FAA TECHNICAL CENTER LETTER REPORT

THE EQUIPMENT MONITOR (EQM)

FOR THE REMOTE MAINTENANCE

MONITORING SYSTEM (RMMS)

by

John L. Wiley

September 1982

U. S. DEPARTMENT OF TRANSPORTATION
FEDERAL AVIATION ADMINISTRATION
TECHNICAL CENTER
Atlantic City Airport, N.J. 08405

INTRODUCTION

PURPOSE.

Task #9 of 9550-AAF-501-78-002 defines the subtask for the development of the Remote Maintenance Monitoring System (RMMS) Equipment Monitor (EQM). This report describes the EQM and the tests conducted to verify its performance.

BACKGROUND

The developmental version of the Equipment Monitor (EQM) has been completed at the FAA Technical Center. The EQM consists of two main parts. The first of these parts is the communication interface to the next higher level processor in the system (e.g., the Remote Monitoring Subsystem Processor (RMSP), or the Maintenance Processor Subsystem (MPS)). The second part is the interfacing of the Test Functional Modules (TFM) to the EQM. This interface not only collects data from the TFM but also allows the EQM to control the TFM's. At the present time the TFM's are in the early developmental stage and therefore this EQM to TFM interface can only be defined in bus hardware and basic software philosophy. As the TFM's are developed, the interfacing EQM software can also be developed. Figure 1 shows the block diagram of the EQM/TFM system configuration.

DISCUSSION

GENERAL.

This section has been divided into four main topics. They are:

1. EQM Bus Structure
2. Remote Monitoring Subsystem Processor Interface
3. Maintenance Processor Subsystem Interface
4. Test Functional Modules Interface

EQM BUS STRUCTURE. Under task #9, a goal is the standardization of the EQM and the TFM. This standardization along with the approach taken, to physically locate most of the TFM's in the EQM, led to the use of common data and control buses between the two devices.

There are several requirements that the interconnecting bus structure should meet. The first requirement is that the bus structure should be a standard throughout the electronic industry. Second, it should have off-the-shelf availability for some boards from various vendors, along with a second source. The last requirement is that a single board should be as small as possible. The reason for this is that at some remote sites there is limited space available for new equipment. Therefore, it is desirable that a single board is no larger than one square foot in size.

The field of bus structures was narrowed down to three, after examining over twenty different buses. The three were STD Bus, S-100 Bus, and Intel's Multibus (See Appendix A for a bus comparison and for data on each bus). The STD card

has the smallest physical dimensions of the three buses (approximately 4.5 inches by 6.5 inches). It meets all of the above bus requirements but has two drawbacks. One is that it has only eight data lines and sixteen address lines, and the other is that it does not allow multi-processing.

The second bus structure considered was the S-100 Bus. The physical dimensions of the card are larger than STD Bus (approximately 10 inches by 5.0 inches). This bus also meets all of the requirements and will handle both 8 and 16 bit microprocessors, along with multi-processing, but S-100 has some problems with standardization between manufacturers. The problem arises over the reserved pins on the backplane connector. Some manufacturers use these reserved pins for one particular signal, where another may use the same pin for a different signal.

The last bus structure to be considered in this report is Intel's Multibus. The Multibus card is the largest of the three, with dimensions of 12 inches by 6.75 inches. This bus not only meets all the requirements but also has multi-processing capabilities and will support a sixteen bit microprocessor. The Multibus has been proposed as an Institute of Electrical and Electronic Engineers (IEEE) standard (number 796). Multibus also allows for an optional battery back-up connector.

After looking at the above bus structures, along with many others, it was decided for reasons of standardization that only one bus would be used between the EQM and the TFM's. The bus structure decided on for the developmental EQM was the Multibus. Based on FAA Specification FAA-G-2100 (reference 1), the EQM will be operating in Environment II facilities (temperature range: -10° C to $+ 50^{\circ}$ C). The components in the EQM built at the Technical Center are commercial grade components (0° C to $+ 70^{\circ}$ C), and were used because of their availability and lower cost. To meet the -10° C to $+ 50^{\circ}$ C requirement two options are available. One is to use industrial grade or CMOS components (range: -40° C to $+ 85^{\circ}$ C). The other is to use commercial grade components with a small electrical heating element to keep to component temperature above 0° C. This latter technique was used for more than $1\frac{1}{2}$ years with microcomputer-based monitors installed at the ILS facilities at the FAA Technical Center (Reference 2).

REMOTE MONITORING SUBSYSTEM PROCESSOR INTERFACE. The EQM is required to send and receive serial data communications to/from the RMSP. At the time the EQM was being developed, this communications link was character-oriented asynchronous ASCII formatted. This format is described in the working paper "Interim Communications Protocol For the Remote Maintenance Monitoring System (RMMS) Program" (reference 3).

The EQM must perform the following tasks through the EQM to RMSP interface:

1. Respond to periodic polls and return one of the following:

    a. alarm data

    b. message data

    c. no alarm or message pending

2. Respond to selection polls and return one of the following:

   a. alarm pending

   b. message pending

   c. ready to receive message

3. Respond to scheduled polls and return one of the following:

   a. alarm data

   b. message data

   c. data requested

4. Perform error checking on all information received

5. Have the ability to change the baud rate of the EQM's end of the serial data channel.

The software to handle the RMSP to EQM interface will be explained under the heading of System Software.

MAINTENANCE PROCESSOR SUBSYSTEM INTERFACE. The EQM may be required to directly interface with the MPS through a bit-oriented serial data channel. The protocol used by this EQM channel is High Level Data Link Control (HDLC). At the time of the developmental work, the MPS (at the Technical Center) did not have a bit-oriented serial data channel. For this reason, the development of an MPS simulator with a HDLC channel was required. The simulator that was developed is described in the letter report "Use of the 8273 Communications Protocol Controller in the Remote Maintenance Monitoring System" (reference 4).

FAA Order 1830.2 (reference 5) requires the EQM to comply with the following:

1. At signalling rates below 1200 bits per second (bps) asynchronous transmission shall be used.

2. At signalling rates above 1200 bits per second, synchronous transmission shall be used.

3. At 1200 bits per second either mode may be implemented.

4. The electrical characteristics must be compatible with the Electronic Industries Association (EIA) Standard RS-449 (reference 6). The requirements for RS-422 (balanced) or RS-423 (unbalanced) is left to the applications requirements.

NAS-MD-790 (reference 7) requires the use of EIA Standard RS-422 (balanced). Because of these requirements, it was decided to incorporate the RS-422 serial data port into the EQM to MPS interface. The requirement for the EQM to use RS-449 exposed a potential problem when interfacing with the MPS. The MPS has been granted a waiver to use an EIA Standard RS-232-C serial data port instead of the RS-449/422. This meant that an adapter would have to be used to convert the RS-449/422 signals to RS-232-C signals and vice-versa. In order to test an adapter, the MPS simulator was designed with an RS-232-C serial data port also. The first tests were run using a commercially available

passive RS-449 to RS-232-C adapter. Letter report "Investigation of the Use of Passive Adapters for RS-449/RS-232-C Interfaces" (reference 8),concluded that there is a potential line length and noise problem with this adapter. Because of these potential problems, the RMMS group at the Technical Center designed and tested an active adapter for RS-449 to RS-232-C interfaces. The information on this adapter and the tests conducted are in the letter report "An Active RS-449/RS-232-C Adapter for RMMS" (reference 9). After writing both of the above reports, and having worked with both RS-449 and RS-232-C, it is this author's conclusion that serial data transmission between the EQM and any higher level system processor should be RS-232-C. Since the EQM will be operating at low data rates (1200 bps or below) and will only have short cables running between equipments (e.g. EQM to modem), RS-232-C can readily support this interface. The main reason for using RS-232-C is for cost savings. The RS-232-C is standard on most commercially available equipment that uses a serial data transmission mode, whereas RS-449 is usually an option at an added expense. Also, the FAA has existing equipment (e.g. terminals, printers, etc,) that are mostly RS-232-C, and to use this equipment with the RS-449 equipment would require that adapters be purchased.

TEST FUNCTIONAL MODULES INTERFACE. The interface between the EQM and most TFM's will be through the Multibus backplane (see section on bus structure). This defines the interface to be a parallel digital interchange. The EQM will send both data and control information to the TFM and will receive data from the TFM over the backplane.

At the time of this report, the TFM's are still in the parameter study stage and have not been fully designed. For this reason the actual software for the EQM to control the TFM's has not been written. The only work that could be done on this interface was to work on the overall philosophy of the interface. The interface software philosophy will be discussed under the heading of EQM System Software.

EQM SYSTEM HARDWARE.

The EQM built at the Technical Center is a Z-80 microprocessor based system. The EQM consists of a Monolithic Systems MSC-8007 board, MPS Communication Interface board, and a Central Data Corporation chassis with power supply. Both boards use the Intel Multibus bus structure. The system diagram is shown in figure 2.

The MSC-8007 board contains a Zilog Z80A microprocessor, 32K bytes of random access memory (RAM), and the capability to accommodate up to 16K bytes of programmable read only memory (PROM). In addition, the board contains three serial data channels, a parallel interface, a priority interrupt controller, and two programmable interval timers. The three serial data channels and the one parallel interface are all independent of the Multibus. The first of the serial data channels is connected to the RMSP (port 3), the second is connected to the local terminal (port 1), and the third is unused at the present time (port 2). The parallel interface and/or the unused serial channel could be used to control an auto-answer, auto-dial modem in future work. All of the serial data channels on the MSC-8007 board are RS-232-C compatible.

The MPS communication interface (MPSCI) board was designed and built at the Technical Center. This board contains an Intel 8273 Programmable Protocol Controller, an Intel 8253 Programmable Interval Timer, an Intel Multibus Interface, and one serial data channel. The serial data channel is RS-449/422 compatible. See figure 3 for a block diagram of the MPSCI.

The local terminal used for the testing of the EQM was a Computer Devices Inc. (CDI) terminal with an RS-232-C port. It was not critical for the terminal to be a CDI, only that it be RS-232-C compatible. The terminal is not required for the EQM to operate, but when used, it provides additional capabilities to the system (e.g. input messages, change baud rates, etc.).

The chassis used is a B1013 "15-slot mother board" and a C1000 cabinet with power supply. This combination made up a complete 15-slot Multibus chassis that contains three power supplies, a cooling fan, a front control panel and a rear panel. The rear panel has pre-punched holes for connectors to be installed. This chassis was designed for stand alone applications and does not require rack mounting. At the present time three slots of the chassis are required for the EQM, leaving the other slots for TFM's. The MPSCI board required two slots because it is built on a wire-wrap board.

EQM SYSTEM SOFTWARE.

The EQM software consists of an initialization program, a main program, and two receiver interrupt programs. The local terminal program is included in the main program. All the software was written in Z-80 microprocessor assembly code and will be run on the MSC-8007 CPU board. The software consists of about 6K bytes of executable code and about 1K bytes of data storage. The executable code is stored in EPROM, and the data is stored in RAM located on the MSC-8007 board.

System operation begins by depressing the reset button on the C1000 chassis. The Z-80A immediately begins executing the initialization program. This program first sets up the interrupt vector addresses and then clears all of the status registers. The next routine is the Main routine. At the present time this program is called "Monit", and was used in the development of the two receive interrupt programs. As mentioned earlier in this report, the actual software to control and collect data from the TFM's has not been written because the TFM development is still in the early stages.

MONIT SOFTWARE. This report will first explain the program "Monit" and then the guide lines that should be followed when writing the Main program for the control and data collection from the TFM's. A simplified flowchart of "Monit" is shown in figure 4. Program operation is as follows:

1. Serial Input and Output (I/O) Ports setup is entered. This routine sets up the serial I/O ports of the MSC-8007 board. The routine sets the baud rate for the local terminal and the RMSP data link. After the setup has been completed the program control is passed to the Command Input Loop.

2. The Command Input Loop is where "Monit" spends most of the execution time waiting for an input from the local terminal. The only time it is not in this loop is when a command is being processed or when an interrupt is being handled. The Interrupt Handling Routine is a separate program and will be explained later in this section of the report The Command Input Loop is divided into two sections, the first is the input from the local terminal, and the second is the checking of that input. When an input is received from the local terminal, the Command Input Loop checks to see if the received data is a valid input. If it is valid, the program transfers control to the requested routine. If it is not a valid

input, an error message is sent to the local terminal and program execution stays in the Command Input Loop.

The following routines are called by the Command Input Loop and all but two return program execution to the Command Input Loop when finished. The two that do not return to the Command Input Loop are the "GO" routine and the "JUMP" routine. These two routines execute a program that has been previously stored in RAM and therefore go to where the stored program requires.

1. Request Data Routine is called when a "Q" is received from the local terminal. This routine allows the person at the terminal to request data, through the RMSP, from any other EQM site. First the routine asks the address of the requesting site and then the address of the site requested to be entered on the local terminal. The program execution is then transferred to the formatting subroutine. This subroutine formats the data for the RMSP and then returns program execution to the Request Data Routine. The Request Data Routine then sets the output flag, so the data will be sent on the next poll from the RMSP.

2. Test Alarm Routine is entered when an "A" is received from the local terminal. This routine sets up a "canned" alarm message that is to be sent to the RMSP. This canned alarm message is stored in memory and is only used for test purposes to simulate an alarm from an EQM. The routine also sets the output flag so the alarm data will be sent on the next poll from the RMSP.

3. Message Routine is called when an "M" is received from the local terminal. The Message Routine is used to send messages from the local terminal to other devices in the system (Note: All EQM to EQM messages are routed through the associated RMSP.) The routine first inputs the message from the local terminal and the calls the formatting subroutine to format the message for the RMSP. Once the message has been formatted, the message routine sets the output flag so the message will be sent on the next poll from the RMSP.

4. Baud Rate Routine is requested by the local terminal (when a "B" is entered) to change the baud rate of one of the three serial ports on the MSC-8007 board. The routine asks which port is to be changed and what the new baud rate will be. The change is then made to the specified port. This change is only made to the port on the EQM and not on both ends of the line, therefore the port on the other piece of equipment will also have to be changed to match the baud rate on the EQM port.

The following routines can also be called by the Command Input Loop routine. These routines were used in the development of the EQM software and can be used to debug the EQM to TFM interface software.

1. Go Routine is called when a "G" is received from the local terminal. This routine will run the stored program that starts at the current address. Remember that this routine does not return to the Command Input Loop but does what the stored program requires. In order to get back to "Monit" a hardware reset is required.

2. Set Memory Routine will set a single memory location with the data that is received from the local terminal. The letter "S" is used to invoke this routine. After the new data has been received, it is stored in the RAM memory.

3. Fill Memory Routine will fill a block of memory with a value received from the local terminal. This routine is invoked by entering the letter "F" on the local terminal. The routine asks what block of memory is to be filled and what value is to be inserted. This routine can be used to clear a block of memory or to test a block of memory in the EQM.

4. Jump Routine is called when a "J" is entered on the local terminal. The routine first asks for the address of the program to be run and then jumps to that address. Once it has reached this address, it transfers the program execution to the Go Routine. Remember that the Jump Routine and the Go Routine do not return program execution to the Command Input Loop routine.

5. Display Register Routine is called when a "R" is received from the terminal. This routine displays the contents of the microprocessor's registers that were stored in the RAM memory by "Monit". This routine allows the contents of any register or pair of registers to be displayed. The register values stored in RAM can be changed by the operator at the local terminal but not the actual microprocessor's registers.

6. Display Memory Routine is used to display a block of memory on the local terminal. The routine is called when a "D" is received from the terminal. This routine is used to examine raw data stored in memory or to examine a status register that is stored in memory.

EQM/TFM SOFTWARE PHILOSOPHY. The EQM's Main program, that will handle the TFM's, will have many of the routines that are now found in "Monit". The basic philosophy of the Main Program is to use subroutines to handle the TFM's. This means that for a particular type of TFM there would be a subroutine to handle the control and data collecting in the EQM, with the subroutine being called by the Main program, when the EQM wants to control or get data from that particular TFM. Thus, the EQM could handle a number of TFM's by having the associated TFM subroutines stored in system memory, and the addresses of these subroutines known by the Main program. This will require the main program for each EQM site to have adaptation data for the site. The main program would have to know the types of TFM's in a particular EQM/TFM card cage and the facility configuration. There are several options available to support these requirements. This report will discuss four ways to implement these requirements but it should be noted that there are other ways to perform this function.

The first option is to have a set of memory addresses assigned to the EQM that is equal to the number of TFM card slots in the EQM/TFM card cage. When a TFM is installed in the card cage, one of these memory addresses must be set on the TFM. This address could be set by the use of jumper wires or dip switches. This address would point to a status register on the TFM that contained the type of TFM that had been entered. Additional status registers on the TFM, could give information on unused ports and parts of the TFM, etc. During start-up, if the EQM does not get a response from a particular assigned address, (i.e; card slot), it assumes the address is unused and goes in to the next address. The EQM would only read these addresses during start-up, and then would store the status and type of TFM in a memory look-up table. The EQM's Main Program would use this table to get the types of TFM's in the system and their addresses. There are a few drawbacks to this option, the first being that, if a TFM has a wrong

address strapped on the board, the EQM would not know that this TFM is in the card cage. The second problem with this option is that the EQM may address an unused address and read in noise from the bus and interpret the noise as a TFM location. The last drawback is the possibility that two TFM's could, through error, have the same address. This would make the EQM try to access both TFM's at the same time with resulting system problems. If any error occurred during the startup of the EQM, the error would be stored in the look up table and this bad data would be used until another restart occurred.

The second option is to assign memory addresses to each type of TFM and have all these addresses stored in the EQM. The EQM would then go through each assigned address to see if there was a TFM of that type in the system. In order to cut down the time of the search, the EQM would only check for multiples of a particular TFM when that particular TFM had already been found in the system. After all the assigned addresses had been checked, the EQM would again make a lookup table of the addresses of the TFM's used in that EQM/TFM card cage configuration. When the TFM was addressed it would give the status of the TFM and could use extra status registers to provide more information to the EQM. Again, there are drawbacks to this option. The first is that there is still the need for jumper wires or dip switches on the TFM's to store the addresses of the TFM. The reason these are still needed is to allow for more than one particular type of TFM to be used in the system. This is still subject to the error of entering the wrong address. There is still the possibility of reading in noise from the bus when the EQM outputs the address of a TFM that is not in the system. The last problem with this option is the time and memory required to search and store an address table of all possible TFM's.

The third option to be discussed, is one that uses status ports on the EQM board. These status ports would be jumper wires or dip switches that would tell the EQM what TFM's are in the system and their addresses. The EQM would first read these status ports and then create a lookup table of the TFM types and address. The EQM would then output the address of each TFM and then read in the status and offset values of each TFM. The values received from the TFM status ports would only be read during startup and then stored in memory for future use. This system still has the problem of a wrong address being set on the jumpers on the EQM. The use of these status ports will eliminate the potential of false readings when there is no TFM in a particular slot, because the EQM will know exactly what TFM's are in the system and their addresses. This option also does not require the memory or time to check through a large table of all the possible TFM's that could be in the system.

The last option for addressing the TFM's is functionally the same as the third option but instead of the status being entered on jumper wires or dip switches, it would be stored in memory. This memory would be a custom package made up for each EQM system. This would allow the EQM to read the addresses and types of TFM's directly from the memory and would eliminate the potential of someone accidently changing the address of a TFM. If a single PROM was used to store the status information, there would still be enough extra memory in this PROM to store other information about the particular EQM. This PROM could be programed at the same time the main program and subroutines were being burned into PROM. After the system is running, if a new TFM was to be added or an old TFM removed, only a new status PROM would have to be made. (Assuming the subroutine to handle the new TFM is already resident in the EQM. If not, then the PROM with the main

program and the subroutines would also have to be changed.) Of the four options considered, this one is recommended for EQM/TFM use.

In all of the options mentioned above, the main program and subroutines could have some extra routines that would not be used by one EQM but may be used in another. This would allow the same main program and subroutines to be used in more than one EQM, therefore, allowing for some standardization between systems software.

INTERRUPT ROUTINES. The main routine was laid out to control only the EQM to TFM interface. The EQM to RMSP (or MPS) interfaces will be interrupt driven. The need to use interrupts for these interfaces is to allow the communication, between the EQM and RMSP or MPS, to have priority over other routines. The interrupt software that was written at the Technical Center, for the RMSP and MPS, was written in two parts. One of these parts will support the communication between the EQM and the RMSP, the other part will handle the interface between the EQM and the MPS. It was assumed, early in the development of this software, that the EQM would interface with one of the two devices and not both devices at the same time. Therefore, the two interrupt routines are not stored in the same memory and were not written to run at the same time on one EQM. At a later date, if the need does arise for the two interrupt routines to run together, this could be done with little modification.

EQM/RMSP INTERFACE. The interrupt software to interface the EQM to the RMSP has been named "Recint". A simplified flowchart of "Recint" is shown in figure 5. "Recint" operates as follows:

1. FLGCHK is called. This routine determines if a complete valid poll has been received. If it is not a complete poll, it clears all the flags that were set and returns from interrupt. If it is a valid poll, FLGCHK determines the type of the poll and then transfers the receive poll from the receive buffer to one of the general buffers. The general buffer is marked as to type of poll that is stored in it, e.g. continuous poll, selection poll, or scheduled poll. After the transfer of the poll has taken place program control moves on to handle the stored poll.

2. GENPOL is called to respond to a continuous poll. This routine first checks to see if there is an alarm pending. If there is, it then transmits the alarm buffer to the RMSP. The alarm buffer was formatted by "Monit" (see earlier decussion of "Monit" software). If there is no alarm pending, GENPOL checks to see if there is a message pending. If so, it transmits the message to the RMSP. Again the message was formatted by the "Monit" software. The last check that GENPOL does is to see if a data request from the RMSP is waiting to be transmitted. If there is, the data is then sent. After all of these checks have been made and there is no information waiting, the routine sends an "End of Transmission" (EOT) and clears the general buffer and returns from interrupt. The EOT is used to tell the RMSP that the communication link is working when there is no other information to be sent. The protocol used in this interface is explained in the working paper "Interim Communications Protocol for the Remote Maintenance Monitoring System (RMMS) Program" (reference 3).

3. SPOL is called to respond to a selection poll. This poll handles messages from the RMSP to the EQM. The routine first checks to see if an alarm or an outgoing message is pending, if so, the RMSP is told not to transmit the message. If nothing is pending, SPOL tells the RMSP to send the message. The incoming message can be up to four data blocks long and is stored in the general buffers. After the entire message has been received, the general buffers are changed to terminal buffers. The "PRINT" routine is then used to output the terminal buffers to the local terminal. There are two important things that the "PRINT" routine does while sending the message to the terminal. First it allows interrupts to be serviced. This means the EQM can still receive polls from the RMSP. Secondly, once a terminal buffer is dumped, it is cleared and returned to a general buffer.

4. RPOL is used when a scheduled poll is received. This routine handles a request for data from the RMSP. The routine first checks to see if there is an alarm or message pending and if so, it jumps to GENPOL. If neither are pending, RPOL then sends the EQM's address and the requested data. The actual transmitting of the data is done through a part of the GENPOL routine. The data that was requested is formatted by the "Monit" routine. After the data has been transmitted the routine clears all the buffers and flags used by RPOL. The routine then returns from interrupt.

The return from interrupt will normally return program execution to the "Monit" routine. There is one case where the program execution may be returned to another interrupt routine. This can occur when a long message has been received from the RMSP and is still being printed on the terminal when another poll occurs. In this special case, printing of the message to the terminal is stopped and the poll is handled. Once the poll response is completed, the program execution returns to the PRINT routine and the message is once again sent to the terminal.

EQM/MPS INTERFACE. The last interrupt software to be discussed is the interface software to handle to EQM to MPS interface. The data protocol for this interface had not been written for the MPS at the time of the development of the EQM. Therefore there was need for a MPS simulator to be developed and built at the Technical Center. This simulator was built using the Intel 8273 Programmable HDLC/SDLC Protocol Controller. The simulator is described in the letter report "Use of the 8273 Communications Protocol Controller in the Remote Maintenance Monitoring System (RMMS)". The EQM interface to the MPS also uses the Intel 8273. The EQM software (EQM73) not only performs the control and initialization of the 8273, but also contains a shortened version of "Monit". This routine was designed to run without any support software. A simplifed flowchart is shown in figure 6. The operation is as follows:

1. START routine initializes the MSC-8007 board and the 8273 prototype board. This routine initializes the following on the MSC-8007 board:

   a. Sets the terminal baud rate to 300 baud

   b. Sets the interrupt addresses

   c. Clears the receiver memory

EQM73 also initializes the following on the 8273 prototype board:

   a. Sets synchronous baud rate to 1200 baud

   b. Sets asynchronous baud rate to 300 baud

c. Sets clock type to asynchronous

d. Sets mode for HDLC and continuous flags

After the initialization has been completed, the program execution goes to the LOOP routine.

2. LOOP is a small version of the Monit routine. Most of the program execution time of EQM73 is spent in LOOP, checking for a terminal entry and waiting for interrupts. When an input is received from the terminal, the routine checks to see if it is a valid input. If it is, the routine then jumps to the routine to handle the input. If it is not a valid input, an error message is sent to the terminal. In either case when the input has been serviced, the program execution returns to the LOOP routine.

3. HELP routine is called when an "H" is entered on the terminal. This routine prints out the list of commands that can be used by the terminal to control the EQM to MPS interface. Figure 7 shows the output of the HELP command as seen on the terminal. Program execution returns to the LOOP routine after the command list has been sent to the terminal.

4. BAUD routine is used is change the baud rate of the EQM to the terminal or the MPS. There are two baud rates associated with the EQM to MPS interface. One is for the synchronous transmission and the other is for the asynchronous transmission. Figure 8 shows the BAUD command format as displayed on the terminal. After the baud rate of a port has been changed, the program execution returns to the LOOP routine.

5. CLKTYP is called to change the EQM to MPS interface communications from asynchronous to synchronous or vice-versa. The format of the terminal input is shown in figure 9. Program execution returns to the LOOP routine.

6. ENTER routine allows a message to be entered on the terminal that is to be sent to the MPS. The message is entered on the terminal in the format shown in figure 10. Any time this routine is used, it erases the old message stored in memory and replaces it with the new message. Program execution returns to the LOOP routine after the new message has been stored in memory.

7. LIST routine outputs the most recent received message or transmitted message that is stored in the RAM memory to the terminal. The format of the command is shown in figure 11. After the message has been sent to the terminal, the program execution returns to the LOOP routine.

8. TXCMD routine transmits the message stored in the transmit message buffer to the MPS. This message was stored in memory by the ENTER routine. After the message has been transmitted to the MPS, the 8273 returns a status report to the terminal. The three status messages are: Clear to Send error, Data Underrun error, and Transmit Frame Complete. The Clear to Send error occurs when the Clear to Send signal is lost from the MPS. The Data Underrun error means that the data was either stopped in the middle of a transmission by the MPS or lost due to a data link problem. TXCMD is invoked by typing the letter "T" on the terminal. The routine returns to the LOOP routine after the message

has been set up and transmission has begun. The transmission of the message is completed by the interrupt routine TXDATA. This routine hands the next part of the message to the 8273, then the 8273 asks for more data. When the 8273 has completed the entire message, it then interrupts to allow the status to be read. TXINT is the interrupt routine that reads in the status and sends it to the terminal.

There are two more interrupt routines that are in EQM73. The first is the RXDATA routine, which inputs the data to memory that is received by the 8273 from the MPS. The second routine, RXINT, is called after the total message has been received and stored in memory. This routine reads the status port of the 8273 to see if the message that was received has any errors. This receive status is then sent to the terminal. A list of the possible status messages are shown and explained in table 1. After either interrupt has been handled, the program execution returns to the LOOP routine.

EQM PROTOTYPE TESTING.

To verify that the I/O side of the EQM operates as designed, the following tests were conducted. The first set of tests were run on the EQM to a bit-oriented MPS simulator interface. The test configuration is shown in figure 12. The following tests were performed on this interface:

1. The EQM and the simulator were run continuously for over 24 hours.

2. Messages were generated and sent by both EQM and the MPS simulator.

3. Messages were received and displayed by both the EQM and the MPS simulator.

4. Communication line problems were simulated in the link between the EQM and the simulator.

5. Equipment problems were simulated at both the EQM and the simulator.

In all tests, the EQM to MSP simulator interface performed as expected.

The second set of tests were run on the interface between the EQM and the RMSP. The test configuration is shown in figure 13. The following tests were performed:

1. The EQM and the RMSP were run continuously for over 24 hours.

2. Messages were sent from the EQM's terminal to the terminals of remote sites through the RMSP.

3. Messages were received from remote site terminals by the EQM after the message had passed through the RMSP.

4. Simulated alarms were sent to the RMSP by the EQM (see note 1).

5. Data requests were sent by the EQM that requested data from the remote site simulators.

6. Data requests were received and answered by the EQM.

7. Communication line problems were simulated in the link between the EQM and the RMSP.

NOTE 1. The EQM was made to resemble a VOR site by preloading the alarm and data buffers with canned VOR data. This was done for test purposes only and this data will normally come from the TFM interface.

In all tests, the EQM performed as expected.

FUTURE MODIFICATIONS.

The designs discussed in this report meet the basic requirements of the RMMS Equipment Monitor. However, the TFM interface is discussed on the basis of the planned approach and has not been tested. There are several modifications that should be added to the EQM in the future. Some of these may require that more tests be run on the EQM. The following list of modifications should be added to the EQM:

1. Change of the protocol to be ICD-1 compatable for asynchronous low speed (1200 baud or below) communication for both the interface to the RMSP and the MPS. It should be noted that at the time the EQM was being developed, the ICD-1 protocol was not resident in either the RMSP or the MPS.

2. Add the required software to the EQM to have a Real-Time Clock in the EQM. At the present time, the real-time clock in the system is only located in the RMSP. This new software could be added to "Monit" or written as part of the Main program. It should be added when the TFM interface software is added to the EQM.

3. Add an auto-dial and auto-answer capability to the EQM. This would make it possible for the EQM to call or receive calls over the public telephone network from the RMSP or the MPS. This link could be used as the main communication link or as a backup link or both. In order to incorporate this option in the EQM, both software and modems would be needed.

4. Develop the EQM software to support the TFM interfaces, as required. The development of the TFM's must be completed before this software can be written.

REFERENCES

1. Electronic Equipment, General Requirements. FAA Specification FAA-G-2100/1b. July 10, 1970.

2. Evaluation of a Remote Tone Signaling Control/Monitor System As Lightning/ Transient Protection for Solid State Instrument Landing Systems. FAA-RD-78-149. James R. Branstetter, January 1979.

3. Interim Communications Protocol for the Remote Maintenance Monitoring System (RMMS) Program. David Wainland, April 1981.

4. Use of the 8273 Communications Protocol Controller in the Remote Maintenance Monitoring System. CT-81-100-15LR. John Wiley, Adam Magoss, September 1981.

5. Policy for use of Telecommunications Data Transfer Standards. FAA Order 1830.2. February 17, 1978

6. General Purpose 27-Position and 9-Position Interface for Data Terminal Equipment and Data Circuit-Terminating Equipment Employing Serial Binary Data Interchange. Electronic Industries Association Standard RS-449 November 1977.

7. Interface Control Document for the Remote Maintenance Monitoring System (RMMS), ICD-1. National Airspace System Configuration Management Document. NAS-MD-790. April 17, 1981.

8. Investigation of the use of Passive Adapters for RS-449/RD-232-C Interfaces. CT-82-100-13LR. John Wiley, December 1981.

9. An Active RS-449/RS-232-C Adapter for RMMS. CT-82-100-33LR. John L. Wiley, Harry L. Brown, April 1982.

TABLE 1. THE RECEIVER ERROR MESSAGES GENERATED BY THE "EQM73" PROGRAM

"EQM73" RECEIVER ERROR MESSAGE

TERMINAL MESSAGES	EXPLANATION OF MESSAGE
CRC Error	The received frame was in the correct format, however, the received Cyclic Redundancy Checkword (CRC) did not match the internally generated CRC.
Abort Detected	An abort has been received from the sending device in the middle of the received message.
Idle Detected	An idle is detected whenever 15 consecutive 1s are received.
EOP Detected	If the End-Of-Poll (EOP) bit has been set in the Operating Mode register, the EOP message will be sent whenever an EOP is received.
Frame < 32 Bits	The number of bits in the received frame, between the flags, is fewer than 32 bits. The number of bits in a valid frame must be 32 bits or greater.
Data Overrun	The data is not all extracted from the 8273 before the next received byte is ready for transfer.
Memory Buffer Overflow	The number of received bits is greater than the number stored in the receiver buffer length counter.
Carrier Failure	The Carrier Detect (CD) pin has gone inactive during reception of the frame.
Interrupt Overrun	The system's interrupt response and service time is not sufficient for the data rates being attempted.

Note: If any of these messages are generated by the 8273, then the received message is discarded.

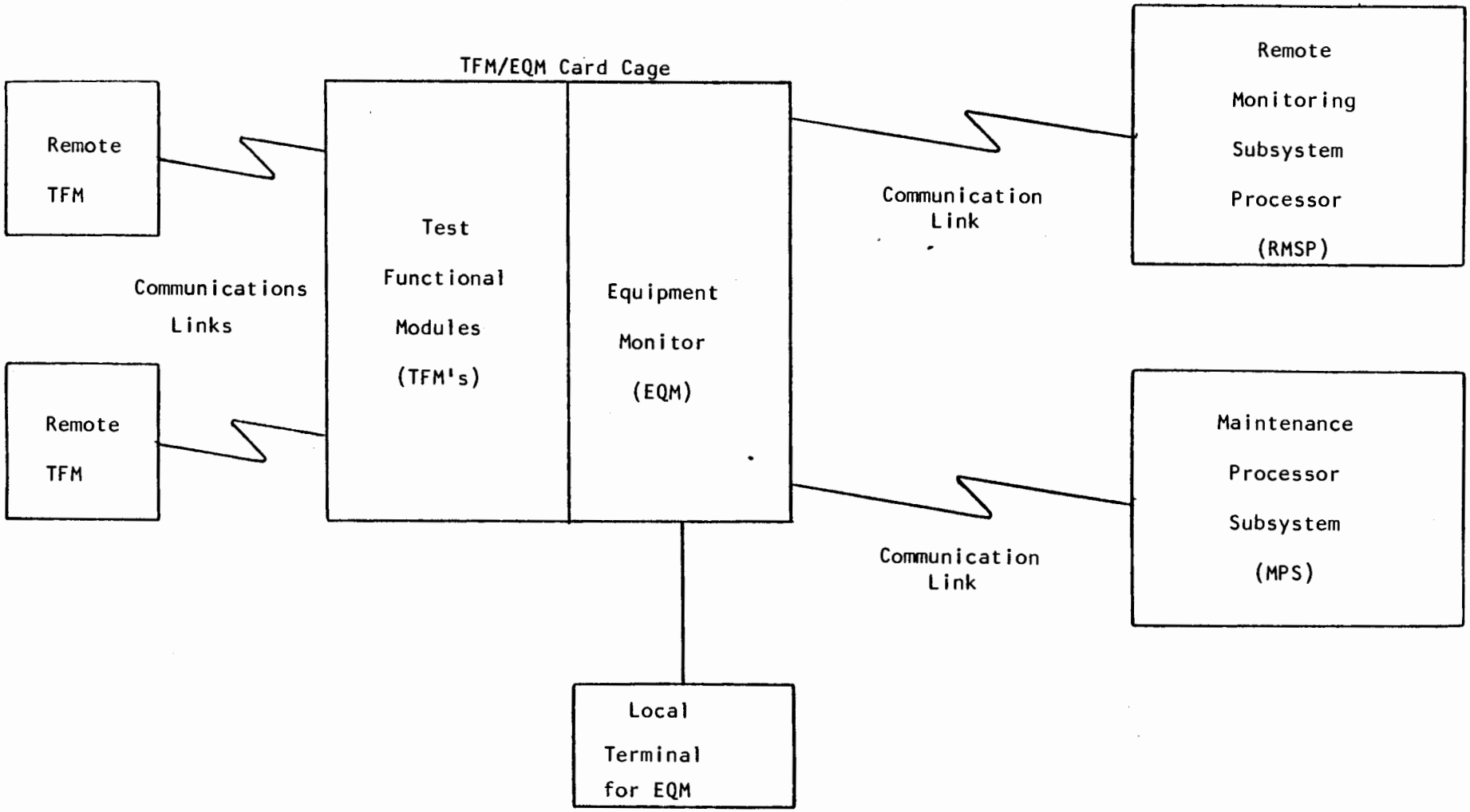
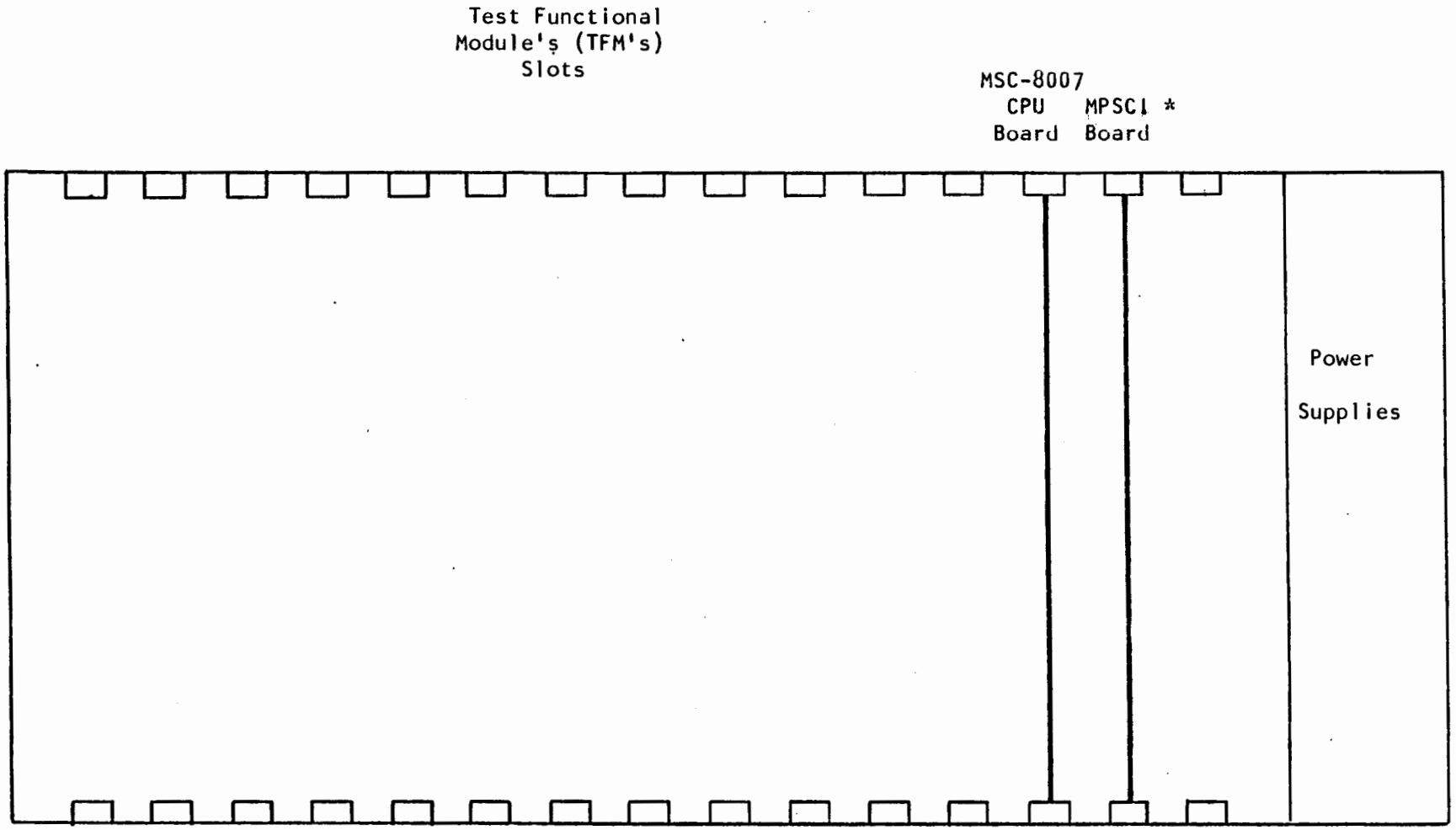


FIGURE 1. BLOCK DIAGRAM OF THE EQM/TFM SYSTEM CONFIGURATION



*NOTE: The Maintenance Processor Subsystem Communication Interface (MPSCI) is built on a wire-wrap card and requires two slots.

FIGURE 2. EQM BOARD LAYOUT IN THE EQM/TFM CARD CAGE

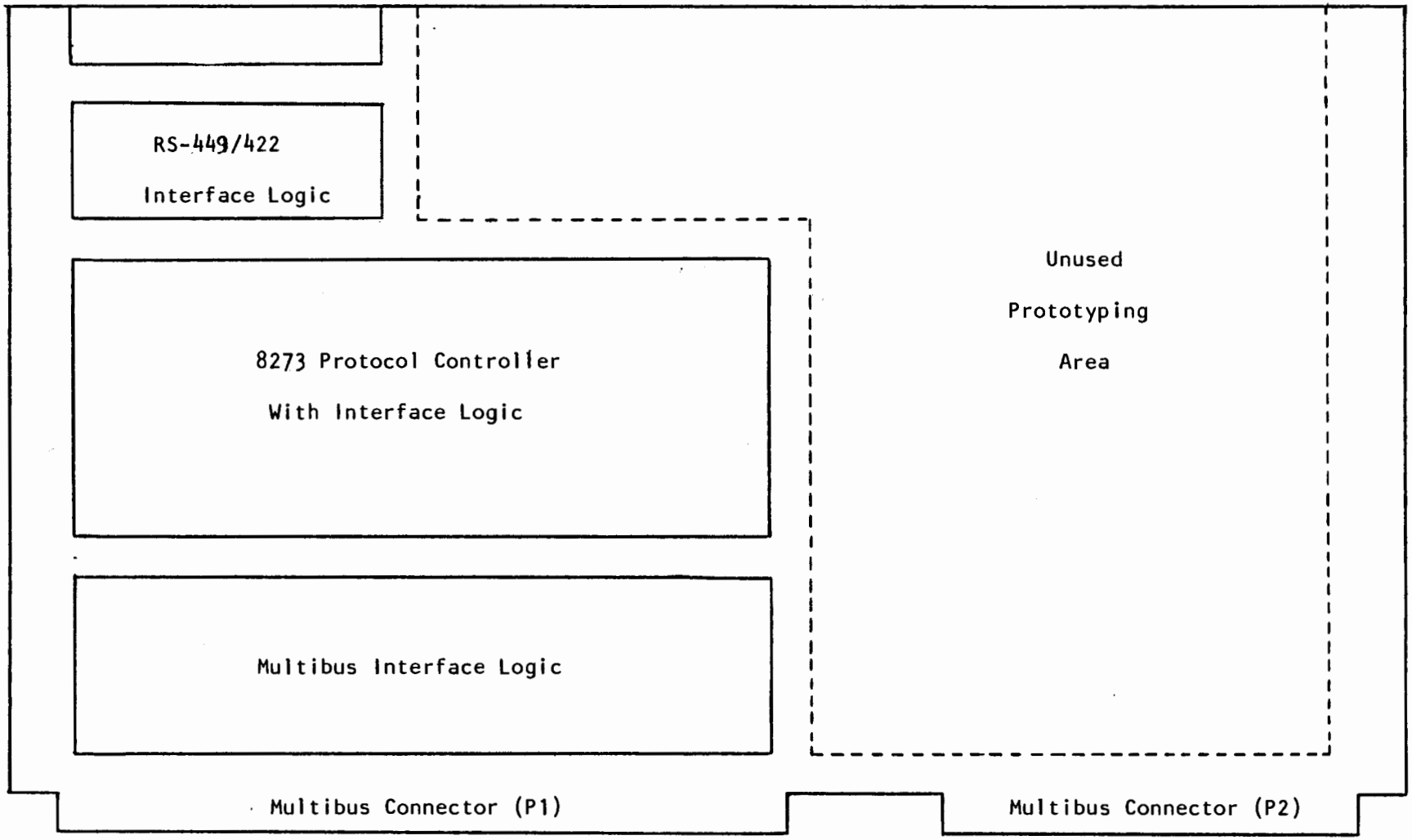


FIGURE 3. BLOCK DIAGRAM OF MAINTENANCE PROCESSOR SUBSYSTEM COMMUNICATION INTERFACE (MPSCI)

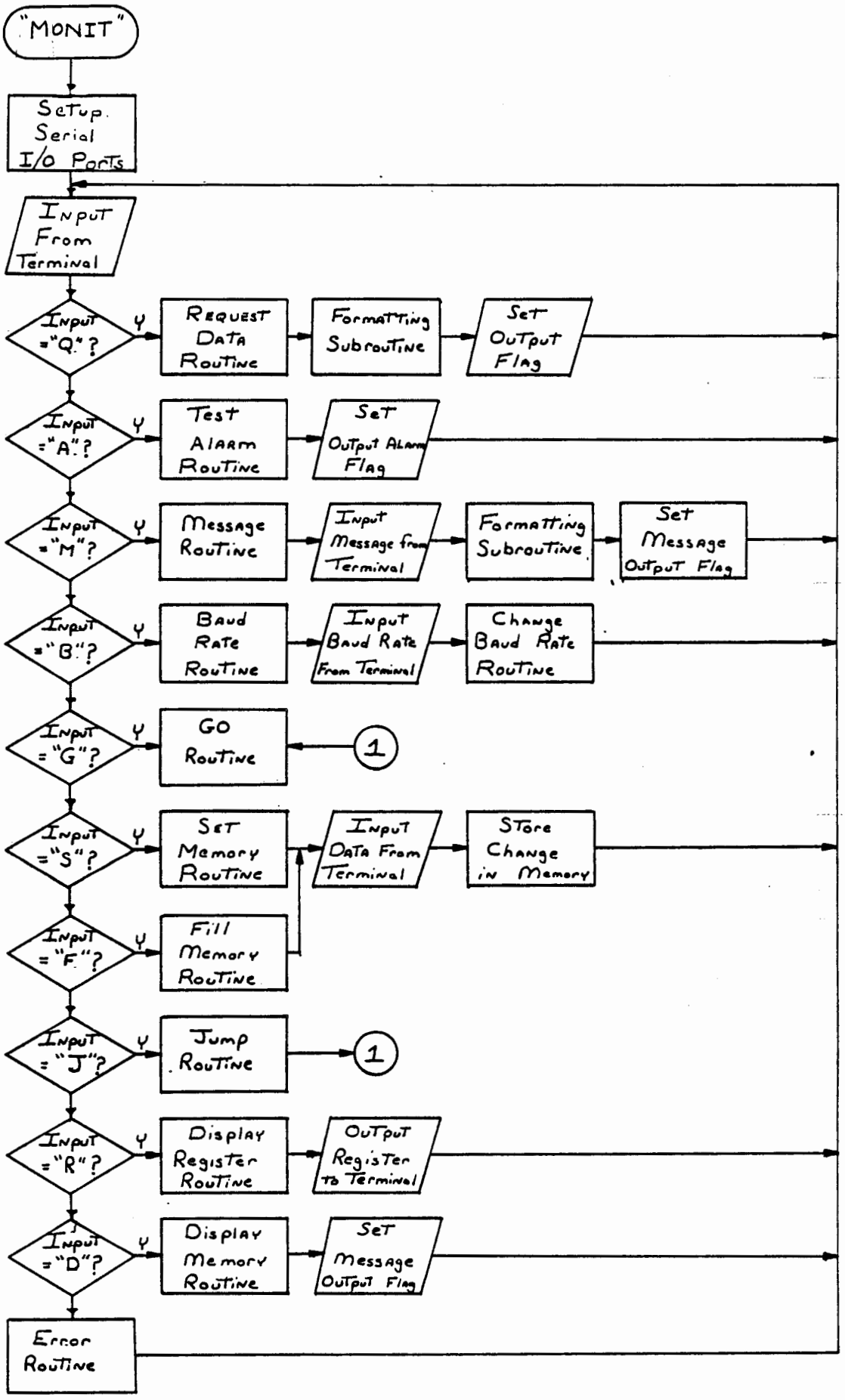


FIGURE 4. SIMPLIFIED FLOWCHART OF "MONIT" PROGRAM

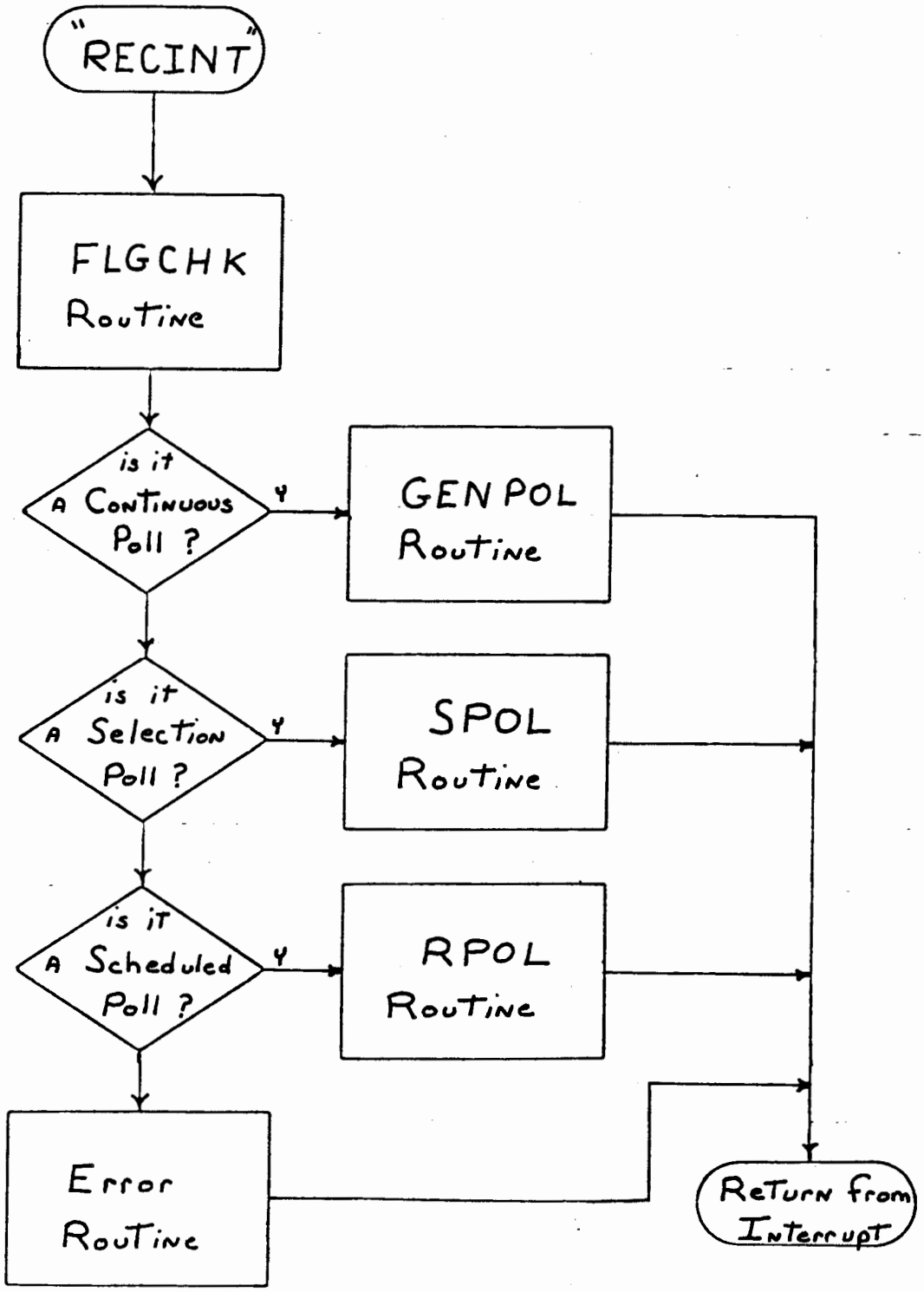


FIGURE 5. SIMPLIFIED FLOWCHART OF "RECINT" INTERRUPT PROGRAM

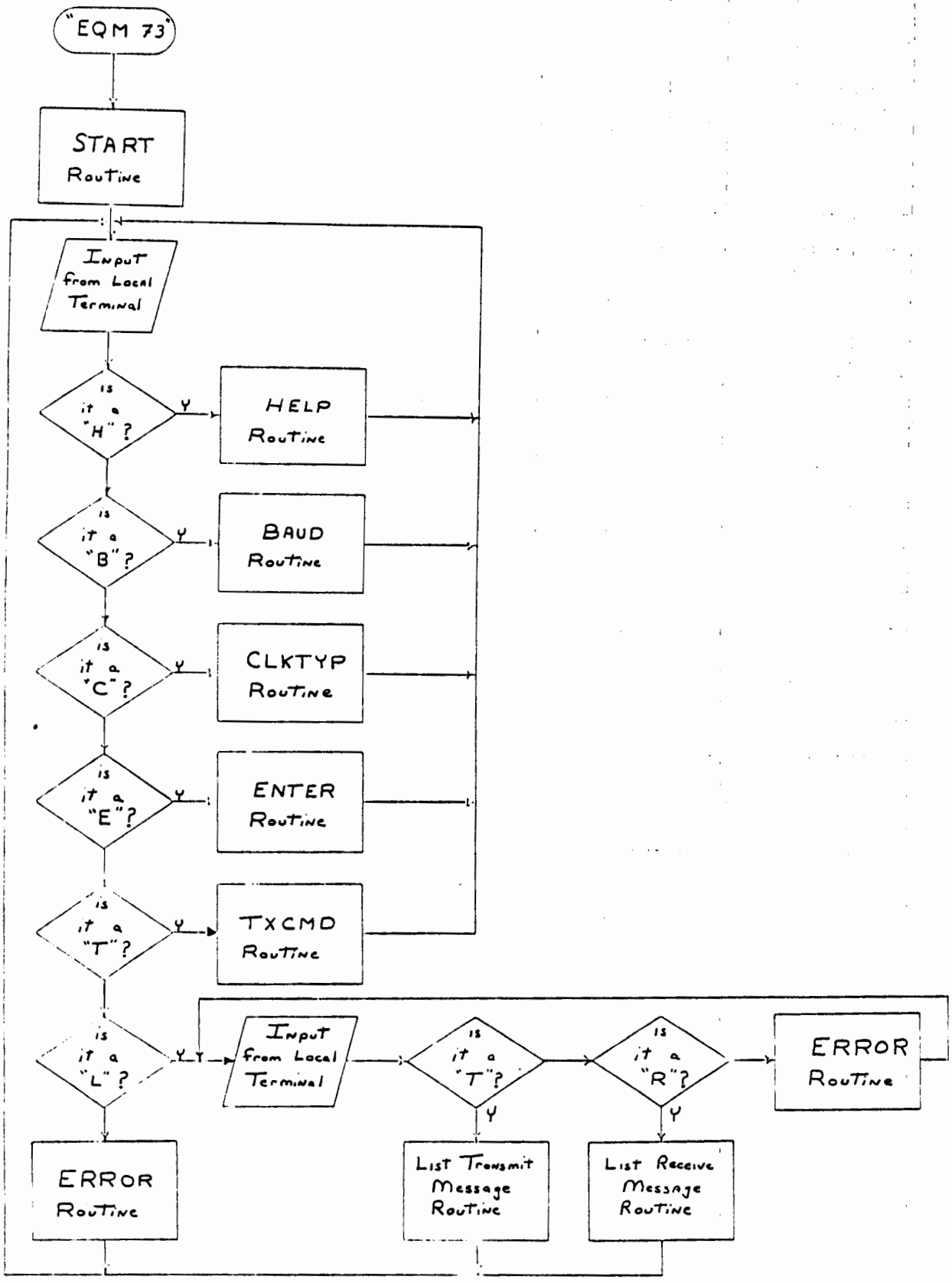


FIGURE 6. SIMPLIFIED FLOWCHART OF "EQM73" INTERRUPT PROGRAM

```
                    8273 COMMUNICATIONS CONTROLLER
                    ******************************

: "H"

   HELP COMMANDS ARE:

                    'B' BAUD RATE
                    'C' CLOCK TYPE
                    'E' ENTER MESSAGE
                    'L' LIST MESSAGE
                    'T' TRANSMIT MESSAGE
```

NOTE The letters that are enclosed between the quote marks are the letters to be entered on the EQM's local terminal.

FIGURE 7. THE "HELP" COMMAND FOR THE EQM WHEN INTERFACED TO THE MPS SIMULATOR

```
                    8273 COMMUNICATIONS CONTROLLER
                    ******************************

- "B"
  BAUD RATE: "H"
  HELP FOR BAUD RATE COMMANDS

                'T' TERMINAL BAUD RATE
                'A' ASYNCHRONOUS BAUD RATE
                'S' SYNCHRONOUS BAUD RATE


  BAUD RATE: "T"

  TERMINAL BAUD RATE NUMBER IS:
  (75 BAUD= #0, 110 BAUD= #1, 150 BAUD= #2, 300 BAUD= #3, 1200 BAUD=
   #5, 2400 BAUD= #6)


  BAUD RATE: "A"

  ASYNCHRONOUS BAUD RATE NUMBER IS:
  (75 BAUD= #0, 110 BAUD= #1, 150 BAUD= #2, 300 BAUD= #3, 600 BAUD=
   #4, 1200 BAUD= #5)


  BAUD RATE: "S"

  SYNCHRONOUS BAUD RATE NUMBER IS:
  (300 BAUD= #0, 1200 BAUD= #1, 1800 BAUD= #2, 2400 BAUD= #3, 3600 BAUD=
   #4, 4800 BAUD= #5, 9600 BAUD= #6, 19200 BAUD= #7)


  (See note on Figure 7.)
```

FIGURE 8. THE "BAUD" COMMAND FOR THE EQM WHEN INTERFACED TO THE MPS SIMULATOR

```
                8273 COMMUNICATIONS CONTROLLER
                ******************************

= "C"

    CLOCK TYPES ARE:

                'A' FOR ASYNCHRONOUS
                'S' FOR SYNCHRONOUS
```

(See note on Figure 7.)

FIGURE 9. THE "CLKTYP" COMMAND FOR THE EQM WHEN INTERFACED TO THE MPS SIMULATOR

```
                8273 COMMUNICATIONS CONTROLLER
                ******************************

= "E"

    ENTER MESSAGE
    "This is a test message entered on the EQM's local terminal.  This
     message can be up to 248 characters long."
```

(See note on Figure 7.)

FIGURE 10. THE "ENTER" COMMAND FOR THE EQM WHEN INTERFACED TO THE MPS SIMULATOR

8273 COMMUNICATIONS CONTROL
****

= "LT"

LIST TRANSMIT MESSAGE
This is the message that was last stored by the "ENTER" command.

= "LD"

LIST RECEIVE MESSAGE
This is the last message received by the EQM.

(See note on Figure 7.)

FIGURE 11. THE "LIST" COMMAND FOR THE EQM WHEN INTERFACED TO THE MPS SIMULATOR

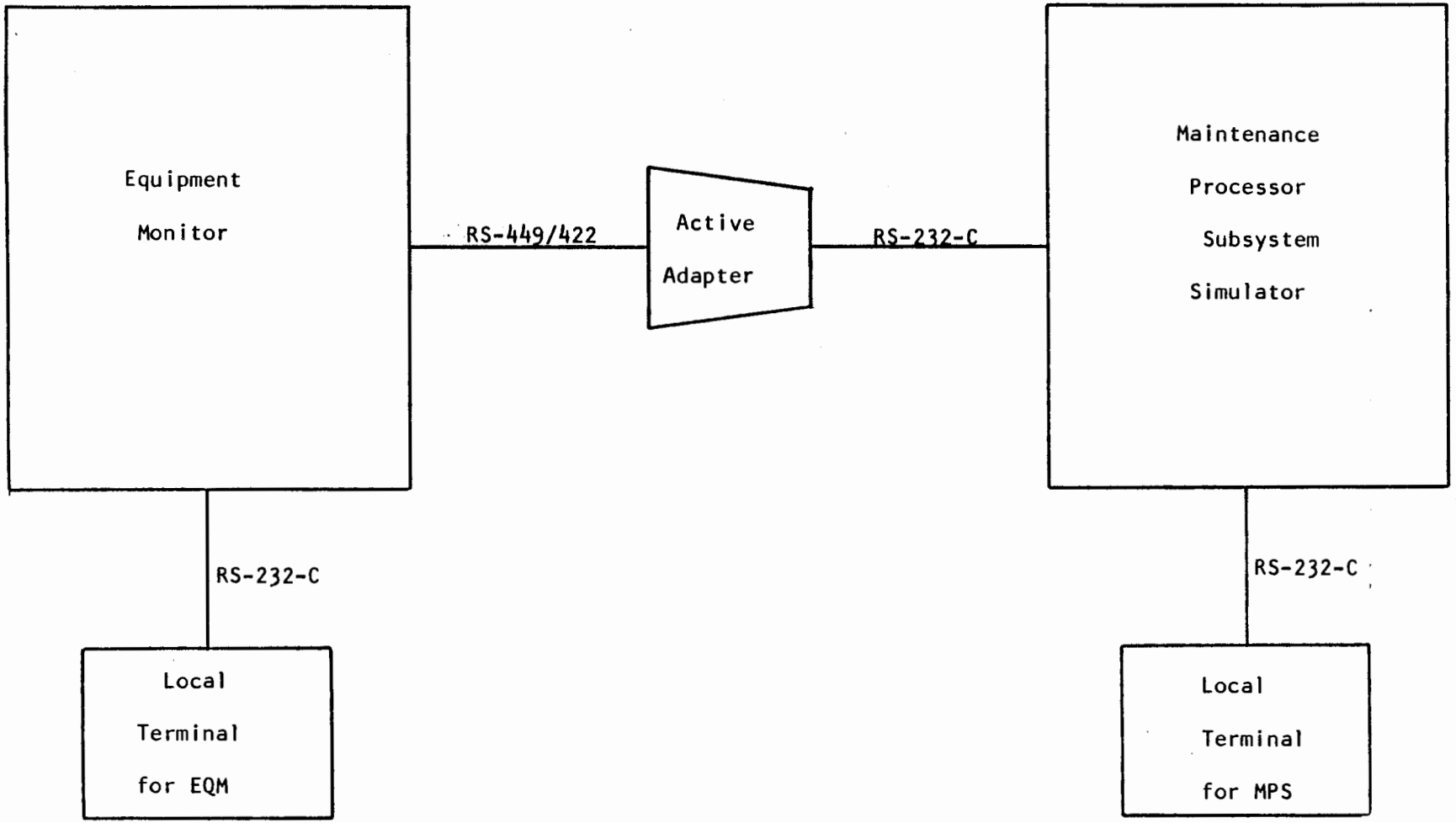


FIGURE 12. TEST CONFIGURATION FOR EQM TO MPS SIMULATOR INTERFACE

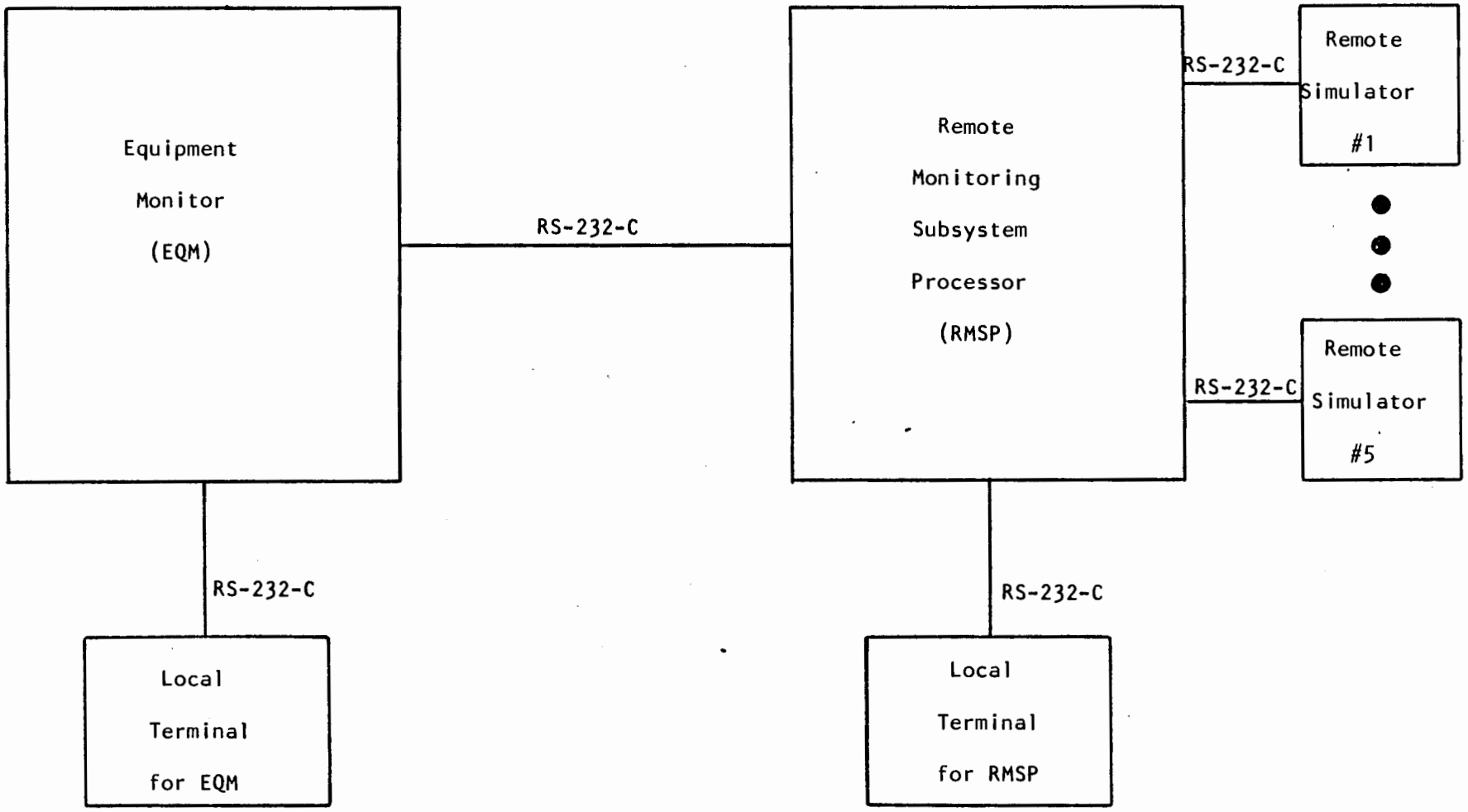


FIGURE 13. TEST CONFIGUREATION FOR EQM TO RMSP INTERFACE

APPENDIX A

This appendix will first compare the three bus structures considered for the EQM and second will show the edge connectors pin numbers with a signal description.

BUS COMPARISON

	STD BUS	S-100 BUS	MULTIBUS
Physical Size	4.5" Wide x 6.5" Long	10.0" Wide x 5.0" Long	6.75" Wide x 12.0" Long
Address Lines	16	20	20
Data Lines	8	16	16
No. of Pins on Edge Connector	56	100	86-P1 60-P2
Multimaster	NO	YES	YES
Interrupt Modes	Priority Chain or Vectored	Priority Chain or Vectored	Priority Chain or Vectored
Voltages			
Required	±5 VDC	+8 VDC & ±16 VDC	+5 VDC & ±12 VDC
Optional			
From AC	±12 VDC	±5 VDC & +24 VDC	-5 VDC & ±15 VDC
Battery	NONE	NONE	-5 VDC & -12 VDC
Standard	Manufacturers Group	Proposed IEEE-696	Proposed IEEE-796

STD BUS EDGE CONNECTOR PIN-OUT

Pin No.	Signal Description	Pin No.	Signal Description
1	+5 VDC	2	+5 VDC
3	Digital Ground	4	Digital Ground
5	-5 VDC	6	-5 VDC
7	D3 Data Bus	8	D7 Data Bus (MSB)
9	D2 Data Bus	10	D6 Data Bus
11	D1 Data Bus	12	D5 Data Bus
13	D0 Data Bus (LSB)	14	D4 Data Bus
15	A7 Address Bus	16	A15 Address Bus(MSB)
17	A6 Address Bus	18	A14 Address Bus
19	A5 Address Bus	20	A13 Address Bus
21	A4 Address Bus	22	A12 Address Bus
23	A3 Address Bus	24	A11 Address Bus
25	A2 Address Bus	26	A10 Address Bus
27	A1 Address Bus	28	A9 Address Bus
29	A0 Address Bus	30	A8 Address Bus
31	Write To Memory	32	Read From Memory
33	I/O Address Select	34	Memory Address Select
35	I/O Expansion	36	Memory Expansion
37	Refresh Timing	38	CPU Cycle Sync.
39	CPU Status 1	40	CPU Status 0
41	Bus Acknowledge	42	Bus Request
43	Interrupt Ack.	44	Interrupt Request
45	Wait Request	46	Non-Maskable Int.
47	System Reset	48	Push Button Reset
49	Clock from CPU	50	AUX Timing
51	Priority Chain Out	52	Priority Chain In
53	AUX Ground	54	AUX Ground
55	AUX +12 VDC	56	AUX -12 VDC

S-100 BUS EDGE CONNECTOR PIN-OUT

Pin No.	Signal Description	Pin No.	Signal Description
1	+8 Volts DC	2	+16 Volts DC
3	External Ready	4	Vectored Interrupt #0
5	Vectored Interrupt #1	6	Vectored Interrupt #2
7	Vectored Interrupt #3	8	vectored Interrupt #4
9	Vectored Interrupt #5	10	Vectored Interrupt #6
11	Vectored Interrupt #7	12	Reserved
13	Reserved	14	Reserved
15	Reserved	16	Reserved
17	Reserved	18	Status Disable
19	Command Control Disable	20	Memory Unprotect
21	Single Step	22	Address Disable
23	Data Out Disable	24	Phase 2 Clock
25	Phase 1 Clock	26	Hold Acknowledge
27	Wait	28	Interrupt Enable
29	Address Line #5	30	Address Line #4
31	Address Line #3	32	Address Line #15
33	Address Line #12	34	Address Line #9
35	Data Output Line #1	36	Data Output Line #0
37	Address Line #10	38	Data Output Line #4
39	Data Output Line #5	40	Data Output Line #6
41	Data In Line #2	42	Data In Line #3
43	Data In Line #7	44	M1
45	Out	46	Inp
47	Memory Read Data	48	Acknowledge Halt
49	2 MHz Clock	50	Ground
51	+8 Volts DC	52	-16 Volts DC
53	Sense Switch Disable	54	External Clear
55	Chassis Ground	56	Reserved
57	Reserved	58	Reserved
59	Reserved	60	Reserved
61	Reserved	62	Reserved
63	Reserved	64	Reserved
65	Reserved	66	Reserved
67	Reserved	68	Memory Write
69	Protect Status	70	Memory Protect
71	Run	72	Ready
73	Interrupt Request	74	Hold
75	Reset	76	Sync (Machine Cycle)
77	Write	78	Data Bus In
79	Address Line #0	80	Address Line #1
81	Address Line #2	82	Address Line #6
83	Address Line #7	84	Address Line #8
85	Address Line #13	86	Address Line #14
87	Address Line #11	88	Data Out Line #2
89	Data Out Line #3	90	Data Out Line #7
91	Data In Line #4	92	Data In Line #5
93	Data In Line #6	94	Data In Line #1
95	Data In Line #0	96	Ack. Interrupt Req.
97	Write/Output Indicator	98	Stack Address Indicator
99	Power-on Clear	100	Ground

*NOTE: Reserved pins will be used for additional address lines and data lines.

MULTIBUS EDGE CONNECTOR PIN-OUT
FOR P-1

	(COMPONENT SIDE)			(CIRCUIT SIDE)		
	PIN	MNEMONIC	DESCRIPTION	PIN	MNEMONIC	DESCRIPTION
POWER SUPPLIES	1	GND	Signal GND	2	GND	Signal GND
	3	+5V	+ 5Vdc	4	+5V	+ 5Vdc
	5	+5V	+ 5Vdc	6	+ 5V	+ 5Vdc
	7	+12V	+12Vdc	8	+12V	+12Vdc
	9	−5V	− 5Vdc	10	−5V	− 5Vdc
	11	GND	Signal GND	12	GND	Signal GND
BUS CONTROLS	13	BCLK/	Bus Clock	14	INIT/	Initialize
	15	BPRN/	Bus Pri. In	16	BPRO/	Bus Pri. Out
	17	BUSY/	Bus Busy	18	BREQ/	Bus Request
	19	MRDC/	Mem Read Cmd	20	MWTC/	Mem Write Cmd
	21	IORC/	I/O Read Cmd	22	IOWC/	I/O Write Cmd
	23	XACK/	XFER Acknowledge	24	INH1/	Inhibit 1 disable RAM
BUS CONTROLS AND ADDRESS	25		Reserved	26	INH2/	Inhibit 2 disable PROM or ROM
	27	BHEN/	Byte High Enable	28	AD10/	
	29	CBRQ/	Common Bus Request	30	AD11/	Address
	31	CCLK/	Constant Clk	32	AD12/	Bus
	33	INTA/	Intr Acknowledge	34	AD13/	
INTERRUPTS	35	INT6/	Parallel	36	INT7/	Parallel
	37	INT4/	Interrupt	38	INT5/	Interrupt
	39	INT2/	Requests	40	INT3/	Requests
	41	INT0/		42	INT1/	
ADDRESS	43	ADRE/		44	ADRF/	
	45	ADRC/		46	ADRD/	
	47	ADRA/	Address	48	ADRB/	Address
	49	ADR8/	Bus	50	ADR9/	Bus
	51	ADR6/		52	ADR7/	
	53	ADR4/		54	ADR5/	
	55	ADR2/		56	ADR3/	
	57	ADR0/		58	ADR1/	
DATA	59	DATE/		60	DATF/	
	61	DATC/		62	DATD/	
	63	DATA/	Data	64	DATB/	Data
	65	DAT8/	Bus	66	DAT9/	Bus
	67	DAT6/		68	DAT7/	
	69	DAT4/		70	DAT5/	
	71	DAT2/		72	DAT3/	
	73	DAT0/		74	DAT1/	
POWER SUPPLIES	75	GND	Signal GND	76	GND	Signal GND
	77		Reserved	78		Reserved
	79	−12V	−12Vdc	80	−12V	−12Vdc
	81	+5V	+ 5Vdc	82	+5V	+ 5Vdc
	83	+5V	+ 5Vdc	84	+5V	+ 5Vdc
	85	GND	Signal GND	86	GND	Signal GND

MULTIBUS EDGE CONNECTOR PIN-OUT FOR P-2

PIN	(COMPONENT SIDE) MNEMONIC	DESCRIPTION	PIN	(CIRCUIT SIDE) MNEMONIC	DESCRIPTION
1	GND	Signal GND	2	GND	Signal GND
3	5VB	+ 5V Battery	4	GVB	+ 5V Battery
5		Reserved	6	VCCPP	+ 5V Puised Power
7	-5VB	- 5V Battery	8	-5VB	- 5V Battery
9		Reserved	10	Reserved	
11	12VB	+12V Battery	12	12VB	+12V Battery
13	PFSR/	Power Fail Sense Reset	14	Reserved	
15	-12VB	-12V Battery	16	-12VB	-12V Battery
17	PFSN/	Power Fail Sense	18	ACLO	AC Low
19	PFIN/	Power Fail Interrupt	20	MPRO/	Memory Protect
21	GND	Signal GND	22	GND	Signal GND
23	+15V	+15V	24	+15V	+15V
25	-15V	-15V	26	-15V	-15V
27	PAR1/	Parity 1	28	HALT/	Bus Master HALT
29	PAR2/	Parity 2	30	WAIT/	Bus Master WAIT STATE
31	↑		32	ALE	Bus Master ALE
33	|		34	Reserved	
35	|		36	Reserved	
37	|		38	AUX RESET/	Reset switch
39	|		40	↑	
41	Reserved		42	|	
43	|		44	|	
45	|		46	Reserved	
47	|		48	|	
49	|		50	|	
51	|		52	|	
53	|		54	|	
55	|		56	|	
57	|		58	|	
59	↓		60	↓	

Notes:

1. PFIN, on slave modules, if possible, should have the option of connecting to INT0/ on P1.
2. All undefined pins are reserved for future use.