DESIGN AND IMPLEMENTATION OF A MEDIUM SPEED COMMUNICATIONS INTERFACE AND PROTOCOL FOR A LOW COST, REFRESHED DISPLAY COMPUTER

By James R. Rhyne and Mart D. Nelson

The University of Houston
Department of Computer Science

INTRODUCTION

Refreshed displays have several advantages over storage tube displays, but the primary advantage is the ability to selectively delete or alter portions of a display without having to re-draw the entire display [1]*. When such displays are used in conjunction with a large scale computer system, interaction with a computer program has new dimension of ease and flexibility. Because such displays must re-draw the entire picture several times each second to prevent it from flickering or fading from view, quite a bit of processor power can be tied up in the task of keeping the picture visible.

Some displays have a special processor, called a display processor, which refreshes the picture on the screen at regular intervals, thus removing a substantial processing load from the large scale computer system. It is possible to buy a display system with a built-in display processor for under $20,000, and recent advances in LSI technology indicate that more such display systems will be available in the future. Most of these low cost displays have been used either as stand-alone systems or have been connected with a large scale computer system by low speed, asynchronous communication lines.

26

DESIGN CRITERIA

We were curious to see what difficulties and benefits would result from using a 40,000 bit/second communication line as the connecting link between an IMLAC PDS 1-D display computer [2] owned by the Department of Computer Science and the University of Houston's Univac 1108 computer system. The obvious benefit is a relatively high speed for data transfers between the 1108 and the IMLAC which permits rapidly changing displays. The main difficulty lies in designing a hardware and software interface which would meet the design criterion of 40,000 bit/second communication without being expensive. This paper describes the hardware and software system which resulted.

Figure 1 shows a block diagram of the IMLAC. It consists of two independent processors which share a common memory. The display processor generates the deflection and beam control currents as it interprets a program contained in the memory; the minicomputer has a general instruction set and is responsible

*Numbers in brackets indicate references.

for starting and stopping the display processor and for communicating with the outside world through the keyboard, teletype, light pen, and communication line.

We began the investigation with a study of the feasibility of communicating at the design goal rate of 40,000 bits/sec. The hardware of both the IMLAC and the Univac 1108 were capable of operating at this speed, but it was not clear that the IMLAC software could handle data coming in at the projected rate. The IMLAC minicomputer has a very limited set of operations; for example, in a single instruction, the accumulator can be shifted only one, two or three bit positions. We identified various design questions which would have to be answered and resolved to answer them all in a manner which would involve the least IMLAC processing time.

We found that we could not guarantee that an overspeed condition at the IMLAC would never take place; certain programming techniques cause the display processor, which has priority in memory access over the minicomputer, to lock the minicomputer from accessing memory over extended periods of time. The communications protocol would have to be capable of detecting and correcting loss of data due to overspeed, as the hardware has no means for detecting loss of data. Furthermore, communication links are subject to occasional error and failure, and loss of data on account of these error conditions must also be corrected.

There are two general techniques for error detection and correction in communications links: the message in error may be re-sent, or enough redundancy may be supplied in the message to allow the errors to be corrected after receipt. The second of these alternatives was rejected because error-correcting codes involve a loss of effective transmission speed due to redundancy, and they require a great deal of processing of the message as it is received. We expected a very low error rate so that the alternative of re-sending messages found to be in error is plausible, and it has the advantage of requiring only minimal processing of the received message.

One typical data transmission technique uses vertical and horizontal parity bits to detect loss of data. The data is sent as seven bits out of eight, with the eighth bit containing the parity of the other seven. We expected that loss of data due to overspeed would occur much more frequently than loss of one or more bits through a transmission error. In an overspeed condition, the vertical parity bit appended as the eighth bit of each group of seven data bits serves no useful function because the entire group of eight bits is lost.

Furthermore, the IMLAC memory is composed of sixteen bit words, and three groups of seven data bits would be required to completely fill a memory word, with five bits of the final group being ignored. The effective transmission rate would then be 67% of the actual transmission rate. The other alternative, of sending 16 groups of seven bits which would exactly fill seven sixteen bit words, was rejected as requiring too much processor time in the minicomputer, especially in view of the limited range of shift instructions noted previously.

Therefore, it was decided that the data would be sent as an eight out of eight bit code, requiring two groups of data bits to fill each IMLAC word. Error checking would be accomplished by using a function which computed a check-

sum for the entire message and appended this checksum to the end of the message. This led to another problem, which is that most communication protocols make use of special codes which signal the beginning and end of messages; in particular, one special code, a synchronization character, is necessary for the hardware to correctly reassemble the serial bit stream into groups of eight bits. If all eight bits of the code are used for data, then these special characters can be found in the message on occasion, and this must not disrupt the communications protocol.

There are currently two proposed standards for communication protocols which use an eight out of eight bit code: the DDCMP protocol developed by the Digital Equipment Corporation [3] and the SDLC protocol proposed by IBM [4]. The SDLC protocol was rejected because it would have required the construction of a special interface which eliminates certain sequences of bits by insertion of a single 1-bit as the message is transmitted and deleting these inserted bits as the message is received. The DDCMP protocol could be implemented without hardware changes to either the Univac 1108 or the IMLAC.

In the DDCMP protocol, messages are of two types: protocol and data. Each data message is divided into two parts: a fixed length header which contains the length of the second part of the message and other error detection and synchronization information, and the body of the message which is variable in length and which follows the header. Both parts of the data message have error check codes appended to them, so that the length can be checked for error prior to receipt of the body of the message. And, since the length of the body of the message is known before the message body is received, there is no need for a special end-of-message character.

Two fields in the header of a data message indicate the sequence number of the data message and the sequence number of the last message correctly received by the sending station. Loss of an entire message can be detected by finding a message which has a sequence number two or more greater than the number of the last message correctly received. The second sequence number field tells the receiving station what messages have so far been correctly received by the sending station and thus eliminates the need for a special message which acknowledges the receipt of each message (the acknowledge, or ACK, message).

The main use of the protocol message, which has a fixed length that is the same as the data message header, is to notify the sending station that a message was received which is in error and to cause the sending station to re-send the data message. If station A sends a data message with sequence number 15 to station B, and B does not receive the data message correctly (i.e. the computed check data is not the same as the check data appended to the message), then B sends a negative acknowledge (NAK) protocol message to A which contains the sequence number 14. Station A, upon receiving the NAK message from B, immediately re-sends all messages having a sequence number greater than 14. Other protocol messages correct sequence number synchronization errors and allow for initial startup of the communications link.

The DDCMP protocol has two disadvantages which we felt should be corrected for our application: one, it is designed to be used with input and output

buffering techniques and although the Univac would have no difficulty in managing input and output buffers, we doubted that the IMLAC would have sufficient processing power for this task; and, two, the DDCMP protocol uses a CRC-16 polynomial to generate the error check code. The CRC-16 check polynomial computation involves a shift and an addition operation for each bit of the data message, and it was clear to us that the IMLAC would not be able to compute this polynomial at the target data transmission rate.

The first disadvantage was overcome by making the Univac 1108 responsible for buffer management in the IMLAC; this implies that the Univac must know where each message is to be stored in the IMLAC memory and that it must specify the IMLAC memory address for each message sent to the IMLAC as a part of the message. The memory address was incorporated in the header part of data messages by extending the size of the header by two bytes. The check code for the header thus verifies the correctness of both the message length and the message address. The IMLAC message processor software is given the message length and the memory address at which the message should be placed before the body of the message is received; as the body of the message is received it is placed directly into the predetermined locations in the memory. A data message coming from the IMLAC to the Univac 1108 contains the starting address of the message body in the IMLAC memory, so that the Univac can be aware of where the message originated in the memory of the IMLAC.

We also decided on a sixteen bit sum function with end-around carry as the appropriate choice for an error check code generating function. This function is reasonably simple to compute and is secure enough for our environment.

IMPLEMENTATION

Having made these design decisions, the software for the IMLAC was designed for our modified version of the DDCMP protocol. Some timing computations were made assuming that the display processor would be running continuously in increment mode; in this mode, it steals every other memory cycle and slows the minicomputer to half its normal operating speed. At the target data rate of 40,000 bits per second, the IMLAC would be being interrupted 5,000 times per second to process eight bit data groups, giving 200 microseconds as the maximum time to process each data byte. The effective rate for memory cycles for the minicomputer is one cycle every 3.6 microseconds, allowing a maximum of 55 memory cycles for the processing of data interrupts for both input and output.

We minimized the processing time associated with each data byte by designing the input and output processes as finite state machines which automatically sequence from each state to the next state. This design was especially easy to implement on the IMLAC because of certain memory locations which are automatically incremented each time they are referenced. These indexing memory locations address a table of jump instructions which allow the proper processing routine to be entered in only three memory cycles.

The interrupt processing routine places interrupts from the communications interface as the highest priority, with light pen, keyboard, asynchronous

communications interface, and display refresh interrupts at a lower priority level. After carefully coding each process for message receipt and transmission, we were able to verify that messages could be sent at full speed over both lines of the communication link without overspeed. Less than ten percent of the processor is available for handling other interrupt conditions when this is occurring, so the display may flicker slightly under these worst case conditions.

The format for the revised DDCMP protocol is shown in the appendix and Figures 2 and 3 show the state diagrams for the input and output processes respectively.

Having established the feasibility of communication at the target rate, we investigated various means of connecting the IMLAC to the Univac. Modems which can send data at 40,000 bits per second are quite expensive, so we decided to hard wire the IMLAC to the Univac. One of us (Nelson) designed and built a clock source and we acquired 200 feet of special, low capacitance, computer grade cable. Anticipating that some applications might be able to tolerate a lower data rate on the IMLAC to Univac side of the communications link in return for additional interrupt processing power in the IMLAC minicomputer, the clock source was made so that the transmission rates could be set by means of switches which give a range of speeds from 75 bits per second to 40,000 bits per second. After some difficulties with the Univac hardware, the hardware connection was made operational in February, 1975.

Since that time, we have run several tests of the communication link and the IMLAC software; in one test, a block of known data was circulated between the Univac and the IMLAC for over an hour at 40,000 bits per second without error; so we feel that the hardware connection is quite stable and error free.

A realtime communications handler was developed for the Univac 1108 so that it could send and receive messages over the communications link to the IMLAC. Under EXEC-8, communications handlers must be realtime programs and they are never swapped from memory. It is not feasible to have a large program locked into memory, as this would very seriously degrade the performance of the system. The 1108 software was designed so that only the handler program, which is about 4K words, needs to be locked into memory; the user program which creates display files runs as a regular conversational program and may be swapped.

EXEC-8 did not provide a mechanism which would allow the communication handler to pass messages to the user program and vice versa. Only an extremely small number of operating systems provide such a mechanism to the general user. We determined that two approaches to implementing a message passing facility were possible: an executive routine could be written to allow one program to send another program a message, or the two programs could share a common area of memory. The former method was felt to be less desirable than the latter because it would require implementation of a new executive function, with the attendant system maintenance difficulties, and because it would have required a substantial enlargement of the resident executive part of EXEC-8.

The implementation of the message passing facility proved to be an interesting exercise in design. Message traffic in one direction can be visualized as a

two stage producer/consumer problem. One program places a message in the common area and then notifies the other program via a semaphore that a message is available. The other program becomes eligible for processor time and eventually removes the message from the common area. Since messages are passed in both directions simultaneously, four activities are required to transfer messages.

If one thinks of each activity as a message pump, then it is immediately apparent that each activity has the same function as the other activities, differing only in the location from which it obtains its input and to which it sends its output. Figure 4 illustrates this. With this in mind, we designed the pump program as a general, re-entrant routine, and it is located in the common area where it is executed by all four activities simultaneously.

The software and communications link has been functioning for over three months now, and we have been able to successfully demonstrate simple animations of stick figures. The animations are created by the 1108 and demonstrate the ability of the 1108 to rapidly alter a display program running in the IMLAC via the high speed communications link.

All of the difficulties which we have encountered thus far have been with the Univac software and hardware. In the message passing facility, we have uncovered what is apparently a design flaw in EXEC-8 that causes occasional system crashes, and we expect that the Univac personnel who have been helping us will have this problem fixed shortly.

The next stage in the development of the graphics subsystem will result in an increase in hardware sophistication and in the implementation of a display file compiler that is designed for implantation in the FORTRAN and COBOL libraries, and in special versions of the LISP, APL, and BASIC interpreters. The hardware development will place the communications protocol handler in an outboard microprocessor which will transfer data to and from the IMLAC memory directly, thus freeing the minicomputer in the IMLAC of the burden of communications processing and facilitating the response of the IMLAC to the light pen and other devices (Figure 5).

ACKNOWLEDGEMENTS

The authors wish to acknowledge the support of this project by The Energy Institute of The University of Houston, and by the Computing Center of the University of Houston. The development of further software and applications is being supported by the National Science Foundation, Grant number, DCR 74-17282.

REFERENCES

1. Newman, William M., and Sproull, Robert F., Principles of Interactive Computer Graphics. McGraw-Hill, 1973.

2. IMLAC Corporation "IMLAC PDS-1D Programming Guide," 1973.

3. Wecker, Stuart "A Message-Oriented Protocol for Interprocessor Communication," Digital Equipment Corporation, May 1974.

4. IBM Corporation "IBM Synchronous Data Link Control General Information," Form Number GA27-3093-0, March 1974.

APPENDIX

PROTOCOL MESSAGE DESCRIPTIONS

Data Message Description and Format

The elements in a data transmission are as follows:

SOM, device address, flags and count, response, message number, memory address, check 1, data, check 2

Note: Preceding the SOM character, sync words are transmitted (octal word 026). These characters cause the receiving port hardware to synchronize to the incoming data so that each successive 8-bit word will be correctly received.

SOM	- the Start Of Message character (8-bits). This, or DLE, must be the first non-sync character to insure that the receiving port has not started due to a false sync word.
device address	- the 8-bit quantity used to specify which device a message is for in a multi-device system
flags and count	- the 16-bit quantity containing a 13-bit count of the number of 16-bit words in data and three bits used as control flags, if required
response	- the response from a device giving the number of the last received error-free message
message number	- the number (in modulo 256) which is assigned consecutively to each transmitted message
memory address	- the 16-bit address in PDS-1 memory at which the received message storage is to begin or from which a transmitted message reading began
check 1	- the 16-bit checksum of the message header formed by adding each successive 16 bits with end-around carry
data	- the 16-bit data elements, the number of which was specified in flags and count
check 2	- the 16-bit checksum of the data elements

Non-data or Protocol Message Descriptions and Formats

Protocol messages are header-only messages which are used for system control and error recovery. The three types of messages are: (1) message acknowledgement or ACK, (2) erroneous message acknowledgement or NAK, and (3) a general purpose message or MES.

The elements of an ACK message are as follows:

DLE	- the Data Link Escape character. This indicates that a header-only message follows.
ACK type	- identification of an ACK message

fill 1 - 8-bit fill word

fill 2 - 16-bit fill word

check - message check word

The elements of a NAK message are as follows:

DLE, device address, NAK type, error, response, message number, fill 2 check

NAK type - identification of NAK message

error - the type of error found, if required

The elements of a MES message are as follows:

DLE, device address, MES type, message, response, message number, fill 2 check

MES type - identification of MES message

message - 8-bit field conveying the required message

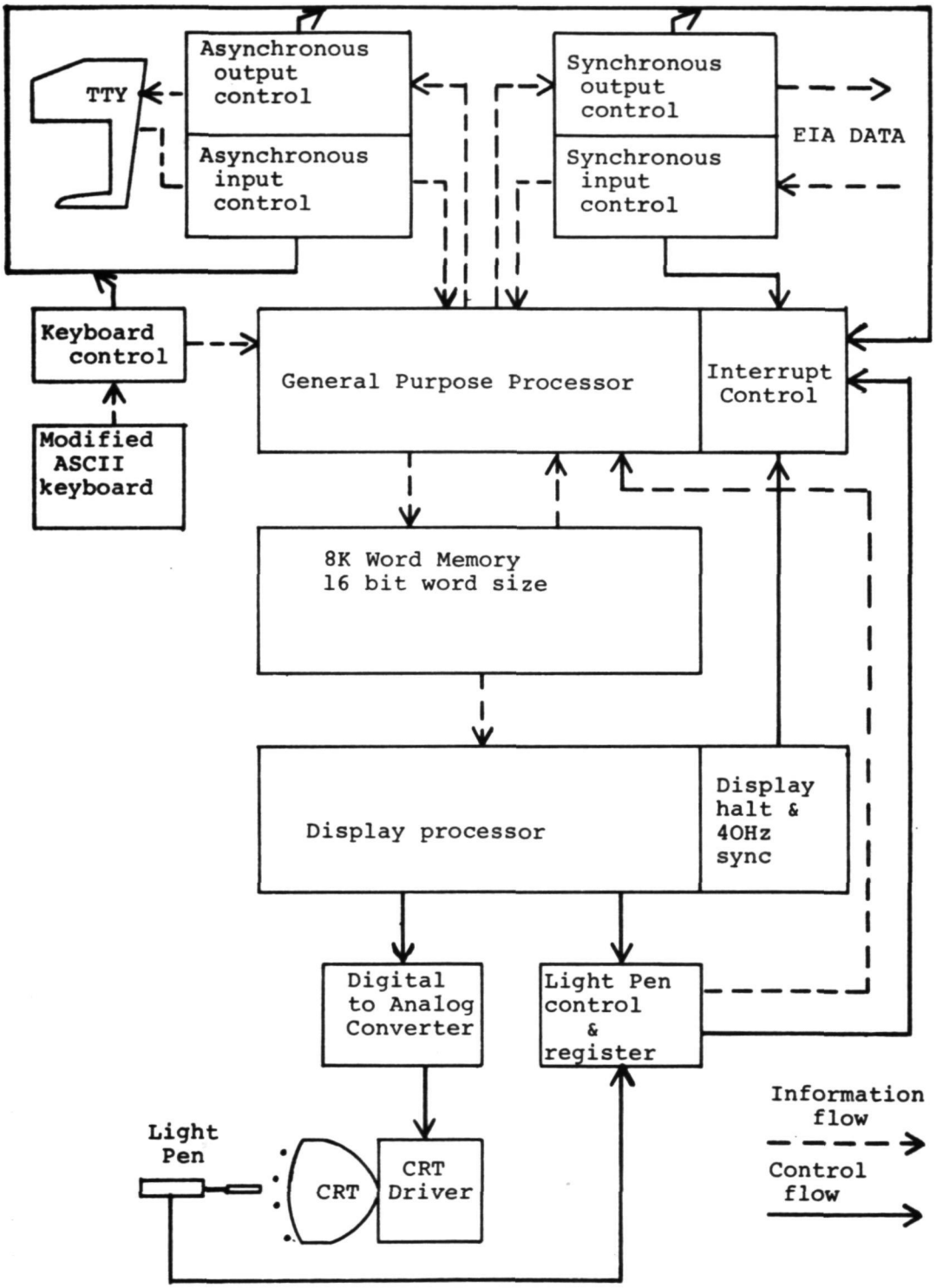

Figure 1.- IMLAC PDS-1D organization.

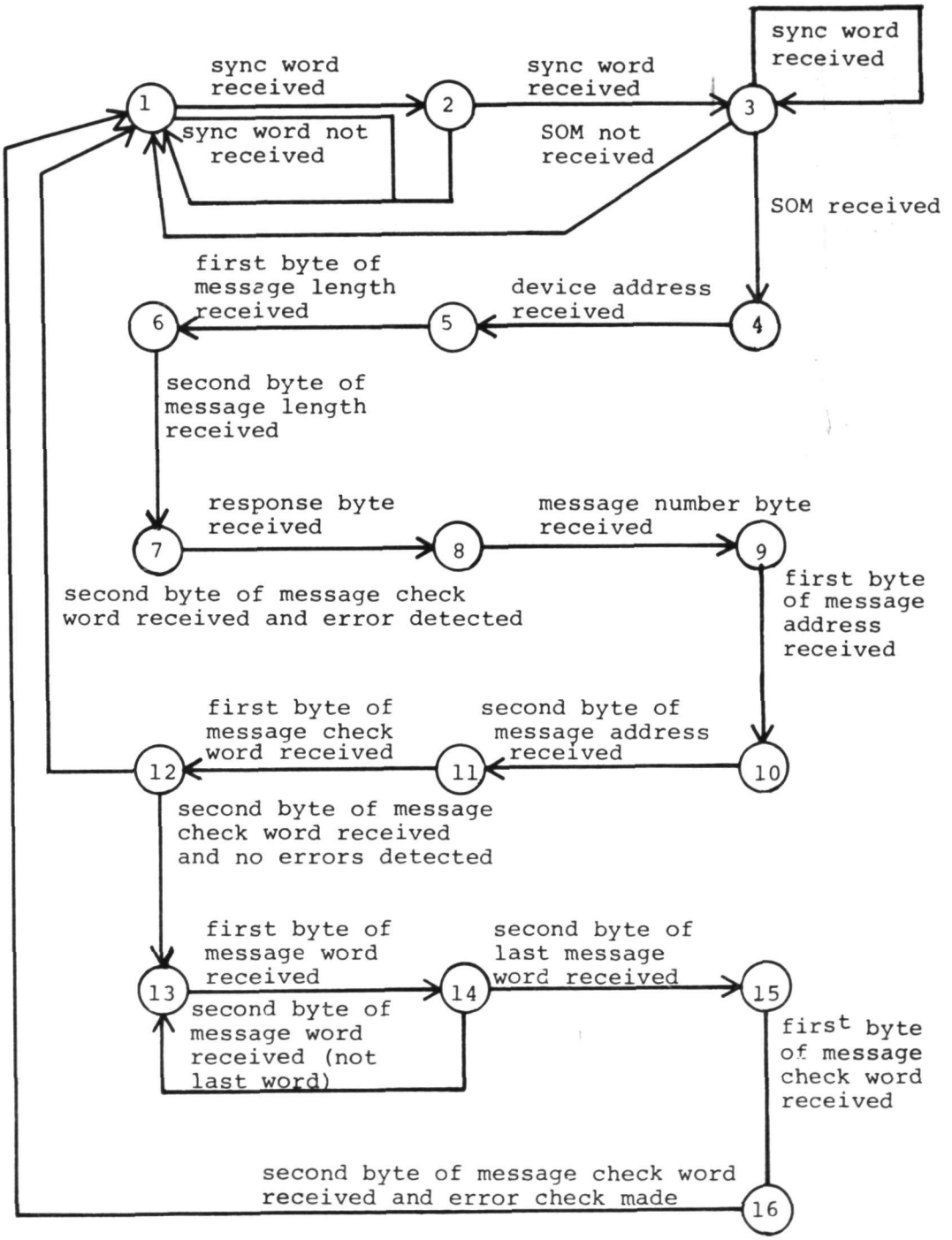

Figure 2.- Receive state transition diagram.

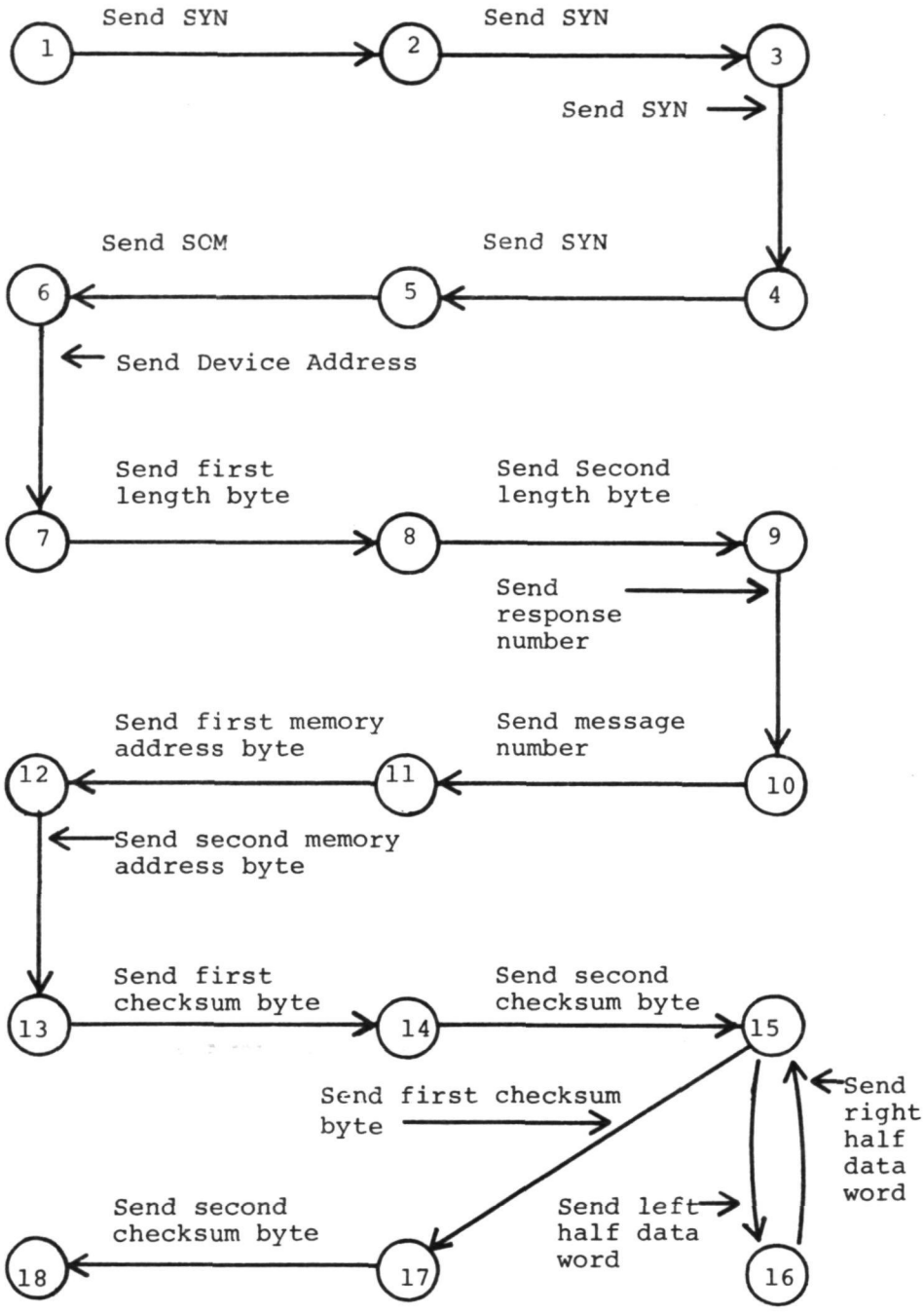

Figure 3.- Send state transition diagram.

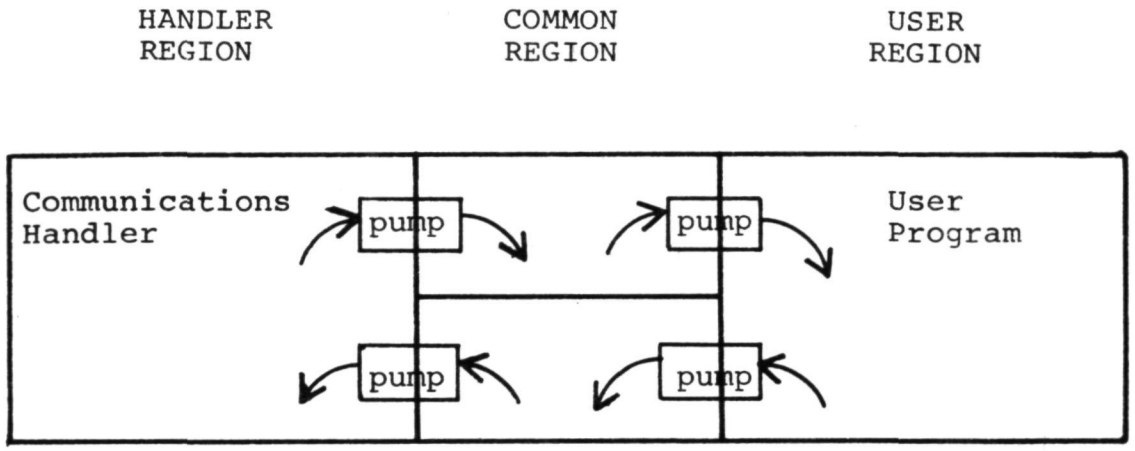

Figure 4.- Interprocess communication scheme.

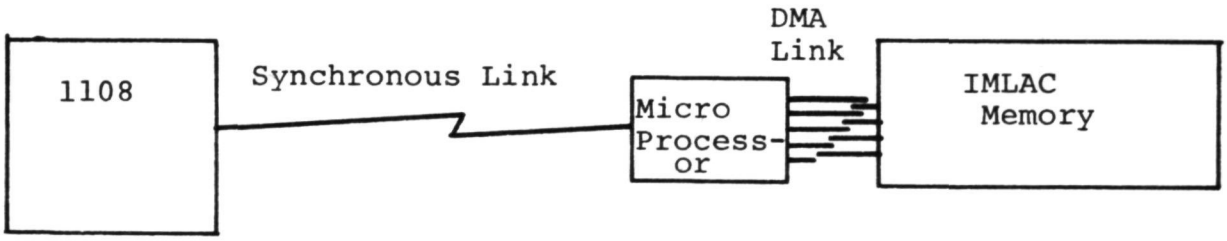

Figure 5.- Proposed data link hardware.