the ABSI Controller are on prototype boards which contain the hardware shaded in figure 4 (not the UART which is on the SDB-80 board itself). The first GWPs are designed and produced by the Gateway Project.

The RIO board is not available until the end of 1978 and will contain 16K of EPROM and a USART. If it is not available in time then we will have to (on a temporary basis) functionally produce it ourselves. The point to be made here is that all hardware is off-the-shelf, standard commercial products except for the two controller boards produced by us.

The stand-alone Gateway Processor connects to another GWP or an X.25 network via a standard 25-pin EIA RS-232-C cable. Each Gateway can provide up to three such connections if configured in a non-network environment but only one such connection if configured to a BBN network.

Each Gateway also connects to the IMP side of one special interface (SPIF) cable. The other end of the SPIF cable connects to any computer vendor's ARPANET local or distant host interface. For Honeywell Series 60 systems the cable connects to an ABSI unit, which is itself cabled to two common peripheral channels in an IOM. The ABSI unit and ABSI SPIF cables are commercially available from private vendors.

The Gateway provides either a local or distant host interface to an ARPANET Host. A panel switch on the Gateway selects the desired interface. Certain other panel controls may prove desirable as the prototype Gateway is developed. These include switches to force a "Host Down" or "IMP Down" condition.

FUNCTIONAL TOP VIEW OF DUAL GATEWAY UNIT

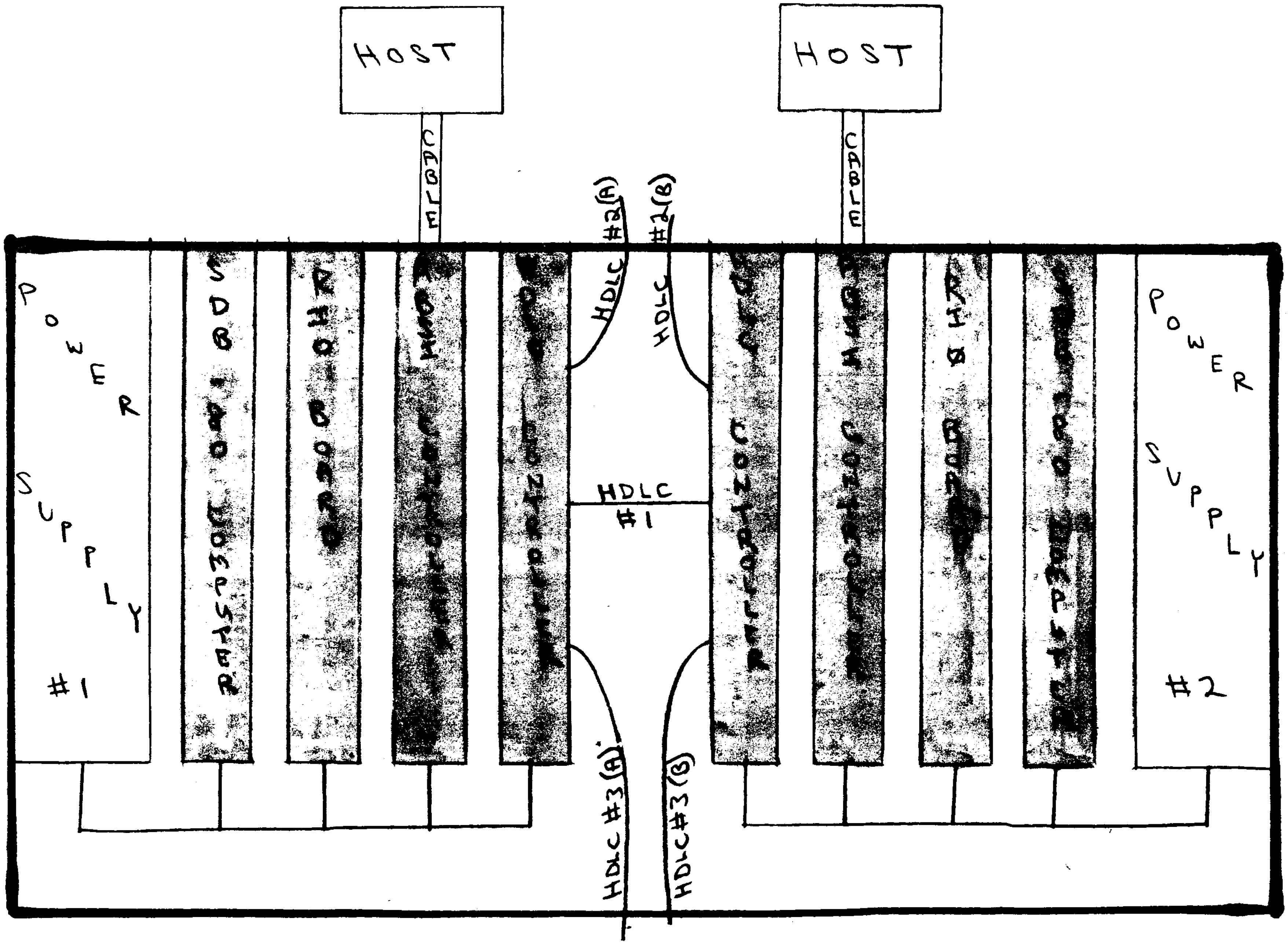


Figure 6

6.0 HARDWARE CATEGORIES

6.1 HDLC CONTROLLER

Currently the HDLC Controller consists of one transmitter and one receiver with the necessary EIA RS-232-C compatible interface signals to comply with CCITT Recommendation X.21bis (physical level of X.25). The logical HDLC interface is managed by a software programmable Zilog or Mostek SIO chip. Channel A of the SIO (at I/O Ports 24 and 25, hex) serves as the receiver. Channel B (at Ports 26 and 27, hex) serves as the transmitter. The SIO manages flag generation and detection, automatic zero-insertion and deletion, and automatic CRC generation and checking. The EIA RS-232-C Data Carrier Detect (DCD) and Clear To Send (CTS) signals are monitored continuously. Changes of state on these signal lines can cause interrupts. Programming techniques and pertinent electrical characteristics are described in Zilog's "SIO Product Specification, March 1978."

Two direct memory access (DMA) chips permit CPU-independent data transfer. Programming techniques are described in Zilog's "DMA Product Specification." The "ready" inputs of the DMA's should be programmed as "active low." The receiver DMA is located at (hex) Port 34, the transmitter DMA at port 30.

A Counter Timer Component (CTC) generates the baud rate for both the SIO receiver and transmitter. Its I/O Port is 2C (hexadecimal). Programming specifications are available in the Mostek 3882 data sheet. The ZC/TO output of channel zero is divided by two and applied to the SIO baud-rate clock input. The system clock runs at 2.458 Mhz. Thus proper selection of the pre-scaler value and the block count will determine most any standard baud rate. The baud rates for X.25 transmission are totally under software control. For MR7.0 any standard rate up to 76.8 kBaud can be used in the X.25 network. Higher rates can be achieved through additional hardware. The table below illustrates the range of transmission rates and how they are achieved.

Baud Rate	CTC Mode	Pre-scaler	Block Count
300	Counter	÷16	256
1200	Counter	÷16	64
4800	Counter	÷16	16
9600	Counter	÷16	8
19.2k	Counter	÷16	4
38.4k	Counter	÷16	2
56.0k	Timer	-	22
76.8k	Counter	÷16	1

6.2.0 ABSI CONTROLLER

The Gateway Processor (GWP) Asynchronous Bit-Serial Interface (ABSI) controller is logically split into two elements, a receiver and transmitter. Serial data from the host enters the GWP through an input shift register in the receiver. An output shift register in the transmitter sends serial data to the Host.

The ABSI controller has three entities that are under program control: (1) an 8-bit transmitter data buffer, (2) an 8-bit receiver data buffer, and (3) an 8-bit command/status register. In addition, a Counter Timer Component (CTC) provides interrupt prioritization and interrupt vectors. Data and the command/status registers and the CTC are accessed by the GWP CPU as I/O ports. Port channel numbers are shown below:

Logical Port	Physical Port No. (hex)
ABSI command/status	10
ABSI receiver data	14
ABSI transmitter data	18
ABSI CTC	1C-1F

The ABSI discipline controls the incoming data rate so that receiver overruns cannot occur. Both the receiver and transmitter have a shift register and a data buffer register to increase data throughput.

6.2.1 Receiver Operation

Following the receipt of the Enable ABSI Receiver command the receiver raises the "Ready for Next Host Bit" signal on the GWP-Host interface. Serial data bits sent from the Host are shifted into the receiver's input shift register. When 8 bits have been received they are loaded into the receiver data buffer, and: (1) the "Receiver Data Available" status bit is set, (2) the receiver's ready line goes low, and (3) an interrupt is signalled (only if it is enabled).

When the receiver data port is read (by an I/O input instruction, or DMA read) the "Data Available" status bit is reset, and the ready line goes high. Upon receipt of the next 8 bits the receiver will not accept any more data from the Host if the receiver buffer is full (has not been read). If (or when) the receiver buffer is empty data is again loaded into the receiver buffer from the input register. The cycle continues to repeat until the "Last Host Bit" signal appears on the interface.

When the last bit of the message is shifted into the input register the receiver performs any necessary padding to produce a full 8-bit byte. One additional byte is generated if necessary (see BBN Report #1822). As the final byte of the message (including padding bytes) is loaded into the receiver buffer the "End of Message" (EOM condition) status bit is set. This bit will not be reset until the receiver is enabled again. As usual, the "Data Available" status bit is also set when the last message byte is loaded into the receiver buffer. However, this bit is reset when the data port is read. If the EOM interrupts are enabled, an interrupt is signalled. The receiver enters the disabled state, and will keep the "Ready for Next Host Bit" interface signal low until enabled.

6.2.2 Transmitter Operation

Once enabled, the ABSI transmitter attempts to transmit to the host all data words output to the transmitter data port. No data should be output until the transmitter is enabled. During the transmission of the last bit (lsb) of the last message byte the ABSI transmitter raises the "Last Imp Bit" signal on the Host-GWP interface. This signals the host that the message is complete.

The transmitter has one buffer register that loads an output shift register. A data word output to the transmitter data port is loaded into this transmitter buffer, causing the "Transmitter Buffer Empty" status bit to reset, and the transmitter ready line to go high. If the shift register is empty the data word is immediately transferred into it.

The movement of data from the buffer to the shift register causes: (1) the "Transmitter Buffer Empty" status bit to be set, (2) the ready line to go low, and (3) an interrupt to be signalled (if enabled). If the transmitter has been given the "Last Byte Follows" command before a data byte is output, only the first event (1) occurs. If the shift register is not empty the data word remains in the buffer until the shift register empties. The CPU reads the status register to prevent overwriting this data.

The transmitter detects the last byte of the message by explicit notification from the GWP CPU. The CPU sends the "Last Byte Follows" command to the command/status port prior to the output of the last message byte to the transmitter data port. When the last byte has been transmitted through the output shift register the "End of Message" (EOM) status bit is set, indicating that the transmitter is disabled and will not acknowledge the "Ready for Next IMP bit" signal from the host. The EOM bit remains set until the transmitter is enabled. The transmitter should not be enabled for the next message transmission until the

EOM/Disabled status bit is set at the end of the current message. Note that all messages transmitted by the ABSI transmitter must contain an integer number of 8-bit bytes. No bit padding is performed by the transmitter.

6.2.3 Status

Status information available at the command/status port is shown below. Most of the status bits are self-explanatory. Note that the bits indicating the EOM condition also indicate the disabled state. Both receiver and transmitter EOM status bits are set by a GWP power-up or GWP reset. The "Disable ABSI Receiver" and "Reset ABSI Receiver" commands set the receiver EOM status bit, the corresponding transmitter commands set the transmitter EOM bit in a similar fashion.

The two error flags are set when the Host or the GWP disconnects its "Master Ready" from its "Ready Test" ABSI interface signal. Once these status bits are set they remain set until cleared by indivdual "Clear Receiver Error Flag" and "Clear Transmitter Error Flag" commands. The two "Master Ready State" bits are set if their respective "Master Ready" and "Ready Test" signals are connected. Otherwise the bits are reset. The data bit numbering shown is not Honeywell standard numbering. "D0" refers to the least significant bit.

Bit	Meaning
D0	Receiver data available
D1	Receiver at End of Message(EOM)/Receiver Disabled
D2	Host or GWP ABSI has been down; Receiver Error flag
D3	GWP Master Ready state
D4	Host Master Ready state
D5	Transmitter Buffer Empty
D6	Transmitter at End of Message(EOM)/Transmitter Disabled
D7	Host or GWP ABSI has been down; Transmitter Error flag

6.2.4 Commands

The commands and their bit values are given below. An "x" implies a don't care condition. The command byte can be logically split into three fields. A single command byte output to the command/status port can specify one action to be taken for each of these fields. The three low-order bits of the command byte (D0-D2) specify transmitter element commands. The next three higher-order bits (D3-D5) specify receiver element commands, while the two highest-order bits (D6-D7) specify general ABSI controller commands.

Bit D0 refers to the least significant bit.

Command Field			Command
DD	DDD	DDD	
76	543	210	
xx	xxx	000	Transmitter NOP (Null command)
xx	xxx	001	Disable ABSI Transmitter
xx	xxx	010	Enable ABSI Transmitter
xx	xxx	011	Reset ABSI Transmitter
xx	xxx	100	Clear Transmitter Error Flag
xx	xxx	101	Last Byte Follows
xx	000	xxx	Receiver NOP (Null command)
xx	001	xxx	Disable ABSI Receiver
xx	010	xxx	Enable ABSI Receiver
xx	011	xxx	Reset ABSI Receiver
xx	100	xxx	Clear Receiver Error Flag
xx	101	xxx	Interrupt on Receiver Data Available
00	xxx	xxx	NOP (Null command)
01	xxx	xxx	Set ABSI Master Ready
10	xxx	xxx	Reset ABSI Master Ready

Enable Commands

The enable commands put the transmitter or receiver into operation. The EOM/Disabled status bit is reset, indicating an active, enabled element. The "ready" lines to the DMA chips are always high (inactive) when an element is disabled. Once enabled, the ready lines assume their correct state.

Disable Commands

The disable commands force the EOM/Disabled status bit to be set. No other alteration of the transmitter or receiver occurs. Any partially received or partially transmitted bytes remain in the shift registers. Data should not be read from or written to the data buffers when the elements are disabled; the results of data buffer I/O when an element is disabled are indeterminate. The disable command halts receiver or transmitter activity. An ensuing enable command will continue that activity.

Reset Commands

A reset command clears all internal registers and prepares the receiver or transmitter for correct operation. In the receiver the EOM/Disabled status bit is set, the "Receiver Data Available" bit is reset, the ready line goes inactive, and the Error Flag is reset. In the transmitter the EOM/Disabled status bit is set, the ready line goes inactive, and the Error Flag is reset, but the "Transmitter Buffer Empty" bit is set. The reset element is put into a disabled state. Note that neither reset command disables or resets the interrupt structure (see "Interrupts" below). To reset the interrupts a hardware reset or software resets to specific CTC channels are required.

Clear Error Flag Commands

Following a disconnection of the Host's or GWP's ABSI "Master Ready" and "Ready Test" signals, the error status bits are set. The "Clear ... Error Flag" command resets the status bit. Two separate flags with individual resets permit differential handling of an error condition by the receiver and transmitter. Note that both status bits can be reset with one command if desired.

Interrupts

Interrupts from the ABSI controller are actually generated by the CTC chip. If interrupts are desired this chip must be programmed with a block count of one, interrupts enabled, in the appropriate channels. For CTC programming information, see the MOSTEK MK3882 data sheet. Channel zero of the CTC (I/O Port 1C) handles the interrupts generated by the receiver and transmitter error conditions (see "Clear Error Flag Commands" above), channel one (I/O Port 1D) handles receiver interrupts, and channel two (I/O Port 1E) handles transmitter interrupts. The CLK/TRG input of each CTC channel should be programmed to decrement the CTC counter on a rising edge. Interrupts may be enabled and disabled by commands to the CTC channels.

If transmitter interrupts are enabled the transmitter will generate an interrupt signifying it is ready to accept data. The first interrupt will occur immediately after the "Enable ABSI Transmitter" command is given. Thereafter an interrupt occurs every time the transmitter data buffer empties. However, no interrupts are generated after the last byte (preceeded by the "Last Byte Follows" command) is loaded into the data buffer. Two interrupt modes are possible for the receiver. Following a power-up or GWP hardware reset the receiver exists in an "Interrupt on End-of-Message" mode. An interrupt will be generated when the last byte of the received message (including any padding) is loaded into the receiver data buffer. The "Interrupt on Receiver Data Available" command sets the receiver into the other interrupt mode. Subsequently, interrupts are generated whenever a new byte is loaded into the receiver data buffer. This interrupt mode is canceled by a power-up reset or GWP hardware reset only (receiver and transmitter resets do not affect it).

Master Ready Commands

The "Set Master Ready" and "Reset Master Ready" commands connect and disconnect the GWP's ABSI Master Ready and Ready Test signals, respectively.

6.2.5 DMA (Direct Memory Access) Operation

The ABSI transmitter and receiver DMA chips should be programmed for active-low ready signals. The ready signals are partially generated from the "Transmitter Buffer Empty" and "Receiver Data Available" status bits. If the receiver or transmitter is disabled, its ready signal is a logical one, or high level, signalling the "not-ready" condition.

The receiver will not signal a "ready" condition once the last message word has been read from its data buffer. Thus the size of the received message can be determined from the receiver DMA block count register. Similarly, the transmitter will not signal a "ready" condition when its transmitter data buffer goes empty if it has been given the "Last Byte Follows" command. The transmitter DMA block count register should contain less than the number of bytes in the message to be transmitted. If it contains one less, for example, the DMA block count runs out (and an interrupt may be signalled by the DMA chip) when the ABSI transmitter has accepted all but the last message byte. The CPU then outputs the "Last Byte Follows" command to the ABSI controller command/status port before output of the final message byte.

6.3 Other Special Hardware

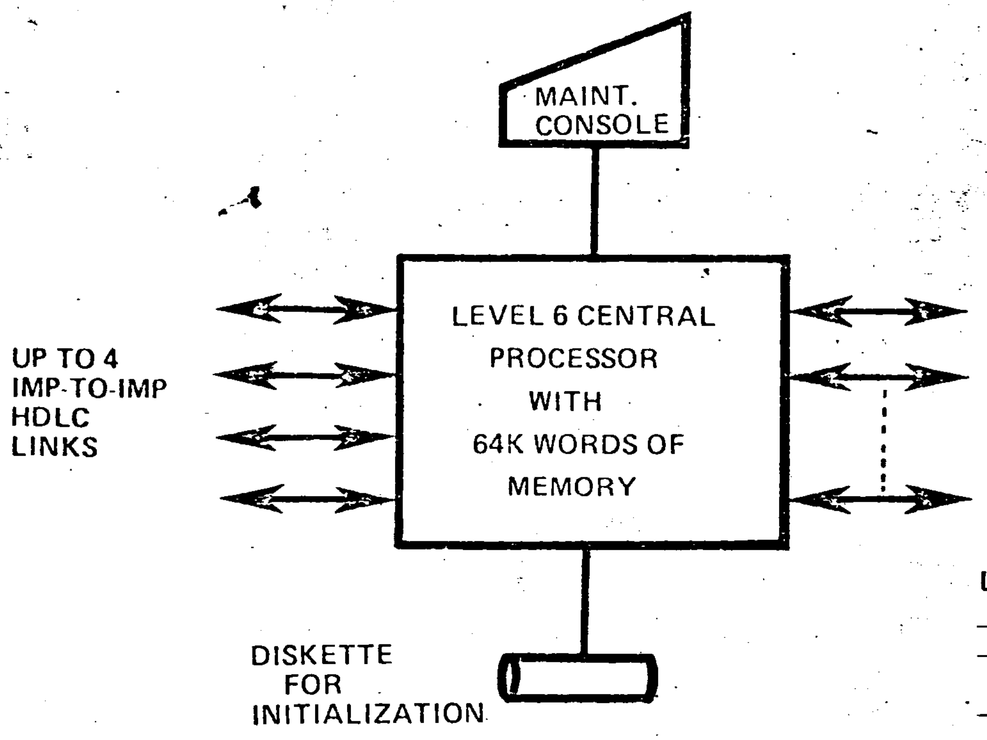
6.3.1 UARTS and USARTS

The Gateway Processor prototype contains additional interfaces used for test and development. These interfaces may be included in the eventual product; that decision must be made as development progresses. The Mostek SDB-80 board contains a UART and free-edge connector to a standard RS-232-C interface. The prototype utilizes this connection to provide a console teletype interface. An additional SIO chip and associated line drivers and receivers establishes a TTY interface for cabling the Gateway to a Multics system. The monitor routine (resident in EPROM) manages this Multics connection, providing a normal terminal interface and down-line loading capability at 1200 baud.

6.3.2 Counter Timer Components

The X.25 software uses Counter Timer Components (CTC'S) to monitor certain timing constraints in the X.25 protocol. One CTC is provided for each X.25 connection. Three of the four channels of the CTC are wired together to create a three-channel timer. The remaining channel is used to generate the transmission baud rate for the X.25 connection. The three-channel timer may generate CPU interrupts upon time-out. The maximum time period for the three-channel timer (with a 2.458 Mhz system clock) is in excess of 29 minutes.

LEVEL 6 INTERFACE MESSAGE PROCESSOR



MAINT. CONSOLE
LEVEL 6 CENTRAL PROCESSOR WITH 64K WORDS OF MEMORY
UP TO 4 IMP-TO-IMP HDLC LINKS
UP TO 12 IMP-TO-HOST HDLC LINKS
DISKETTE FOR INITIALIZATION

LEVEL 6 IMP:

- MONITORS HOST & LINE TRAFFIC
- DISTRIBUTED ADAPTIVE ROUTING (ALTERNATE ROUTING & LOAD LEVELING)
- SELF-MONITORING DATA $\longrightarrow$ NETWORK MEASUREMENT CENTER
- POWER FAILURE DETECTION
- AUTO SHUTDOWN & RESTART
- TOTAL THROUGHPUT OF ~ 100Kb/SEC.

Figure 7

DESIGN APPROACH

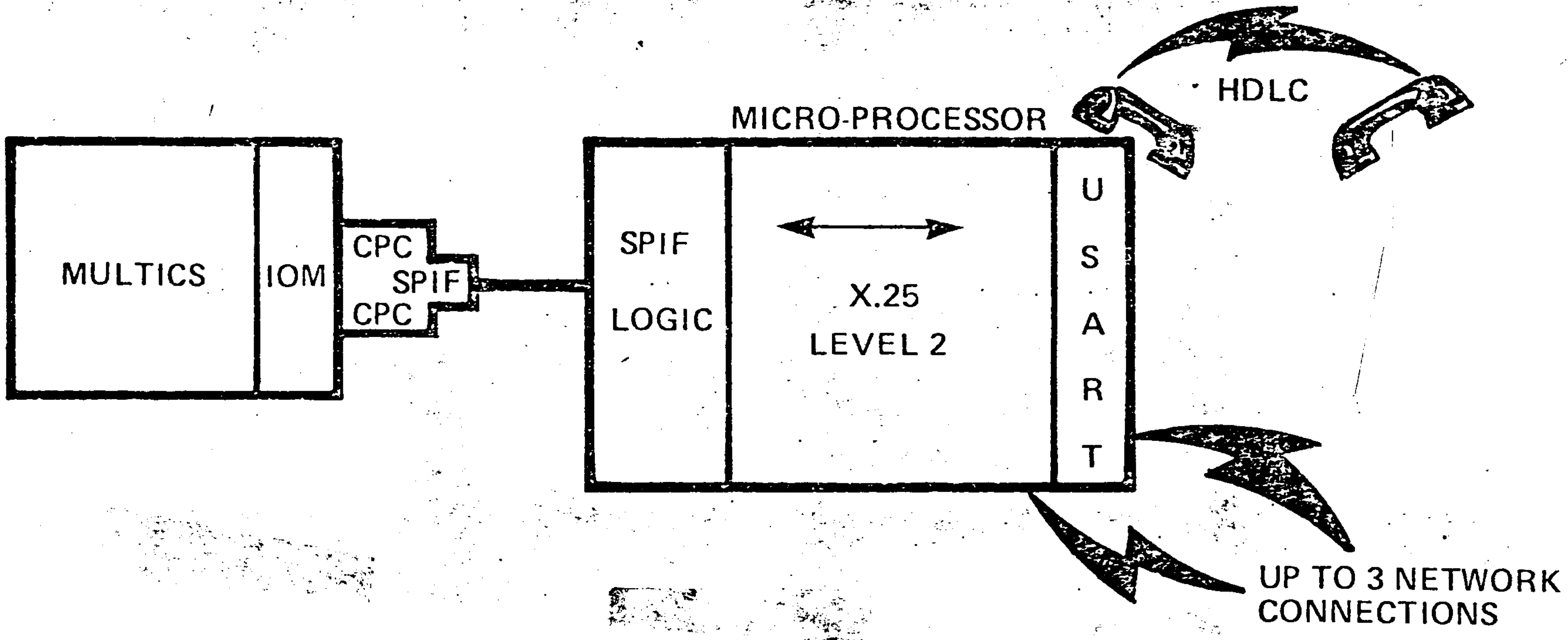


Figure 8