SLAC-PUB-2420
October 1979
(M)

PROGRAMMABLE SYNCHRONOUS COMMUNICATIONS MODULE*

D. Horelick
Stanford Linear Accelerator Center
Stanford University, Stanford, California 94305

and

Daresbury Laboratory
Daresbury, England

1. INTRODUCTION

The purpose of this paper is to describe in general the functional characteristics of a programmable, synchronous serial communications CAMAC module with buffering in "block" format. It is intended that both bit and byte oriented protocols can be handled in full duplex depending on the program implemented. In particular network protocols such as X.25 (HDLC) to 48 KBPS[1] and Bi-Sync to 9.6 KBPS are expected to be supported, but of course other applications are possible, limited only by the hardware and operating speed. The general requirements are summarized in Appendix A.

The main elements of the module are a Signetics 2652 Multi-Protocol Communications Controller (MPCC), a Zilog Z-80A 8 bit Microprocessor with PROM and RAM, and FIFOs for buffering the data.

This work was done by the author while on leave at Daresbury Laboratory, Science Research Council, Daresbury, England, during 1978-79. Many individuals at Daresbury who made this work possible are described in the acknowledgements. An earlier version of this paper was published as a Daresbury internal report.

* Work supported in part by the Department of Energy under contract No. DE-AC03-76SF00515 and by Science Research Council, Daresbury, England.

[1]KBPS: Kilo Bits Per Second.

(Invited talk presented at the Joint AT-E Division Seminar, Los Alamos, N. Mexico, November 14, 1979.)

2. THE MICROPROCESSOR

The basic role of the microprocessor is to set up and control the communications chip (MPCC), and to handle the various data flows in the module, as well as providing the necessary processing power to deal with various non-standard elements of protocols. For example it must detect the end of a received block in Bi-Sync, and signal the end of block in both Bi-Sync and X.25 transmissions since these functions are not handled by the MPCC chip.

A Z-80A microprocessor is used, and the reader is referred to manufacturer's literature for detailed description and instruction set. Z-80A architecture is based on the INTEL 8080 and the well-known 8080 instruction set is a subset of the Z-80A, which includes additional instructions such as "block transfers" and individual bit testing and manipulation. (See Appendix B.) The CPU has a dual internal register set for improved interrupt response and two 16-bit index registers for indexed addressing.

The clock rate is 4.0 MHz. Some typical processor execution times are as follows. Add time from memory to accumulator is 1.75 μsec, and load time from memory to accumulator is 1.75 μsec, assuming the memory address is in a 16-bit register pair. Input or output from/to an 8-bit I/O port takes 2.5 sec. To set or reset a bit in an internal register takes 2 μsec; the same operation on a memory bit takes 3.75 μsec.

It is planned to use the micro in a "polling" mode in which it monitors various status bits in order to control the module. All data transfers within the module are implemented as memory mapped transfers for flexibility in programming, and speed.

3. GENERAL PHILOSOPHY OF OPERATION

The general operational philosophy of this module is that of sending and receiving blocks of data in order to optimize the CAMAC data flow. That is, in reception of data, data bytes are stored in a buffer memory (FIFO) in the module until the complete block is received. The module then generates a CAMAC interrupt. The higher level processor (the CAMAC computer) responds, reads the word count and error status, followed by a single address CAMAC block transfer until the entire block is transferred. This transfer can take place at the maximum CAMAC speed of 1 MHz, limited only by the speed of the particular CAMAC system. In the reverse direction the module generates an interrupt when buffer space is available for a complete block of transmit data. The CAMAC processor responds by sending a complete block in single address mode at maximum speed. The important point is that there is only a single interrupt for each block of data instead of an interrupt for each byte, as is done in non-buffered modules. Thus the hardware-software overheads for dealing with the CAMAC interrupt environment are virtually eliminated, and the overall data transfer rate is greatly increased. To further improve CAMAC transfer rates, packed data bytes are transferred as 16-bit words, both in the transmit and receive directions. (Eight bit transfers are optional.)

A key system parameter in this environment is CAMAC block transfer capability which must be optimized to achieve the full capabilities of the system. In many cases best performance will be achieved by transferring the CAMAC data in a DMA mode — that is, moving data into/out of computer memory without program intervention. Hopefully CAMAC systems employing this module will have high performance DMA capabilities.

The possible effects of "blocking" data in the module upon overall network delay should be carefully considered in each application, as well as the complexities of multiple modules in a single crate.

4. GENERAL DESCRIPTION OF THE MODULE

Figure 1 shows the general organization of the module. For flexibility in adapting to new components and new requirements the module is packaged on three boards, one the CPU and closely associated components such as PROM and RAM, another the communication oriented hardware, the last contains the FIFOs. The boards communicate via a simplified Z-80A bus. One can then more easily accommodate major changes either in the communications and buffering hardware, or the processing/control hardware as determined by changes in requirements, or the available technology to do the job. It is even possible to replace the function of the CPU board with a microsequencer or high speed Z-80A emulator should certain applications need this capability.

The Z-80A bus is a conventional micro computer bus with a 16 bit address, an 8 bit data bus and 13 control signals. The components relating to the microprocessor are 4K bytes of static RAM based on INTEL 2141-5 and 1K bytes of EPROM based on the INTEL 2708-1. Expansion of both PROM and RAM will be provided on the CPU board up to a total of 4K EPROM, 8K RAM. A conventional asynchronous port is provided on the board using the TMS 6011 or equivalent. This is interfaced to the outside world via a standard V.24 (RS-232C) or current loop port operating at selectable rates up to 9600 baud; a VDU can be used for direct operator interaction with the module to facilitate hardware/software development and debugging.

Further components on the board are eight programmable status LED's to indicate module status, eight sense switches, a reset push button, an interrupt push button and a dedicated LED for CPU "HALT". Other than power, there are no Dataway connections on this board.

The bus signals are transmitted using a reduced Z-80A bus. That is, address space is limited to 64 bytes by comparing the 10 MSB of the 16 bit address to a selected "page number". A single "enable" line, plus the 6 LSB of address are then transmitted to the other boards. This simplifies the board interface and the address decoding. Drivers/receivers isolate the signals on the boards.

The data FIFOs to buffer transmitted and received frames of data are based on 16 pin, 4 bit × 64 word FIFO chips, Am 2841, and the memory space of 640 bytes is somewhat larger than the minimum requested. FIFO technology appears to be far behind RAM developments, and the design of a RAM based FIFO with 2K bytes of space was considered. Appendix C indicates the status of this work, which is a possible alternative. Other possibilities are the 256 byte FIFO's (Z-FIFO) being developed by Zilog, but these are not yet available. Using the 4 × 64 FIFOs, 20 chips for each direction are required.[2] All transactions between the FIFOs and the communications chip (MPCC) are routed via the microprocessor. In order to handle 16 bit words the FIFOs appear as two memory locations on the bus; bit ordering can be jumper selected and it is also possible to operate in single byte mode (non-packed) with the same buffer capacity, by reconnecting the FIFOs using jumpers. The transmit FIFO must generate a CAMAC interrupt (L) when space is available for another block. This is accomplished by a jumper connection on the FIFOs selectable every

[2]A prototype module with a reduced FIFO capacity of 384 bytes (12 chips) was actually built.

128 bytes. (Every 64 bytes in byte mode.) Of course both transmit and receive FIFOs are capable of 1 MHz operation on the CAMAC Dataway.

Closely related components are the control FIFOs. These contain block length and other status information concerning the data contained in the larger FIFO data buffers, and are independently loaded and read by the micro and the Dataway. They are again based on the 4 × 64 word FIFO chips, and are two bytes wide at the Dataway, but are only 64 words deep. They each appear as two memory locations on the microprocessor bus. Note that a CAMAC interrupt (L) is generated when there are any control words in the Receive Control FIFO.

The Signetics 2652 Multi-Protocol Communications Chip is used to handle the routine protocol tasks. It handles both bit and byte protocols and is specified to work to 1 MBPS. It has been selected since its performance has been verified by others, and it is readily available. All interactions with the micro are on a byte basis, and the MPCC mode of operation is programmed by the micro in the initialize routine. Its data and control registers will appear as several memory locations on the bus.

This chip accomplishes many protocol functions for X.25. For example it does "zero" insertion and deletion, generation and detection of FLAGS, CRC-CCITT generation and checking. It therefore produces a byte stream of binary data in receive mode, and signals the end of block. In transmit mode the micro must load the control register in the chip to terminate a block, using word count information which comes from the transmit control FIFO. In byte oriented protocols such as Bi-Sync, its processing is more limited, although it does generate and detect SYNC (two characters), generate and check CRC-16 or VRC, but not LRC, which is the responsibility of the processor. It does not deal with DLE insertion and deletion, nor does it detect the end of the block. In general then, the responsibility of the micro is considerably greater in the older byte formats than the newer X.25 environment. To more completely understand the overall capabilities of the chip, and the resulting implications in the software, the reader is advised to refer to the Signetics 2652 data sheet.

Standard MODEM control signals are programmed via a register accessible as a memory location. Likewise, status of the MODEM will be monitored by the program of the micro, accessed again as a memory location. These actions can be taken either in an initialize routine, or in the operating program.

The overall control of the module is determined by a "system status word" which indicates to the micro which system component needs service. In this way the complexities and delays of interrupt hardware and interrupt subroutines can be avoided. This word is read at the end of a task to determine the next task. The system status bits are:

(1) Received data byte available from MPCC, must be read within one byte time.

(2) Transmitter buffer empty from MPCC, must be filled within one byte time.

(3) Transmit Control FIFO contains control words, indicating data is to be transmitted.

(4) Receiver Status has changed, indicating end of block in HDLC, error, abort or receiver overrun.

(5) Byte in CAMAC input register to be read.

(6) No byte in CAMAC output register, i.e., CAMAC processor has read the output byte.

The latter two conditions are of lower priority since they are in a non-time critical environment.

A patchable OR function of the system status bits is also provided for an optional maskable interrupt, to permit rapid response to time critical events, although polling of the system status register is the preferred mode of operation.

The CAMAC input register is used for general control of the module from the CAMAC processor — such as start operation, stop, program loading mode, self-test, initialize, MODEM control and other conditions as determined by the program. Likewise the CAMAC output register is used to transmit up to 256 conditions (8 bits) of module status to the CAMAC processor, to acknowledge commands, etc. again as determined by the program. These CAMAC operations are synchronized by LAMs on the Dataway side, and by status bits on the microprocessor side.

The communication board is completed by a CAMAC decoding/control subsystem and a CAMAC interrupt structure consisting of a conventional LAM status, mask, and request register. These LAMs interrupt the CAMAC processor when data blocks are to be written or read, when the CAMAC registers require service, or when an illegal operation takes place in reading or writing FIFOs.[3] Figures 2 and 3 give details on these elements of the system.

A RAM, PROM, FIFO memory map is shown in Figure 4. A preliminary prototype front panel is shown in Figure 5. Note that there are

[3]The latter feature, FIFO error flags, was omitted from the prototype for simplicity.

connectors for transmit data ready and receive data ready that can be used as DMA Sync signals.

5. TYPICAL OPERATION - HDLC OR X.25

The following description gives an example of the sequence of operations in the HDLC environment assuming the program is loaded, the MPCC chip is initialized, the MODEM is set properly and data is flowing. The actual sequence is of course software controlled.

The MPCC chip searches the received data stream for a FLAG byte. The micro is in a loop, testing system status. When the MPCC assembles the first byte after the FLAG the micro detects this from the status byte, reads the MPCC chip (which resets the status bit), starts accumulation of a byte count and sends the byte into the receive data FIFO. This continues, with the micro alternating the FIFO addresses to pack the data into 16 bit words. At the end of the block another status bit is set by the MPCC; the micro then reads the last data byte and the MPCC status byte which indicates error conditions, uncompleted bytes, etc. The micro then fills any uncompleted words, sends one or two words of data block delimiters (such as all zeroes) and a block sequence count into the receive data FIFO, and sends two bytes of status information to the receive control FIFO. This status information includes a block sequence, eight or nine bits of byte count (or a CAMAC word count if that is more appropriate) plus the block status information from the MPCC chip, which consists of five bits. The control information reaching the CAMAC end of the receive control FIFO generates a CAMAC interrupt. The CAMAC processor reads the 16 bit control word (word count and status), followed by a sequential (block transfer) readout of the data FIFO. Proper blocking is confirmed in the CAMAC processor by checking the block delimiters. Of course, data from the next block may be received and sent to the data FIFO while the CAMAC readout process is underway.

At the same time data bytes are being transmitted by the MPCC chip. The CAMAC processor receives an interrupt when the transmit data FIFO can accept a block of data. The CAMAC processor then sends a block of data as a sequence of 16 bit words, followed by a block delimiter word (again a word of zeroes is a possibility) and a block sequence count. It then sends a control word of 16 bits which includes the byte count, 8 or 9 bits, and any specific control information. The micro monitors the system status word. When a transmit control word is present, and it is not presently transmitting a block, the micro reads the two bytes of transmit control information, sets up a byte count, sends a Transmit Start of Message to the MPCC and waits for a Transmitter Buffer Empty status from MPCC. It then reads the first byte from the transmit data FIFO and sends this to the MPCC chip, which resets Transmit Buffer Empty. This process continues until the micro byte count indicates the end of the block. After the last byte is sent to the MPCC the micro sets Transmit End of Message and the MPCC chip follows the last byte with the Frame Check Sequence followed by the closing FLAG. The micro then checks the block delimiter and the transmitter status (to make sure no transmitter under-run occurred). In the event of error the CAMAC processor is alerted via the L signal and the CAMAC output register. Of course CAMAC loading of the data FIFO may be taking place while data transmission is occurring.

It is interesting to note that the transmit and receive FIFOs tend to act in an opposite manner. That is, the receive FIFO tends to run partially full, and the transmit FIFO tends to run partially empty. This is as it should be in order to keep the communication line efficient and buffered from the CAMAC processor.

It is emphasized that the operations described above must be done concurrently. Thus monitoring of the status register must organize the tasks in the proper sequence so that they are completed in the required time. At 48 KPBS for example, the received data byte must be read within 160 μsec after the character is assembled, and the transmit byte must be sent to the MPCC within 160 μsec after it is demanded. End of block, or beginning of block can of course occur at any time. It is interesting to note that when the closing FLAG is detected, there is at least a 3 byte delay (480 μsec) before the first byte of the next frame is detected. Some estimates of the Z-80A timing to accomplish the basic byte handling tasks are shown in Appendix D. Optionally, one can respond to status changes with an interrupt generated by an OR function of the status bits.

Some questions have been raised concerning the use of the MPCC to do error generation/checking, since there is no longer an end-to-end check in the CAMAC processors of the on-line message. Where this is of concern it can be handled in two ways:

(1) Include an error check in the information field which can then be generated/checked in the higher level CAMAC processor.

(2) Disable the error generation/checking of the MPCC chip in the initialize routine. In this case the CAMAC processor has the burden of this task, but at the same time you achieve better overall error

control. On the other hand, there is now only a one byte delay between blocks instead of the 3 bytes mentioned above.

6. OTHER PROTOCOLS

Many other protocols can be operated with this module with the MPCC in the byte mode. Due to the variability of these protocols it is not the purpose of this paper to discuss the operation of these. The processor may have various tasks in byte protocols, but the common element is that the micro must determine the end of block in receive mode. The module will be able to handle many variations of byte protocols, but the maximum operating rate depends upon the processing load of the microprocessor.

In one case of CDC format the block length is 1050 bytes, but the mode is half-duplex, and the required rate is up to 4.8 KBPS. In this case the block will be divided into three shorter blocks, with an interrupt and CAMAC block transfer for each of the sub-blocks. A bit in the control word will be used to indicate when the block is finally terminated.

7. PROGRAM LOADING

It is planned that the module contains the program to handle one line protocol at a time. In order to change this program, or to initialize the module from a cold start it is suggested to download the module from the CAMAC processor via the transmit data FIFO. Starting from the reset state, Z.S2 or N.F(25).A(15), the module typically awaits commands from the CAMAC input. If a program load is specified, then the micro enters a PROM boostrap routine which is initiated by the transmit control FIFO after the transmit data FIFO is loaded. This routine loads RAM with the data bytes from the transmit FIFO in sequence. When this is complete the micro signals the CAMAC processor using the CAMAC output byte which raises a LAM. This process continues until the entire program is loaded, which may take several such cycles, since one can load only 640 bytes at a time.[4] When the process is complete a similar procedure can take place using the receive data FIFO to re-transmit RAM contents to the CAMAC processor for verification. Note that this procedure also verifies proper operation of the four FIFOs, the micro, and the CAMAC input-output registers.

After this procedure is successfully completed, operation of the proper program is initiated by sending a program starting address via the CAMAC input register. Of course, once the a protocol handling program has been debugged, it is desirable to eliminate the start-up procedure, and avoid the possibility of losing the program through error. In this case, EPROM can be used for the program. However, it is required to retain some RAM for the stack and working registers where these are being used.

8. TESTING FEATURES

A complex module of this type demands self-test capability, necessary to verify proper operation in a larger network, and to assist diagnosis in debugging and repair. The very operation of loading and verifying described above already suggests a partial self-test. Further tests are to be provided, most likely in PROM for immediate access.

[4]This figure is reduced to 348 bytes for the prototype.

In development, it is possible to test the CPU board as an entity. A first test verifies operation with a VDU echo check. This checks the VDU, the connections, the UART, and certain elements of the processor. A second and very important test verifies the RAM by writing and reading every location using a checkerboard or similar pattern which exercises every bit. The VDU is used to control and display results of the test. A further memory test is "static hold" in which a pattern is written into memory and read back for verification at a later time, perhaps several seconds to minutes. Semiconductor memories sometimes exhibit the disturbing failure of losing data after a short time.

Self-tests to be performed on the complete module under software control include:

(1) Write known pattern into receive data FIFO, read out via CAMAC and verify.

(2) Same on receive control FIFO.

(3) Load known pattern into transmit data FIFO and use micro to confirm it.

(4) Same on transmit control FIFO.

(5) Exercise MODEM controls, read back MODEM status using a test plug on the V.24 connector.

(6) Verify CAMAC input and output registers by writing from CAMAC into the CAMAC input register and echoing this on the output register (to exit this routine a special code will have to be used, such as all "ones").

(7) Finally, a complete test in which the serial output and input of the MPCC chip are tied together, and a complete block is sent from

the CAMAC processor, sent and received by the MPCC, read out and verified by the CAMAC processor against the original message.

This list is not meant to be complete, but simply indicates the type of tests that can be performed to debug, verify operation, and diagnose faults.

9. SOFTWARE DEVELOPMENT

One of the keys to success of this project is the ability to develop and debug software. Thus it is important to accomplish as much software development on the local computing center as possible. Assembly of Z-80A code using the computing facility is an obvious requirement. This assembly should go as far as possible in providing feedback to the programmer. It is also quite important to have a software simulator available which can then run the machine code and thereby provide even more feedback to develop proper software. Of course, how far one can go with a simulator of this sort is debatable, since it obviously cannot run with real machine timing or I/O. However, the closer the simulation approaches the ideal, the faster and easier checkout will be when it runs in the actual module.

It is a further requirement that code developed on the local computer should be downloaded into the module via CAMAC.

Interactive routines are necessary which will permit an operator using the VDU, when connected, to display locations and areas of memory, and to modify particular locations, as well as initiate the program at desired memory locations. During the course of hardware development it is necessary to produce various EPROMS to exercise portions of the system.

These programs should be produced on the local computer and directly downloaded into EPROM. Facilities for reading and verifying EPROMS should also exist.

In the event that proper facilities on the local computer cannot be provided or depended upon, then a development system should be strongly considered. The ZDS-1/40 and Futuredata are standalone disk-based systems with software support and real time emulation for checkout of the user system.

10. ACKNOWLEDGEMENTS

The development of this module has been a collaborative effort of many people at Daresbury. These include J. Alexander, D. Hines, P. Kummer, H. Kirkman, E. C. G. Owen, A. C. Peatfield, M. Powell, and I. Smith. In particular, the basic concepts of using the FIFOs, and the nature of the LAM structure were suggested by A. C. Peatfield. I. Smith and D. Hines provided the preliminary timing analysis — Appendix D. D. Hines developed the Z-80 assembler, and M. Powell wrote the initial testing programs and the first operating program. P. Kummer provided Requirements — Appendix A.

The author wishes to acknowledge the efforts of co-designer J. Davis, who was especially helpful throughout this project.

Construction of the prototype was handled efficiently by the Daresbury Electronics Shop under K. Evans.

APPENDIX A

MODEM DRIVER MODULE FUNCTIONAL REQUIREMENTS

Required:	HDLC (X.25, level 2)
	Checksum calculation/checking
	Elimination of synchronism
	Framing (flag detection)
	Error status reporting
	Cheap, simple module
Useful:	Byte oriented protocols (requires intelligence in module)
Note:	The HDLC chip does all of "required" except elimination of synchronism.

Recommendations

1. Byte protocols should only be included if the buffering required to eliminate synchronism needs a micro to organize it sensibly.
2. The module should be capable of handling I-frame lengths of 278 bytes.
3. The number of buffers required is:

2 input data buffers	(278 bytes each)
3 input control data buffers	(4 bytes each)
2 output data buffers	(278 bytes each)

Note: This is a minimum requirement — more would be nice but not essential.

Speed

48	KBPS	HDLC	(full-duplex)
9.6	KBPS	Byte	(full-duplex)

Byte Protocols

HASP	upto	430 bytes/block	half-duplex, i.e., only 1 buffer required.
(BSC)			upto 9.6 KBPS
CDC	upto	1050 bytes/block	half-duplex
			upto 4.8 KBPS

Prototype

Using FIFO buffers then a prototype module with at least 320 bytes in each direction will be of use for:

— our present communications protocol (Daresbury)

— testing of X.25 level 2.

APPENDIX B

SOME GENERAL COMMENTS ABOUT THE DIFFERENCES BETWEEN THE 8080A, 8085A and Z-80A MICROPROCESSOR CHIPS

(1) The Zilog Z-80A is 100% upwards software compatible with the INTEL 8080 family; all 8080 code will run on the Z-80A.

(2) Z-80A clock rate (4 MHz) is twice as fast as "conventional" 8080A, although not as fast as the newest 8085A-2 (5 MHz). Instruction execution times range from 1 μsec to 5.5 μsec.

(3) Z-80A bus has 8 bit bidirectional data, 16 bit address, and data bus control signals directly from the chip. The 8080A requires another chip (8228) to generate some control signals; the 8085 has a partially multiplexed address/data bus. Thus the Z-80A appears to have the simplest, most easily used bus.

(4) Z-80A requires single phase TTL clock; 8080A requires two phase clock, generally supplied with another chip (8224); 8085 is fully integrated and requires only a crystal.

(5) Z-80A requires only +5v d.c. supply while 8080A requires +5v and -5v and +12v d.c.; 8085 requires only +5v d.c.

(6) The 8085 has a serial I/O port. Neither the 8080A or Z-80A has this capability integrated into the chip.

(7) The Z-80A has built in dynamic RAM refresh circuitry. The other two do not.

(8) The 8085 instruction set is the same as the 8080A except for two additional instructions associated with an expanded interrupt system.

(9) The Z-80A has <u>two</u> sets of internal registers, for the A, B, C, D, E, H, L and F registers. These can be interchanged with two instructions in 2 μsec. This operation is quite useful in interrupt handling.

(10) The Z-80A has two 16 bit index registers not present in the 8080A or 8085. These permit indexed addressing from a base address. Relative addressing (relative to the program counter) is also provided in the Z-80A for jump instructions, but not in the 8080A or 8085.

(11) The Z-80A has instructions for independent setting, resetting and testing of bits in any of the registers, or in memory. These instructions are not available in the 8080A/8085. They could be quite useful in the communications application, or in any general control environment.

(12) The Z-80A has some interesting "block move" and "block search" instructions not available in the 8080A/8085. In "block move" bytes are transferred from/to sequential bus addresses (blocks). They can be executed singly or automatically repeated until $BC = 0$ (BC = word count). In "block search" sequential bus addresses (blocks) are compared to the byte in the accumulator. Again this can be executed singly or continuously until $BC = 0$. The Z flag = 1 if comparison is achieved.

(13) Similar "block modes" for input and output are available in the Z-80A. The 8080A/8085 input/output is based upon single 8 bit transfers to/from the accumulator with an 8 bit address code on $A_0 - A_7$ (multiplexed in the case of the 8085). The Z-80A has this

mode, of course, and in addition can move data to/from sequential memory addresses either singly or repeated until $B = 0$, with an 8 bit address code on $A_0 - A_7$.

(14) Interrupt systems of all three chips are somewhat different, with the Z-80A probably being the most flexible.

(15) All three devices have bus signals to "hold" the processor for slower I/O or memory. All three also have bus request signals to "tri-state" address, data and control, in order to permit DMA and other devices to gain access to the bus. Differences between the devices in this area seem to be relatively minor, although terminology is quite different.

(16) This comparison is not intended to be complete, objective or unbiased.

APPENDIX C

RAM BASED FIFO

Due to the low capacity of available FIFOs compared with RAMs some work was done on the design of a FIFO based on fast 1K × 4 RAMs. Using four such chips one can design an equivalent FIFO function with a capacity of 2K bytes. This is possible since the maximum dataway speed of 1 MHz permits interleaved read-write cycles in a fast RAM (200 nsec). The general approach is shown in Fig. C-1 for the receive data FIFO. Although the design of the transmit data FIFO is similar it is recognized that a complication is added by the functional requirement of generating a LAM when the transmit FIFO is ready to receive another block or frame.

APPENDIX D

PRELIMINARY TIMING ESTIMATES FOR X.25

The flow chart (Fig. D-1) shows a preliminary estimate of Z-80A timing to handle both a transmitted byte and a received byte in X.25 mode. This work assumes:

(1) MPCC-chip status is read in a system status byte.

(2) All applicable registers are memory mapped.

(3) FIFO ports are each separate memory addresses (each FIFO byte has a separate memory address).

(4) 16 bit CAMAC words.

(5) 4 MHz Z-80A operation.

Timing of other operations such as end of block, beginning of block, are not included in this preliminary estimates.

The flow chart indicates that it takes 106.5 μsec to handle both bytes. At the required synchronous speed of 48 KBPS each byte must be serviced in 167 μsec in order to avoid receive over-run or transmitter under-run errors. It appears likely from this estimate that the module will handle X.25 at 48K BPS.

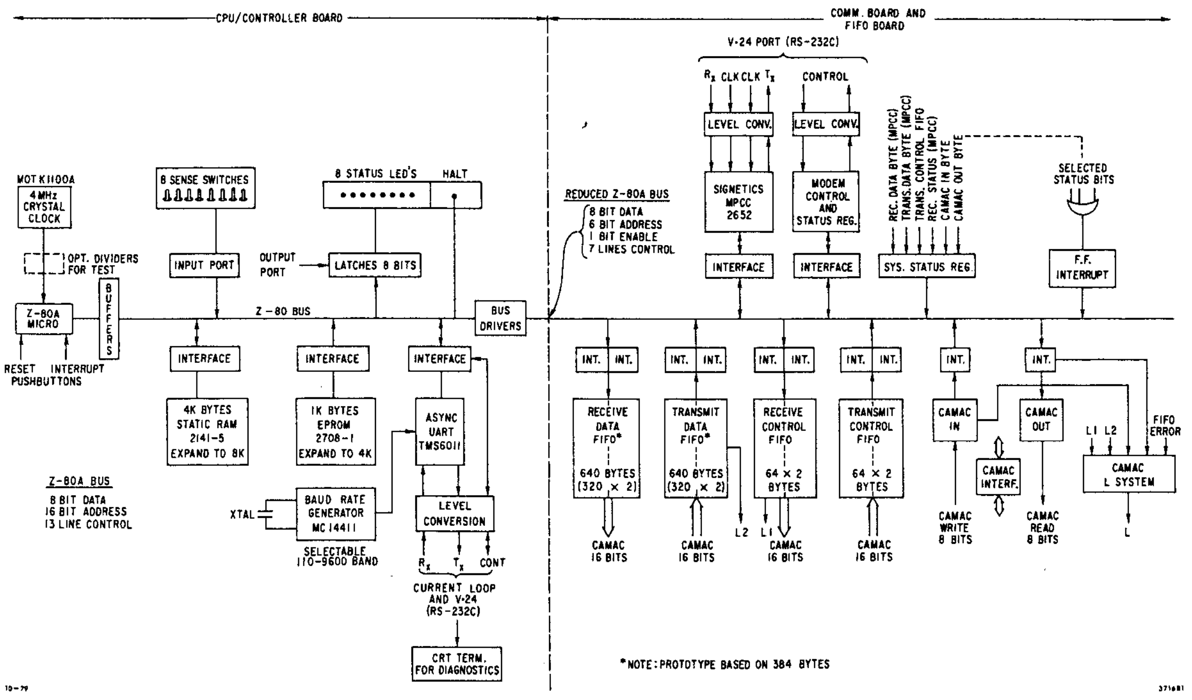


Fig. 1. Block Diagram.

COMMAND	DESCRIPTION	BITS	X	Q
N.F(0).A(0)	Read Rec. Data FIFO[1]	16 or 8	1	Data Avail.
N.F(16).A(0).S1	Write Trans. Data FIFO[1]	16 or 8	1	Space Avail.
N.F(1).A(0)	Read Rec. Control FIFO[1]	16	1	Data Avail.
N.F(17).A(0).S1	Write Trans. Control FIFO[1]	16	1	Space Avail.
N.F(1).A(I)	Read CAMAC Output Reg.[2]	8	1	Data Ready
N.F(17).A(1).S1	Write CAMAC Input Reg.[2]	8	1	Reg. Ready
N.F(25).A(15).S1	Initialize Module, Clear All FIFOs, Reset Z-80A,[3] Clear Interrupts where applicable, Set LAM Masks to "0".		1	--
Z.S2	Same as N.F(25).A(0)		1	--
N.F(1).A(14)	Read LAM Req. Reg.	5	1	0
N.F(1).A(13)	Read LAM Mask Reg. "1" = Enable	5	1	0
N.F(17).A(13).S1	Write LAM Mask Reg. "1" = Enable LAM	5	1	0
N.F(1).A(12)	Read LAM Status Reg.[2] "1" = Error	12	1	0
N.F(8).A(15)	Test L		1	L State

Notes:

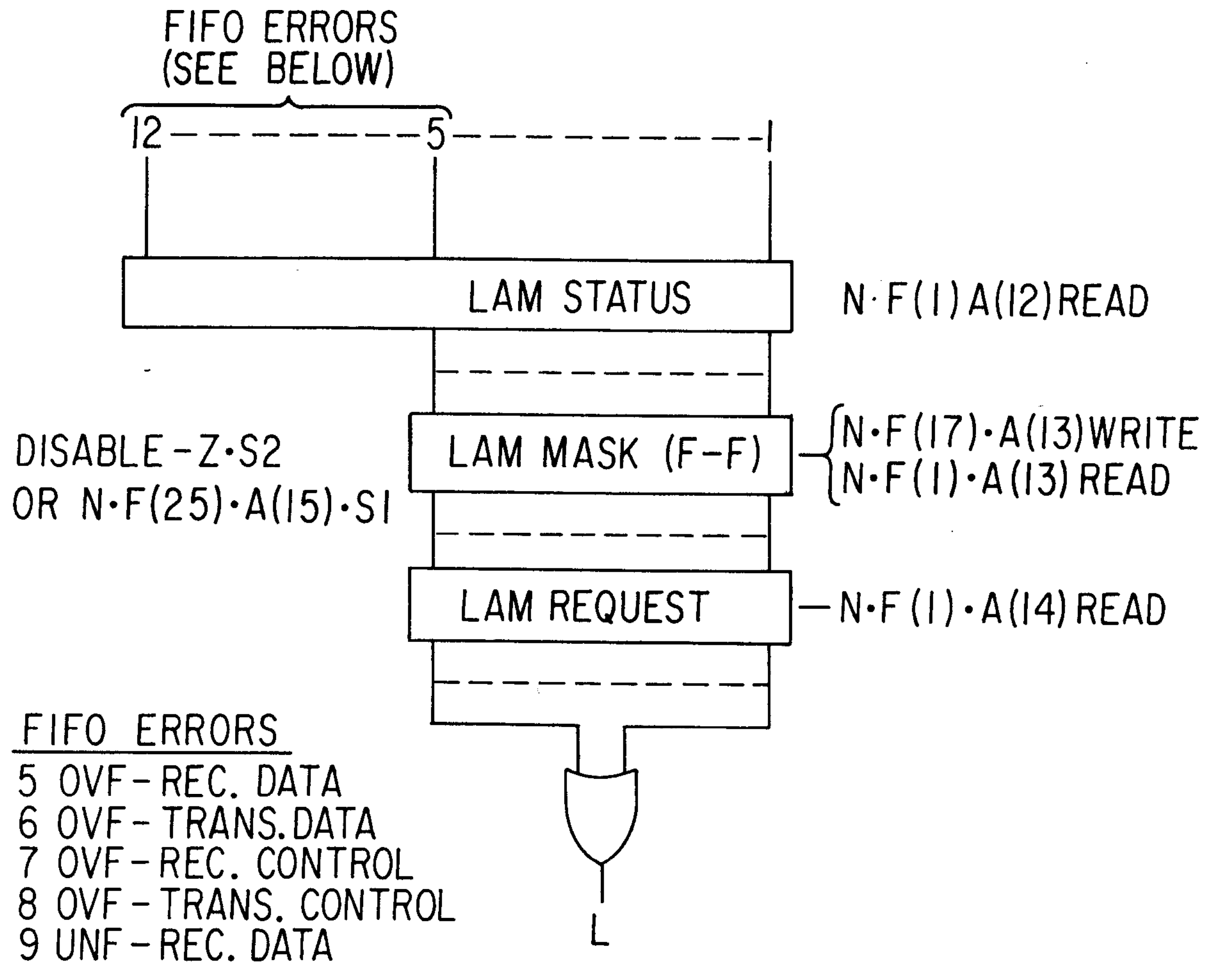
(1) These operations are CAMAC single address block transfers. FIFO clocks on S1 (Write), S2 (Read).

(2) This operation clears the respective LAM Status on S2. In the case of N.F(1).A(12) it clears only the error bits (Bits 5-12).

(3) Micro executes from memory location 0 after a reset.

(4) Prototype module commands slightly different.

Fig. 2. CAMAC Commands.



FIFO ERRORS

5 OVF-REC. DATA
6 OVF-TRANS.DATA
7 OVF-REC. CONTROL
8 OVF-TRANS. CONTROL
9 UNF-REC. DATA
10 UNF-TRANS. DATA
11 UNF-REC. CONTROL
12 UNF-TRANS. CONTROL

OVF: ATTEMPTED WRITE TO FULL FIFO
UNF: ATTEMPTED READ TO EMPTY FIFO

LAM REQUEST BITS

1 REC. CONTROL FIFO HAS WORD (2 BYTES)
2 TRANS. DATA FIFO HAS SPACE
3 CAMAC INPUT BYTE READ BY MICRO
4 CAMAC OUTPUT BYTE LOADED BY MICRO
5 ERROR IN FIFOS ("OR" OF STATUS BITS 5-12)

10-79
3716A2

Fig. 3. CAMAC Interrupt-(L) System.

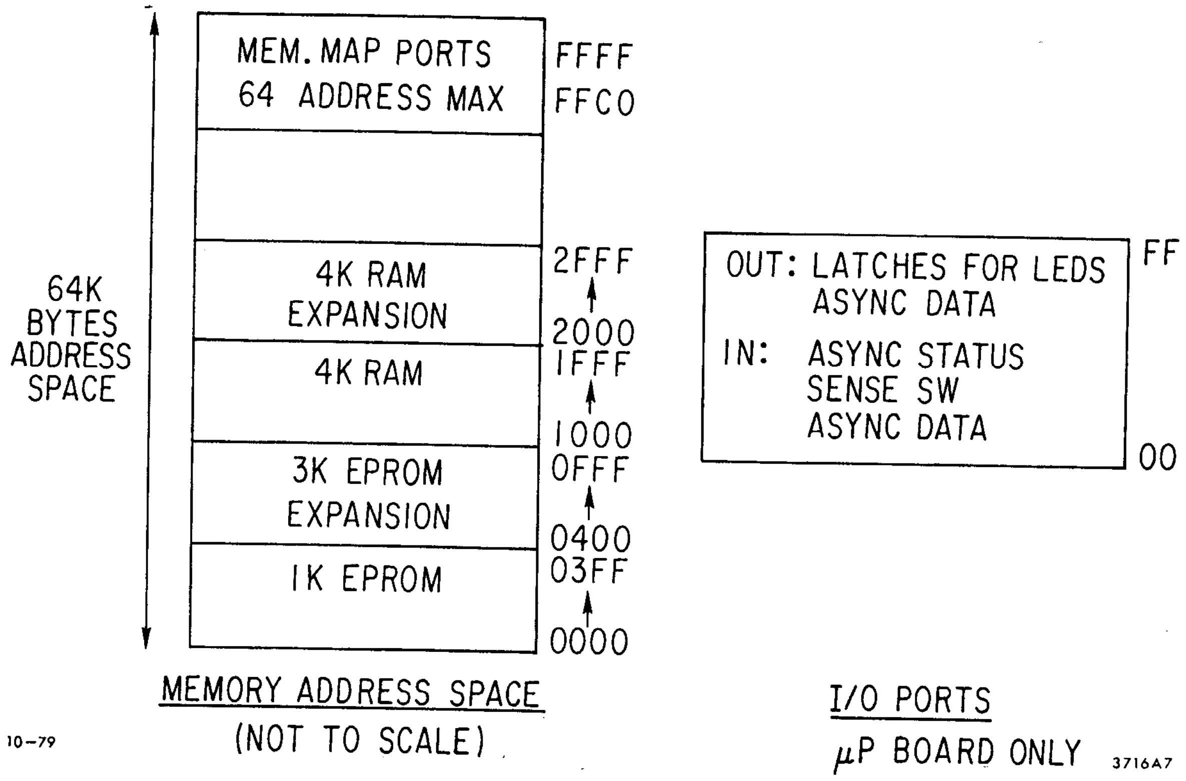


Fig. 4a. Memory Map.

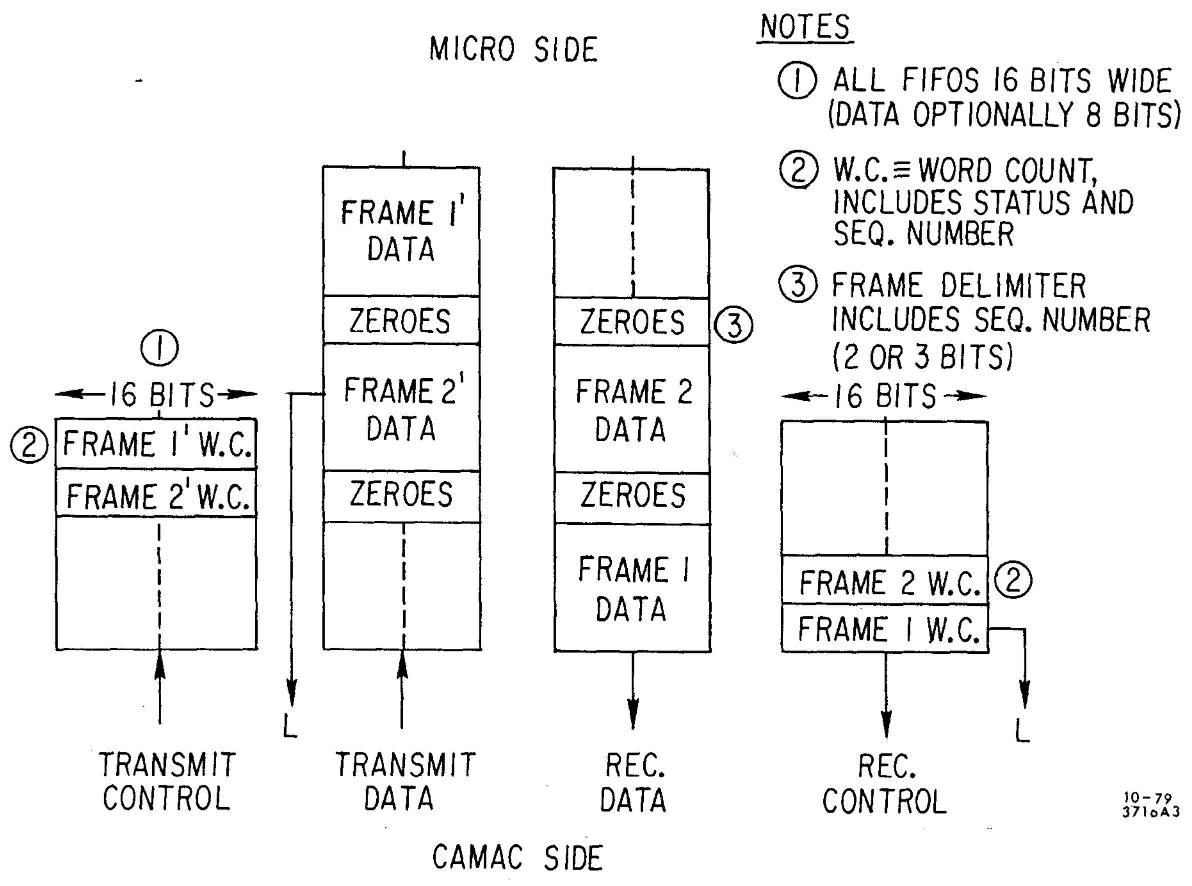


Fig. 4b. FIFO Map (Example).

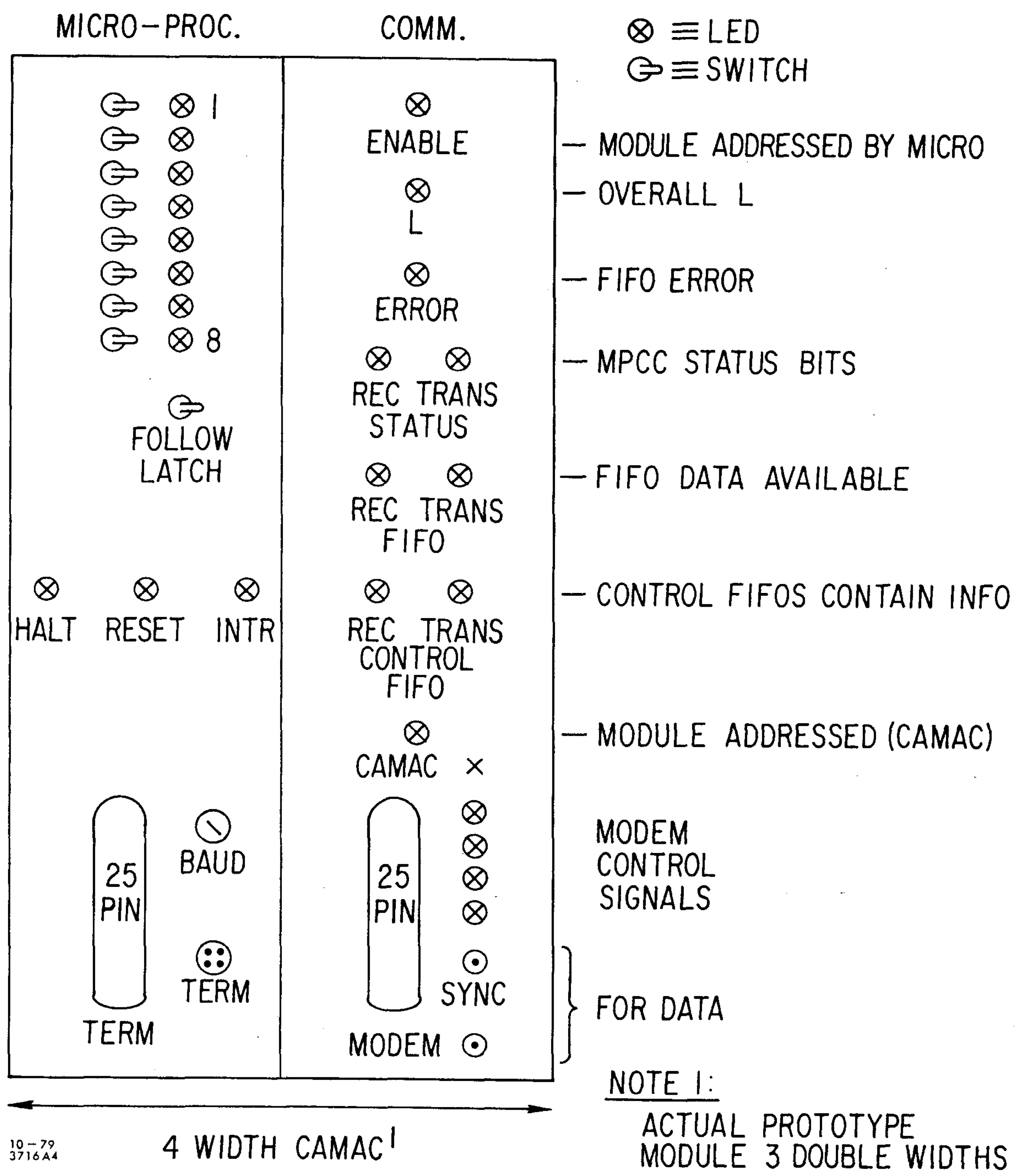


Fig. 5. Prototype (Wire-Wrap) Module Front Panel.

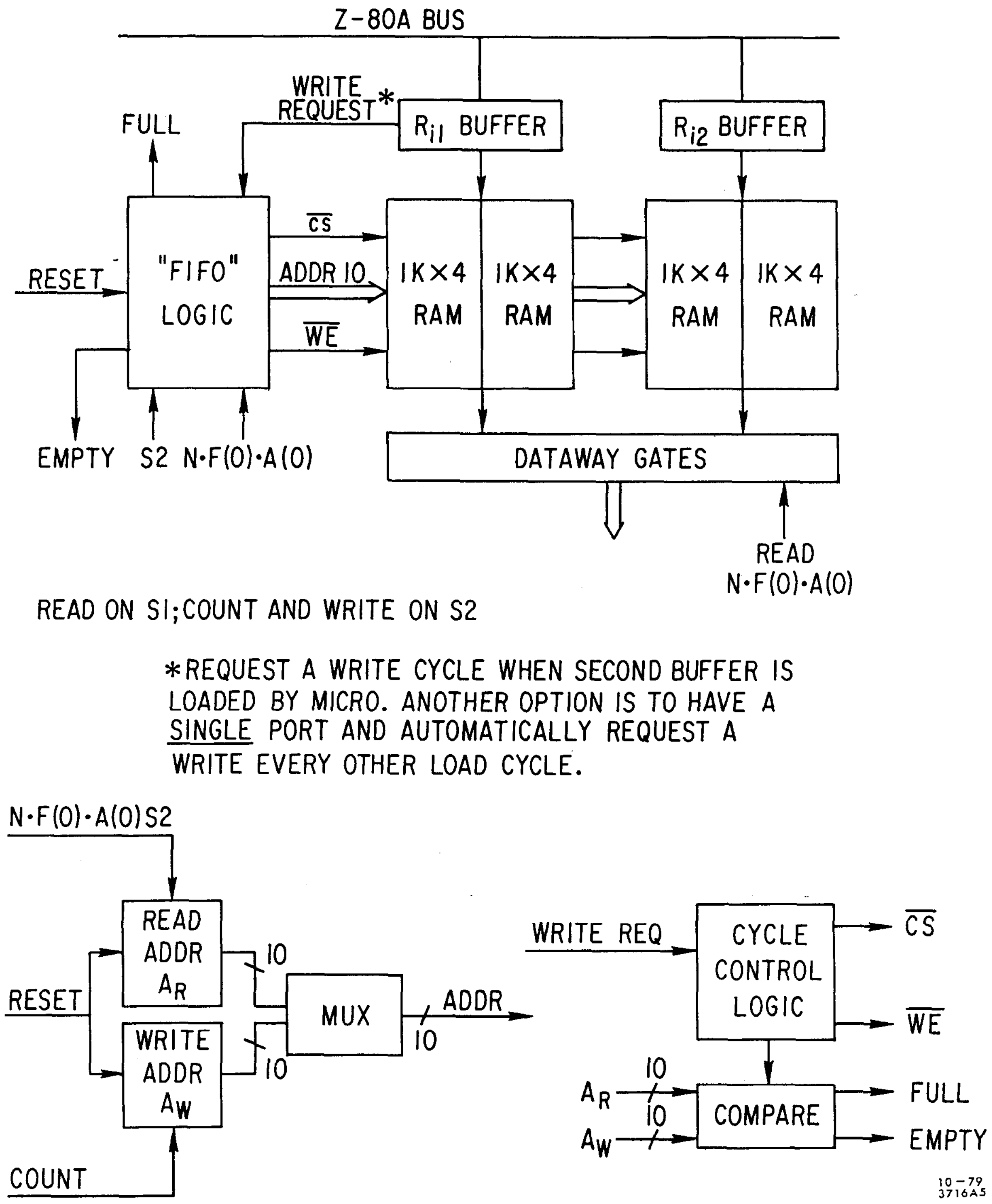


Fig. C-1. RAM Based FIFO (Receive).

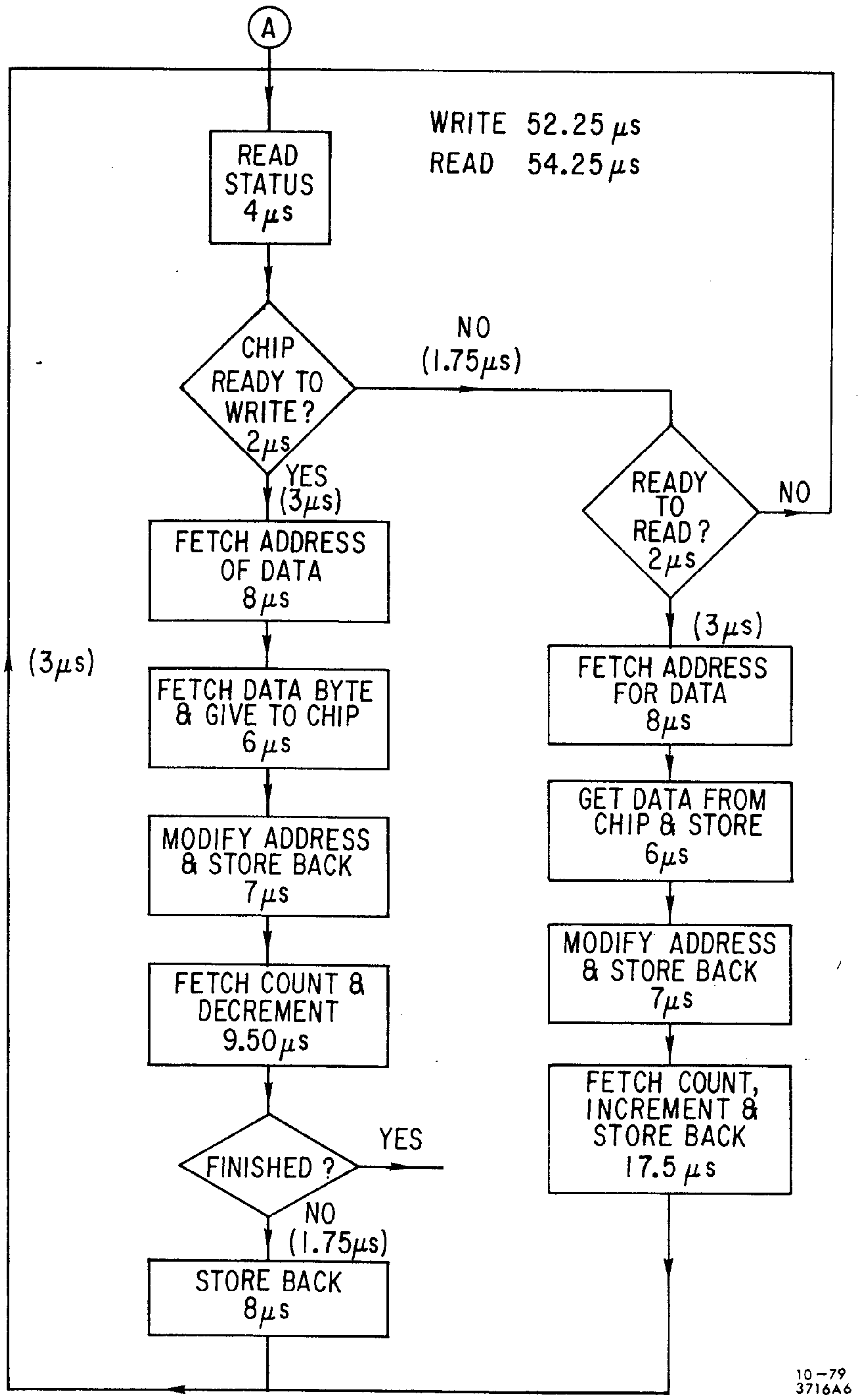

A
WRITE 52.25 μs
READ 54.25 μs
READ STATUS 4μs
CHIP READY TO WRITE? 2μs
NO (1.75μs)
YES (3μs)
FETCH ADDRESS OF DATA 8μs
FETCH DATA BYTE & GIVE TO CHIP 6μs
MODIFY ADDRESS & STORE BACK 7μs
FETCH COUNT & DECREMENT 9.50μs
FINISHED ?
YES
NO (1.75μs)
STORE BACK 8μs
(3μs)
READY TO READ? 2μs
NO
(3μs)
FETCH ADDRESS FOR DATA 8μs
GET DATA FROM CHIP & STORE 6μs
MODIFY ADDRESS & STORE BACK 7μs
FETCH COUNT, INCREMENT & STORE BACK 17.5 μs
10-79
3716A6

Fig. D-1. Flow Chart — Read and Write Byte.