SUMMARY

We have designed an analog data collection and control board. It was designed to be connected to the S100 bus. Software was written to use the features of this D/A and A/D converter.

The buses and standards described are intended to make the job of interfacing easier. To plug the device into a system with no extra work is every interface designer's dream. We have seen how the many users of the S100, CAMAC, IEEE-488 and EIA-RS232C standards create a large need for standard-compatible devices, modules, and systems. If at all possible, *stay within a standard.* The design will be easier and your time may be spent on the harder problems.

Parallel and serial bus standards, methods of communication between modules, and an actual bus interface example were presented. The S100 bus is the most popular parallel bus used now, with over 600 different types of compatible boards being produced. The serial RS232C standard is the most popular standard for data communications, and versions of data formatting are used, with modems, to store and retrieve data from cassettes and cartridges, as described in Chapter 4.

Power supplies are the heart of a system. Regulation, stability, and some design parameters have been discussed. The OEM solution is obviously one of the best, as the power supply manufacturer is a specialist in regular and custom supply requirements.

LIE: 21-45

LESSON NO. MIT-7

7(22) ⊗

THE MULTIPLEXER — A CASE STUDY

INTRODUCTION

This system is intended to concentrate 32 EIA RS232C-compatible terminals onto a single two-way high-speed transmission line. Each terminal has buffered output and character-by-character input. Thus, the host computer can spend less time executing the multiplexing task.

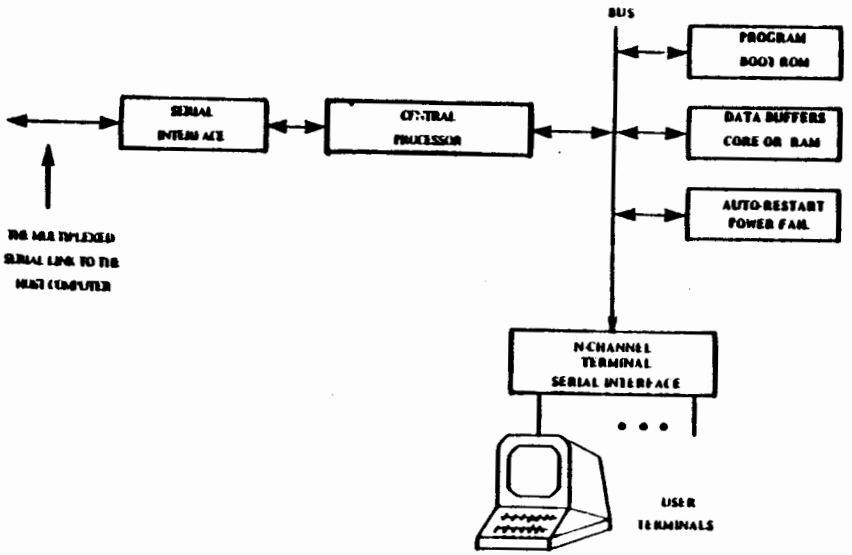


Fig. 7-0: A 32-Channel Multiplexer

Designed for a PDP 11/70, the system is also applicable, with only code changes in the host machine, to almost any host computer. The cost of providing this function is $50 per channel, as compared to usually around $250 per channel. The system is also cost-effective in clusters of fewer than 32 terminals.

The system uses the 8080 microprocessor, 8251 USRT, 8259 interrupt controller, and other components in the 8080 family. The system has no modem-control features, as it was intended to be at the site of the terminals, saving even more money in man-hours of time and cost of wire for connection. This does not even include the cost benefit of fewer telephone lines and modems.

THE SPECIFICATIONS

The task of connecting a large number of terminals to a time-sharing facility always presents the engineer with a number of problems. Most have to do with the interconnection headaches of modems, telephone wiring, patchboards for testing, and internal machine interfacing.

Remotely-located concentrators would eliminate many problems. The new problem: cost. The design goal here is to service 32 terminals at an input rate never exceeding 30 characters-per-second, and an output rate as fast as possible. Given that the 8080A could execute roughly 300 instructions in the time between characters at 9600 baud, if it were to service 32 terminals on input, it would have to have fewer than 300 instructions in the polling loop for the terminals. Any time left over would be used for output. The code would have to be thought out byte-by-byte, with all coding being carefully optimized. A prototype was built, under the assumption that it could service at least 16 terminals in a degraded mode.

The typical statistics of our input was a maximum of 150 baud for any second, and a rate of 50 baud for all 32 terminals combined. Thus, when completed, the multiplexer could handle a maximum of 150 baud on all 32 at once, or a maximum of 300 baud on one. The output was a minimum of 300 baud for all 32 at once, and typical 6000 baud when there was a specific demand from a single user.

ARCHITECTURE

The architectural block diagram is presented in Fig. 7-1. Each terminal has its own USART, because each needs a dedicated serial interface. The USARTs are grouped into fours and then placed onto cards, which are on the 8080A system bus. There are 8,192 bytes of RAM for data storage, and 1,024 bytes of EAROM for program, in the system. Lastly, there is an interrupt controller and high-speed-channel card, which is on the bus.

Each terminal, through its USART, has a 128-character buffer associated with it, for buffering output to the terminal. This takes 4,096 bytes of the available RAM. The terminals-to-host queue is 256 characters long. These lengths were chosen to optimize the communication-channel transfers. The method will not be discussed here.

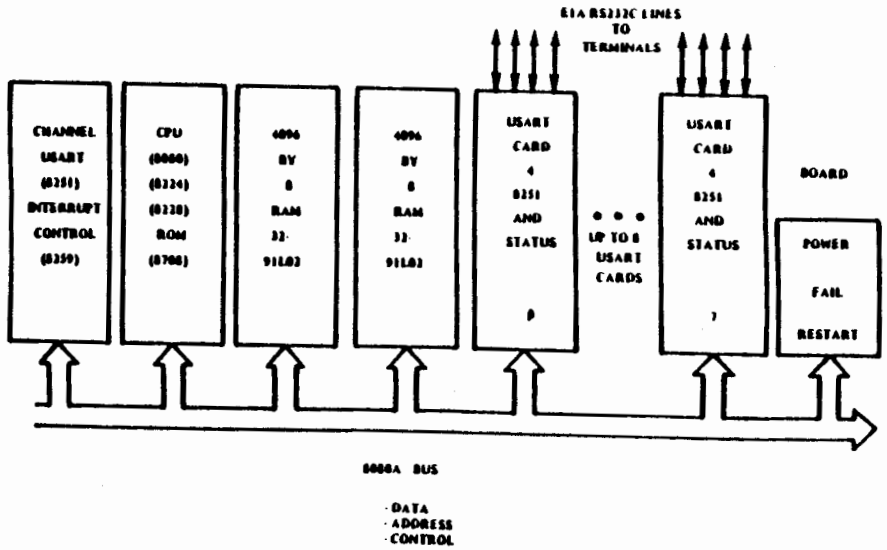


Fig. 7-1: Multiplexer Block Diagram

There are three processes, running one at a time: input-output service polling routine, host-to-terminal buffer interrupt process, and terminal-holding-queue to host interrupt process. They will be described in the following section.

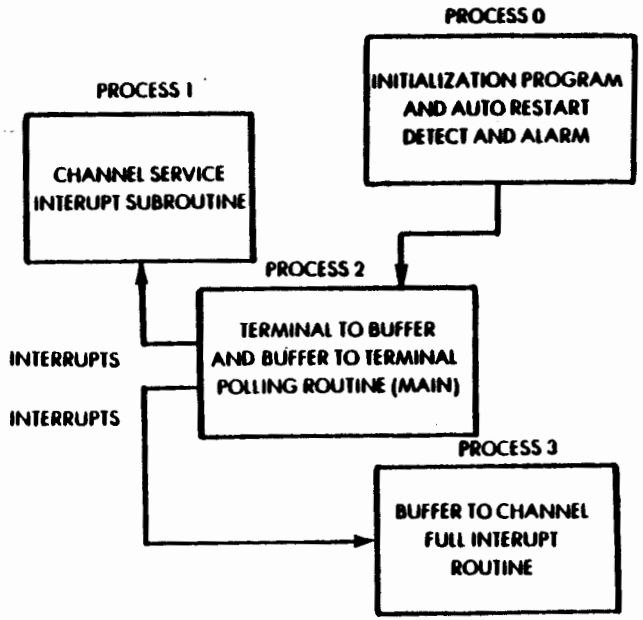
SOFTWARE

A flowchart of the software appears in Figs. 7-2, 7-3, 7-4 and 7-5. The software can be divided into four parts: the initialization routine,

the polling routine, the interrupt routine to fill terminal buffers from the host, and the interrupt routine to empty the terminal-to-host waiting queue.

The initialization runs only when reset, then the latter processes may run, one at a time. They communicate only through the output data buffers and share no other common memory space, other than pointer tables.

The initialization routine clears all memory, sets up tables, finds which boards are plugged in, resets all USARTs, and will print out errors, if a debug board is installed. This is roughly all the system housekeeping. It sets the stack pointer, resets and sets the mode, speed, and number of bits-per-word on the USARTs. This section of the program is 60% of the code used for the whole application.



TOTAL 8080A BYTES FOR PROGRAM: 526 BYTES! LESS THAN ¼ OF THE 2708 USED

Fig. 7-2: Multiplexer Software: Overall Program Flow

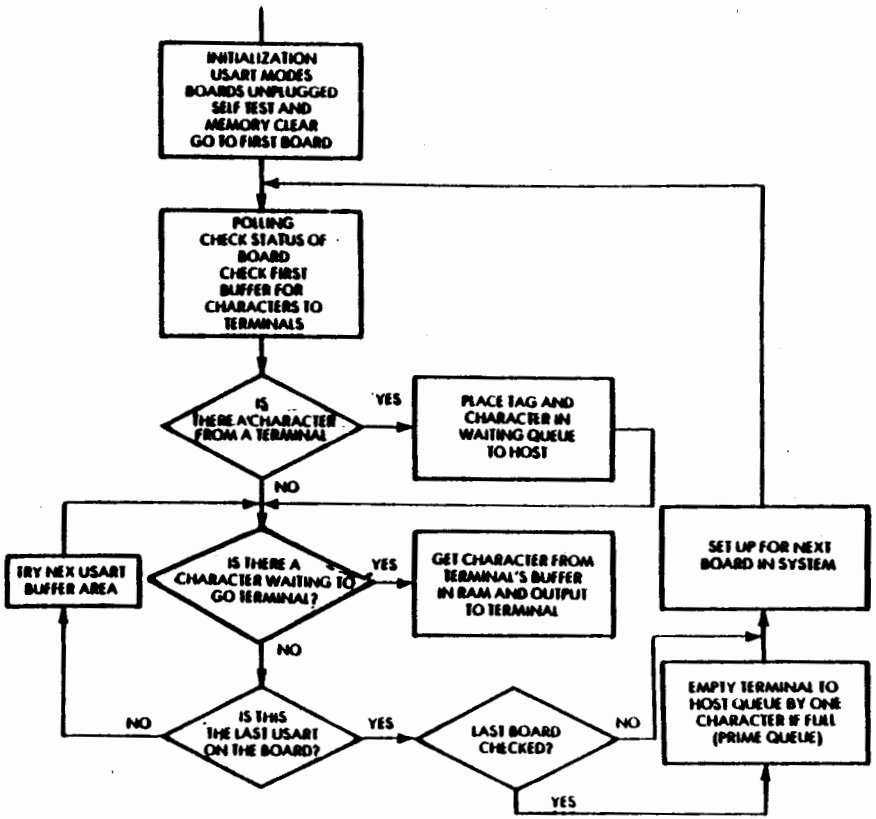


Fig. 7-3: Multiplexer Software: Polling Loop

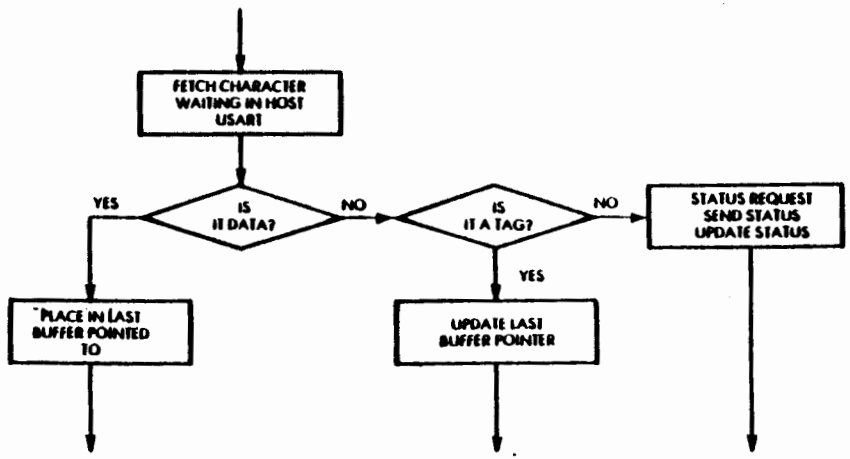


Fig. 7-4: Multiplexer Software: Host to Mux Interrupt

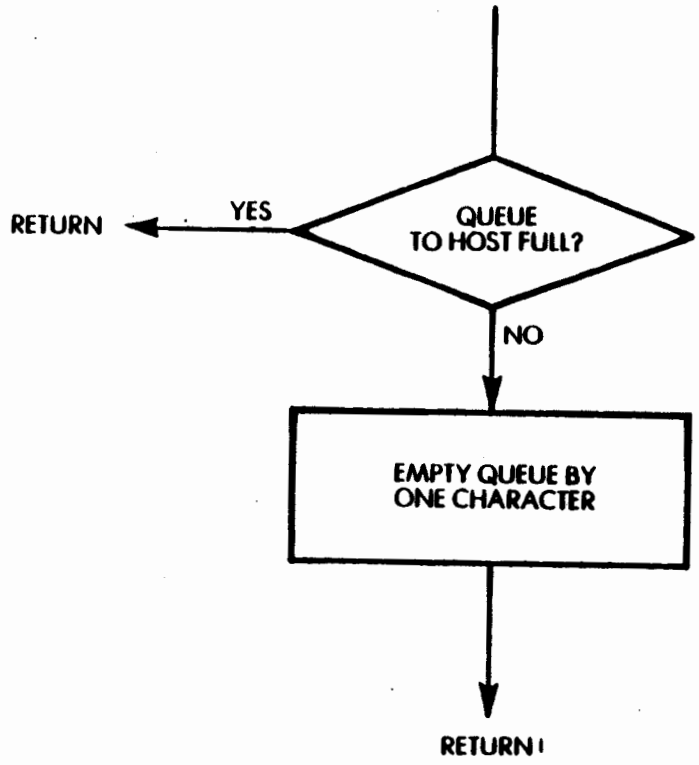


Fig. 7-5: Multiplexer Software: Mux to Host Queue Interrupt

The polling routine goes through the list set up by the initialization program, testing to see if there has been a character typed by a terminal, or if there is data in a buffer, to be output to a terminal. Thus, each of the 32 terminals is serviced once during each pass. If the channel-to-host is busy (it takes 1 millisecond to transmit a character at 9600 baud), the characters are put into a waiting queue that will be serviced when the "channel-not-busy" interrupt comes in. If the channel is not busy, the waiting queue is emptied by one character, and the character currently waiting is placed at the end of the line, in the queue. In this way, the queue-service routine is primed and will continue to interrupt, when not busy, to empty all the characters waiting for the channel. The format used for data transmission is the following: the tag for that terminal is sent first, and then the character is sent to the host, via the queue routine. Each board has its own priority table, so that only one input is processed, per pass, per board. After a character is transmitted, or, if a board has no characters, the buffer area for each terminal is then checked to find if there is an output character pending (these are placed in the buffer by the host-interrupt routine). If so, the buffer gives its character to the USART to be transmitted, and all the pointers are updated. When there are no incoming characters, and no buffer is full, the system still polls each board for input, and each USART buffer for output.

The channel-queue-interrupt routine looks at the queue, and transmits a character, if there is one waiting; otherwise it returns. This routine will not be called again by interrupt, until the polling routine primes it by sending a character.

The host-interrupt routine waits for information to come from the 11/70, or host machine, before it executes. When a character is received, and ready, an interrupt is generated that then starts this interrupt process. This process checks the incoming character and, if it is data, places it in the appropriate output buffer area. After this, polling resumes. Other characters from the host perform status requests, data-tag-switch, and soft-restart commands.

The host-interrupt routine may interrupt at any time during polling. It first saves the status vector of the machine, then picks up the character that caused the interrupt. If the most-significant-bit (MSB) is a "1", the character is a tag, or a command. If it is a tag, it is stored, so that the following data characters are loaded into the buffer pointed to by the last tag.

The most-significant-bit could also mean that it is a command. The commands allowed are: "status-request," "status-change," and "soft-restart." "Status-request" will send back a status tag followed by the status of that USART. "Status-change" will take the next character, and transfer it to the USART control register. This can be used to turn ports on or off, and change baud rate by a factor of four. "Soft-restart" will reinitialize the entire system. Caution is advised in the use of these controls: do not expect the data buffers to be unaffected by their use! This is because these commands require more time than is allowed to poll all the terminals. Thus, interrupts are locked out and characters may be lost. These commands are usually used to re-intitialize the system from the host, after the host crashes.

The most-significant-bit being "0" means that the character represents data. This character is then loaded into the last place in the buffer pointed to by the last tag. All following characters will load into the same buffer, until a new tag is sent.

The CPU and PROM Module

In Fig. 7-6, we see the 8080 CPU board schematic. This board contains all of the necessary CPU interface circuitry along with one 2708 programmable ROM and the necessary bus buffers.

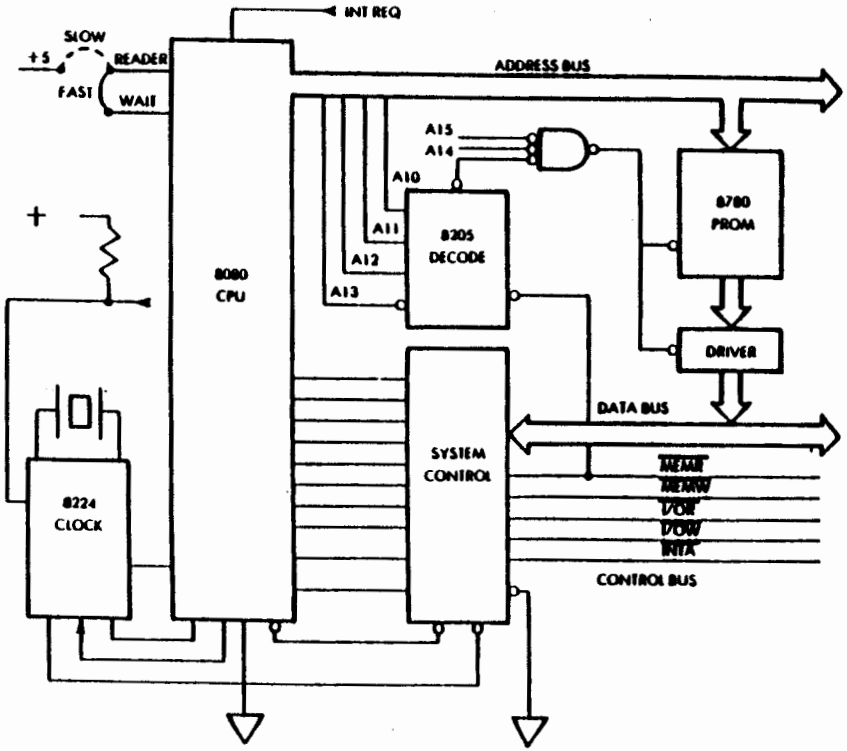


Fig. 7-6: CPU Board Schematic

The 8080 needs a clock and a system controller. These functions are provided by the 8224 and the 8228 chips, respectively. The 8224 provides the necessary timing from the 18-megahertz crystal to drive the two-phase clock of the 8080. It also provides the reset signal synchronization necessary.

The 8228 system controller provides the system with the control bus and also buffers the data bus, so that all of the modules in the system can be driven with no load limitation.

Also on this board are 1,024 bytes of EPROM provided by the 2708. Notice that the selection of this device is fully decoded. The EPROM will only respond to addresses from "0000" hexadecimal to "03FF" hexadecimal. This is where the multiplexer program resides.

The selection is done as follows: all address bits A10 through A15 must be low, to enable the EPROM, as well as the $\overline{\text{MEMR}}$ signal. The first four of those signals, along with this $\overline{\text{MEMR}}$, go to a 1-of-8 decoder, an 8205. If all of these are zero, then the first output is selected. Then this output is checked with the last two address lines. If all are zero, then the $\overline{\text{CS}}$ is held low, selecting the EPROM. The EPROM bus driver, an 8212, is also enabled at this time to drive the appropriate cells' data onto the data bus, to be read by the processor.

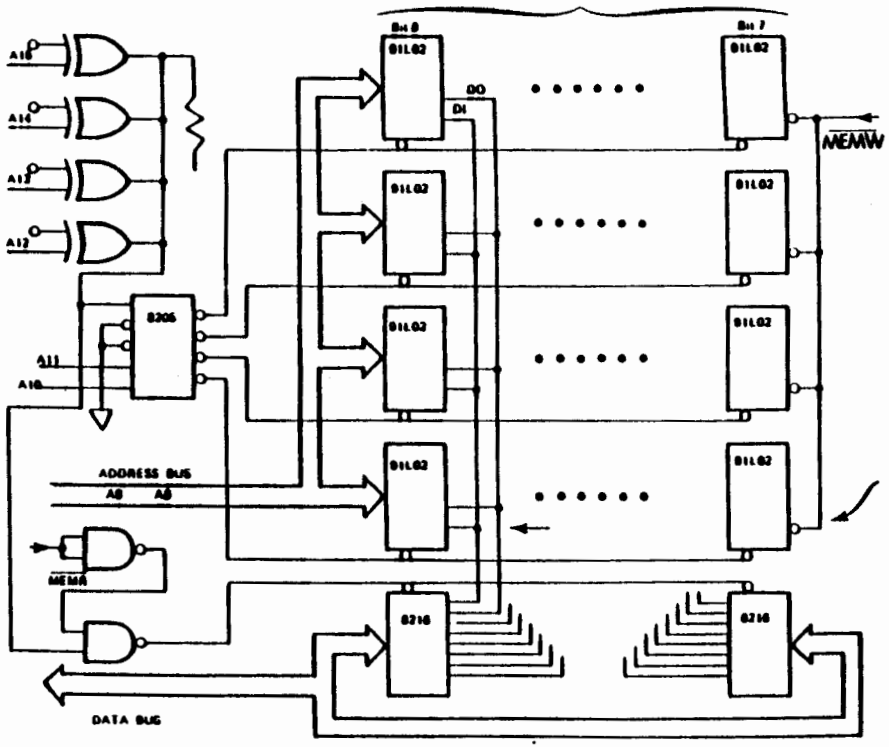


Fig. 7-7: RAM Board

RAM Modules

There are two memory cards in this system. They are both identical, except one is for addresses "1000" hexadecimal through "1FFF" hexadecimal, and the other is for addresses "2000" hexadecimal through "2FFF" hexadecimal. These two cards provide 8,192 bytes of RAM storage.

Each card contains 32 static 1,024 × 1-bit RAM chips, bus drivers and receivers, and address-selection logic.

A single RAM chip can store 1,024 bits of information. In order to store 4,096 × 8 bits, we need to organize these chips into a *memory array*. Note that we need one chip for each bit, and that we need four sets for 4,096 bytes.

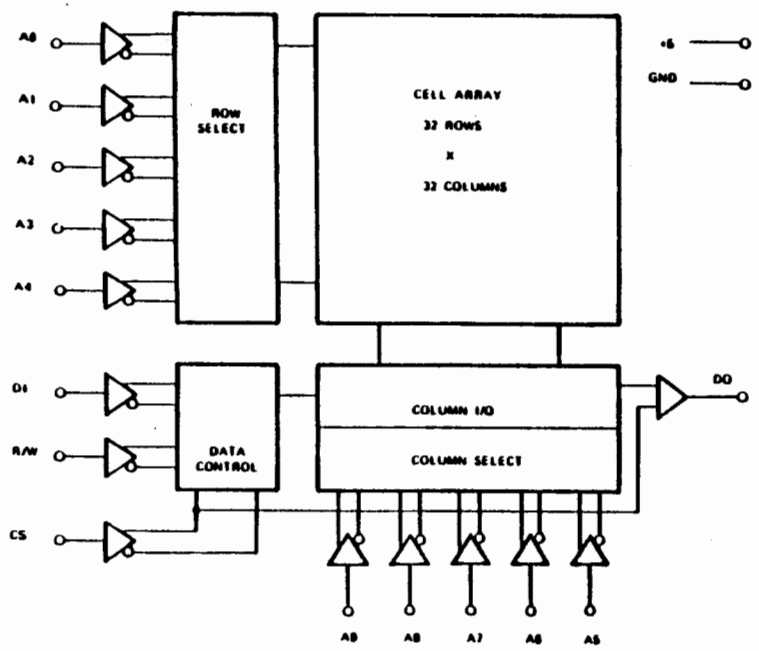


Fig. 7-8: Detail of 91L02C

Since, for any group of 1,024 bytes, eight 91L02s will need to be enabled, the chip-selects for each of the groups of eight are tied together. From there, these four group-selects go to a 1-of-8 decoder.

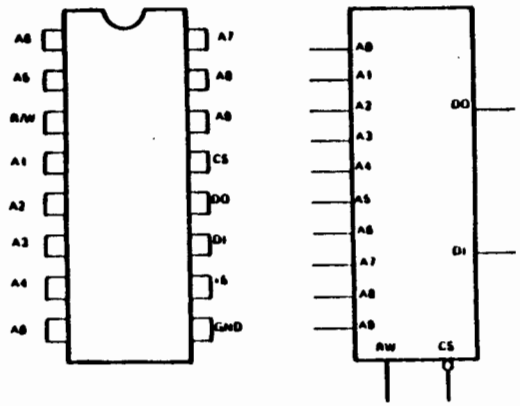


Fig. 7-9: Pinout of 91L02C

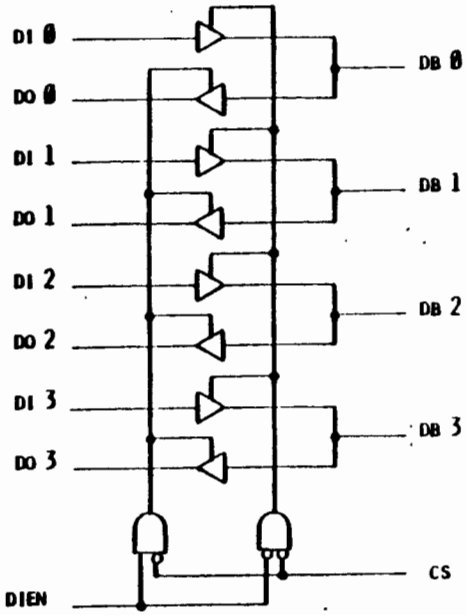


Fig. 7-10: 8216 Bidirectional Bus Drivers

The data bits are bussed from each group in the direction perpendicular to the chip select. All bit 0s should be tied together, as well as bit 1s, bit 2s, bit 3s, etc. Since 91L02's cannot drive the bus directly, all input data lines come from an 8216 bidirectional bus driver and receiver. In a similar fashion, all data outputs from the 91L02s go to the 8216 bidirectional bus drivers. An illustration of the 8216 appears in Fig. 7-10.

Two of these devices will provide a standard method of listening to, and driving, the data bus. The DIEN signal controls whether the bus is driven by the 8216, or whether the bus is listened to. The CS enables the outputs to drive both the bus and the D0 outputs. If CA is high, all of the DB and D0 pins are in the high-impedance state.

The direction of data-flow is determined by the MEMR signal. When it is low, the RAM will put data out onto the DI lines of the 8216s. The bus-drivers will be enabled, to drive the 8080 data bus with this data. At all other times, the memory array listens to the bus. The only time it will write data into the memory is when the MEMW signal goes low and the chips are selected.

The address selection is performed in a way so that the address of the board may be selected by jumper wires. The low ten address bits go directly to the 91L02s. The next two bits go to a 1-of-8 decoder (8205) to select one of the four sets of eight memory chips. The enable line of the 8205 comes from a wire-ANDed combination of exclusive-or (XOR) gates.

Only when all of the outputs from these four gates are high will the memory board be enabled. Each XOR gate compares an address bit with a jumper wired to "1" or "0". If both are identical, the output will be "0". If they are different, the output will be "1". To set these jumpers for the right address, we set the jumper to the opposite of what the high four address bits should be. If we want "0010", for A15-A12, the jumpers should be tied to "1", "1", "0", "1", respectively. In this way, the board will respond only when an address lies in the area of $0010XXXXXXXXXXX_2$. This is pages "20" through "2F" hexadecimal, or "2000" through "2FFF" hexadecimal. *Exercise for the alert reader: What should the jumpers be for "1000" through "1FFF"?*

The USART Board

In Fig. 7-11, the basic card for all the terminals' interface is shown. This card contains four 8251 USARTs, a baud-rate clock generator, and a priority-encoded status-generation PROM.

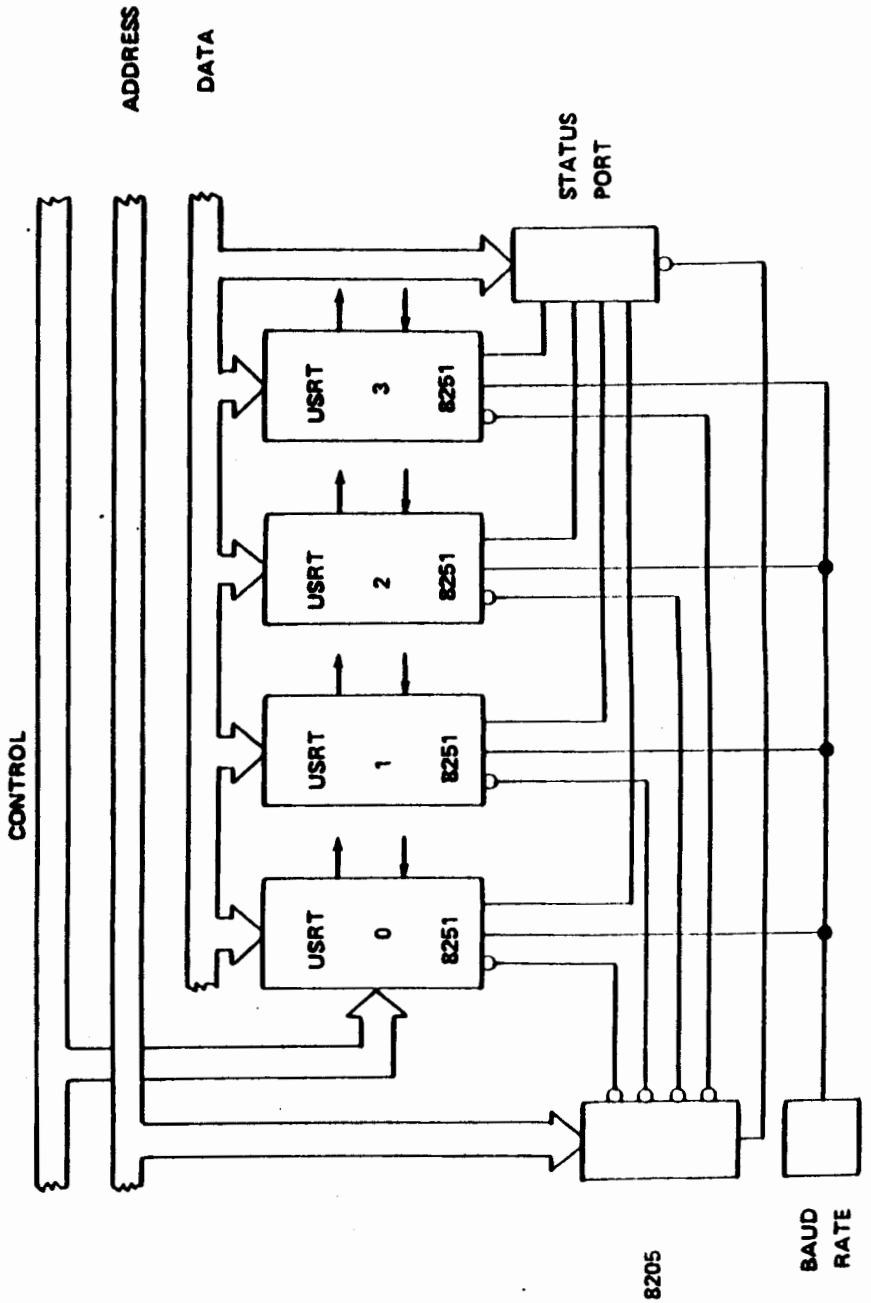


Fig. 7-11: USART Board

The 8251 is the basic serial interface element. Grouped four to a card, they are connected on their data buses to form an on-card data bus. Similar to the memory card, this on-card bus is buffered by 8216s onto the system bus. This is because the 8251 cannot drive more than eight other LSTTL loads. The 8251 is selected by implementing an address-decoding technique, using an 8205. Note how these devices are memory-mapped input-output. That is, since the same signals that control memory (MEMW, MEMR) control the USARTs, they appear as memory locations. According to our memory map, when bit A15 is high, we are addressing input-output. This corresponds to locations from "8000" to "8FFF" hexadecimal. Note that since the lower eight address lines are not decoded, these are "don't cares" in our memory-mapped I/O map.

The first card starts at "80XX" (where "XX" means that these bits do not matter) and, since each USART has two registers (input-output and control), the address ends at "87XX" hexadecimal. The next card goes from "88XX" to "8FXX", and so on, with the last card addressed by "B8XX" to "BFXX". The even page-addresses are the status registers, and the odd ones are the data-in and data-out registers.

Note also that there is a special PROM on the card, which is decoded by a separate decoder. Its address is "70XX" for the first card and "77XX" for the last card. The function of this PROM is to place on the data bus the actual address of the USART which has received a character from its terminal. How is this done? Each of the "RxRdy" lines on the USARTs indicate whether a character has been received. These four lines, one from each USART, are tied to the *address lines* of the PROM.

One of 16 possible bytes may be selected by the decoding. The fifth address bit is jumpered to a one or a zero. In this way, the same PROM can be used for board 0 or board 1, by placing in the other 16 locations the addresses for board 1, and setting the jumper on the fifth address bit to a 1 (jumper to zero for even, one for odd). What are these 16 locations? They are simply a table of the addresses "81", "83", "85" and "87" hexadecimal for board zero, and "89", "8B", "8D", "8F" for board one. Similar PROMs are made for the other six boards.

The values are placed in such a way that the first location in the PROM is a byte of zeroes. That way, when no USART has a character and all RxRdy lines are low, the byte of status is all zeroes, indicating that there is "nothing" to do for this board. If it is not zero, then a character is waiting. To make sure that it is easy to tell which USART is waiting, the next location contains the value "81": if the first USART is waiting, and all the others are not, the program will receive an "81" from the status PROM. The program can then use this value to directly address the actual character waiting. What is more, the value "81" can be masked, to form the tag for the data fetched.

The next two locations contain "83", the next four "85", and the next eight, "87". In this way, a priority table is formed so that, as each USART is serviced, the next one waiting will be serviced in turn.

This method of addressing the status PROM allows the program to use only a few instructions to identify which USART, out of 32 possible ones, is ready with a character, fetch the character, and generate the proper tag from that status information.

There are two interface chips to take the TTL serial inputs and outputs from the USARTs and convert them to EIA-RS232C +12 and −12 volt-serial pulses. These are simple level-translator integrated circuits.

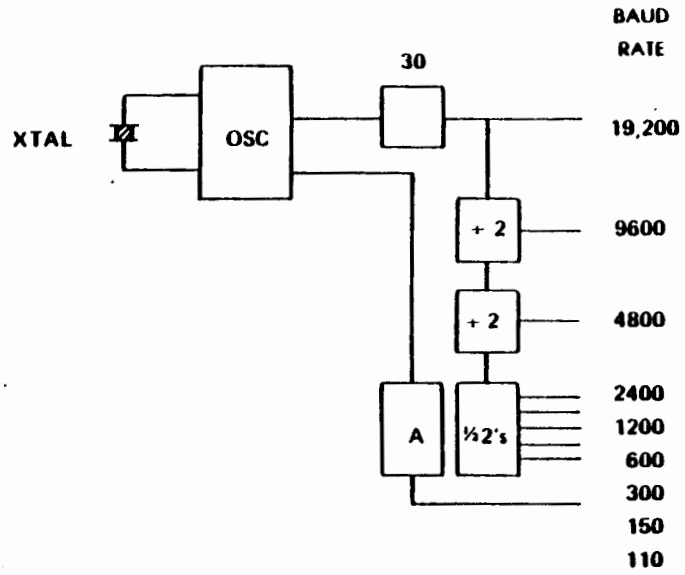


Fig. 7-12: Baud-Rate Generator

The last section consists of an astable multivibrator, synchronized by a crystal, to provide the timing for the serial-bit clocks. Two simple dividers are on each board to provide the USARTs with all of the common serial rates. This is shown in Fig. 7-12.

The Host Interface Board

This module contains: the host USART, the interrupt controller, and a baud-rate generator for the host-to-multiplexer communication rates. It appears in Fig. 7-13.

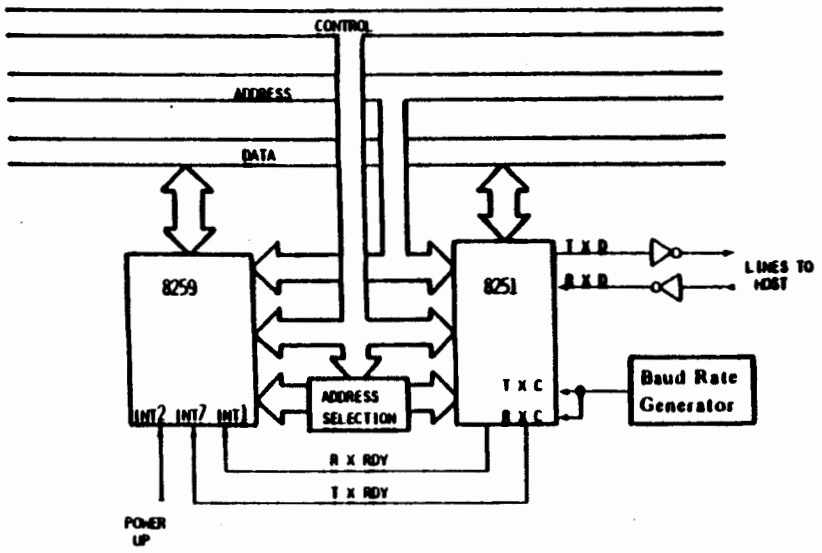


Fig. 7-13: Host Interface Board

The units on this board are addressed as input-output ports, instead of memory locations. The USART is addressed as ports "F9" and

PORTS F7 AND F8 ARE PIC

CONTROL	ADDRESS	DATA	OPERATION	
WRITE I/O				
WRITE I/O	F8	32	SETS LOW ADDRESS FOR CALL	FOR INT 1
WRITE I/O	F7	00	SETS HIGH ADDRESS FOR CALL	FOR INT 1
WRITE I/O	F8	F2	SETS LOW ADDRESS FOR CALL	FOR INT 7
WRITE I/O	F7	00	SETS HIGH ADDRESS FOR CALL	FOR INT 7
WRITE I/O	F7	70	ENABLES ONLY INT 1 AND INT 7	
WRITE I/O	F8	A0	SETS ROTATING PRIORITY RESET MODE	

Fig. 7-14: PIC Software Load Format

```
0000                    ORG 0H          ;INITIALIZATION STARTS
0000 00         RST0:   NOP
0001 31F72F             LXI SP,2FFFH    ;SET THE STACK POINTER
0004 F3                 DI              ;DISABLE THE INTRUPTS
0005 C30700             JMP INIT        ;SYSTEM RESTART UPON RESET
0008 C5         RST1:   PUSH B          ;HOST TO MUX RST VECTOR
0009 D5                 PUSH D          ;PUSH STATUS VECTOR
000A E5                 PUSH H
000B F5                 PUSH PSW
000C CD4900             CALL INT70      ;INT70 GETS THE CHARACTER FROM
000F 3E08               MVI A,0008H     ;HOST--DECODES IT AND RETURNS.
0011 D3F8               OUT 00F8H       ;INTERRUPT CONTROLLER RESET INT 1
0013 F1                 POP PSW
0014 E1                 POP H
0015 D1                 POP D           ;FLAG
0016 C1                 POP B           ;POP STATUS VECTOR
0017 EF                 RST 5           ;PRIME QUEUE
0018 FB                 EI

0019 C9                 RET
0020                    ORG 0020H
0020 CDC700     RST4:   CALL SWD50      ;SOFTWARE RESET
0023 C7                 RST 0

0028                    ORG 0028H
0028 F5         RST5:   PUSH PSW        ;SAVE A AND FLAGS
0029 DBFA               IN 00FAH        ;READ THE USRT STATUS
002B E601               ANI 0001H       ;CHK FOR TXRDY
002D CA3100             JZ POPAF        ;IF USRT IS BUSY RETURN
0030 FF                 RST 7           ;ELSE CALL RST7 FOR FIFO SERVICE
                                        ;TO CHK IF ANYTHING IS IN THE
                                        ;FIFO TO SEND TO 11/70
0031 F1         POPAF:  POP PSW
0032 C9                 RET
0038                    ORG 0038H
0038 C5         RST7:   PUSH B          ;MUX TO HOST RST VECTOR
0039 D5                 PUSH D
003A E5                 PUSH H          ;CHANNEL NOT BUSY
003B F5                 PUSH PSW
003C CD1802             CALL OINT       ;OINT IS OUTPUT A CHARACTER
003F 3E08               MVI A,0008H
0041 D3F8               OUT 00F8H       ;FROM QUEUE
0043 F1                 POP PSW
0044 E1                 POP H
0045 D1                 POP D
```

Fig. 7-15: Example of Interrupt Control

RST0;	Hardware Initialize.
RST1;	Character from Host has arrived.
RST4;	Soft-reset on program fail ROM detect.
RST5;	Channel to Host is not-busy check. Mux to Host buffer queue should be emptied.
RST7;	Channel to Host is not-busy. Check buffer queue for characters, if any transmit, if not, return.

Fig. 7-16: Vectors in Software

"FA" hexadecimal, for control and data, respectively. There is a duplicate of the baud-rate circuit here to generate the "TxC" and "RxC" signals for the host-to-multiplexer USART, as these rates may differ from any of the others in a typical system.

The interrupt controller takes the "RxRdy" and TxRdy" signals from the USART and generates two interrupt vectors, number 1 and number 7. Number 1 is to signal that a character has been received from the host and should be processed, and number 7 indicates that the USART can be reloaded to transmit another character to the host.

The 8259 interrupt controller is set up by the initialization routine, to call the service routines at the proper locations and service the interrupts on a rotating basis. After an interrupt has been serviced, the software will reset the corresponding bit flag in the 8259, and proceed with polling, until a new interrupt arrives.

Fig. 7-14 illustrates the initialization procedure of the PIC and Fig. 7-16 presents the interrupt-handling code at the beginning of memory.

PICTURES OF MULTIPLEXER PROTOTYPE P.C. BOARDS:

Fig. 7-17: CPU

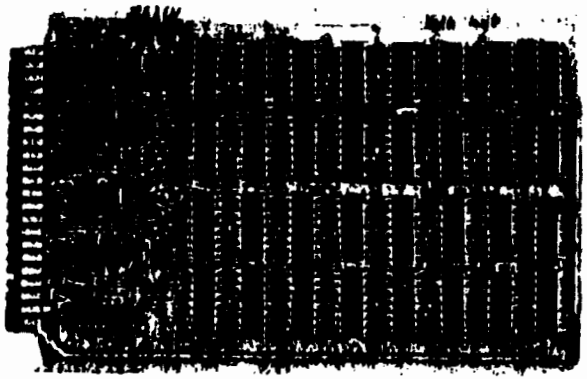


Fig. 7-17: RAM

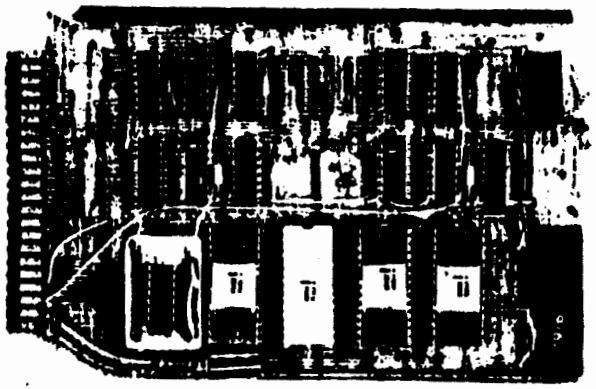


Fig. 7-19: Terminals' USARTs

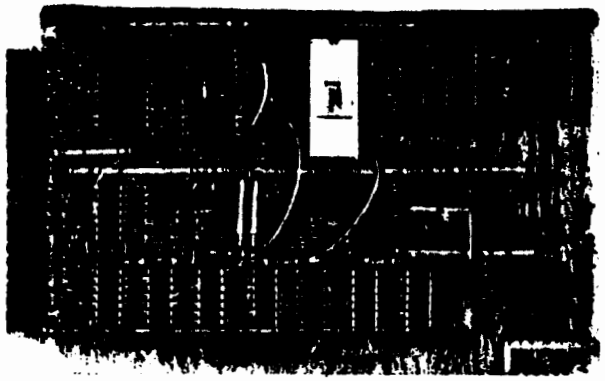


Fig. 7-20: Host and Interrupt Control

The channel-to-host was set to 9600 baud in both directions. The characters from each terminal must be echoed, as this is a full-duplex system. For every character generated, the host must process and return the echo. There are 24 Lear-Siegler ADM-3s terminals, set to 9600-baud input and output. There are also four 300-baud terminals and four 300-baud dial-up lines on the multiplexer.

Typical averaged input rate is ten characters-per-second. Average output rate is 200 characters-per-second. Buffers in the host, for characters waiting for output channel, are 95% of the time empty, indicating the host can get rid of data as fast as the channel can handle it, rather than as fast as the terminals can print. Maximum rates measured are 15 characters-per-second on input, and 620 characters-per-second on output. The maximum and typical figures were obtained over a 17-hour period, when 90% of the terminals on the multiplexer were in use.

Error rates were entirely due to the channel, or at least indistinguishable from other errors, such as operator errors and host errors.

Photographs of the printed-circuit boards appear in Figs. 7-17, 7-18, 7-19, 7-20.

CONCLUSION

In this chapter, a complete interface was described. A step-by-step discussion of how each component was integrated into a module, how the modules created a subsystem and then the overall system, should enable the reader to follow through almost any other microprocessor interface application. This particular application utilizes most of the techniques discussed in previous chapters: interrupts, memory and I/O management, integrating special techniques for software reduction in hardware, and external device interface were used here.

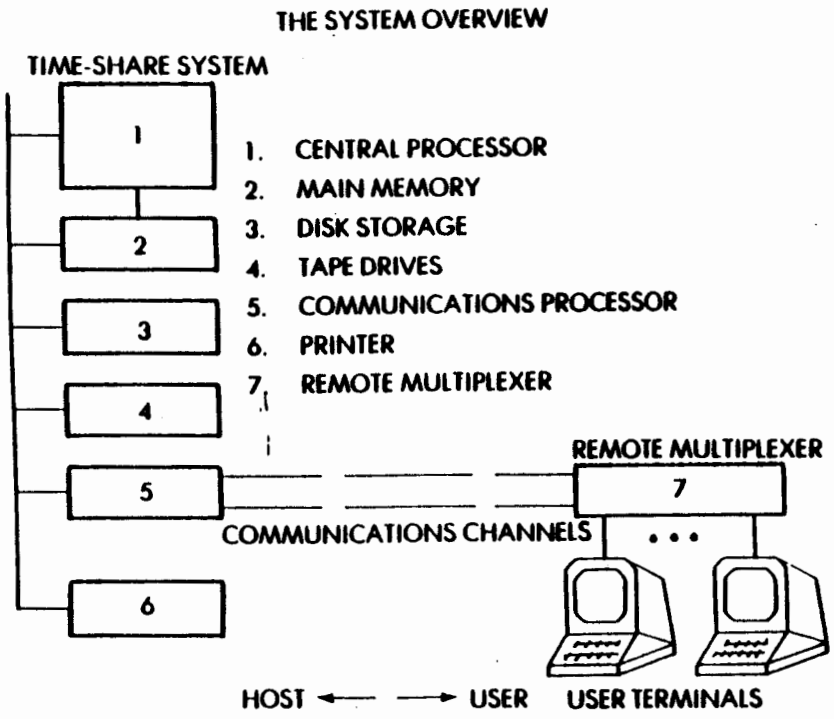


Fig. 7-21: Overall System

IIE; 21-45
IBSON NO. MIT-T

8 (37) (X)

TESTING

INTRODUCTION

What do you do when it doesn't work? What went wrong and why? The debugging process, also known as testing or trouble-shooting, is an integral part of any system design. Murphy's Law usually holds: if anything can go wrong, it will.

When faced with a misbehaving system, there are a number of techniques available to the designer for identifying and correcting problems. In this chapter, the causes of common problems, and their solutions, will be presented. Problems such as component failure, software failure, and noise-induced failure will be analyzed, and methods for identifying them will be presented.

The tools necessary in order to identify and locate these problems will also be described: voltmeter, logic probe, signature analyzer, oscilloscope, digital analyzer, in-circuit emulator, emulator, and simulator.

Finally, a case history of the "One Bit in 16,384" will be presented. The example illustrates the debugging phase in the actual design of the multiplexer presented in Chapter 7.

WHAT GOES WRONG?

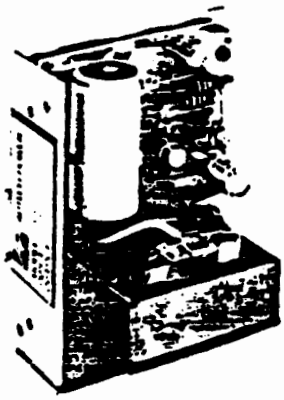
Four essential problems may arise in a system: wiring fault—a short or open circuit; component failure—including wrong value components; software bugs; and noise or interference—either internal or external.

SERIES REGULATED

MULTIPLE OUTPUT MICROPROCESSOR POWER SUPPLIES

TRIPLE OUTPUT: 75 MA TO 3 AMP

MICROPROCESSOR/GENERAL PURPOSE SERIES



TEAPS SERIES

A versatile and economical solution to many DC power applications, IC regulated.

Not available through Distributors
Contact the Engineering Sales Office near you.

A.C. Input: 115/230V ± 10%, 47-63 Hz.

D.C. Outputs: See table. Outputs #2 and #3 are independent and isolated, either output may be grounded positive or negative. Like voltage outputs may be paralleled to provide higher current outputs.

Transient Response: Less than 50 microseconds.

Temperature Coefficient: 0.02%/°C (0.0005%/°C typical).

Overshoot: No turn-on, turn-off or power failure overshoots.

Control: Screwdriver adjust potentiometers for output #1. Outputs #2 and #3 are fixed outputs.

Protection: Automatic foldback current limiting for overload through short circuit. Overvoltage protection on 5V outputs; optional on other outputs.

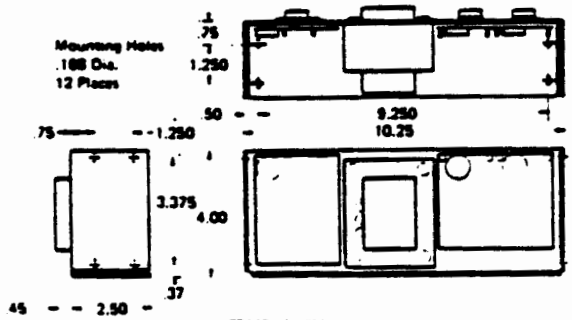
Cooling: Convection, rated at 40°C in free air. Derate 2.8%/°C above 40°C.

Operating Temperature: 0 to 65°C.

Storage Temperature: -20 to +80°C.

MODEL NUMBER	OUTPUT #1		OUTPUT #2		OUTPUT #3	
	Vdc	Amps	Vdc	mA	Vdc	mA
TEAPS-1	4.75 to 5.25	3	+12	100	-12	100
TEAPS-2	4.75 to 5.25	3	+15	100	-15	100
TEAPS-3	4.75 to 5.25	3	+12	150	- 5	75
TEAPS-4	4.75 to 5.25	3	+12	1A	-12	1A

Ripple: 5mV (PK/PK). Regulation: ±0.05% line; ±0.10% load.

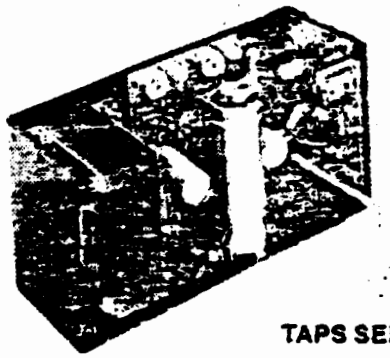


TEAPS 1, 2, and 3 DIMENSIONS
4" x 4.4" x 4.87"

Weight:
Teaps 1, 2, 3 2.6 lbs.
Teaps 4 5.25 lbs.

TRIPLE OUTPUT: 4 TO 12 AMP

MICROPROCESSOR/GENERAL PURPOSE SERIES



TAPS SERIES

Economy supplies with IC regulation and top quality components for long term reliability.

U.L. Recognized (File No. E58512)
CSA Certified (File No. LR44914)

A.C. Input: 115/230V ± 10%.

D.C. Output: See table below. All outputs are independent and isolated, any output may be grounded positive or negative. Like voltage outputs may be paralleled to provide higher output voltage.

Transient Response: Less than 50 microseconds.

Stability: ±0.2% for 24-hour period after 30-minute warm-up.

Overshoot: No turn-on, turn-off or power failure overshoots.

Temperature Coefficient: 0.02%/°C (0.0005%/°C typical).

Control: Screwdriver adjust potentiometer for ±5% voltage adjustment on 5V output. Outputs 2 and 3 are adjustable ±9 to 12V standard; ±12 to 15V if specified.

Remote Sensing: Provided on all 5V outputs.

Protection: Automatic foldback current limiting overload and short circuit protection; crowbar overvoltage protection is included on 5V outputs.

Cooling: Convection, rated at 50°C in free air. Derate 2.8%/°C when operating above 50°C up to 65°C. Optional 70°C units are available.

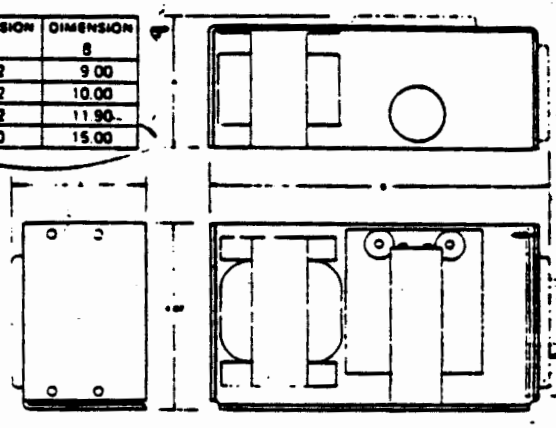
Operating Temperature: 0 to 65°C.

Storage Temperature: -20 to +80°C.

MODEL NUMBER	RATING		REGULATION		RIPPLE (PK/PK)
	Vdc	Amps	Line	Load	
TAPS 1	5V	4.0	±0.1%	±0.1%	5mV
	±9-12*	0.5	±0.1%	±0.1%	5mV
TAPS 2	5V	6.0	±0.1%	±0.1%	5mV
	±9-12*	1.0	±0.1%	±0.1%	5mV
TAPS 3	5V	9.0	±0.1%	±0.1%	5mV
	±9-12*	1.5	±0.1%	±0.1%	5mV
TAPS 4	5V	12.0	±0.1%	±0.1%	5mV
	±9-12*	3.0	±0.1%	±0.1%	5mV

*Also available with ±12-15V output. Specify if desired by adding "-2" to model number.

MODEL	DIMENSION A	DIMENSION B
TAPS 1	3.62	9.00
TAPS 2	3.62	10.00
TAPS 3	3.62	11.90
TAPS 4	4.50	15.00



Weight:
Taps 1 7.5 lbs.
Taps 2 7.75 lbs.
Taps 3 12 lbs.
Taps 4 16 lbs.

Mounting Holes:
Taps 1, 2, 3 .187
Taps 4 .218