An Application of a Single Chip Slave Microcomputer as an Intelligent Interface

Spring 1979

Thomas J. Riordan

University of Central Florida

Find similar works at: https://stars.library.ucf.edu/rtd

University of Central Florida Libraries http://library.ucf.edu

Part of the Engineering Commons

STARS Citation


This Masters Thesis (Open Access) is brought to you for free and open access by STARS. It has been accepted for inclusion in Retrospective Theses and Dissertations by an authorized administrator of STARS. For more information, please contact lee.dotson@ucf.edu.
AN APPLICATION OF A SINGLE CHIP
SLAVE MICROCOMPUTER AS AN INTELLIGENT INTERFACE

BY

THOMAS JAMES RIORDAN
B.S.E., Florida Technological University, 1978

THESIS
Submitted in partial fulfillment of the
requirements for the degree of Master of Science in
Engineering in the Graduate Studies Program of the
College of Engineering of University of Central Florida
at Orlando, Florida

Spring Quarter
1979
ABSTRACT

This thesis describes the operation of an intelligent controller existing between a computer system and a peripheral. The intelligent controller is implemented with an INTEL universal peripheral interface, UPI-41, single chip slave microcomputer. The interfaces between the controller and the computer system, and the controller and the peripheral are described in detail, as are the firmware and internal facilities of the controller itself. An evaluation of system operation is presented and, finally, the organizational philosophy of the system is discussed. Located within the appendicies are a system schematic and listing of the controller program.
TO: DR. HERBERT C. TOWLE
ACKNOWLEDGEMENT

The author wishes to thank Mr. Albert Marshal at the Orlando Naval Training Center for his involvement in the preparation of this thesis. A special thanks goes to Mrs. Doug Caldes for her efforts in typing the final draft.
# TABLE OF CONTENTS

ACKNOWLEDGMENT ........................................ iii

LIST OF FIGURES ........................................ v

I. INTRODUCTION ....................................... 1

II. SYSTEM DESCRIPTION ................................ 2

- SBC 80/20 to Controller Interface .................. 3
- Controller to ADM-3A Interface ...................... 9
- Control Switches to Controller Interface .......... 14

III. UPI-41 SINGLE CHIP MICROCOMPUTER ............ 16

IV. UPI-41 CONTROL PROGRAM .......................... 20

- Initialization Procedures ......................... 20
- Data Reception and Storage ......................... 23
- Data Decode and Message Transmission ............. 29

V. SYSTEM EVALUATION .................................. 37

VI. CONCLUSION ........................................ 43

APPENDIX 1 CONTROLLER SCHEMATIC .................... 44

APPENDIX 2 UPI-41 CONTROL PROGRAM .................. 46

APPENDIX 3 ALTERNATE BAUD RATE GENERATION SCHEME .. 54

LIST OF REFERENCES .................................... 63
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>System Block Diagram</td>
</tr>
<tr>
<td>2.</td>
<td>SBC 80/20 to Controller - Data Transfer Connections</td>
</tr>
<tr>
<td>3.</td>
<td>SBC 80/20 to Controller - Clock Connections</td>
</tr>
<tr>
<td>4.</td>
<td>SBC 80/20 to UPI-41 Data Byte</td>
</tr>
<tr>
<td>5.</td>
<td>SBC 80/20 Input Source Configuration</td>
</tr>
<tr>
<td>6.</td>
<td>Serial Transmission Character</td>
</tr>
<tr>
<td>7.</td>
<td>ADM-3A Screen Use</td>
</tr>
<tr>
<td>8.</td>
<td>Character String Transmitted to ADM-3A</td>
</tr>
<tr>
<td>9.</td>
<td>Control Switches to Controller Interface</td>
</tr>
<tr>
<td>10.</td>
<td>UPI-41 Single Chip Microcomputer Block Diagram</td>
</tr>
<tr>
<td>11.</td>
<td>UPI-41 Control Program Memory Map</td>
</tr>
<tr>
<td>12.</td>
<td>UPI-41 RAM Memory Map</td>
</tr>
<tr>
<td>13a.</td>
<td>UPI-41 Control Program Flowchart - Processing Loop</td>
</tr>
<tr>
<td>13b.</td>
<td>UPI-41 Control Program Flowchart - Interrupt Service Routine</td>
</tr>
<tr>
<td>14.</td>
<td>Interrupt Service Routine Flowchart</td>
</tr>
<tr>
<td>15.</td>
<td>Fixed Base FIFO Operation</td>
</tr>
<tr>
<td>16.</td>
<td>Moving Base FIFO Operation</td>
</tr>
<tr>
<td>17.</td>
<td>Register Bank 1 Map</td>
</tr>
<tr>
<td>18.</td>
<td>Register Bank 0 Map</td>
</tr>
<tr>
<td>19.</td>
<td>Data Access Segment Flowchart</td>
</tr>
<tr>
<td>20.</td>
<td>UPI-41 Instruction Cycle</td>
</tr>
<tr>
<td></td>
<td>Description</td>
</tr>
<tr>
<td>---</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>21.</td>
<td>UPI-41 Character Transmission Subroutine</td>
</tr>
<tr>
<td>A3-1.</td>
<td>UPI-41 Timer/Counter</td>
</tr>
<tr>
<td>A3-2.</td>
<td>Rate Mismatch Transmission</td>
</tr>
<tr>
<td>A3-3.</td>
<td>Serial Transmission Misread</td>
</tr>
</tbody>
</table>
I. INTRODUCTION

This thesis describes the hardware and software of an intelligent controller system. The controller regulates communication between a microcomputer system, an Intel SBC 80/20 (1), a CRT display, the Lear Seigler ADM-3A (2,3), and a set of control switches. Controller components include:

1. An Intel Universal Programmable Interface, UPI-41, single chip microcomputer (4)
2. A 7474 Dual D Flip Flop (5)
3. A 9602 Dual One Shot (6)
4. And a 75188 RS-232C Inverting Line Driver (7)

The system schematic and the UPI-41 control program are located in appendices 1 and 2 respectively.

The system description is divided into four sections: in section II, a functional summary and a component interface description are presented. Performance criteria are also established in this section. Section III describes the facilities available within the UPI-41 and explains their use in the present application. Section IV describes the UPI-41 control program and section V evaluates the system with respect to assumption validity, performance criteria, and maximum system capabilities. Finally, in the conclusion a discussion of the general organizational philosophy of the system is presented.
II. SYSTEM DESCRIPTION

The function of the intelligent controller is to receive parallel data from the SBC 80/20, decode the data, and cause a message to appear on the ADM-3A screen based upon the content of the data received. The control switch settings also affect controller operation, but only secondarily.

A block diagram showing the system component relationships appears in figure 1.

![System Block Diagram](image)
The following describes the three component interfaces shown in figure 1: SBC 80/20 to controller, controller to ADM-3A, and control switches to controller.

**SBC 80/20 To Controller Interface**

The SBC 80/20 to controller interface is comprised of three sets of connections. The first set, consisting of 8 data lines and 1 control line, are the data transfer connections. The second set consists of the clock connections, while the third set consists of only one connection, the initialization connection.

**Data Transfer Connections**

The 8 data lines of the data transfer set connect an 8 bit output port on the SBC 80/20 to the 8 bit Interface Register of the UPI-41. There are six I/O ports on the SBC 80/20 numbered 1 through 6 (1). These ports are divided into the Group A ports, 1 - 3, and the Group B ports, 4 - 6. Each port group corresponds to a single 8255 Programmable Peripheral Interface, PPI (8). Port 4 of Group B is programmed as an output port and used for the SBC 80/20 to UPI-41 data connection.

To transmit data to the UPI-41, the SBC 80/20 places data on port 4 and sends a Data-Available pulse to the UPI-41 over the control line. The Data Available pulse is software generated and is transmitted through port 3 of the Group A 8255. The length of the Data Available pulse is set by the time required to execute the instructions necessary to change the logic level of the control line.
twice, first from high to low, then from low to high. For the SBC 80/20 this results in a 10 microsecond pulse. The maximum pulse length to the UPI-41 is set at twice the instruction cycle length, or 6.5 microseconds (4); therefore, the 10 microsecond Data Available pulse is sent to the one shot within the controller where it is shortened to 1 microsecond. The 1 microsecond pulse from the one shot supplies the WR input to the UPI-41. On the rising edge of this pulse the data on the SBC 80/20 output port is latched into the UPI-41 Interface Register. SBC 80/20 to controller data transfer connections are illustrated in figure 2.
Each byte of data transferred from the SBC 80/20 to the UPI-41 contains two kinds of information encoded into separate fields within the byte. The three most significant bits contain a source identifier encoded in straight binary, and the four least significant bits contain a message identifier, also in straight binary, see figure 3. Bit four is not used.

![Data Byte Diagram](image)

**Fig. 3. SBC 80/20 to UPI-41 data byte**

The rate of data transfer from the SBC 80/20 to the controller can be characterized by three separate data transfer rates of which the last two will be of interest. The first two rates are determined by the SBC 80/20 input configuration, figure 4, while the third is determined by the input configuration in combination with the SBC 80/20 data processing rate.

The SBC 80/20 input configuration consists of 5 input sources, where each source contains a data latch and a service request line. When data is latched into one of the sources, the SBC 80/20 receives a service request signal from that source. For each
Fig. 4. SBC 80/20 input source configuration
service request that the SBC 80/20 responds to, a data byte will be sent to the controller.

The first data transfer rate is the average transfer rate and occurs when the 5 sources are initiating service requests at their nominal rate. The second data transfer rate is a peak average rate, and occurs when all 5 sources are initiating service requests at their maximum rate of 12 per second. This condition results in a peak average rate of $12 \times 5$, or 60 transfers per second. The third data transfer rate is the maximum rate, and occurs anytime there are simultaneous service requests to the SBC 80/20. This rate is determined by the processing rate of the SBC 80/20. Analysis using: (1) real time emulation under control of Intel's In Circuit Emulator, ICE-80 (9,10), (2) tabulation of instructions executed and their execution time (11), and (3) experimental determination (11), indicates that the SBC 80/20 processing rate is approximately 200 inputs per second.

As indicated before, the peak average transfer rate of 60 transfers per second, and the maximum transfer rate of 200 transfers per second are the relevant quantities characterizing the data transfer interface.

To keep up with the SBC 80/20 over extended periods, the processing rate of the UPI-41 must equal or exceed the SBC 80/20 peak average transfer rate, and to keep up with the SBC 80/20 when simultaneous service requests have occurred, the reception rate of the UPI-41 must equal or exceed the SBC 80/20 maximum transfer rate.
The requirement on the UPI-41 processing rate will be used in the sequel to determine the baud rate used in the controller to ADM-3A interface, while the requirement on the UPI-41 reception rate will be used to establish the necessity of a data queue within the UPI-41.

One final point is that there are no provisions for the UPI-41 to indicate that it is ready to accept a data transfer from the SBC 80/20. Thus, the data queue mentioned above will be filled by an interrupt driven procedure. This technique will assure that a data byte has been removed from the Interface Register before an additional data transfer can occur.

Clock Connections

The clock connections supply the UPI-41 clock inputs, X1 and X2. A single line from the SBC 80/20 supplies the controller with a 9.216 megahertz clock which the SBC 80/20 makes available as the BCLK output. Within the controller, the BCLK frequency is divided in half by a D flip flop within the 7474. This division is necessary to bring the BCLK frequency within the 1 to 6 megahertz operating range of the UPI-41 (4). The Q and \( \bar{Q} \) outputs of this flip flop supply the UPI-41 inputs, X1 and X2, with a 180° out of phase 4.608 megahertz clock. While the UPI-41 is capable of generating its own clock by connecting a crystal to the X1 and X2 inputs, the BCLK frequency is used since the standard asynchronous communication frequencies can be derived from it. The clock connections are shown in figure 5.
Initialization Connection

The initialization connection is between the INIT output of the SBC 80/20 and the RESET input of the UPI-41. A low going pulse on this line causes the control program of the UPI-41 to begin execution at location 0.

Controller to ADM-3A Interface

The controller to ADM-3A interface consists of a single line which originates from line 0 of port 1 on the UPI-41, passes through the 75188 inverting line driver, and terminates on the Receive Data, RXD, input of the ADM-3A. The line driver converts the TTL output of port 1, 0 – 5 volts, into RS-232C logic levels of ± 12 volts.

Information is transmitted from the UPI-41 to the ADM-3A serially using 7 bit ASCII code under the RS-232C communication
protocol (2). For this application, the number of bits per character has been minimized by using a single stop bit and no parity bit. For a given serial transmission rate this configuration will result in the fastest possible character transmission time. This time is an important consideration, as each parallel byte received by the controller from the SBC 80/20 will require a 22 character message to be transmitted. With the single start bit, the 9 bit serial character appears as shown in figure 6.

![Serial transmission character](image)

Fig. 6. Serial transmission character

Each data byte received by the UPI-41, except as noted in the next section, causes a string of 9 bit characters to be sent from the UPI-41 to the ADM-3A, a 24 line by 80 character CRT display.

The function of the ADM-3A is to provide three kinds of information concerning the SBC 80/20 inputs to an observer. The ADM-3A displays a message, indicates the SBC 80/20 source corresponding to the message, and reflects the order of input occurrence. The message is indicated by the characters displayed on the screen. The source is indicated by dividing the ADM-3A screen into 5 columns of equal width, with the first column reserved for source 1 messages, the second column for source 2 messages, and so on for the five
sources. The order of inputs is indicated by scrolling the display 1 line each time a message is displayed.

For a screen width of 80 characters, and not allowing an overlap of columns, the message field for each source is limited to the integer portion of 80/5, or 16 characters. The ADM-3A screen use is illustrated in figure 7.

To implement the function of the ADM-3A as described above requires that 22 characters be sent to the ADM-3A for each SBC 80/20 to controller transfer. The 22 characters are sent in 3 groups: a cursor control group, a message group, and a display control group.

The first group sent, the cursor control group, contains four characters which cause the cursor of the ADM-3A to position itself at the beginning of one of the five message columns. The first two control characters "escape" and "equals", activate the ADM-3A cursor positioning logic, while the next two characters are interpreted as the X and Y coordinates of the new cursor position, respectively. The Y coordinate sent is always the same, 037H, and selects the bottom line of the display. The X coordinate is determined by the SBC 80/20 input source.

The second group sent, the message group, contains 16 characters. These characters will be printed on the screen of the ADM-3A in the message field whose beginning was established by the cursor positioning control group.

The third group sent, the display control group, contains the remaining 2 characters. These characters, a carriage return and line feed, cause the display to scroll up one line in prepara-
Fig. 7. ADM-3A screen use
tion for the next control group 1 sequence.

The complete 22 character string appears as shown in figure 8.

```
1 2 3 4
ESC = 37H
Control Group 1
```

```
1 16
Data * * * Data
Message Group
```

```
1 2
ODH OAH
Control Group 2
```

Fig. 8. Character string transmitted to ADM-3A

The final aspect of the controller to ADM-3A interface is the serial transmission rate to be used. Having now established (1) the number of characters sent by the controller to the ADM-3A per SBC 80/20 input, 22, the number of serial bits per character, 9, and (3) the UPI-41 processing rate requirement, 60 transfers/sec, a minimum serial transmission, or baud, rate can be computed as

\[
\text{minimum baud rate} = \frac{22 \text{ characters}}{\text{SBC 80/20 transfer}} \times \frac{9 \text{ bits}}{\text{character}} \times \frac{60 \text{ SBC 80/20 transfers/second}}{}
\]

or 11,880 bits per second. The next highest, indeed the highest, baud rate at which the ADM-3A can receive data is 19,200 baud. This value must necessarily be chosen as the data transmission rate.
Control Switches to Controller Interface

The control switches to controller interface is a 5 line connection between 5 control switch outputs and the 5 least significant inputs of port 2 on the UPI-41. The design of port 2 on the UPI-41 is such that if nothing is connected to a port line, the line will read as a logic one, whereas, if the line is grounded through a 1k resistor, the port will read a logic zero (3). The control switch to controller connections are shown in figure 9.

During the processing of a data byte by the UPI-41, the binary source identifier is translated into a linear select code which is then compared with the switch settings on port 2. If the switch corresponding to the source identifier is set in the abort
position, a logic 0 is present and a message will not be sent. This is the exception referred to in the controller to ADM-3A interface description. If the switch is set in the display position a logic 1 will be present and a message will be sent.

This concludes the overall system description. The next two sections will describe the principal device within the intelligent controller, the UPI-41 single chip microcomputer.
III. UPI-41 SINGLE CHIP MICROCOMPUTER

The UPI-41 single chip microcomputer provides the intelligence of the intelligent controller. The block diagram below illustrates the facilities available within the UPI-41.

Fig. 10. UPI-41 single chip microcomputer block diagram
RAM within the UPI-41 serves three purposes: it contains the registers, the subroutine and interrupt stack, and the variable data storage locations. The distribution of the 64 RAM locations between these three functions is shown in figure 12.

The registers in bank 0 are designated R0-R7, while those in bank 1 are designated R0'-R7'. Only one register bank at a time can be addressed. Bank selection is accomplished by executing a special select register bank X, SEL RBX, instruction where X is either 0 or 1. The registers of bank 0 are used for data processing and message transmission, while those of bank 1 are used for queue control.

The UPI-41 contains a rather sophisticated timer which was evaluated for use as the bit interval generator for UPI-41 to ADM-3A serial transmission. As several difficulties were encountered, see appendix 3, the use of the timer, while representing a possible area for future research, was rejected in favor of a software timing approach. The software timing routine will be described, along with the rest of the UPI-41 control program, in the next section.
Fig. 12. UPI-41 RAM memory map
IV. UPI-41 CONTROL PROGRAM

The UPI-41 program is written in MCS-48/UPI-41 assembly language (12). The program was assembled using a cross assembler (13) operating on an Intel Microcomputer Development System, MDS-800 (14,15). The machine code was burned into the EPROM program memory of the UPI-41 (4) using an Intel Universal Prom Programmer (16) and the Universal Prom Mapper Software (17). The assembly of the program and the burning of the EPROM were done under control of the Intel System Implementation Supervisor, ISIS II (18), operating from an Intel Dual Floppy Disk Drive (19).

The program description is divided into three parts:
1. Initialization procedures
2. Data reception and storage
3. Data decode and message transmission

The program listing is located in appendix 2, a flow chart appears in figure 13a and 13b.

Initialization Procedures

The first section of the UPI-41 program performs functions which are necessary prior to data reception. These functions are the initialization of registers and the initialization of the ADM-3A screen. The values placed in the various registers will be explained as they are encountered within the program. The screen initializa-
Initialize

A. constants
B. cursor

Start

Data in Queue?

No

Yes

Fetch Queue Data
Decrement Queue Status
Decode Identifier

Output Desired?

No

Yes

Position Cursor

Decode Message

Enable Interrupts

Send Character Bit

Disable Interrupts

Character Complete?

Yes

Message Complete?

No

Yes

Start

Fig. 13a. UPI-41 control program flowchart - processing loop
Fig. 13b. UPI-41 control program flowchart - interrupt service routine
tion procedure consists of clearing the screen and positioning the cursor in the bottom left hand corner. The screen is cleared by transmitting a special character, 01AH, to the ADM-3A (3), while the cursor is positioned using the 4 character cursor positioning sequence described previously in the controller to ADM-3A interface section.

As the final step in the initialization procedures, the UPI-41 enables itself to data reception by outputting a logic zero to port 2 line 7. This port line is connected to the UPI-41 chip select, CS, input as shown in the schematic in appendix 1. Since all port lines are in the logic high state following a system reset, UPI-41 input is disabled until the output instruction is executed.

Data Reception and Storage

When data is written into the UPI-41 interface register by the SBC 80/20, an interrupt request is generated. Upon recognition of the interrupt, the interrupt vector jump at locations 3 and 4 in program memory is executed, and the interrupt service routine, lines 118 through 134 in appendix 2, is entered. The interrupt routine inputs the data from the interface register and places the data in a queue. A flowchart of the interrupt service routine appears in figure 14.

It was pointed out in the section describing the SBC 80/20 to controller interface that the UPI-41 reception rate requirement would necessitate the data queue. The necessity for the queue can be shown as follows:
Interrupt

1) Select Register Bank 1
2) Save accumulator

Yes Queue Full?

No

1) Increment status register
2) Input data from interface register
3) Store data at "Put Pointer"

"Put Pointer" Wraparound?

Yes

Load "Put Pointer" with 1 less than stack base

Increment Put Pointer

Restore Accumulator

Return and Restore Status

No

Fig. 14. Interrupt routine flowchart
Unless the 19,200 baud rate can meet the UPI-41 reception rate requirement as well as the processing rate requirement, it is necessary to provide a data queue to prevent data from being overwritten in the interface register. For this condition to be met, the 19,200 baud rate must be proportionately greater than the 11,800 minimum baud rate by at least the proportion of the reception rate requirement to the processing rate requirement, or

\[
\frac{19,200}{11,800} > \frac{200}{60}
\]

as this is not true, a queue must be maintained.

To meet the storage requirements a First In First Out, or FIFO, stack is implemented in the variable data storage area of the RAM memory. See figure 12. A FIFO stack allows data to be retrieved so that order of entry is preserved. The operation of a FIFO stack can be conceptualized by considering a storage mechanism where data inputs are stacked one on top of the other as they arrive, and where data removal is accomplished by pulling from the bottom. As an entry is removed, all remaining entries move down one location. This operation is illustrated in figure 15.

The problem with this implementation is in moving the remaining data entries down. For N remaining inputs, the operation requires 2N memory accesses and 5N program steps as shown below:

1. Increment pointer
2. Load data byte - first memory access
3. Decrement pointer
4. Store data byte - second memory access
5. Increment pointer
A more efficient algorithm uses a "get data" pointer as well as the "put data" pointer used in the implementation above. The get data pointer allows the "bottom" of the stack to move upward as data is removed from the stack. This eliminates the necessity of moving each of the remaining inputs down. Instead, the get data pointer is incremented once each time data is removed. The put data pointer always identifies the next location available for data storage and the get data pointer identifies the location of the next value to be removed. The only problem with this implementation is that unless data memory is infinitely long, storage locations will run out at some point. This condition being unacceptable, a "top-of-stack" must be defined, and as the pointers reach the top they must wrap around. In this application the top-of-stack has been made coincident with the top of RAM, making the last location ad-
dress 63 and giving a stack size of \((63 - 32) + 1\), or 32 locations. As each pointer reaches location 63 it is returned to location 32 instead of being incremented further. Implemented in this manner, the number of steps required for a data removal is independent of \(N\) and, for the UPI-41, has a maximum value of 5 as indicated by lines 85 through 89 of the program listing.

For either implementation some way of determining when the stack is full must be available. For the two pointer implementation, the queue full condition is easily detected by maintaining a queue status value which indicates how many entries are presently on the stack. If a check of the queue status register indicates that the queue is full additional data must be rejected to avoid overwriting of the earliest entry with the newest entry. Since the UPI-41 has been designed to meet the processing rate requirement, it follows that the maximum stack usage must be less than or equal to the number of SBC 80/20 input sources, or 5; therefore, the queue full condition can never occur in this application. Use of the queue status value in this application, then, is limited to determining when data is available on the stack. Figure 16 illustrates the operation of the moving base FIFO stack.

The put and get data pointers, the queue status, and the constants used to determine the pointer wraparound and queue full conditions are located in register bank 1. Also, since the data reception routine is entered in response to an interrupt, another bank 1 register is allocated for accumulator storage. Finally, one register is used for temporary data byte storage during computations.
Fig. 16. Moving base FIFO operation

Registers 0 and 1 are the only locations which can serve as pointers into the variable data storage area; therefore, the get and put data pointers are defined as the contents of registers 0 and 1 respectively, the other locations are assigned arbitrarily as per figure 17.
Data Decode and Message Transmission

Once data is placed in the queue by the interrupt service routine, a check of the queue status register, lines 80 and 81 of the program listing, will indicate that data is available for processing. The program will then enter the main program loop, line 82, where the data decode and message transmission function begins.

This section of the program can be divided into 3 segments:
1. Data access
2. Source processing
3. Message processing

Data Access

The function of the data access segment is to remove a data byte from the queue and perform the transition between register bank 1 operation and register bank 0 operation. The data removal
steps are reminiscent of the steps performed in the interrupt rou-
tine, while the bank transition is accomplished by placing the data in the accumulator and then selecting the new register bank. A flowchart is shown in figure 18.

Register bank 0 is used for the remainder of the program. All locations within this bank are assigned arbitrarily as shown in figure 19.

| Register 0 | Serial Transmission Counter |
| Register 1 | Unused |
| Register 2 | Relay Counter |
| Register 3 | Message Identifier |
| Register 4 | Linear Select Source Identifier |
| Register 5 | Binary Source Identifier |
| Register 6 | Message Length Constant |
| Register 7 | Data Byte |

Fig. 19. Register bank 0 map

Source Processing

The function of the source processing segment, lines 93 through 98, is to use the source identifier portion of the data byte to (1) determine whether a message transmission is desired and (2) position the cursor at the proper place on the ADM-3A screen. The source processing segment calls three subroutines; MASK, LOCSET, and TAB.
Fig. 18. Data access segment flowchart
Subroutine MASK, lines 144 through 156, converts the binary source identifier into the linear select identifier through the use of the lookup table located at MSKD AT, line 143. The subroutine then performs the comparison with the port 2 control switch lines and sets a flag according to the result.

Subroutine LOCSET, lines 161 through 168, sends the characters which activate the cursor control logic and the Y coordinate value to the ADM-3A.

Subroutine TAB, lines 157 through 160, converts the binary source identifier into the proper X coordinate value and completes the cursor positioning sequence by transmitting the coordinate value to the ADM-3A.

Message Processing

The function of the message processing segment, lines 99 through 112, is to convert the message identifier portion of the data byte into the page 3 address of the message string, output the message string, scroll the ADM-3A display one line, and return to the queue status checking loop.

The page 3 address of the message string is produced by multiplying the binary message identifier by 16. Thus, the message identifier is converted into the starting address of a 16 character string which makes up the message. The multiplication is accomplished by swapping the high and low order nybbles of the data byte and then masking out the low order nybble. This operation is equivalent to four left shifts and, therefore, multiplies the source
identifier by $2^4$, or 16.

Subroutine STROUT, lines 169 through 175, uses the message address produced by the preceding multiplication and the string length constant contained in register 6 to control the transmission of the 16 character message string to the ADM-3A.

The CRLF procedure, lines 107 through 110 cause the scroll of the ADM-3A display by sending the carriage return line feed combination.

Finally, register bank 1 is selected so that when the jump at line 112 occurs the register bank containing the queue status value, R2', will be addressed by the WAIT loop.

This completes the description of the control program except for the subroutine which controls character transmission. This function is accomplished by the OUTPUT subroutine, lines 176 through 191.

It was noted in the description of the clock connection, section II, that the 4.608 megahertz clock input to the UPI-41 would be used to generate the proper communication frequency. The following discussion explains this process and the operation of the OUTPUT subroutine.

Each instruction in the UPI-41 instruction set consists of either 1 or 2 instruction cycles. Each instruction cycle consists of 5 machine states and each state consists of 3 clock periods. See figure 20.
The instruction cycle execution rate, then, is \( \frac{1}{15} \) of the input clock rate, or 307,200 instruction cycles per second. The instruction cycle execution rate divided by 16 produces the serial transmission rate of 19,200 baud. Therefore, a bit interval, i.e. the time a serial bit should be present on port 1 during transmission, is exactly 16 instruction cycles. A 9 bit character can be transmitted by constructing a loop which places a new serial bit on the port 1 transmission line every 16 instruction cycles.

The OUTPUT subroutine, figure 21, expects the 7 least significant accumulator bits to hold the 7 bit ASCII representation of the character to be sent. As 9 bits are required to send a complete character, including the start and stop bits, the 8 bit accumulator and the carry bit are catenated to form a 9 bit register. The accumulators most significant bit and the carry bit serve as the stop and start bits respectively. Once the 9 bit register is set up with the character, the bits are sent by successively rotating the bits into the least significant bit position of the accumulator.
and then outputting the accumulator to port 1.

Instructions 1, 2, and 3 set up the character, the transmission loop begins at line 180. Note that the number of instruction cycles required for each instruction in the transmission loop is shown to the right of the instruction.

For the first eight bits transmitted, program execution proceeds through the steps indicated 1 through 8. As can be verified by the reader, 16 instruction cycles are executed between bit changes.

Program flow for the final bit proceeds through the steps indicated A through E. While this sequence requires only 9 instruction cycles, analysis of the complete program shows that for any set of conditions a minimum of 8 additional instruction cycles will be required to reach the initial output instruction for a new character. Thus, a minimum of 9 + 8, or 17, cycles will be executed exceeding the minimum of 16 by 1 cycle. But, as there is no maximum length for the stop bit since its level corresponds to the non-active, or "marking" state, the value 17 is acceptable.

The instruction executed just prior to entry into the bit transmission loop disables interrupts, while the instruction just before the return reenables them. Interrupts must be disabled during transmission of a character since the occurrence of an interrupt service routine would insert extra instruction cycles, thereby destroying the integrity of the software timing loop.
As a concluding remark on the UPI-41 control program, it is noted that starting on page 51 of the listing a sample set of message strings is shown.

The program listing referred to throughout this section is the assembly listing produced during the assembly of the UPI-41 control program source file. This version of the program was used for the system evaluation to be presented in the next section.
V. SYSTEM EVALUATION

In section II equation (1), the minimum UPI-41 to ADM-3A serial transmission rate was computed by considering only the character string transmission time. It was assumed at that point that processing done in decoding the input data would be insignificant compared to the processing done within the serial transmission routine. The validity of this assumption can be evaluated by determining from the control program the percentage of instruction cycles which are executed outside of the serial transmission routine.

As determined in section IV, sixteen instruction cycles are required per serial bit transmitted. Since 22 characters at 9 bits per character are sent for each input, the number of transmission instruction cycles executed for each string transmission is

\[ 16 \text{ instruction cycles/bit} \times 9 \text{ bits/character} \times 22 \text{ characters/string} \]

or

\[ 3168 \text{ instruction cycles/string} \quad (3) \]

Analysis of the UPI-41 program shows that 236 instruction cycles are executed external to the serial transmission routine. Thus, the percentage of instruction cycles executed outside of the serial transmission routine is

\[ \frac{236 \text{ instruction cycles}}{3168 \text{ instruction cycles}} \times 100 \text{ percent}, \text{ or } 7.45 \text{ percent} \quad (4) \]
The assumption, then, gave a minimum baud rate within 10% of the exact figure.

To meet the UPI-41 processing rate requirement requires that the minimum baud rate of 11,880 computed in equation (1) be adjusted upward by 7.45 percent. The new minimum baud rate becomes

\[ 11,880 \text{ bits/sec} + (0.0745 \times 11,880 \text{ bits/sec}) \], or 12,765 bits/sec (5)

However, since the baud rate chosen was already 19,200 bits per second due to the ADM-3A receiving rate constraints, no program or hardware adjustments are necessary.

Transmission at 19,200 baud instead of the minimum baud provides an immediate system expansion capability of

\[ \frac{19,200 \text{ bits/sec} - 12,765 \text{ bits/sec}}{12,765 \text{ bits/sec}} \times 100 \text{ percent} \] (6)

or approximately 50 percent. This expansion can be in terms of the number of SBC 80/20 input sources or the source input rate, see figure 5. If the source input rate is maintained at 12 inputs per second, the number of sources could be expanded to 7.5, or realistically 7. Conversely, if the number of sources is taken as a constant, the input rate can be expanded from 12 inputs per second to 18 inputs per second.

The expansion to 18 inputs per second would require no additional system expansion, while the expansion to 7 inputs would require either a wider display screen or a slight overlapping of the message columns. Overlapping the columns would be preferable since
a wider screen would necessitate a new display device.

It was shown in section II that if the UPI-41 reception rate requirement was to be met, a data queue would have to be maintained. While a necessary requirement, however, the queue is not sufficient to guarantee that the reception rate requirement will be met. The guarantee must be provided by insuring that the worst case interrupt response time meets the requirement. The worst case response occurs when an input occurs just as interrupts are disabled following entry into the serial transmission loop. The interrupt response time for this condition will be essentially equal to the character transmission time. The worst case reception rate will be one over this time and can be represented as

\[
\frac{1}{19,200 \text{ bits/sec}} \times 9 \text{ bits/transfer}
\]

or 2133 transfers per second. This value exceeds the required UPI-41 data reception rate of 200 transfers per second by better than an order of magnitude.

Comparing the above order of magnitude difference with the 50% difference computed in relation to the UPI-41 processing rate requirement, it is seen that the constraints on system expansion are primarily associated with data processing rather than data reception, and, therefore, with the serial transmission rate. System expansion beyond 7 sources or 18 inputs per second would only be possible by increasing the serial transmission rate. As 19,200 baud is the highest transmission rate supported by the ADM-3A, any expa-
sion would require going to a different display device.

In terms of the UPI-41, the limits on system expansion are related to its maximum serial transmission rate, the input identifier field of the data byte, and the queue size.

If the input clock rate to the UPI-41 was increased to 5.76 megahertz, the highest clock rate which is both within the 1 - 6 megahertz operating range and an integral multiple of the standard communication frequencies, the baud rate could be quadrupled to 76,800 baud. This baud rate corresponds to a serial bit time equivalent to 5 instruction cycles at the 2.604 microseconds per instruction cycle established by the 5.76 megahertz clock. By decreasing to a minimum the number of instructions executed in the bit generation loop of the OUTPUT subroutine as follows:

```
Loop 1: OUTL P1, A ;2 cycles
        RRC A ;1 cycle
        DJNZ RO, Loop1 ;2 cycles
```

the 5 instruction cycle requirement can be met. For the 76,800 bit per second transmission rate, the system expansion percentage becomes:

\[
\frac{76,800 \text{ bits/sec} - 12,765 \text{ bits/sec}}{12,765 \text{ bits/sec}} \times 100 \text{ percent} \quad (8)
\]

or approximately 500 percent.

Now, the number of sources could be increased to \(5 + (5 \times 5)\) or 30, or the maximum input rate could be increased to \(12 + (12 \times 5)\), or 72 inputs per second. The changes required for the
increased input rate would be the alteration of the output subroutine to the form shown above.

An increase in the number of sources to 30, however, must be checked for plausibility against two other system constraints: the queue size, and the source identification field of the SBC 80/20 to controller data byte.

The queue size of the UPI-41 is 32 locations; thus, even if it is assumed that the queue could be filled before any input could be completely processed, the minimum number of inputs that could be handled would be the queue size plus the entry being processed, or 33. Since the baud rate already limits the input sources to 30, this is not a limiting factor. The source identification field is currently 3 bits wide as necessitated by 5 sources allowing a possible identification of $2^3$ or 8 sources, see figure 4. If the undedicated bit is added to the source identifier field, then $2^4$, or 16 sources could be supported. As this value is less than the limitation imposed by the serial transmission rate constraint, the source identification field becomes the limiting factor on input source expansion.

While not directly related to the scope of this thesis, it is noted that expansion of the source input rate is a real possibility as other input sources could be easily attached which have higher characteristic input rates. For 5 source inputs, the SBC 80/20 is limited to servicing a source input rate of 40 inputs per second. Thus, with a display device having a higher serial reception rate, the controller could easily handle the maximum source
input rate imposed by the SBC 80/20. Expansion to greater than 5 sources would require rather extensive hardware and software changes within the SBC 80/20. For the source input rate of 12 inputs per second, the SBC 80/20 could handle 16 input sources. This value also compares favorably with the maximum capability imposed by the UPI-41, also equal to 16 sources.
VI. CONCLUSION

The use of the UPI-41 as described in this thesis follows a current trend in the design of computer control systems. This trend is characterized by the spreading of the "intelligence" throughout the entire system. Systems designed in this fashion are grouped under the general classification of "distributed processor systems". The present application can be further classified as a master/slave distributed processor system, where the SBC 80/20 computer functions as the master and the UPI-41 functions as the slave. Distributed processor systems provide for a certain degree of parallelism within a control task, and thereby increase the operating speed of the overall system. As the cost of processing elements continues to decrease, distributed processor systems will become increasingly prevalent in computer control applications.

The design and construction of the intelligent controller system was completed during the summer/fall quarter of 1978. In many tests since that time the system has performed as expected.
APPENDIX 1

CONTROLLER SCHEMATIC
APPENDIX 2

UPI-41 CONTROL PROGRAM
### Register Bank 0

<table>
<thead>
<tr>
<th>LOC</th>
<th>OBJ</th>
<th>SEQ</th>
<th>SOURCE STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>*</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>*</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td>*</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td>*</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td>*</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td>!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td>REGISTER BANK 0</td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td>REGISTER 0(R0) 9 BIT SERIAL TRANSMISSION COUNTER</td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td>REGISTER 1(R1) UNUSED</td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
<td>REGISTER 2(R2) COUNT FOR VARIABLE DELAY</td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
<td>REGISTER 3(R3) CHARACTER STRING INDEX REGISTER</td>
</tr>
<tr>
<td>19</td>
<td></td>
<td></td>
<td>REGISTER 4(R4) SOURCE IDENTIFIER (LINEAR SELECT)</td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
<td>REGISTER 5(R5) SOURCE IDENTIFIER (BINARY)</td>
</tr>
<tr>
<td>21</td>
<td></td>
<td></td>
<td>REGISTER 6(R6) MESSAGE STRING LENGTH CONSTANT</td>
</tr>
<tr>
<td>22</td>
<td></td>
<td></td>
<td>REGISTER 7(R7) PARALLEL DATA TRANSFER</td>
</tr>
<tr>
<td>23</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
<td>REGISTER BANK 1</td>
</tr>
<tr>
<td>25</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
<td></td>
<td>REGISTER 0(R0) GET DATA POINTER</td>
</tr>
<tr>
<td>28</td>
<td></td>
<td></td>
<td>REGISTER 1(R1) PUT DATA POINTER</td>
</tr>
<tr>
<td>29</td>
<td></td>
<td></td>
<td>REGISTER 2(R2) QUEUE STATUS</td>
</tr>
<tr>
<td>30</td>
<td></td>
<td></td>
<td>REGISTER 3(R3) ACCUMULATOR STORAGE</td>
</tr>
<tr>
<td>31</td>
<td></td>
<td></td>
<td>REGISTER 4(R4) UNUSED</td>
</tr>
<tr>
<td>32</td>
<td></td>
<td></td>
<td>REGISTER 5(R5) WRAPAROUND CONSTANT=153D</td>
</tr>
<tr>
<td>33</td>
<td></td>
<td></td>
<td>REGISTER 6(R6) QUEUE STATUS CONSTANT=224D</td>
</tr>
<tr>
<td>34</td>
<td></td>
<td></td>
<td>REGISTER 7(R7) TEMPORARY DATA WORD STORAGE</td>
</tr>
<tr>
<td>35</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>37</td>
<td></td>
<td></td>
<td>PORT 1 SERIAL TRANSMISSION ON BIT 0</td>
</tr>
</tbody>
</table>
PORT 2 LINES 0-4 USED AS A MASK INPUT TO INHIBIT TEXT STRING OUTPUT. LINE 7 USED TO ENABLE CHIP SELECT.

0000 44 ORG 0
0000 040A 45 JMP 10D ; PRESERVE INTERRUPT VECTORS
0003 46 ORG 3D
0003 0458 47 EXTINT: JMP INROUT ; JUMP TO INTERRUPT ROUTINE

000A 50 ORG 10D
000A B889 51 INIT: MOV R0,#050 ; INITIAL SET TRANSMISSION COUNTER
000C 231A 52 MOV A,#1AH ; ASCII CHAR TO CLEAR CRT SCREEN
000E 342E 53 CALL OUTPUT
0010 23FF 54 MOV A,#0FFH ; START OF NESTED DELAY
0012 BAFF 55 LOOP: MOV R2,#0FFH
0014 3441 56 CALL DELAY
0016 07 57 DEC A
0017 5412 58 JNZ LOOP ; END OF NESTED DELAY
0019 3419 59 CALL LOCSET ; SET UP CRT TO ACCEPT X COORD VALUE
001B 2320 60 MOV A,#20H ; X VALUE FOR COLUMN 1
001D 342E 61 CALL OUTPUT ; ROUTINE TO SEND ASCII CHARACTER
001F D5 62 SEL R81
0028 B920 63 MOV R0,#32D ; INITIAL VALUE FOR READ MEMORY POINTER
0022 B920 64 MOV R1,#32D ; INITIAL VALUE FOR WRITE MEMORY POINTER
0024 BA00 65 MOV R2,#8 ; CLEAR QUEUE STATUS REGISTER
0026 REE0 66 MOV R6,#224D ; 224 + 32 AVAILABLE LOCATIONS IN RAM
0028 BDC1 67 MOV R5,#1930 ; 193 + 63(LAST RAM ADDRESS) = 256 =>$ OVERFLOW
002A 237F 68 MOV A,#7FH ; NUMBER WILL KEEP LINES 0-6 AS INPUTS
002C 3A 69 MOV P2,A ; AND WILL ENABLE THE CHIP SELECT WHICH
002D 05 70 EN I ; IS TIED TO LINE 7

002E FA 80 WAIT: MOV A,R2 ; ENABLE EXTERNAL INTERRUPTS
002F C62E 81 JZ WAIT ; GET QUEUE STATUS
0031 F0 82 START: MOV A,#R0 ; IF QUEUE EMPTY NO ACTION

; GET DATA FROM RAM LOCATION
49

0032 AF 83 MOV R7, A ; STORE DATA
0033 CA 84 DEC R2 ; DECREMENT QUEUE STATUS REGISTER
0034 FD 85 MOV A, R5 ; 193 DECIMAL
0035 68 85 ADD A, R8 ; CHECK FOR LAST ACCESS BEING @ TOP OF RAM
0036 963A 87 JNZ CONT ; ONE LESS THAN BOTTOM OF RAM
0037 B81F 88 MOV R8, #31D ; INCREMENT GET DATA POINTER
0038 83 T10Y R7,A ; RETRIEVE DATA

0039 89 CONT: INC R8 ; STORE DATA
003A 18 90 MOV A, R7 ; CHECK TO SEE IF OUTPUT DESIRED
003B C5 91 SEL R80 ; IF FLAG SET WAIT FOR NEW DATA
003C 3485 92 MOV R7, A ; SET UP ADON TO ACCEPT X-COORD VALUE
003D 8658 93 CALL MASK ; GET DATA ID FROM PROCEDURE MASK STORAGE
003E 3419 94 JFB ESCAPE ; LOCATION
003F FD 95 CALL LOCSET ; TAB OVER TO LOCATION CORRESP TO DATA #
0040 96 96 MOV A, R5 ; RETRIEVE BB/20 DATA
0041 FF 97 CALL TAB ; PUT CODE FOR MESSAGE ACCESS
0042 183 AF 98 ANL A, #8FH ; IN UPPER 4 BITS TO ALLOW ACCESS TO
0043 53F8 101 MOV R3, A ; 16 MEMORY LOCATIONS PER SHOT TYPE
0044 185 102 MOV R6, #16D ; MASK OUT LOW ORDER BITS
0045 3486 103 ANL A, #8FH ; STORE RELATIVE ADDRESS OF CHAR STRING
0046 8658 104 MOV R3, A ; STRING LENGTH COUNTER
0047 BE18 105 MOV R6, #16D ; PROCEDE TO OUTPUT ASCII STR
0048 3426 106 CALL STROUT ; LINE FEED
0049 238A 107 CRLF: MOV A, #AH ; CARRIAGE RETURN
004A 342E 108 CALL OUTPUT ; RETURN TO CORRECT REG BANK FOR WAIT LOOP
004B 2380 109 MOV A, #0DH
004C 342E 110 CALL OUTPUT
004D 2380 111 ESCAPE: SEL R81
004E 842E 112 JMP WAIT

004F 842E 113 ;
0050 A1 114 ;
0051 115 ;
0052 116 ;
0053 117 ;
0054 D5 118 INROUT: SEL R81 ; INTERRUPT REG BANK
0055 CA 119 MOV R3, A ; SAVE ACCUMULATOR
0056 FE 120 MOV A, R5 ; 224D
0057 68 121 ADD A, R2 ; CHECK FOR QUEUE FULL
0058 68 122 JZ QUEUEFUL ; INCREMENT QUEUE STATUS REGISTER
0059 68 123 INC R2 ; INPUT DATA
005A 68 124 ADD A, D88 ; FROM SR INTERRUPT STORE FF
005B 68 125 IN A, D8B ; STORE NEW DATA
005C 68 126 MOV A, R8L A ; 193D
005D FD 127 MOV A, R5
0065 69 128 ADD A, R1 ; CHECK TO SEE IF STORE WAS
129 0066 966A 129 IN LAST AVAILABLE RAM LOCATION
0068 B91F 130 JNZ CONT1 ; IF NOT THEN CONTINUE
006A 19 131 MOV R1, #31D ; BOTTOM OF QUEUE
006B FB 132 CONT1: INC R1 ; INCREMENT PUT DATA REGISTER
006C 93 133 QUEFUL: MOV A, R3 ; RESTORE ACCUMULATOR
134 RETR ; RETURN FROM INTERRUPT
135 ; & REENABLE INTERRUPTS
136 ;
137 ;
138 ; SUBROUTINES IN SECOND PAGE OF MEMORY
139 ;
140 ;
141 ;
142 0100 47 ORG 256D
143 MSKDAT: DB 1D, 2D, 4D, 8D, 16D
144 0101 87
145 0102 04
146 0103 08
147 0104 18
148 0105 5310 144 MASK: ANL A, #3E8H ; MASK OUT 5 LOW ORDER BITS
149 0107 47 145 SWAP A ; DATA ID IN BITS 1, 2, 3
150 0108 77 146 RR A ; IN BITS 0, 1, 2
151 0109 8D 147 MOV R5, A ; STORE DATA CODE
152 010A 87 148 DEC A ; DATA CODE (FILE) 0–4
153 010B 83 149 MOV R, #A ; GET LOOKUP VALUE FOR CURRENT DATA
154 010C AC 150 MOV R4, A ; STORE VALUE
155 010D 0A 151 IN A, #2 ; GET DATA MASK
156 010E 5C 152 ANL A, #4 ;
157 010F 65 153 CLR F0 ;
158 0110 9613 154 JNZ CONT2 ; JUMP IF DATA NOT MASKED
159 0112 95 155 CPL F0 ; SET FLAG INDICATING MASK
160 0113 83 156 CONT2: RET ; RETURN FROM SUBROUTINE
161 0114 17 157 TAB: INC A ; CREATE CORRECT DIGIT FOR HIGH BYTE
162 0115 47 158 SWAP A ; PUT IN HIGH BYTE
163 0116 342E 159 CALL OUTPUT ; RETURN FROM SUBROUTINE
164 0118 83 160 RET ;
165 0119 231B 161 LOCSET: MOV A, #18H ; ASCII ESCAPE (REF ADM MANUAL
166 162 ; RE CURSOR POSITIONING)
167 011B 342E 163 CALL OUTPUT ; ASCII EQUALS
168 011C 233D 164 MOV A, #30H ; ROW 24 OF ADM TERMINAL
169 011D 342E 165 CALL OUTPUT
170 0121 2337 166 MOV A, #37H ;
171 0122 342E 167 CALL OUTPUT
172 0123 83 168 RET ; RETURN FROM SUBROUTINE
0126 FB 169 STROUT: MOV A,R3  ; RETRIEVE PAGE 3 ADDRESS OF ASCII STRING
0127 E3 170  ; GET ASCII CHARACTER
0128 1B 171 MOVPL A, #A
0129 342E 172 INC R3  ; NEXT CHARACTER
012A EE26 173 CALL OUTPUT
012B 83 174 DJNZ R6, STROUT  ; HAVE ALL CHAR BEEN OUTPUT
012C 83 175 RET  ; RETURN FROM SUBROUTINE
012D 97 176 OUTPUT: CLR C  ; CLEAR CARRY BIT
012E E7 177 RL A  ; POSITION START BIT
012F A7 178 CPL C  ; SET CARRY BIT
0130 15 179 DIS I  ; DISABLE INTERRUPTS
0131 39 180 LOOP1: OUTL PL,A  ; PUT SERIAL BIT ON TRANSMIT LINE
0132 EE39 181 DJNZ R8, CONTINUE  ; JUMP TO CONTINUE IF NOT DONE
0133 8889 182 MOV R8, #89D  ; RELOAD BIT COUNTER
0134 05 183 EN I  ; ENABLE INTERRUPTS
0135 83 184 RET  ; RETURN FROM SUBROUTINE
0136 67 185 CONTINUE: RRC A  ; ROTATE NEXT BIT INTO TRANSMIT POSITION
0137 00 186 NOP  ; EVEN CYCLE COUNT
0138 8A01 187 MOV R2, #01  ; SET DELAY LENGTH
0139 3441 188 CALL DELAY  ; CALL DELAY SUBROUTINE
013A F432 189 JMP LOOP1  ; CONTINUE LOOP EXECUTION
013B EA41 190 DELAY: DJNZ R2, DELAY  ; VARIABLE DELAY DEPENDING ON R2
013C 83 191 RET  ; RETURN FROM SUBROUTINE

013D

0300 197 ORG 300H
0300 4D455353 198 DB 'MESSAGE ZERO'
0300 41474520
0300 5545524F
0300 28202020
013E 199 ORG 310H
013F 4D455353 200 DB 'MESSAGE ONE'
0310 41474520
0310 4F4E4520
0310 28202020
0311 281 ORG 320H
0312 4D455353 282 DB 'MESSAGE TWO'
0313 41474520
0313 54574F20
0314 28202020
0315 283 ORG 330H
0316 4D455353 284 DB 'MESSAGE THREE'
0317
org 34h
message four

org 35h
message five

org 36h
message six

org 37h
message seven

org 38h
message eight

org 39h
message nine

org 3ah
message ten

org 3bh
message eleven

org 3ch
message twelve
ORG 300H
DB "MESSAGE THIRTEEN"
ORG 3EOH
DB "MESSAGE FOURTEEN"
ORG 3FOH
DB "MESSAGE FIFTEEN"
END

USER SYMBOLS
CONT 003A CONT1 006A CONT2 0113 CONTIN 0139 .
CRFL 0050 DELAY 0141 ESCAPE 0058 EXTINT 0063
INIT 006A INROUT 005B LOCSET 0119 LOOP 0012
LOOP1 0132 MASK 0105 MSKDAT 0100 OUTPUT 012E
QUEFUL 006B START 0031 STROUT 0126 TAB 0114
WAIT 002E

ASSEMBLY COMPLETE. NO ERRORS
APPENDIX 3

ALTERNATE BAUD RATE GENERATION SCHEME
This appendix describes the use of the UPI-41 8 bit timer as an alternate way of generating the bit times for serial transmission.

The UPI-41 timer, figure A3-1, can be programmed to operate in a number of different modes:

1. Internally clocked - the timer counts up one every 32 UPI-41 instruction cycles
2. Externally clocked - the timer count rate is controlled by an external clock source. In this mode the maximum count rate is once every three UPI-41 instruction cycles
3. Set flag on overflow - in this mode the timer flag, TF, will be set when the counter overflows. The timer flag can be tested by conditional branch instructions
4. Interrupt on overflow - an interrupt request is generated on timer overflow. If the timer interrupt is enabled a call to location 7 of program memory is executed

Operating the UPI-41 timer in the "interrupt on overflow mode" permits the UPI-41 to execute a processing task and transmit serial data concurrently. The rationale for operating the UPI-41 in this manner is to increase the system throughput by using the instruction cycles which are wasted in the software timing implementation to do other processing tasks. The following discussion describes some of the analysis necessary to operationalize the above concept within the UPI-41 control program. Two key points will be made: (1) the gain in system throughput for this applica-
External Clock → T1

8-Bit Timer/Counter

÷32

÷15

Timer Flag

Interrupt

UPI-41 Internal Bus

Fig. A3-1. UPI-41 Timer/counter
tion would be very small, and (2) there will be an increase in the probability of data transmission errors from close to zero for the software timing implementation to some unknown but fairly significant probability for the timer/interrupt implementation.

Consider the case where the processor and the timer are operating synchronously at their maximum respective rates. This condition corresponds to a UPI-41 input clock frequency of 6 mega­hertz and a timer input frequency of 133.3 kilohertz, resulting in a UPI-41 instruction cycle period of 2.5 microseconds per instruction cycle and a timer count rate of one count per 3 instruction cycles, or one count per 7.5 microseconds (4). Synchronization of the instruction cycle clock with the timer clock forces all timer interrupts to occur at the same point within an instruction cycle, while operating at the maximum permissible clock rates gives the finest time resolution. Both of these conditions appear desirable for transmission using the timer/interrupt technique.

The timer/interrupt transmission technique works by placing the transmission control loop within the timer interrupt service routine. When a character is available for transmission, it would be placed in a character storage register within the register bank used for the interrupt routine. Character transmission would be initialized by calling the interrupt service routine directly, and the remaining bits of the character would be sent as timer interrupts occurred. There exist two cases where the timer/interrupt technique can be used. The first case requires that there be sufficient processing tasks between all character transmissions so
that the processor does not end up waiting for the previous character to be transmitted. This condition is not met in the present application, as all processing is done before character transmission begins. The second case is where the characters to be output are placed in a queue. In this way the transmission of characters can be independent of the program arrangement. For either implementation the details of the interrupt routine will be essentially identical, with the queued implementation requiring a couple of instructions to check the status of the queue. Assuming either implementation, the details of the interrupt routine can be inspected.

The first issue addressed will be the timing involved, after which a discussion of the number of program steps required for implementation will be presented. For transmission at 19,200 bits per second, the bit transmission time is $1/19,200$, or 52.08 microseconds, corresponding to a time interval in instruction cycles of 52.08 microseconds $\times 1/2.5$ microseconds per instruction cycle, or 20.8 instruction cycles. The interrupt routine must be set up, then, so that bit changes will occur at intervals of 20.8 instruction cycles. As the timer can only generate interrupts in integral multiples of the instruction cycle, the best approximation to this time will be off by .2 instruction cycles. Over a 9 bit character this will result in an error of $9 \times .2$, or 1.8 instruction cycles, corresponding to a time shift in the final bit sent of 1.8 instruction cycles $\times 2.5$ microseconds per instruction cycle, or 4.5 microseconds, see figure A3-2. Since the ADM-3A reads the data bits at their nominal center, however, it is unlikely that this baud rate/interrupt rate
"mismatch" alone would cause data misreads.

**Fig. A3-2. Rate mismatch transmission**

In addition to the rate mismatch error, another error arises due to the inability to predict where within an instruction execution the timer interrupt will occur. Even though an interrupt is constrained, through synchronization with the instruction cycle execution rate, to occur at the same point within an instruction cycle, it is not known whether it will occur during a one cycle or a two cycle instruction. This condition leads to an additional variation of one instruction cycle per bit interval generated. If the timer interrupt routine is set up based on the assumption that all interrupts occur during 1 cycle instructions, and the worst case condition occurs, all interrupts occur during 2 cycle instructions, then a 2.5 microseconds/bit x 9 bits/character, or 22.5 microseconds/character error will result. If the rate mismatch error is included a total error of 22.5 + 4.5, or 27 microseconds/character will occur. As shown in figure A3-3 this condition
would cause a misread of the final bit.

Sampling Clock

Read 1

Read 9

Ideal Transmission

1 2 3 4 5 6 7 8 9

Worst Case Transmission

1 2 3 4 5 6 7 8 9

\[ \rightarrow 2.7 \text{ microseconds} \quad \text{27} \rightarrow \quad \text{microseconds} \]

Fig. A3-3. Serial transmission misread

Conversely, if all interrupts were assumed to occur during 2 cycle instructions, the rate mismatch error would act favorably by reducing the final bit error to \(22.5 - 4.5\), or 18 microseconds, which would appear to be an acceptable error.

Additional analysis which might prove fruitful in reducing the possibility of error would be a statistical study of the program to determine the relative probability of an interrupt occurring within a two cycle instruction versus its probability of occurrence in a one cycle instruction. Once the probabilities were established, the intervals generated by the timer could be set up so that some would assume interrupt occurrence during a 1 cycle instruction and some would assume occurrence during a 2 cycle instruction.

One final approach to eliminating the transmission error
problem might be to step the input crystal frequency down from its maximum value of 6 megahertz to 5.76 megahertz. As this frequency is an integer multiple of the baud rate, the rate mismatch error is eliminated. However, the error due to the variation in interrupt occurrence point becomes slightly larger.

The program steps required to implement the interrupt service routine determine the amount of processing time that will remain available for other processing tasks.

Using the software timing implementation described in section V as a model, it is noted that a minimum of 5 instruction cycles are required to control the character transmission loop. If no additional instructions were required to implement the timer/interrupt scheme, 21-5, or 16 instruction cycles would be available for other processing tasks per bit sent. For the present application, this would allow the processing for several additional inputs to be completed during transmission of the message characters for one. Unfortunately, a considerable number of additional instructions must be executed for the interrupt implementation as shown in the program flow listing below:

1. Jump to interrupt routine - 2 instruction cycles
2. Save accumulator - 1 instruction cycle
3. Load accumulator with character - 1 instruction cycle
   The loop control instructions would be executed at this point
4. Store accumulator into character storage reg. - 1 instruction cycle
5. Stop timer - 1 instruction cycle
6. Load accumulator with timer count - 1 instruction cycle
7. Load timer from accumulator - 1 instruction cycle
8. Start timer - 1 instruction cycle
9. Restore accumulator - 1 instruction cycle
10. Return from interrupt - 2 instruction cycles

These additional steps result in a rather dramatic decrease in the number of instruction cycles available for other processing tasks. Summing the cycle count in the right hand entries, reveals that 12 of the 16 cycles are used. Clearly, under these conditions the timer/interrupt implementation does not appear to be worthwhile.

Some relationship should exist between a processors instruction cycle execution rate, timer count rate, and the desired serial transmission rate which would indicate when it might be cost effective to do the analysis necessary to use a timer/interrupt approach. Additional research on the subject might reveal such a relationship.
LIST OF REFERENCES


17. ISIS-II Prom Mapper Version 2.0. Santa Clara, California: Intel Corporation. (Computer Program)
