ECE 395 — μProcessor Laboratory

Version 1.3

Dr. Sol Rosenstark

Department of Electrical and Computer Engineering
New Jersey Institute of Technology
Newark, New Jersey

Contents

Acknowledgements

The students who finished their projects in the Spring of 1997 helped correct the design as well as compile the parts list. Among them I would like to thank Vinnie Cascio, Aamish Kapadia, Leo Hendriks and Dave Harrison. The latter was also instrumental in modifying Antonakos's monitor so it would function correctly with this design. In addition I would like to thank Eric Staub and Craig Budinich for lending me their boards so that I could spend the 1997 end-of-year semester-break to thoroughly rewrite the monitor and add the features which I think a good monitor requires. I would like to express my appreciation to Idan Mandelbaum for testing out the design with the M68EC000 chip which needed to supplant the now defunct M68008 originally used in this design.

Dave Harrison should be thanked additionally for testing the design described in this manual and for later adding a disassembler to the monitor, a rather challenging task.

Introduction

Since their invention in 1971, microprocessors have been used in the design of all kinds of electrical and other equipment. It is well known that most microwave ovens use microprocessors to read the input keypad and translate this into signals to control the on-time as well as the power of the magnetron, which is the device that supplies the energy used in cooking the food. That is a relatively modern application of microprocessors. In cases of older designs, microprocessor devices replace troublesome mechanical systems. An example of this is found in the ignition system of a modern automobile.

In the good old days, the spark distributor contained centrifugal weights as well as a vacuum diaphragm device to control the ignition spark-advance needed at various engine speeds and loads. This method was very imprecise. Under certain climactic conditions, the spark advance weights got glued in place through the congealing of the very lubricant that was supposed to promote their movement. Vacuum diaphragms punctured with age. This represented a maintenance headache, not to mention the fact that the spark advance curve was limited in the degrees of freedom that could be incorporated into its design.

In a modern vehicle the entire spark advance operation is taken care of by a microprocessor device. There are two electromechanical transduces to inform the microprocessor of the crankshaft and camshaft position. In addition there is data from a throttle position sensor, a mass air flow sensor, an oxygen sensor as well as others too numerous to mention. Using the input data, the microprocessor determines, through a well designed program, the spark timing required. Not only does this design eliminate a number of unreliable mechanical devices, but the efficacy of this design is further enhanced by the fact that the software can be readily modified if it is desired to change the engine operating characteristics. The use of microprocessors thus gives unprecedented flexibility to the engine designer.

It should now be apparent from the above discussion that a microprocessor device uses more than just software. To be useful, it must be interfaced to the real world. It must act on externally gathered data and control the action of some device.

General Objectives of this Laboratory

It is the objective of this laboratory to give the students the opportunity to acquire some hands-on experience with the various workings of a microprocessor. To attain that familiarity, all students will acquire an individual printed circuit board (PCB) version of a microprocessor based single board computer (SBC). After adding a few components and populating the board with chips, they will make it operational by programming an EEPROM with the latest Monitor program. This SBC will have adequate interfacing capabilities to enable the students to perform some useful input/output tasks. Programming of the device will be done in assembly language. Programming in a higher level language would prevent the user from becoming intimately familiar with the microprocessor.

This laboratory is intended to help educate engineers rather than computer scientists. As a consequence, some of the experiments described in this manual deal with software and some deal with interfacing. This is to reinforce the idea discussed in the introduction, that a microprocessor is not of much use if it is not interfaced to the real world.

References

The students should have the first reference listed below from a previous course. The other references are suggested, and may be helpful, but are not absolutely essential.

- James L. Antonakos, *The 68000 Microprocessor, Hardware and Software Principles and Applications,* Fourth Edition, Prentice Hall, 1999. Since it is used as a textbook in our microprocessor course, every student taking this laboratory should have a copy.

- Alan Clements, *68000 Family Assembly Language,* PWS Publishing Company, Boston, 1994. Excellent book when it comes to explaining code. His presentation is a masterpiece on the subject of orderly parameter passing using the LINK instruction.

- Intel Manual, or spec-sheet, for the 8251A PCI (Programmable Communication Interface).

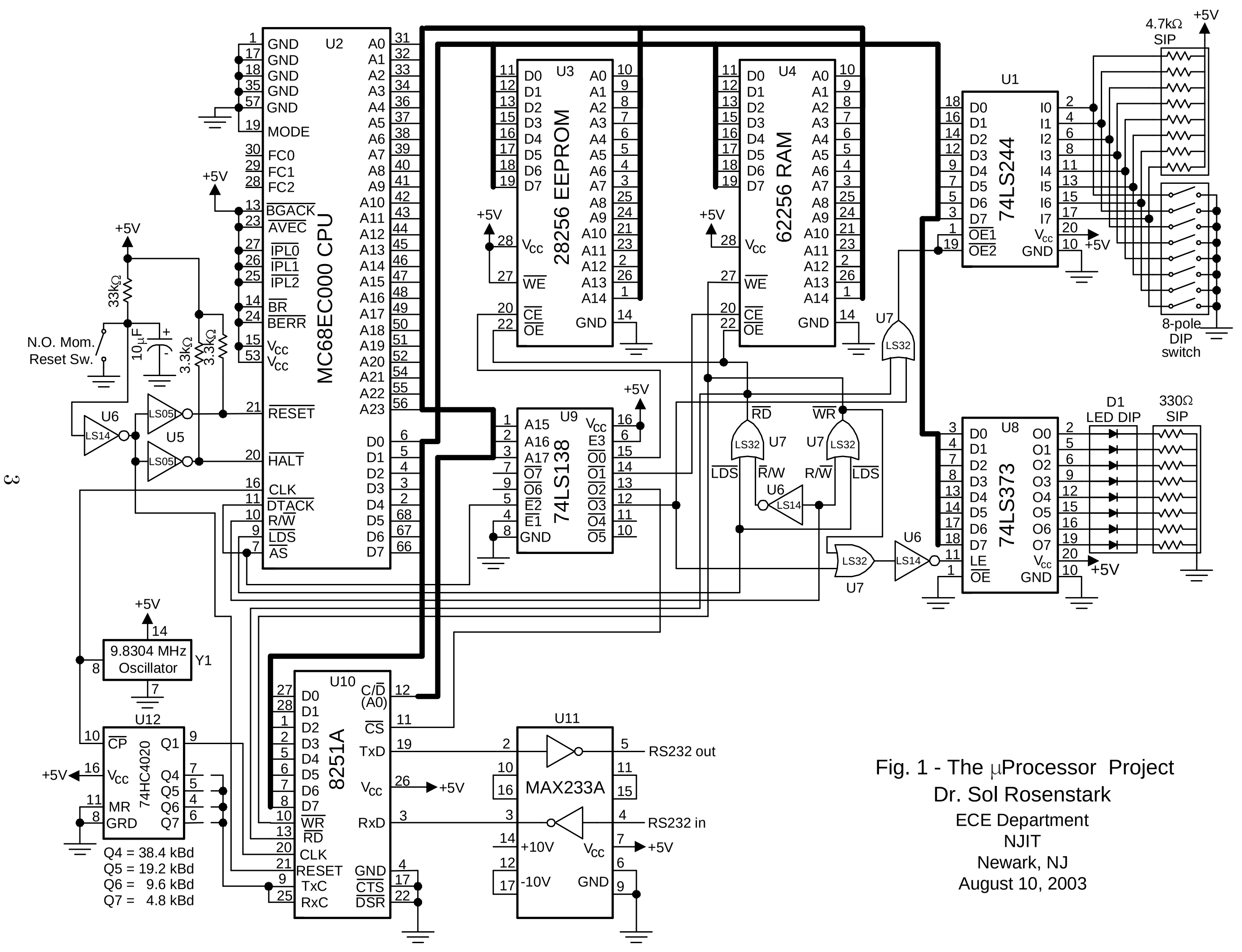

Fig. 1 - The μProcessor Project
Dr. Sol Rosenstark
ECE Department
NJIT
Newark, NJ
August 10, 2003

Explanation of the Single Board Computer (SBC) Design

The single board computer used in this course is built around an MC68EC000 CPU. It is designed to be interfaced to another computer, such as a PC, using a serial port connection. The microcomputer is controlled by a monitor program available on the author's website.

The Motorola MC68EC000 CPU

An MC68EC000 CPU has replaced the originally used MC68008, because the latter went out of production. The MC68EC000 is capable of being operated in both 8-bit and 16-bit mode. We choose to use it as an 8-bit CPU in order to minimize the cost of the SBC. With this choice, the number of ROM and RAM chips is cut by a factor of two.

If the MODE pin on this CPU is grounded then, after the CPU is reset, it wakes up operating in 8-bit mode. Unlike the MC68000, this chip lacks the signals used for interfacing to old M6800 support chips. In particular it lacks the $\overline{\text{VPA}}$ signal, which was also used to inform the CPU that autovectored interrupts were desired. This signal is replaced with the autovectoring signal $\overline{\text{AVEC}}$. In all other respects this chip is identical with the MC68000 CPU.

When the MC68EC000 is operated in 8-bit mode, the $\overline{\text{UDS}}$ signal is of no consequence, but $\overline{\text{LDS}}$ is used to inform the peripheral chips that valid data is present on the data bus during a write cycle, or to inform the peripherals to put data on the data bus during read cycles. For 8-bit operation, an A0 address pin is provided as well.

The 28C256 32 kbyte EEPROM

The 28C256 electrically erasable programmable read only memory (EEPROM) was chosen for its quick reprogramming capabilities. This is in contrast with an equivalent 27256 EPROM which would need to be erased with a UV light eraser before being reprogrammed. It was also chosen because it has an adequate storage capability for the latest version of the monitor program.

The 62256 32-kbyte RAM

The 62256 static random access memory (RAM) chip stores 32-kbytes and is a reasonable choice for maintaining a stack and any reasonably sized programs which can be downloaded from a personal computer (PC).

The 74LS138 3 to 8-line Decoder

The 74LS138 3 to 8-line decoder chip was chosen to give this system a capability of addressing 8 different devices. The decoding used is definitely not unique and all peripheral devices can be found at more than one address.

From the schematic diagram it is clear that the 32-kbyte 28C256 EEPROM is enabled with the $\overline{O0}$ output from the decoder. It, therefore, occupies the memory address space 0 - 7FFFH. The 62256 32-kbyte RAM chip is enabled with the $\overline{O1}$ output from the decoder. It is consequently addressable in the memory address space 8000H - 0FFFFH. Since this decoding is not unique, the chip can be accessed at other memory locations as well.

The 8251A PCI chip is selected with the $\overline{O2}$ output from the decoder. It is also connected to the A0 address line, so it is addressable, non-uniquely, as two memory mapped ports at 10000H and 10001H. The 74LS244 input port chip, as well as the 74LS373 output port chip, are selected with the $\overline{O3}$ output from the decoder. It is clear that both ports are non-uniquely mapped into memory location 18000H. There are additional decoder outputs which are reserved for future expansion.

The Serial Port Interface Chips

An RS-232 communication interface requires a universal asynchronous receiver transmitter (UART) to transform transmitted data from parallel to serial form and received data from serial to parallel form. For our serial interface we use the fast Intel 8251A (programmable communication interface) PCI. Since that chip is not described in Antonakos's book, a later chapter of this manual will deal with it in detail.

A UART deals with data using TTL compatible voltages, that is 0 and 5 V. The RS-232 interface requires that the TTL signals be inverted and converted to a bipolar form with voltages ranging from ±3 V to ±15. Many years ago the very popular 1488 and 1489 chips were used for this purpose. Their disadvantage is that they require the use of additional +12 V and −12 V power supplies. The MAX233A TTL - RS232 interface chip does away with that need in that it generates the +10 and −10 voltages internally.

Most UARTs use a clock signal that is 16 times (16×) the actual bit rate of the RS-232 interface. The I8251A is no exception. In our case the clock signal is provided by dividing the 9.8304 MHz oscillator signal using the 74HC4020 ripple counter. The baud rates available are indicated in the schematic.

The Parallel Port Interface Chips

There are many different ways to create a parallel port interface. Using a sophisticated parallel port chip, such as the MC68230 or the I8255, produces 24 pins of I/O which can be configured in many different ways. This requires that the instructions for the chip be studied and understood in order to learn what data must be written to the chip's registers in order to configure it for the desired operation. The alternative is to use unsophisticated ICs and bypass those difficulties entirely.

The 74LS244 is a tri-state octal buffer chip. When its output is enabled, it transfers the data on the input pins, I0-I7, to the data bus which is connected to pins D0-D7. This data is put on the bus when the output enable pins $\overline{\text{OE1}}$ and $\overline{\text{OE2}}$ on the 74LS244 chip are pulled down. The signal for this purpose is obtained by ORing the 74LS138 decoder output pin $\overline{\text{O3}}$ signal with the $\overline{\text{RD}}$ signal.

The 74LS373 octal transparent latch implements an 8-bit output port. The $\overline{\text{OE}}$ pin is grounded so that the tri-state output is always enabled. This way any data sent to the output port is permanently available to any peripheral device connected to it. The data that is supplied on the D0-D7 pins is loaded into the 8-bit buffer when the latch enable pin on the 74LS373 is pulled up. The signal for this is supplied by a 74LS14 inverter which gets its input from the 74LS138 decoder output pin $\overline{\text{O3}}$ signal which has been ORed with the $\overline{\text{WR}}$ signal.

Now that we have discussed the address decoding for all the chips in this system we are ready to summarize it in a table. Table 1.1 gives the addresses (non-unique) for the various components of the system.

The System Clock

There are numerous ways of constructing precision oscillator circuits. When the price, and trouble taken, are considered it is quickly concluded that the simplest

Table 1.1: Decoding Table for the SBC

Device	address range (hex)
EEPROM MONITOR	0000 - 7FFF
Static RAM	8000 - FFFF
Serial Port Data Register	10000
Serial Port Control/Status Register	10001
‖ Input Port	18000
‖ Output Port	18000

thing to do is to use an integrated oscillator. Such an oscillator possesses only three pins. One is used for connecting to +5 V, one for ground and one for the clock output.

The 9.8304 MHz oscillator output is used to drive the 68EC000 microprocessor directly. It also serves as the input to the 74HC4020 binary ripple counter to produce a number of lower frequencies. Thus the counter's Q1 output is used to clock the UART at a slightly slower 4.9152 MHz. The Q4 output produces a frequency of 614.4 kHz. This is 16 times the frequency needed at the $T \times C$ and $R \times C$ terminals of the UART to get it to communicate at a bit rate of 38.4 kHz. Some other operating bit rates are indicated on the SBC schematic.

The Reset Circuit

To reset the CPU it is necessary to pull down the $\overline{\text{RESET}}$ and $\overline{\text{HALT}}$ pins for at least 100 ms. The CPU begins to execute initialization routines when these pins are subsequently permitted to rise to 5 volts. A single pole, normally open, momentary switch which is buffered by a 74LS14 schmitt inverter is used to reset the CPU. The time constant of the RC circuit which is connected to the switch is 330 ms. The output signal of the 74LS14 schmitt inverter is applied to two 74LS05 open collector inverters. These are used to hold the $\overline{\text{RESET}}$ and $\overline{\text{HALT}}$ pins at low voltage long enough for the power supply voltage to reach 5 volts, thus causing CPU reset on power up.

The $\overline{\text{RESET}}$ and $\overline{\text{HALT}}$ pins are bidirectional. It is therefore essential to keep them independent of each other. That is the reason they are connected in the manner shown. Open collector devices have for output a transistor with the collector only connected to the output pin. When the output is in a high state the output transistor is cut off so that the output terminal is effectively connected to an open circuit. The 3.3 kΩ resistor pulls the terminal up to +5 V. If the CPU then decides to output a voltage on either the $\overline{\text{RESET}}$ or the $\overline{\text{HALT}}$ pin, it can do so without causing any conflicts. This would not be the case with a conventional inverting gate, for example a 74LS04, which has a totem pole output.

Decoding of the $\overline{\text{RD}}$ and $\overline{\text{WR}}$ signals

The two 7432 OR gates combine the CPU's $\overline{\text{LDS}}$ data strobe signal and the CPU RD/$\overline{\text{WR}}$ signal to generate separate $\overline{\text{RD}}$ and $\overline{\text{WR}}$ signals. Those signals are used directly as inputs to the 8251A PCI chip. The $\overline{\text{RD}}$ signal is used to enable the output of the 28C256 EEPROM.

The 62256 RAM chip has different needs. Its output must be disabled when an attempt is made to write to the chip. A destructive conflict could occur if the RAM chip were to put data on the data pins while an external device were to do it at the same time. Connecting the $\overline{\text{RD}}$ signal to the $\overline{\text{OE}}$ pin on the 62256 RAM avoids possible conflicts. The $\overline{\text{WR}}$ signals of the decoder is connected to the $\overline{\text{WE}}$ pin of the RAM and its function is to let this chip know that a memory write is desired.

It is noteworthy that this chip can never be asked to perform a memory read and a memory write simultaneously. The connections are such that the two functions are mutually exclusive.

Supplying the $\overline{\text{DTACK}}$ Response

The $\overline{\text{DTACK}}$ pin must be activated some time after the assertion of the address strobe $\overline{\text{AS}}$ and subsequently deactivated after the negation of $\overline{\text{AS}}$. This pin is read by the CPU one clock cycle after the address strobe $\overline{\text{AS}}$ is activated. With a 10 Mhz clock it means that it is read 100 ns later. Since all of our peripherals are capable of functioning with a CPU clocked at 10 MHz, the simplest thing to do is to connect the $\overline{\text{DTACK}}$ pin directly to the $\overline{\text{AS}}$ signal. This way the $\overline{\text{DTACK}}$ pin is asserted at the same time as the $\overline{\text{AS}}$ signal, and deactivated when the $\overline{\text{AS}}$ goes high.

The 8251A Programmable Communication Interface

This Intel chip is capable of both synchronous and asynchronous bidirectional serial communication hence it is also referred to as a Universal Synchronous-Asynchronous Receiver Transmitter (USART). Synchronous communication can be used if both ends of the connection agree to certain communication protocols. It is more commonplace to use asynchronous communications using a Universal Asynchronous Receiver Transmitter (UART), so our discussion will be confined to that form of operation.

This UART contains two registers addressed as two ports. One port is the command/status register and the other port is the data register. The UART is initialized by writing to the command register. When this port is read it supplies status information. The data port contains the last byte received as well as the byte which is to be transmitted.

If the UART is in the reset state then it expects to be initialized first with a single MODE instruction which can then be followed by any number of COMMAND instructions. The bits of the mode instruction, designated

$$S_2, S_1, EP, PEN, L_2, L_1, B_2, B_1$$

have the following interpretation:

- S_2, S_1 determines the number of stop bits. The choices are the following:
    1. $S_2, S_1 = 00$ is illegal.
    2. $S_2, S_1 = 01$ for 1 stop bit.
    3. $S_2, S_1 = 10$ for $1\frac{1}{2}$ stop bits.
    4. $S_2, S_1 = 11$ for 2 stop bits.
- $EP = 1$ for even parity, $EP = 0$ for odd parity.
- $PEN = 1$ to enable parity, $PEN = 0$ to disable.
- L_2, L_1 determines the data length. The choices are the following:

1. $L_2, L_1 = 00$ for 5 bits.
2. $L_2, L_1 = 01$ for 6 bits.
3. $L_2, L_1 = 10$ for 7 bits.
4. $L_2, L_1 = 11$ for 8 bits.

- If $B_2, B_1 = 00$ then it specifies the SYNCH mode of operation and the preceding MODE bits take on a completely different meaning. We wish to use the asynchronous mode only, in which case these bits specify the frequency of the UART clock in relation to the baud rate. The choices are the following:

  1. $B_2, B_1 = 01$ for a $1\times$ clock speed.
  2. $B_2, B_1 = 10$ for a $16\times$ clock speed.
  3. $B_2, B_1 = 11$ for a $64\times$ clock speed.

Any number of COMMAND instructions can follow the MODE instruction. The bits of the command instruction, designated by

$$X, IR, RTS, ER, SBRK, R \times E, DTR, T \times E$$

have the following meaning:

- The X bit has no use.
- If $IR = 1$ then the UART is reset. This has the same effect as pulling up the UART's reset pin. This command can be issued at any time, but will not have the proper effect if the UART expects a mode instruction.
- $RTS = 1$ makes the $\overline{\text{RTS}}$ pin go to zero. It is used if one prefers to connect the $\overline{\text{RTS}}$ output pin to the $\overline{\text{CTS}}$ input pin to avoid handshaking.
- $ER = 1$ resets all flags in the status register. Has no purpose if flags are not being used.
- $SBRK = 1$ forces the $T \times D$ pin low for an appropriate amount of time thus sending a break signal. Some systems stop what they are doing (such as a screen dump) and return to control mode. This signal is not of much interest nowadays.
- $R \times E = 1$ enables receiving of data.
- $DTR = 1$ makes the $\overline{\text{DTR}}$ pin go to zero. It is used if one prefers to connect the $\overline{\text{DTR}}$ output pin to the $\overline{\text{DSR}}$ input pin to avoid handshaking.
- $T \times E = 1$ enables transmission of data.

Analysis of the above information leads us to the conclusion that, after reset, sending a 4E H MODE byte to the control register should initialize the UART for serial transmission with 1 stop bit, no parity, 8-bit format with a 16× clock. This can be followed by a 27H COMMAND word to the control register to enable transmission, reception, and to pull down the $\overline{\text{RTS}}$ and $\overline{\text{DTR}}$ pins.

Intel suggests that after power is applied one cannot be absolutely sure that the UART is in the reset state before beginning its initialization. It is therefore more prudent to send the bytes 0AA H, 40 H, 4E H and 27 H. If the chip is in the reset state then 0AA H will be taken as a proper mode instruction and 40 H will then be the command instruction telling the UART to enter the reset state. If, on the other hand, the chip is not in a reset state, it will take the 0AA H as a command instruction, which will do no harm, and the subsequent 40 H will be taken as a command instruction, causing it to go into the software reset state. After that, the 4E H and 27 H do their normal job.

The following simple subroutine can be used for the UART initialization

```
dreg    equu    $10000   ; data port
csreg   equ     $10001   ; control status port
;
serinit lea     csreg,a0
; The next 2 lines are to get the UART
; to a reset state in case it has not
; been reset prior to initialization.
        move.b  #$aa,(a0)
        move.b  #$40,(a0)
; Now that we are sure the UART is reset,
; we proceed with a mode instruction to
; obtain operation with 1 stop bit,
; no parity, 8 data bits and a 16x clock.
        move.b  #$4e,(a0)
; COMMAND instruction. RxE = 1 to enable
; reception, TxE = 1 to enable transmission,
; Also make RTS* = 0 and DTR* = 0.
        move.b  #$27,(a0)
        rts
```

When the command register is read it supplies status information. The bits of the status byte are designated by

$$DSR, SYNDET, FE, OE, PE, T \times E, R \times RDY, T \times RDY$$

Only the two least significant bits are of interest to us. They are the following:

- If $R \times RDY = 1$ then there is a new byte of data in the receive buffer.

- If $T \times RDY = 1$ then the transmit buffer is empty and we can go ahead and load a new byte.

The following short subroutines can be used to communicate with the serial port.

```
dreg    equ     $10000  ; data port
csreg   equ     $10001  ; control status port
davbit  equ     2       ; receive data available mask
bfebit  equ     1       ; transmit buffer empty mask
;
; Get a char, mask with 7fH. Char in D1.B
charin  move.b  csreg,d0   ; get status
        andi.b  #davbit,d0 ; Mask for char input status
        beq.s   charin     ; No char, then loop back
        move.b  dreg,d1    ; Get the char
        andi.b  #$7f,d1    ; Get rid of bit 7 as a precaution
        rts                ; Char in d1.b
;
; Send a char from D1.B
charout move.b  csreg,d0   ; get status
        andi.b  #bfebit,d0 ; Mask for buffer empty status
        beq.s   charout    ; Not empty, then loop back
        move.b  d1,dreg    ; Send the char
        rts                ; Done
```

The EEPROM Monitor Program

To get this microcomputer to operate it is necessary to put a monitor program into an EPROM or EEPROM. This design uses a 28C256 EEPROM because it is sufficiently large to accommodate the program that we wish to use as a system MONITOR. This is the monitor program that has been quite thoroughly reworked by Dave Harrison and myself. The program MON210.ASM can be found on my website. Antonakos's latest assembler and emulator are there as well. The monitor program is assembled using Antonakos's assembler. The resultant .HEX file is then used to burn the EEPROM as described in the next section.

Programming the EEPROM Monitor

There are PCs in room 204F, 211F as well as in room 318F, which are interfaced with Xeltek programmer pods. To burn the EEPROM go to the \SP\BIN subdirectory and type SP<ent> then follow the steps below.

- Type <F9> to select the programming of GALs or EEPROMs, and then <esc> to get out.
- Type <F7> and use < * > <ent> to select the device manufacturer, and then <esc> to get out.
- Type <F8> and use < * > <ent> to select the device number, and then <esc> to get out.

The proper data for the EEPROM should now appear in the right middle of the screen. Use the FILE menu to load the .HEX file that you will need to program the chip. Use the BUFFER then EDIT feature to verify the data that has been loaded, assuming that you have taken the trouble to memorize a few bytes of your code.

Go to the DEVICE screen to program the chip. It is self explanatory from here.

If your EEPROM fails to program under its proper brand name, then program it as a XELTEK 28C256 chip. No harm is done since all the programming is done by toggling 5 volts on and off in a specific sequence at various pins.

Parts List

The assembled SBC printed circuit board can be purchsed from ACL Equipment Corporation, Livingston, NJ. Their telephone number is 973-740-9800. The parts list below is furnished for the completeness of this manual. The students can purchse the entire kit from ACL.

Most of the part numbers below correspond to Jameco catalog listings, but any reasonably priced parts supplier will do.

Quan	Part #	Description
	The following solder-tail sockets are required.	
1	152696	68-pin PLCC socket (CPU)
1	102744	28-pin ZIF socket (EEPROM)
2	40336	28-pin socket (UART,SRAM)
4	38631	20-pin socket (MAX233,244,373,DIP LED)
3	37410	16-pin socket (74HC4020,74138, 8-pole DIP switch)
4	62050	14-pin socket (74LS32,74LS14,74LS05,oscillator)
	Miscellaneous additional parts needed.	
1		PCB board
1	104951	DB9S318 female, rt. angle, 9-pin RS-232 socket
1	152346	side entry 2-pin terminal block
1	152354	side entry 3-pin terminal block
1	137672	2.1mm female power jack for PCB
14	15270	0.1 uF bypass capacitors
5	94369	10 uF capacitors, minimum voltage rating 16V
1	24660	4.7 kOhm SIP resistor
1	97851CP	330 Ohm SIP resistor
1	109516	4 rows of 2-contact solder header
1	119010CP	N.O. Mom. PCB switch
1		33 kOhm resistors 1/4W
4		3.3 kOhm resistors 1/4W

ICs needed for populating the PC board.

1		MC68EC000FN10	Motorola CPU	(68-pin)
1		28C256-12	EEPROM	(28-pin)
1		62256-12	Static RAM	(28-pin)
1		NEC8251AFC	NEC UART	(28-pin)
1		MAX233	RS-232 driver & receiver	(20-pin)
1		74LS244	Octal buffer	(20-pin)
1		74LS373	Octal latch	(20-pin)
1		74HC4020	Baud rate generator	(16-pin)
1		74LS138	3 to 8 address decoder	(16-pin)
1		74LS32	Quad 2-input OR gate	(14-pin)
1		74LS14	Hex inverter	(14-pin)
1		74LS05	Hex inverter (OC)	(14-pin)
1		9.8304 MHz	4-pin oscillator	(14-pin)
1	38842	8 pole DIP switch		(16-pin)
1		10-segment LED display R.S. # 276-081, also Digi-Key # P10723-ND		(20-pin)

The items below are needed to make the board functional.

1	199638	Serial cable, straight through, male to female
1	17301CP	5V, 1A regulated DC wall x'former with 2.1mm female plug
4	101282	20 pin Header SIPP sockets with WW bottoms
1	22023	Shorting block (Berg jumper)
1		6.5" X 2.2" Proto-board to fit the mounting hole pattern of 1 7/16" X 6 1/8". The Global Specialties UBS-100 will do nicely as well as one made by E&L Instruments.
4		Threaded feet to support the PC board

Completing the Assembly of the SBC

Table 1.2: Chip Socket Identification Table

Chip	Chip Number
74LS244	U1
68EC000 CPU	U2
28C256 EEPROM	U3
62256 RAM	U4
74LS05	U5
74LS14	U6
74LS32	U7
74LS373	U8
74LS138	U9
8251A UART	U10
MAX233 serial Driver	U11
74HC4020	U12
9.8304 MHz oscillator	Y1
8 pole DIP switch	SW1
10-segment LED display	D1

To complete the assembly of the SBC the student needs to insert all the ICs carefully into their appropriate sockets, if this has not been previously done by the supplier. Correspondence of pin 1 on the chip to that of the socket should be given special attention. The EEPROM containing the MONITOR program goes into the Zero Insertion Force (ZIF) socket. The ZIF socket was provided in case the student should want to add additional software to the EEPROM in the future. Extra care should be exercised installing the MC68EC000 CPU as its socket is turned 90° counterclockwise. To install the chips into their correct sockets consult table 1.2.

Solder into place the three 20 pin header SIPP sockets making sure that the wire wrap pins face down and the socket ends face up. Break one 20 pin header into two equal parts and solder in the two resultant 10 pin headers with the wire wrap pins facing down. Mount the Proto board using the holes provided and install the legs on the board to make sure the wire wrap pins don't get bent out of shape when the printed circuit board is put down.

The board is now ready for use. You need only select a baud rate by installing a jumper on two pins of the 8-pin header, J5, then connect the power supply and the serial communications cable.

Interfacing the SBC to a Personal Computer

The bidirectional serial interface which appears at the bottom of the microcomputer schematic is designed to connect the microcomputer to a personal computer. In this kind of connection we have to make the distinction between Data Terminal Equipment (DTE) and Data Communications Equipment (DCE).

Generally, the equipment that controls the communication is considered DTE and the PC is in that category. A MODEM or a printer with a serial interface is considered DCE. DCE is either the intermediary or terminus for the DTE. The designation has a bearing on the configuration of the serial connector used with each device.

Modern DTE serial ports, as found on the backs of the latest PCs, use a 9-pin D-subminiature (D-sub) male connector. The DTE uses pin 3 of the 9-pin connector for sending data, pin 2 for receiving data and pin 5 for ground.

The SBC is considered DCE. A 3-wire flexible cable, about 4 feet in length, should be used to connect the RS-232 serial signals coming from the MAX233 chip to a 9-pin D-sub female connector. This 9-pin female connector should have pin 3 connected to the RS-232 input on the MAX233 chip, and pin 2 connected to the RS-232 output from that same chip. Pin 5 is connected to ground.

The baud rate can be set at something fast, such as 38,400 baud, and the PC communication software should be set to match that rate. If all is connected correctly then communication should be established immediately.

The most straightforward program to use for communicating with the SBC is HyperTerminal. But it leaves a great deal to be desired, because it is slow in transferring ASCII HEX files from the PC to the SBC. A better choice is the public domain communication program TTermPro (a.k.a. TeraTermPro). It has no shortcomings, but should it prove troublesome in ASCII HEX file transfer then just go to **SETUP**, then **SERIAL PORT** and specify a 30 msec/line time delay. It won't affect the transfer rate noticeably, but will make the file transfer reliable.

Future Uses of the SBC

The Proto board was furnished to facilitate the interfacing experiments in the Microprocessor laboratory. But the SBC can be used for projects which need the decision making power of a microprocessor.

If you decide to use the SBC as part of your senior project then you simply remove the Proto board to gain access to a fairly big wire-wrap area. Wire wrapping is more reliable than using a Proto board for large interfacing projects and you can also get more chips into a given area. Only about 35% of the EEPROM is filled with the MONITOR program. There is plenty of memory left for adding software that is needed to operate devices that are interfaced to the SBC.

You should keep the above options in mind before you consider getting rid of your SBC.

The Experiments

Experiment 1 - Acquiring and Completing the SBC

Once the preceding tutorial material has been read and understood, the students should proceed with the procurement and final assembly of the microprocessor. As described above, this involves the programming of the EEPROM, the installation of the chips, the soldering-in of the wire-wrap headers, the installation of the Proto board and the installation of the feet to support the board. This can all be accomplished before the second meeting of the class.

During each course meeting the instructor should check each students' progress. This is to make sure that procrastinators do not get rewarded with high grades. The idea is to maintain momentum from the first day on. Failure to demonstrate competence in the course through the demonstration of a substantial number of competently completed experiments should result in course failure.

Experiment 2 - Developing Software for the SBC

Although you may have written much of the software that follows in the microprocessor lecture course, you now have the opportunity to test the software in its native environment. Previously you could only test software using a cross-emulator on a PC. It is therefore worthwhile to perform the experiments below in order to acquire greater familiarity with the MC68EC000 microprocessor.

Finding the Position of a Letter in a String

The students should use the Antonakos's cross-assembler to write a program for finding the position of the letter 'x' in any one of the following strings:

1. `I have never seen a better experiment.`
2. `This is an overly short and dull experiment.`
3. `This is a fairly simple but exciting experiment.`
4. `This is an interesting though not long experiment.`
5. `Hmm, this isn't as dull as all that, as experiments go.`

This short program can then be downloaded into the microprocessor and tested.

The string should be stored in the RAM that immediately follows the required program. This is done using the DC.B directive. The position of the letter 'x' in the string should be displayed on the LEDs of the parallel output port.

Now modify the string in the SBC, by using the monitor's EDIT feature, to replace the 'x' with another character. When the string contains no 'x' then 0FFH should be displayed on the LEDs of the parallel output port.

Moving a string in RAM

Write and debug a program for moving a string of bytes of arbitrary length from one memory location to another. The A0 register is initialized to point to the beginning of the data source, the A1 register is initialized to point to the beginning of the data destination, and the D1 register is initialized to contain the count. This program will pick up the data one byte at a time and deposit it one byte at a time. The transfer is complete when the (16 bit) D1 register is down to zero. The program should first fill the destination area with 0FFHs so that the success of the move will be perfectly clear.

Even though your string may be quite short, your software must be prepared to transfer up to 64 kbytes of data.

Addition of Numbers with Keyboard Input

The trap #0 monitor feature can be used to obtain ASCII characters in D1.B from the PC keyboard. Numerical characters (digits), '0' to '9', should be converted to BCD by stripping off the 30 H. Suppose there are 4 BCD digits, e.g. d_4, d_3, d_2, d_1.These can be converted to a HEX number, useful for further arithmetical use, by utilizing the looping routine:

$$\text{HEX number} = [(d_4 * 10 + d_3) * 10 + d_2] * 10 + d_1$$

If, for example, the decimal number typed-in is 8102, then the resultant HEX number should be 1FA6 H

Using the above information, write a subroutine which will accept a decimal number from the keyboard, maximum length of 4 digits, and convert it to a HEX number in a D-register.

The trap #1 monitor feature can be used to send ASCII characters in D1.B to the screen. To display the decimal equivalent of a HEX number on the screen, the decimal digits contained in the HEX number have to be made available. This is done by continuously dividing the number by 10 D. Take, for example, the number 23AF H. We divide it by 10 D to obtain 23AF H / 10 D = 391 H, with a remainder of 5. The 5 is the least significant decimal digit of this number. When this is ORed with 30 H it becomes the ASCII 35 H. The process is repeated so that 391 H / 10 D = 5B H with a remainder of 3, and so on. The result for the above number should be '9135' in ASCII.

Using the above information, write a subroutine which will take a HEX word in a D-register and print the decimal equivalent on the screen.

When the above subroutines are finally available write a program that will perform the addition of two 4 digit numbers obtained from the keyboard and display the result on the screen. For example if you type: 4562 + 9371 = then the program should skip a line and type out on the screen: 4562+9371 = 13933.

Prelab Assignment

Prepare the commented assembly language program well before the experiment is performed, and bring the software to class on a diskette.

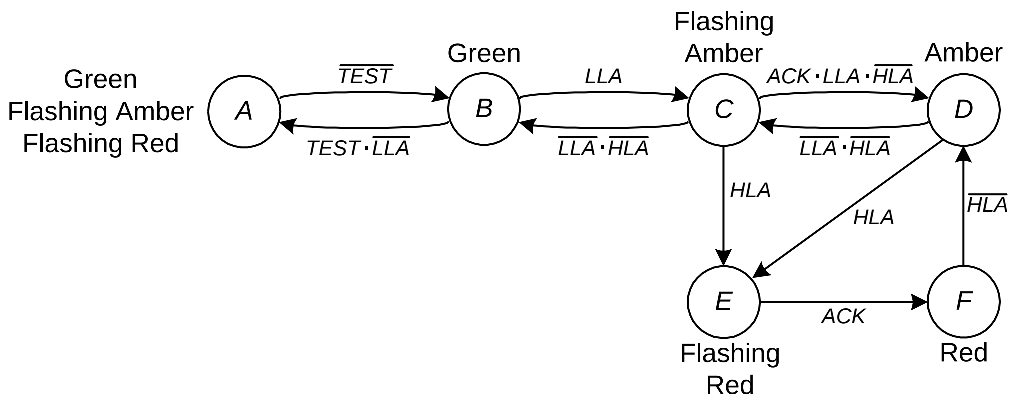
Green
Flashing Amber
Flashing Red
A
$\overline{TEST}$
$TEST \cdot \overline{LLA}$
Green
B
LLA
$\overline{LLA} \cdot \overline{HLA}$
Flashing
Amber
C
$ACK \cdot LLA \cdot \overline{HLA}$
$\overline{LLA} \cdot \overline{HLA}$
Amber
D
HLA
HLA
$\overline{HLA}$
E
Flashing
Red
ACK
F
Red

Figure 1.2: State diagram of the annunciator circuit.

Experiment 3 — An Event Driven Annunciator System

In an event driven sequential device, the next state is determined by the present state of the device as well as by the state of the inputs. A sequential circuit has to have memory devices, such as D or J-K flip-flops as well as supporting combinatorial circuitry.

The difficulty with the design described above is that once it is implemented it is difficult to modify. If a change in the sequence is desired then a new printed circuit board layout may be necessary and perhaps even the addition of more ICs. This difficulty is circumvented by using a microprocessor or a microcontroller at the heart of the design. A change in the design might merely involve a reprogramming of the ROM that controls the device.

The event driven sequential circuit will be implemented in this experiment is not clock driven. It is consequently called a free running circuit because the output responds to an input change and not to a clock. It is clearly the input changes which drive the circuit, so the circuit is called event driven. Other names commonly used are nonpulse circuits or asynchronous circuits.

The state diagram for the sequential circuit that we wish to design is shown in figure 1.2. It is a two alarm system which might be used in a factory to signal that various levels of faults are occurring on the production line. One application may be in process control, where the fault could indicate an elevated pressure. A flashing amber light would indicate the first stage of pressure change, a potential hazard. A large change in the system, shown by a flashing red light would indicate an emergency condition.

The operation of the circuit is best described by the state diagram of figure 1.2. The circuit operates in the following manner:

1. With no fault-signal present the system is stable, it is in state B, and the GREEN light is on.

2. When the signal LLA is present, indicating a minor fault, the state changes to C, a FLASHING-AMBER light comes on and the GREEN light goes off. If the fault disappears ($\overline{LLA}$), the annunciator returns directly to the normal GREEN state.

3. When the system is in the minor-fault (FLASHING-AMBER) state, an operator can intervene to clear the minor fault by pushing the acknowledge button which contains a momentary contact switch. The presence of the ACK signal for a mere fraction of a second changes the system to the steady-AMBER state, telling supervisory personnel that someone is trying to clear the fault. If the minor fault is cleared ($\overline{LLA}$) then the annunciator returns to the normal (GREEN) state after 2 clock pulses.

4. If the system is in the AMBER or FLASHING-AMBER state and the major fault signal (HLA) is received, the system signals a major fault by changing to the FLASHING-RED state. Even if this signal is only momentary, this condition will be maintained indefinitely. The presence of the ACK signal for a mere fraction of a second causes a transition to the steady-RED state, telling supervisory personnel that someone is trying to clear the major fault. If the major fault is cleared ($\overline{HLA}$), the annunciator starts on its path to the normal state and finally attains it if there is no low level alarm ($\overline{LLA}$).

5. A test pushbutton is included to check the condition of all the lights. On $TEST$, the GREEN, FLASHING-AMBER and FLASHING-RED lights should be on. This is the A state.

On the prototyping board interface 3 LEDs with the parallel output port. Keep in mind that LEDs have a constant voltage drop across them of approximately 1.6 volts. A resistor of approximately $330\,\Omega$ must be wired in series with each LED to limit its current to $10\,\text{mA}$. The existing DIP switch can be used for the various input signals. Labeling the switches and LEDs with a small amount of masking tape will go a long way to keeping the final presentation comprehensible.

Write the software required to implement the annunciator circuit. Demonstrate the working model to the instructor.

Prelab Assignment

1. Procure some LEDs, of different colors if possible, as well as some 330Ω resistors.
2. On a diskette, prepare the commented assembly language program to drive the LEDs according to the specifications shown in figure 1.2.

Table 1.3: Stockroom Component Kit

Quantity	Component
1	IC puller
1	IC 311 Voltage Comparator
1	IC 411 Op Amp
1	IC 741 Op Amp
1	IC 7400 Quad 2 input NAND gate
1	IC 7408 Quad 2 input AND gate
1	IC 7432 Quad 2 input OR gate
1	IC 7474 D Flip-Flop
1	IC 7476 J-K Flip-Flop
1	DAC0808 Digital to Analog Converter
2	1.25 kΩ resistor
1	5 kΩ resistor
1	2.5 kΩ resistor
1	15 pF capacitor

Experiment 4 — Testing and Simulating Some ICs

The SBC as an IC Test Jig

The student should obtain from the stockroom the parts kit containing the components listed in table 1.3.

Test the three (3) combinatorial chips using the SBC. The microprocessor should be used to output a voltage test pattern to the chip using the output port, and to read back the chip output using the input port. Pass or fail should be signaled on completion of the test by putting a 0 or 0FF H in D0.B.

The above should be demonstrated for each of the combinatorial chips. A working chip can be made to look defective if one of its input pins is pulled from the socket and left hanging in mid-air. This feature can be used to demonstrate failure of a chip.

The SBC Used to Simulate D and J-K flip-flops

The SBC will be used to simulate a D type flip-flop. The inputs are D and CLOCK and the outputs are Q and $\overline{Q}$. The input should be read when CLOCK is high and should be transferred to the output when the CLOCK signal drops low. The software should be written to guarantee that no false triggering can take place due to a sudden noise spike.

Repeat the above experiment for a J-K type flip flop.

Prelab Assignment

Prepare the commented assembly language program to accomplish the tasks required and bring it to the lab.

Experiment 5 — Testing the Serial Port

Clock Tolerance for the UART

Turn off the SBC power and disconnect the jumper that connects the 74HC4020 baud rate output to the T×C and R×C pins on the 8251A UART. Attach the output of a square wave generator to the T×C and R×C pins of the UART. Using an oscilloscope, adjust the wave generator's frequency to 614.4 kHz. Power up the SBC and load a program whose function is to send repeatedly the ASCII letter 'e' to the screen. The screen should fill up with the letter 'e'.

Increase the frequency output of the generator until the screen output just becomes "flaky." Observe the frequency at which this happens. Repeat the above steps but this time decreasing the frequency output of the generator.

Calculate the positive and negative percentage frequency deviation that can be permitted for the T×C and R×C clock. Can you explain why this is so?

Observation of the Serial Waveform

Again load a program whose function is to send repeatedly the ASCII letter 'e' to the screen. As before, the screen should fill up with the letter 'e'. Attach an oscilloscope to the TRANSMIT terminal of the SBC and observe the waveform. Recall that in normal operation the SBC sends 8 bits, no parity and 1 stop bit.

In the monitor program you can find the routine SERINI. It is used for initializing the UART for 8 bits, no parity and 1 stop bit operation. It is reproduced below for your convenience, along with a routine to waste a little bit of time, to give the initialization a chance to take place before the serial port goes into use. You need only add the few lines of code to the routine SEND.

```
csreg   equ     $10001 ;control/status port
        org     $8000
serexp  bsr.s   serini
        bsr.s   waste
; Your program for sending an 'e' repeatedly
; goes here
send    ---
        ---
serini  movea.l #csreg,a0
        move.b  #$aa,(a0)
        move.b  #$40,(a0)
        move.b  #$4e,(a0)
        move.b  #$27,(a0)
        rts
waste   move.b  #100,d3
w1      nop     ;Waste some time to make sure
        nop     ;reinitialization takes hold
```

```
subq.b  #1,d3
bne.s   w1
rts
end     serexp
```

- Loading and execution of the above SEREXP program will again allow you to observe the waveform at the SBC's TRANSMIT terminal. It should be the same as that observed previously.
- In the part of this manual entitled "The 8251A Programmable Communication Interface," you will find an explanation of the UART initialization routine SERINI. Refer to it to determine how to modify the initialization routine for sending 7 bits, no parity and 1 stop bit. Execute the SEREXP program with this modification and again observe the waveform at the SBC's TRANSMIT terminal. Sketch it and determine if it corresponds to what can be expected. You'll have to reset the SBC, and perhaps reload the communication program to return the operation of both to normal.
- Repeat the above experiment, but this time modify the initialization routine for sending 7 bits, odd parity and 1 stop bit. Again observe the waveform at the SBC's TRANSMIT terminal. Sketch it and determine if it corresponds to what can be expected. As above, you'll have to reset the SBC, and perhaps reload the communication program to return the operation of both to normal.

Prelab Assignment

Type up the software needed for this experiment and bring to the lab on a diskette.

Experiment 6 — DAC Interface with the SBC

Objectives

This experiment is used to familiarize students with interfacing computers to the analog world. This is necessary because in the real world there are multitudes of analog devices which need to be interfaced to digital equipment. To interface computers and digital devices to the analog world we commonly use A/D (analog to digital) and D/A (digital to analog) converters. The A/D converter takes an analog signal and converts it to a digital value. The output is the ratio of the input voltage to the reference voltage of the A/D converter. The D/A converter does the opposite. It takes the digital value and generates an analog voltage which is proportional to a reference voltage.

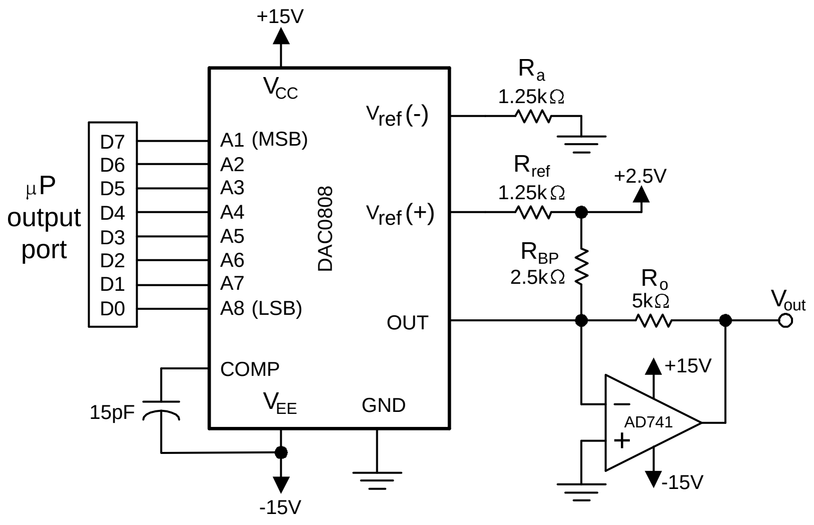

Figure 1.3: The DAC connection to the microprocessor board.
R_{BP} is used only in the bipolar configuration.

In this experiment a DAC0808 8-bit digital to analog converter is interfaced to the microprocessor parallel I/O output port as shown in figure 1.3. This setup will then be used for the generation of square, sine and triangular waveforms.

The D/A converter used in this experiment is the DAC0808 included in the stockroom parts kit mentioned in experiment 4, table 1.3. The specification sheets can be obtained from the *National Semiconductor Linear Data Book* available in the stockroom.

Prelab Assignments

1. Prepare commented assembly language programs to generate the waveforms in the following section. Prepare flow charts for these programs. Do not replicate common portions of the code or flow charts.

2. Compute the current flow out of the DAC of figure 1.3 when it sees the following data bytes:

   (a) 30H

   (b) 0A1H

   (c) 0FAH

   Assume that the reference voltage $V_{ref} = 5.0$ V and that $R_{ref} = 5\text{K}\Omega$.

Lab assignments

Connect the DAC and associated components to the output port of the microprocessor. Write programs to output a binary sequence so as to generate:

1. A Square Wave (both unipolar and bipolar)

2. A Sawtooth Wave (both unipolar and bipolar)

3. A Sine Wave (bipolar only)

The waves should be periodic, with a period controllable by a scale stored in a specific RAM address. Select a reasonable range of frequencies, taking into account the dynamic properties of the DAC and of your microcomputer. Display the output waveforms on an oscilloscope.

Experiment 7 — The Logic Analyzer

Objectives

The objective of this experiment is to familiarize students with the use of a logic analyzer for trouble shooting digital devices. This is to be accomplished through an examination of the bus signals of the SBC.

Introduction

The instrument that is used a great deal in the troubleshooting of electrical circuitry is the oscilloscope. It permits the viewing of waveforms in great detail, determining rise-times and fall-times of pulse waveforms and the viewing of any kind of analog signals. Oscilloscopes are capable of displaying two traces at once, and this is not too useful in studying the behavior of digital circuits. This is where the logic analyzer becomes very useful.

This piece of test equipment can display multiple signals at once, but it must be understood that it does not give a detailed presentation of the signal. When you read the logic analyzer manual you discover its limitations. It squares up all waveforms that it is examining. It looks for transitions of digital states from low to high and conversely. Thus, a sinewave will look like a square wave on a logic analyzer. So it should not be used for the study of analog waveforms. Its strong point is its ability to display multiple waveforms when troubleshooting digital circuitry.

The Experiment

Obtain one of the four logic analyzers from the stockroom. You'll find that it has 4 pods with 16 leads each. The SBC signals that the analyzer needs to see are brought out to the single row 20-pin headers that you installed earlier on the board. Hook-up leads to the clock signal CLK, the 8 data bus signals D7–D0, the 15 address bus signals A14–A0, $\overline{\text{RD}}$, $\overline{\text{WD}}$, and $\overline{\text{AS}}$.

An examination of the monitor program of the SBC will reveal that it is usually busy getting a keyboard command. The GETLIN routine is responsible for this. It loops to the label GETCHAR within that routine. Setting the analyzer to trigger on the HEX address of GETCHAR will cause it to capture the bus signals of interest.

- Compare the waveforms obtained with the timing waveforms for the **synchronous read** appearing in the Supplementary Notes for the prerequisite micrprocessor lecture course.

- Have the logic analyzer show the data obtained in HEX number form. See if it corresponds to the listing of the GETCHAR routine.

Write a short program with an endless loop for writing a word of some data continuously to address `$8100`. Set the analyzer to trigger at the start of this loop.

- Compare the waveforms obtained with the timing waveforms for the **synchronous write** appearing in the Supplementary Notes for the prerequisite micrprocessor lecture course.
- Have the logic analyzer show the data obtained in HEX number form. See if it corresponds to the listing of the GETCHAR routine.
- Since the CPU is operating in 8-bit mode, writing a word to memory should require two physical memory write accesses. See if this is indeed the case.

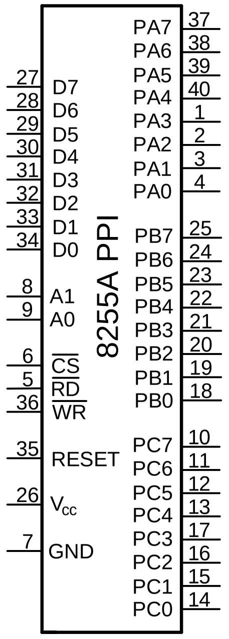

Fig. A.1 - 8255 pinout

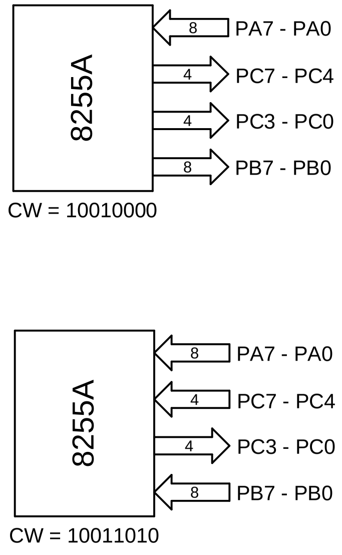

Fig. A.2 - Two examples of mode 0

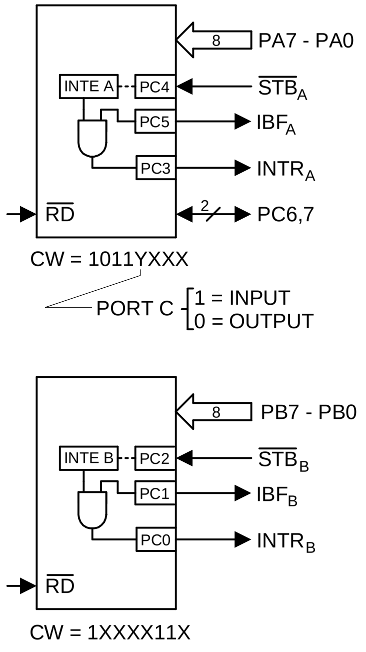

Fig. A.3 - Two examples of mode 1 INPUT

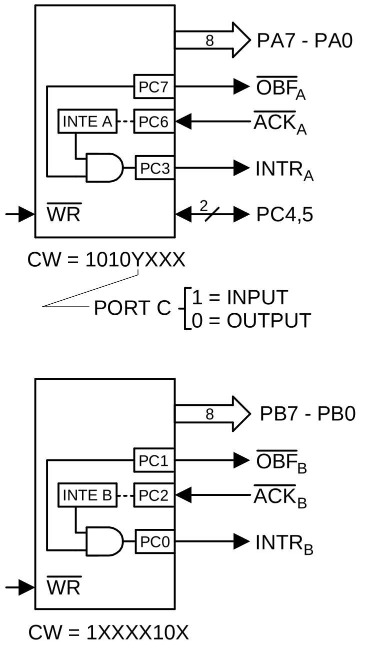

Fig. A.4 - Two examples of mode 1 OUTPUT

INPUT CONFIGURATION

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
I/O	I/O	IBF_A	$INTE_A$	$INTR_A$	$INTE_B$	IBF_B	$INTR_B$
GROUP A					GROUP B		

OUTPUT CONFIGURATION

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
$\overline{OBF}_A$	$INTE_A$	I/O	I/O	$INTR_A$	$INTE_B$	$\overline{OBF}_B$	$INTR_B$
GROUP A					GROUP B		

Fig. A.5 - Mode 1 Status Word Format

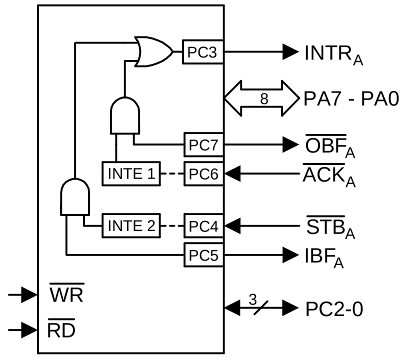

Fig. A.6 - Mode 2

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
$\overline{OBF}_A$	$INTE_1$	IBF_A	$INTE_2$	$INTR_A$	☒	☒	☒
GROUP A					GROUP B		

Fig. A.7 - Mode 2 Status Word Format

Dr. Sol Rosenstark
ECE Department
NJIT
Newark, NJ
June 10, 1997

Appendix A — Optional Parallel Port Expansion

Introduction

The parallel interface discussed so far was based on the 74373 and the 74244 chips. That is an inflexible design inasmuch as the direction of the ports cannot be changed and also because no handshaking features are available. If additional parallel interface capability is required then it is better to add a chip that has the features which were missing in the previous design. The Intel 8255A Programmable Peripheral Interface (PPI) packs a lot of versatile parallel interface into one 40-pin chip. Its pinout is shown in figure A.1.

The PPI has 24 pins of parallel I/O which can be configured in many different ways. These pins are divided into two basic groups, group A and group B. Group A consists of port A and the upper part of port C (bits 4 - 7). Group B consists of port B and the lower part of port C (bits 0 - 3). The assignment of the pins of port C to the two groups changes slightly with the modes of operation.

The ports of this chip can be operated in a number of different modes. In mode 0 the port A, port B, and the two halves of port C can be set up individually for either input or output. In mode 1 each group can be set up for either strobed input or strobed output, with some pins of port C used for handshaking. In mode 2, which is only available in group A, port A can be used for strobed bi-directional data transfer, with 5 pins of port C used for handshaking.

Interfacing to the CPU

The PPI can be interfaced to either 8-bit or 16-bit CPUs. It is easiest to discuss the operation of the PPI in connection with an 8-bit CPU. The question of how this chip is interfaced with a 16-bit CPU, such as the Motorola 68EC000 operating in 16-bit mode, is left as an exercise to the reader.

The PPI has 8 bidirectional data pins, D0 - D7, which are connected directly

Table 1.4: Decoding Table for the PPI

Register	Pin Designation	Address
Port A	PA0 – PA7	20000 H
Port B	PB0 – PB7	20001 H
Port C	PC0 – PC7	20002 H
Control Word Register (CWR)		20003 H

to the CPU data bus. This chip has separate $\overline{\text{RD}}$ and $\overline{\text{WR}}$ pins, which in our microprocessor project would be interfaced with the two signals with that same designation, derived from the 74LS32 OR-gates. The RESET pin would be tied to the RESET signal used for the 8251A serial interface chip.

The chip select signal $\overline{\text{CS}}$ can be connected to one of the 74LS138 decoder outputs. If it were tied to the output $\overline{\text{O4}}$ then it would be decoded starting at 20000 H. The device has two pins marked A0 and A1. With our CPU these pins can be connected to the correspondingly marked CPU address bus pins. The PPI is then addressed according to the decoding table 1.4.

Selecting the Operating Modes of the PPI

As mentioned in the introduction, the ports of the PPI are divided into two groups. Port A and the upper part of port C (bits 4 - 7) belong to group A. Port B and the lower part of port C (bits 0 - 3) belong to group B. The association of the parts of port C to the two groups is rather loose and bit 3 is shifted to group A when that group is used in mode 1 or mode 2.

Assigning Modes and Port Directions

The device can be operated in three modes. The modes of operation are selectable by writing a control byte to the CWR register. The bits of the control byte have a specific grouping. The control byte bits, designated

D7, D6, D5, D4, D3, D2, D1, D0

are used to select the following modes of operation:

- D7 is the mode set flag. It must be 1 to activate mode setting.
- D6, D5 are used to select the mode of operation of group A. They are used as follows:
  1. D6, D5 = 00 selects mode 0.
  2. D6, D5 = 01 selects mode 1.
  3. D6, D5 = 1X selects mode 2.
- D4 determines the direction of port A. Input = 1 and Output = 0.

- D3 determines the direction of the upper half of port C. Input = 1 and Output = 0.
- D2 is used to select the mode of operation of group B. D2 = 0 selects mode 0 and D2 = 1 selects mode 1. This group has no mode 1.
- D1 determines the direction of port B. Input = 1 and Output = 0.
- D0 determines the direction of the lower half of port C. Input = 1 and Output = 0.

Individual Bit Set/Reset Capability of Port C

In setting the mode of operation of the PPI we needed to write to the CWR using D7 = 1. Writing to the CWR with D7 = 0 can be used to set or reset specific bits of port C, one bit at a time. This assumes that port C is being used for output and not for input. If a group is in mode 0 then its portion of port C can be written to directly. In the two other modes the only way to change the output bits of port C is to use the bit set/reset method mentioned above.

To use the port C individual bit set/reset method, the control byte bits, designated

D7, D6, D5, D4, D3, D2, D1, D0

are used as follows:

- We need D7 = 0 for port C bit setting to take place.
- D6, D5 and D4 are "don't cares."
- D3, D2 and D1 determine which bit of port C will be affected, with 000 designating bit 0, 001 designating bit 1 and so on.
- The value of D0 determines the value that the bit will take.

It should be noted that when ports are used for output, the last value written to them can be read back. This does not apply to the CWR.

The Operating Modes of the PPI

In the previous section it was explained how to obtain the various modes of operation. Now we need to become familiar with how those modes affect the operation of the ports of the PPI. But before we proceed, a few important observations are in order.

If any group is programmed in mode 0 for output, then the data can be sent to those pins by simply writing to port C. This is not true if a group is programmed in any mode other than mode 0. If it is desired to write any data to port C, when the group to which that port belongs is in mode 1 or 2, then this must be done on a bit by bit basis using the bit set/reset method. Hence if pins PC6,7 are used for output, in one of the two higher modes, then the data must be sent to them using the bit set/reset method.

In any mode, if any pins of port C are used for input then the data can be obtained by simply reading port C.

Mode 0 — Simple Input/Output

In mode 0 the ports are used for input or output without any handshaking. As an example, writing the control word CW = 10010000 to the CWR will cause mode setting to take place, and configure group A for input in mode 0. The upper part of port C will be configured for output. This CW will also configure group B for output in mode 0. The lower part of port C will be configured for output. This port configuration is demonstrated in figure A.2.

As another example, writing the control word CW = 10011010 to the CWR will configure group A for input in mode 0. The upper part of port C will also be configured for input. Group B will also be configured for input in mode 0. The lower part of port C will be configured for output. This port configuration is also demonstrated in figure A.2.

Mode 1 — Strobed Unidirectional Data Transfer

In mode 1 the ports are used for input or output with handshaking. If one group is used in mode 1, the other group can be used in any available mode. In mode 1 the five pins PC3-7 are assigned to group A. This leaves PC0-2 of port C assigned to group B. In figure A.3 we see how to configure group A and group B in mode 1 for input. In figure A.4 we see how to configure group A and group B in mode 1 for output. It is important to note that if group A is used on mode 1, then two pins of this group are left over for simple I/O. When group B is used in mode 1, however, then the three pins of lower port C are all assigned to handshaking duty. This is clearly demonstrated in figures A.3 and A.4.

The handshaking for the upper diagrams of figure A.3 and figure A.4 will be explained in detail. The other two cases can then be understood without further explanation.

When group A is used for input in mode 1, as is the case for the upper diagram in figure A.3, then the external device checks the “Input Buffer Full” pin, $\mathrm{IBF}_{\mathrm{A}}$. If this pin is low, indicating that the input buffer is empty, then the external device puts a byte on pins PA7-PA0 and strobes it in by momentarily pulling down the $\overline{\mathrm{STB}}_{\mathrm{A}}$ pin. The $\mathrm{IBF}_{\mathrm{A}}$ goes high and stays in that state until the data is read by the CPU. If the INTE A bit is set then the $\mathrm{INTR}_{\mathrm{A}}$ pin also goes high. This can be used to interrupt the CPU and to inform it to call an interrupt routine to read port A. The INTR_A stays high until the data is read by the CPU at which point the input cycle can be repeated.

The CPU can keep itself informed of the progress of the data transfer by reading port C. The bit patterns for INPUT and OUTPUT are shown in figure A.5. If the IBF bit is set then the CPU knows that a new byte is waiting in the input buffer and that it can be read.

When group A is used for output in mode 1, as is the case for the upper diagram in figure A.4, the CPU reads the status word in port C and determines if the “Output Buffer Full” $\overline{\text{OBF}}_\text{A}$ bit is inactive high. If that is the case then it can write a byte to port A. This causes the $\overline{\text{OBF}}_\text{A}$ status bit to become active low signaling the CPU that the port A buffer is full. In addition the $\overline{\text{OBF}}_\text{A}$ pin on the PPI goes low.

The external device checks the $\overline{\text{OBF}}_\text{A}$ pin. If this pin is low, indicating that the output buffer is full, then the external device reads the data and momentarily strobes low the $\overline{\text{ACK}}_\text{A}$ pin. In response to this the $\overline{\text{OBF}}_\text{A}$ pin goes high informing the peripheral that there is no new data for it to read. The corresponding $\overline{\text{OBF}}_\text{A}$ status bit also goes high informing the CPU that the data has been read by the peripheral. If the INTE A bit is set then the INTR_A pin also goes high. This can be used to interrupt the CPU and to inform it to call an interrupt routine to send a new byte to port A.

In figures A.3 and A.4 the INTE flip-flops are shown connected with dashed lines to the bit in port C that controls them. Thus, for setting INTE A, for the case shown in the upper diagram of figure A.3, it is necessary to use the bit set feature of the CWR to write a 1 to PC4. Reading of port C produces the status information shown in figure A.5 and the status of the INTE bits can be verified this way.

Mode 2 — Strobed Bidirectional Data Transfer

Mode 2 can be used only for group A for bidirectional data transfer with full handshaking. Group B can be used in mode 0 or mode 1 at the same time. The utilization of the pins is demonstrated in figure A.6. Careful examination of this figure reveals that it is composite of the upper two diagrams of figures A.3 and A.4. The handshaking which was explained for mode 1 applies here as well. The only difference is that there are two INTE flip-flops for the INTR_A pin. One controls the interrupt enabling for output, the other for input.

As in the case of mode 1, reading of port C produces the status information shown in figure A.7. If interrupts are used for both input and output, the fact that there is only one INTR_A pin means that the CPU must consult the status word to determine whether an input or output should be performed.