Microcontroller based System Design

Objective:

The objective of this project is to design a Microcontroller based trainer kit with provisions for Port Access, Serial and Parallel Interfacing with the PC. An onboard ADC is also provided.

Microcontroller Overview:

The main difference between a Microprocessor and a Microcontroller is that a Microcontroller has inbuilt peripherals like timers, USART, Interrupt Controller etc. whereas in a Microprocessor these have to be interfaced as separate IC's. In this project we use the Microcontroller AT89C51 from the ATMEL Corp. The AT89C51 is a low-power, high-performance CMOS 8-bit microcomputer with 4Kbytes of Flash programmable and erasable read only memory (PEROM). The device is manufactured using Atmel's high-density nonvolatile memory technology and is compatible with the industry-standard MCS-51 instruction set and pinout. The on-chip Flash allows the program memory to be reprogrammed in-system or by a conventional nonvolatile memory programmer. By combining a versatile 8-bit CPU with Flash on a monolithic chip, the Atmel AT89C51 is a powerful microcomputer which provides a highly-flexible and cost-effective solution to many embedded control applications. Some of the features of the AT89C51 Microcontroller are listed below.

Features

• Compatible with MCS-51™ Products

• 4K Bytes of In-System Reprogrammable Flash Memory

– Endurance: 1,000 Write/Erase Cycles

• Fully Static Operation: 0 Hz to 24 MHz

• Three-level Program Memory Lock

• 128 x 8-bit Internal RAM

• 32 Programmable I/O Lines

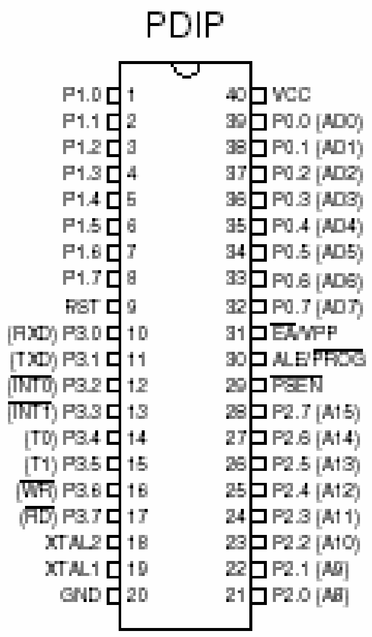
• Two 16-bit Timer/Counters

• Six Interrupt Sources

• Programmable Serial Channel

• Low-power Idle and Power-down Modes

Pin Configuration:



Pin Description:

VCC Supply voltage.

GND Ground.

Port 0

Port 0 is an 8-bit open-drain bi-directional I/O port. As an output port, each pin can sink eight TTL inputs. When 1s are written to port 0 pins, the pins can be used as high impedance inputs. Port 0 may also be configured to be the multiplexed low order address/data bus during accesses to external program and data memory. In this mode P0 has internal pull-ups. Port 0 also receives the code bytes during Flash programming, and outputs the code bytes during program verification. External pull-ups are required during program verification.

Port 1

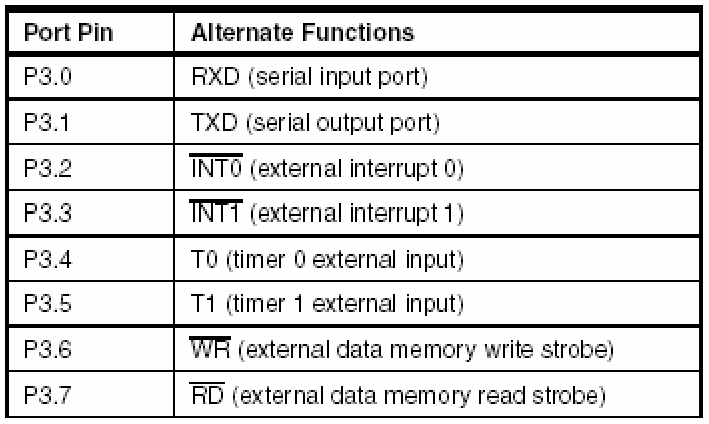
Port 1 is an 8-bit bi-directional I/O port with internal pull-ups. The Port 1 output buffers can sink/source four TTL inputs. When 1s are written to Port 1 pins they are pulled high by the internal pull-ups and can be used as inputs. As inputs, Port 1 pins that are externally being pulled low will source current (IIL) because of the internal pull-ups. Port 1 also receives the low-order address bytes during Flash programming and verification.

Port 2

Port 2 is an 8-bit bi-directional I/O port with internal pull-ups. The Port 2 output buffers can sink/source four TTL inputs. When 1s are written to Port 2 pins they are pulled high by the internal pull-ups and can be used as inputs. As inputs, Port 2 pins that are externally being pulled low will source current (IIL) because of the internal pull-ups. Port 2 emits the high-order address byte during fetches from external program memory and during accesses to external data memory that uses 16-bit addresses (MOVX @ DPTR). In this application, it uses strong internal pull-ups when emitting 1s. During accesses to external data memory that uses 8-bit addresses (MOVX @ RI), Port 2 emits the contents of the P2 Special Function Register. Port 2 also receives the high-order address bits and some Control signals during Flash programming and verification.

Port 3

Port 3 is an 8-bit bi-directional I/O port with internal pull-ups. The Port 3 output buffers can sink/source four TTL inputs. When 1s are written to Port 3 pins they are pulled high by the internal pull-ups and can be used as inputs. As inputs, Port 3 pins that are externally being pulled low will source current (IIL) because of the pull-ups. Port 3 also serves the functions of various special features of the AT89C51 as listed below:



Port Pin	Alternate Functions
P3.0	RXD (serial input port)
P3.1	TXD (serial output port)
P3.2	$\overline{\text{INT0}}$ (external interrupt 0)
P3.3	$\overline{\text{INT1}}$ (external interrupt 1)
P3.4	T0 (timer 0 external input)
P3.5	T1 (timer 1 external input)
P3.6	$\overline{\text{WR}}$ (external data memory write strobe)
P3.7	$\overline{\text{RD}}$ (external data memory read strobe)

Port 3 also receives some control signals for Flash programming and verification.

RST

Reset input. A high on this pin for two machine cycles while the oscillator is running resets the device.

ALE/PROG

Address Latch Enable output pulse for latching the low byte of the address during accesses to external memory. This pin is also the program pulse input (PROG) during Flash programming. In normal operation ALE is emitted at a constant rate of 1/6 the oscillator frequency, and may be used for external timing or clocking purposes. Note, however, that one ALE pulse is skipped during each access to external Data Memory. If desired, ALE operation can be disabled by setting bit 0 of SFR location 8EH. With the bit set, ALE is active only during a MOVX or MOVC instruction. Otherwise, the pin is weakly pulled high. Setting the ALE-disable bit has no effect if the microcontroller is in external execution mode.

PSEN

Program Store Enable is the read strobe to external program memory. When the AT89C51 is executing code from external program memory, PSEN is activated twice each machine cycle, except that two PSEN activations are skipped during each access to external data memory.

EA/VPP

External Access Enable. EA must be strapped to GND in order to enable the device to fetch code from external program memory locations starting at 0000H up to FFFFH. Note, however, that if lock bit 1 is programmed, EA will be internally latched on reset.

EA should be strapped to VCC for internal program executions. This pin also receives the 12-volt programming enable voltage (VPP) during Flash programming, for parts that require 12-volt VPP.

XTAL1

Input to the inverting oscillator amplifier and input to the internal clock operating circuit.

XTAL2

Output from the inverting oscillator amplifier.

Analog to Digital Converter Overview:

8bit µP Compatible A/D Converter(ADC0804):

General Description:

The ADC0804 is CMOS 8-bit successive approximation A/D Converter. This ADC appear like memory locations or I/O ports to the microprocessor and no interfacing logic is needed. Differential analog voltage inputs allow increasing the common-mode rejection and offsetting the analog zero input voltage value. In addition, the voltage reference input can be adjusted to allow encoding any smaller analog voltage span to the full 8 bits of resolution.

Features:

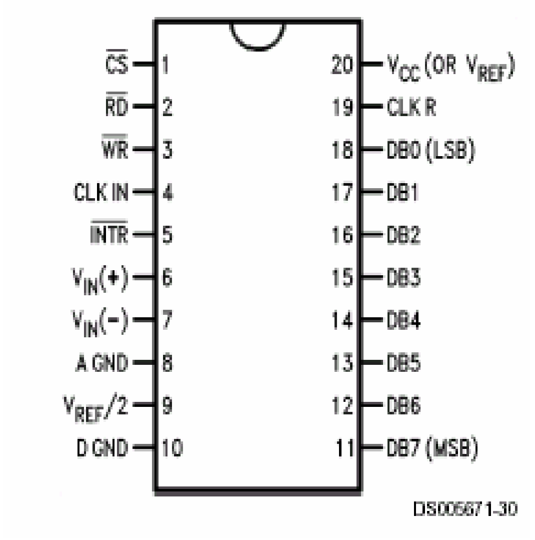
- Compatible with 8080 µP derivatives—no interfacing logic needed - access time - 135 ns
- Easy interface to all microprocessors, or operates “stand alone”
- Differential analog voltage inputs
- Logic inputs and outputs meet both MOS and TTL voltage level specifications
- Works with 2.5V (LM336) voltage reference
- On-chip clock generator
- 0V to 5V analog input voltage range with single 5V supply
- No zero adjust required
- 0.3" standard width 20-pin DIP package
- 20-pin molded chip carrier or small outline package

- Operates ratio metrically or with 5 VDC, 2.5 VDC, or analog span adjusted voltage reference

Key Specifications:

- Resolution 8 bits
- Total error ±1/4 LSB, ±1/2 LSB and ±1 LSB
- Conversion time 100 μs

PIN Configuration:

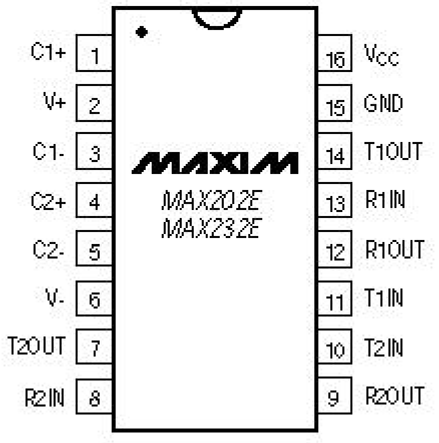


Serial Port Interfacing Overview:

MAX 232 IC Overview:

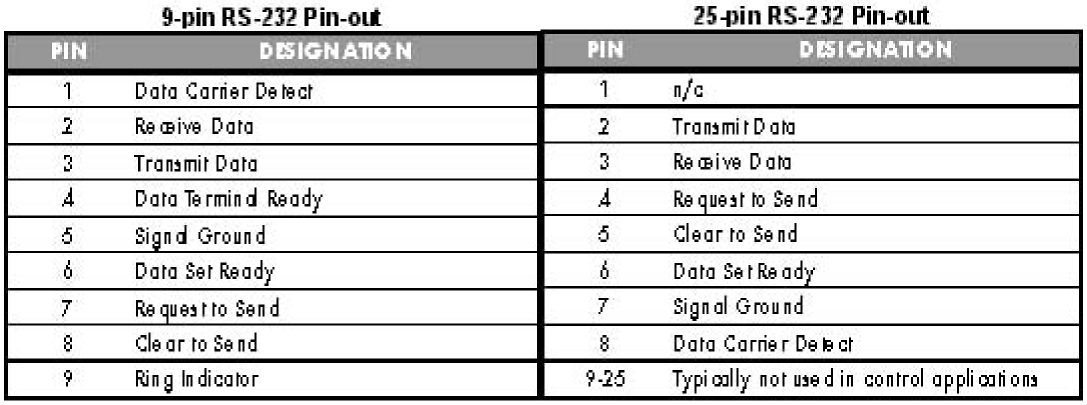
The 89C51 has a built in serial port that makes it very easy to communicate with the PC's serial port but the 89C51 outputs are 0 and 5 volts and we need +10 and -10 volts to meet the RS232 serial port standard. The easiest way to get these values is to use the MAX232. The MAX232 acts as a buffer driver for the processor. It accepts the standard digital logic values of 0 and 5 volts and converts them to the RS232 standard of +10 and -10 volts. It also helps protect the processor from possible damage from static that may come from people handling the serial port connectors. The MAX232 requires 5 external 1uF capacitors. These are used by the internal charge pump to create +10 volts and -10 volts. It includes 2 receivers and 2 transmitters so two serial ports can be used with a single chip.

Pin Configuration:



RS-232 Pinouts

Standard RS-232 pin-outs for IBM compatible computers are shown below. There are two configurations that are typically used: one for a 9-pin connector and the other for a 25-pin connector.



9-pin RS-232 Pin-out

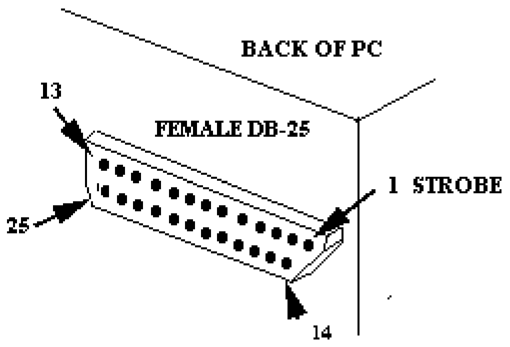
PIN	DESIGNATION
1	Data Carrier Detect
2	Receive Data
3	Transmit Data
4	Data Terminal Ready
5	Signal Ground
6	Data Set Ready
7	Request to Send
8	Clear to Send
9	Ring Indicator

25-pin RS-232 Pin-out

PIN	DESIGNATION
1	n/c
2	Transmit Data
3	Receive Data
4	Request to Send
5	Clear to Send
6	Data Set Ready
7	Signal Ground
8	Data Carrier Detect
9-25	Typically not used in control applications

The 25 pin connector has some pins that are not used for data transmission. These pins are mainly used for loop-back testing of the port. Note that the designations for pins 2 and 3 on the 9-pin connector are the exact opposite of what they are for pins 2 and 3 on the 25-pin connector.

Parallel Interfacing Overview:



The Parallel Port is the most commonly used port for interfacing home made projects. This port will allow the input of up to 9 bits or the output of 12 bits at any one given time, thus requiring minimal external circuitry to implement many simpler tasks. The port is composed of 4 control lines, 5 status lines and 8 data lines. It's found commonly on the back of your PC as a D-Type 25 Pin female connector. There may also be a D-Type 25 pin male connector. This will be a serial RS-232 port and thus, is a totally incompatible port.The parallel port has 4 function types for a total of 25 pins: data (8 pins), control (4), status (5) and ground (8). To understand the function of the data, control and status types, consider what happens when you print something on your printer. The printer prints out alphanumeric characters onto paper (thus using the data lines). Sometimes it does a carriage return and linefeed (hence using the control lines). Sometimes, the printer doesn't print because you ran out of paper, or you forgot to have the printer on-line (status lines). Thus the printer has a number of input and output related function types. The 8 data lines are used for 8 digital OUTPUT lines. For example, you can turn on 8 different motors. The 5 status lines are used for 5 digital INPUT lines. Thus you can interface 5 different sensors, like pushbuttons. The 4 control lines can be used for 4 additional digital output lines (thus 4 more motors!).

PARALLEL PORT PINOUT

PIN NO.	FUNCTION	TYPE
1	STROBE	CONTROL
2	DATA BIT 0	OUTPUT
3	DATA BIT 1	OUTPUT
4	DATA BIT 2	OUTPUT
5	DATA BIT 3	OUTPUT
6	DATA BIT 4	OUTPUT
7	DATA BIT 5	OUTPUT

8	DATA BIT 6	OUTPUT
9	DATA BIT 7	OUTPUT
10	ACKNOWLEDGE	STATUS
11	BUSY	STATUS
12	PE: PAPER TRAY EMPTY	STATUS
13	PRINTER ON-LINE	STATUS
14	AUTO LINEFEED AFTER (CR) CARRIAGE RETURN	CONTROL
15	PRINTER ERROR	STATUS
16	INITIALIZE PRINTER	CONTROL
17	SELECT/DESELECT PRINTER	CONTROL
18-25	UNUSED/GROUND	

Baud Rate:

In addition to voltage ranges, RS-232 Protocol identifies how fast data is transmitted between two ports. This transmission speed is defined as Baud Rate—roughly equivalent to the number of bits transmitted per second. Typically, the baud rate will vary between 1200 to 19200. Common baud rates are as follows: 1200, 2400, 4800, 9600, and 19200. Notice that each baud rate indicated sequentially here is twice that of the previous baud rate.

Introduction to the General Purpose Microcontroller Interface Board

The interface board that has been developed has the following extensions

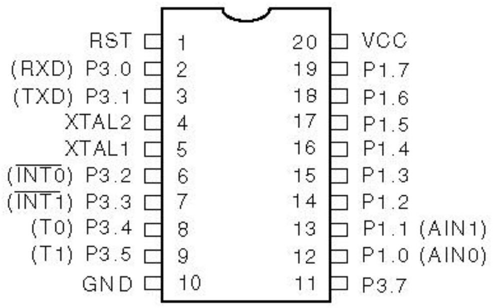
- Communicating with the PC using Parallel Port
- Serial Port Communication
- ADC Interfacing
- Microcontroller Ports available for further interfacing
- Plug and Play Microcontroller feature

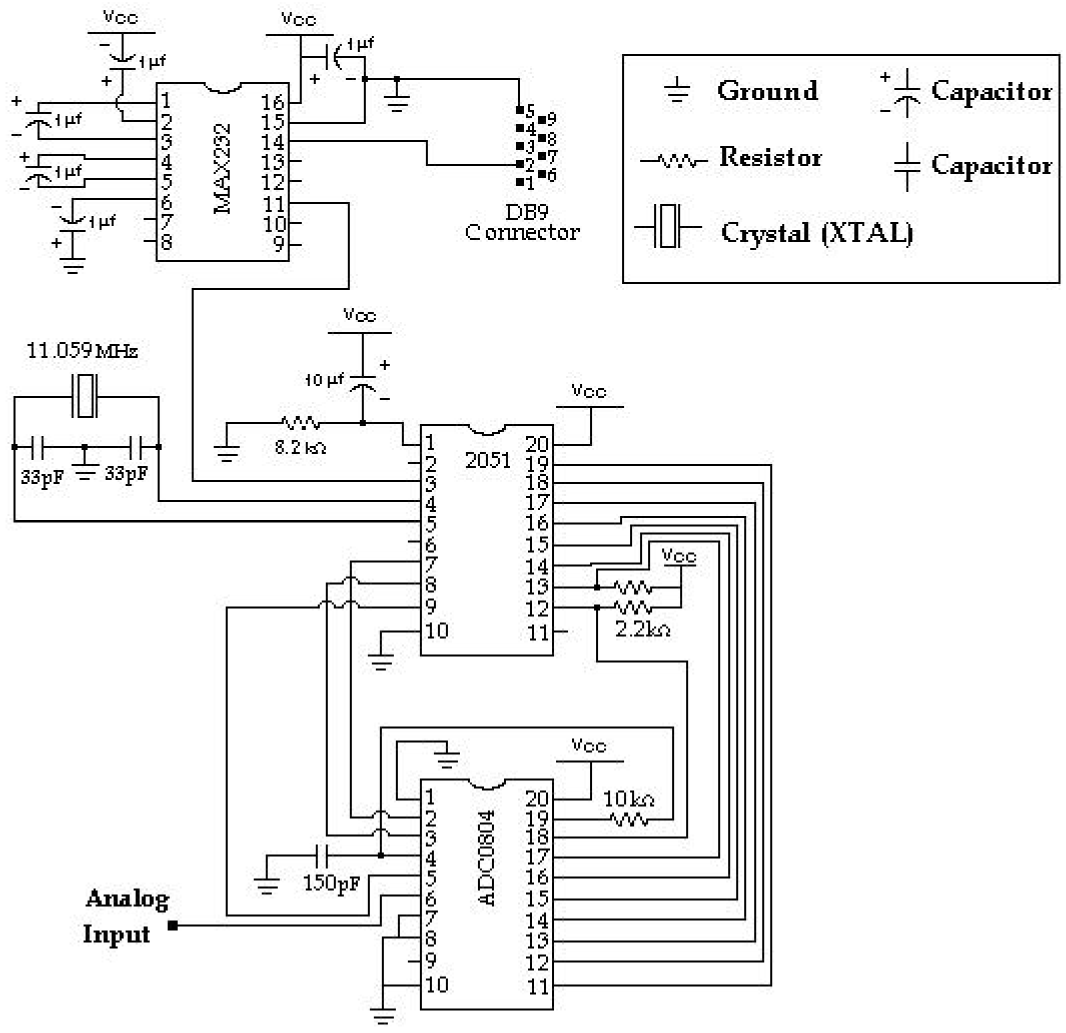
Circuit Diagram and Description:

The Interface that has been developed has features for accessing the four ports of that are present in the microcontroller and using them to drive any circuit such as a Stepper motor driver, DC motor driver etc. that can be built on a Bread Board and interfaced with the Interface board.

There is also provision for receiving and transmitting data via the parallel port of the PC and use this data to control any circuit or processing on the PC. The Microcontroller’s port 2 has been interfaced with the DB 25 connector. Therefore care should be taken while programming that any data to be processed or received must be such that it is sent through PORT 2.

An on board ADC is interfaced to the PORT 1 of the Microcontroller as shown below in the fig. (Remember that 2051 used in the fig is a 20 pin microcontroller and hence the corresponding pins in 8051 must be looked into). The pin outs of the 2051 is shown below





The above circuit explains both the concepts of interfacing an ADC as well as sending the aquired data to the PC through the PC's Serial Port (DB 9). It uses an ADC0804 chip to convert from analog to digital, an AT89C2051 microcontroller to control the ADC0804 and send data to the PC, and a MAX232CPE chip to convert the signals from and to RS232 levels for sending and receiving from the PC.

The easiest way to do analog to digital conversion is to use an IC such as the ADC0804 that does the work for you. The analog voltage is applied to pin 6 and the result is available at pins 11 through 18. We will connect pin 1 (Chip Select) to ground so that the chip is always enabled. (If you wanted to use more than one ADC you could use this pin to control which chip is currently enabled). Connect pin 7 (Vin -) to ground. The ADC0804 includes an internal oscillator which requires an external capacitor and resistor to operate. Connect the 150 pF capacitor from pin 4 to ground and the 10k ohm resistor from pin 4 to pin 19.

To control the ADC0804, we will use 3 lines from the 2051. Connect pin 2 (Read) from the ADC0804 to pin 7 (P3.3) of the 2051. Connect pin 3 (Write) to pin 8 (P3.4). Connect pin 5 (Interrupt) to pin 9 (P3.5). The 8 bit Output Data from the ADC0804 will be connected to Port 1 of the 2051. The 2051 pins 12 and 13 do not have internal pull up resistors so external pull up resistors are required.

Conversion Process stages:

The 2051 controls the analog to digital conversion process. The conversion process has several stages.

*Stage 1)*To trigger a new conversion, we must make pin 3 (Write) low and then return it to the high state. The conversion process starts when Write goes high (rising edge triggered).

Stage 2) When the conversion process is complete, pin 5 (Interrupt) will go low. *Stage 3)* When we see pin 5 (Interrupt) go low; we must make pin 2 (Read) low to load the new value into the outputs D0 - D7.

Stage 4) Next we read the values into the 2051 Port 1.

Stage 5) Finally, we return pin 2 (Read) to the high state. The next conversion can be started immediately.

The 2051 has a built in serial port that makes it very easy to communicate with the PC's serial port but the 2051 outputs are 0 and 5 volts and we need +10 and -10 volts to meet the RS232 serial port standard. The easiest way to get these values is to use the MAX232. The MAX232 acts as a buffer driver for the processor. It accepts the standard digital logic values of 0 and 5 volts and converts them to the RS232 standard of +10 and -10 volts. It also helps protect the processor from possible damage from static that may come from people handling the serial port connectors. The MAX232 requires 5 external 1uF capacitors. These are used by the internal charge pump to create +10 volts and -10 volts. Most computers use a 9 pin DB9 male connector so a 9 pin female connector has been selected.

Programming the Microcontroller

To program a microcontroller one must have two basic tools. One is the hardware-programmer board and the second is the software-for downloading the program. The hardware programmer board is used to hole the chip and enable the computer to download the program into the micro. The programmer board is serially connected to the computer.

The soft wares you will need to program your micro will be two fold. One is the compiler and the other to download the program written. The compiler program automatically checks for errors in your program and then compiles it into a hex file. This hex file is stored in the same directory in which you have stored your original program file. The compiler software used be us is ASEM software. The downloading software used here is the UNIPROG. The uniprog can be run in only the DOS mode. This is very user friendly software and can be operated easily. This helps in erasing previous files and downloading new programs into the Micro. We will help you out how to program the micro with step by step descriptions as follows.

Steps involved in writing a program to the Micro:

1> Write the required program in the Notepad of windows with the syntaxes being followed as per the rules of programming.

2> Save the written file in **.asm** extension. Now the assembly file has been created.

3> Next run this file in ASEM software by specifying the path of the file.

4> After this the ASEM software will check for errors and then convert this into **.hex** file.

5> Now run the system in the Dos Mode.

6> Next run the UNIPROG software

7> It automatically checks the IC placed in the board.

8> Then specify the hex file to be downloaded.

9> If any program is in the IC,you can erase it.

10>Now download the program and hence you are ready to work with the IC.

UNIPROG is an open engine for support any hardware chip programmer. It's implement an open architecture based in hardware-abstract algorithms and chip definitions, that can work in any hardware. With Uniprog the development of new and powerful programmers is not limited by lack of software or the need of test chips.

ASEM-51 is a two-pass macro assembler for the Intel MCS-51 family of microcontrollers. It is running on the PC under MS-DOS, Windows and Linux. The DOS real-mode assembler ASEM.EXE requires only 256 kB of free DOS memory and MS-DOS 3.0 (or higher). The new protected-mode assembler ASEMX.EXE requires a 286 CPU (or better), and at least 512 kB of free XMS memory. The new Win32 console-mode assembler ASEMW.EXE requires a 386 CPU (or better) and Windows 9x, NT, 2000 or XP. The new Linux assembler asem requires a 386-based Linux system. The new HTML documentation set requires a 90 MHz Pentium (or better) and a web browser. The ASEM-51 assembly language is a rich subset of the Intel standard that guarantees maximum compatibility with existing 8051 assembler sources. ASEM-51 can generate two sorts of object files: Intel-HEX format, which is directly accepted by most EPROM programmers, and absolute OMF-51 format, which is required for many simulators, emulators and target debuggers. Thus ASEM-51 is suitable for small and medium MCS-51-based microcontroller projects in hobby, education and business. However, ASEM-51 has been designed to process also very large programs!

Features:

- fast, compact, reliable, easy to use, and well-documented
- easy installation, almost no configuration required
- command line operation, batch and networking capability
- DOS (RM and PM), Win32 and Linux binaries available
- Intel-compatible syntax
- five location counters, one for each of the MCS-51 address spaces
- assembly-time evaluation of arithmetic and logical expressions

Sample Experiments

1. Serial Interfacing:

As explained in the previous pages the serial interfacing involves a USART and a Level Shifter. The USART is built in the 8051 and we use the MAXIM IC 232 for level shifting. The Connector is used to connect the PC (in this example) or any RS232 compatible device.

Assembly Program:

```
        MOV  TMOD,#20H ;timer 1, mode 2
        MOV  TH1,#-3   ;9600 baud
        MOV  SCON,#50H ;8-bit, 1 stop bit, REN enabled
        SETB TR1       ;start timer 1
AGAIN:
        MOV  A,#"E"
        ACALL TRANS
        MOV  A,#"C"
        ACALL TRANS
        MOV  A,#"E"
        ACALL TRANS
        SJMP AGAIN     ;keep doing it
;serial data transfer subroutine
TRANS:
        MOV  SBUF,A    ;load SBUF
HERE:
        JNB  TI,HERE   ;wait for last bit to transfer
        CLR  TI        ;get ready for next byte
        RET
        END
```

The above code transmits the word “ECE” to the PC through the serial port at a baud rate of 9600bps. [1] The code is written is any text editor and compiled using a 8051 Assembler software and downloaded onto the Microcontroller.

2. ADC Interfacing:

The interfacing circuit is explained in the circuit description along with the steps necessary to initiate and perform Analog to Digital Conversion. In the program code given below the analog input is converted and the value is sent to the PC through the serial port.

Program Code:

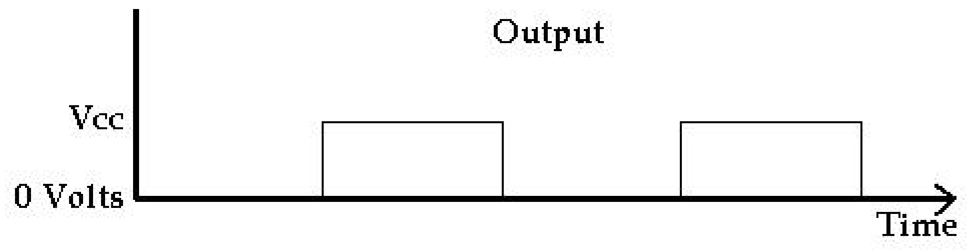
```
; Port1 - ADC Result (Input)

; P3.5 - ADC Complete (Input Active Low)

; P3.4 - Start ADC (Output Active Low)

; P3.3 - Load values to output (Output Active Low)

;***********************************************************************

; RESET                    ;reset routine

ORG   0H                 ;locate routine at 00H

 AJMP   START              ;jump to START

;***********************************************************************

  ORG   25H                ;locate beginning of rest of program

;***********************************************************************

INITIALIZE:                 ;set up control registers

   MOV   TH1, #0FDH          ;Set up for 9600 baud rate

   MOV   SCON, #01010000B     ;Mode = 8 bit UART
  MOV   TMOD, #00100001B      ;Sets Timer1 to 8 bit auto reload
  MOV   TCON, #01000000B      ;Turns Timer1 on
  RET
;***********************************************************************
;      Real code starts below.
;***********************************************************************
; This routine transmits the value in A through the serial port.
SEND:
  CLR     TI            ;Clear Timer1 Flag
  MOV     SBUF, A        ;Transmit Byte
WAIT:
  JNB     TI, WAIT       ;Wait for transmission to be completed.
  RET
;***********************************************************************
START:                         ;Main program (on power up, program jumps to this point)
  MOV   SP, #030H             ;Set Stack Pointer to 30H
  ACALL INITIALIZE             ;Set up control registers
  ORL P1,#0FFH                 ;Make Port 1 an input port
  SETB P3.5                    ;Make P3.5 an Input pin
LOOP:
  CLR P3.4                     ;Prepare for Analog to Digital Conversion
  SETB P3.3                    ;Set Read to High
  SETB P3.4                    ;Start Analog to Digital Conversion
```

```
WAITFORDONE:
  JB P3.5, WAITFORDONE       ; Wait until Conversion is complete
  CLR P3.3                   ; Load 8 bit value to ADC0804 Outputs
  MOV A,P1                   ;Read ADC0804 Output
  ACALL SEND                 ; Send Value to PC
  SETB P3.3                  ; Disable ADC0804 Output
  AJMP LOOP                  ; Go to LOOP (jump back to point labeled LOOP)
  END                        ; End program
```

3. Speed Control of DC Motor (using PWM):

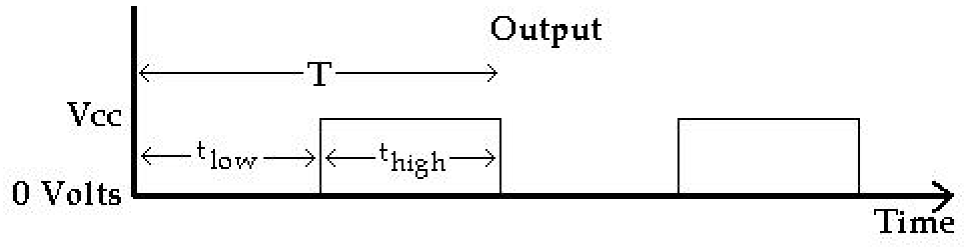
Principle of PWM:

It is easy to use a microcontroller to turn LED's on and off in almost any pattern you want. But you can only turn the LED on and off. So what if you want to control the brightness of the LED? The same problem comes up in robotics where you want to control the speed of a motor with a microcontroller. It is not good enough to just turn the motor on and off. To control the brightness of the LED or speed of the motor you have to control the amount of current going through the device. One solution that may occur to you is to quickly turn the LED or motor on and off. The current only flows when the output is low (for microcontrollers LED circuits are usually wired so current flows into the microcontroller when the output is low. The output of your microcontroller will look like the following square wave.

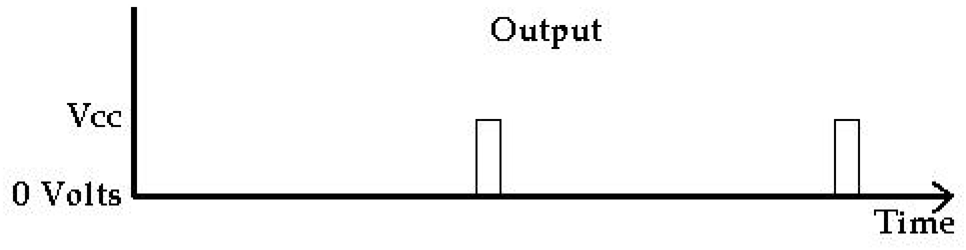


If you turn an LED or motor on and off fast enough then it will appear to stay on continuously and since there is less current flowing overall the LED will appear less bright and the motor will run at a slower speed. With this solution you can make the LED flash on and off as slow as 30 times a second but any slower and you start to see the LED blinking which is not the desired result. Or, for the motor, it will lose its smooth operation and get jerky. The solution does not work very well because the LED is still rather bright at 30 times a second.

Rather than changing the number of times the output goes on and off, we change how long the output stays on and off. Let's take a closer look at one output cycle. An output cycle consists of a low period, tlow and a high period, thigh. tlow + thigh = T, where T is the period (length of time) for one output cycle. thigh is also called an output pulse, or just pulse.

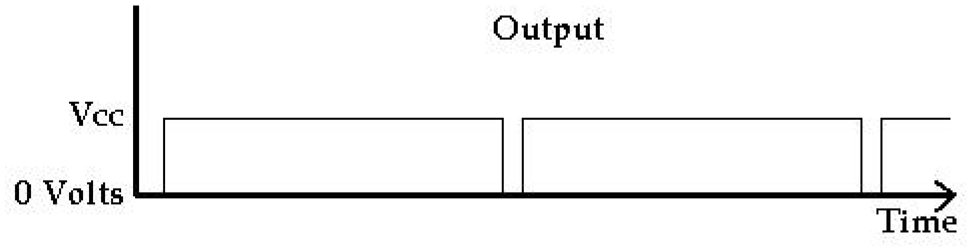


We will always keep T the same so that there is always the same number of output cycles per second. If we increase the width of thigh then we must decrease tlow to keep T the same. If we decrease thigh then we must increase tlow. For the case that we make thigh small then the output looks like the following.



You can see that the output is 0 most of the time and the LED or motor will be on most of the time.

For the case that we make thigh large then the output looks like the following.



The output is Vcc most of the time which turns off the LED. The current only flows through the LED for the brief time that the LED is on during tlow. But since we are still turning the LED on and off very fast (we will use about 100 times a second in the examples below), you can not see the LED blinking and it appears very dim. The total current that flows through the LED is low. For the motor it will smoothly turn at a low speed. So we can control the brightness of the LED or the speed of a motor by changing the width of thigh. This is the secret of Pulse Width Modulation

Program Code:

```
;*********************************************************************
;RESET                    ;reset routine

   ORG   0H                ;locate routine at 00H

    AJMP   START             ;jump to START

    ORG   25H               ;locate beginning of rest of program

INITIALIZE:                ; set up control registers

   MOV A,#0FFH

   MOV P1,A

   MOV TMOD,#00H            ; set timer 0 to Mode 0 (8 bit Timer with 5 bit prescalar)
   SETB TR0                ; turn on timer 0

   MOV PSW,#00H

   SETB EA                      ; Enable Interrupts (each individual interrupt must also be enabled)

   SETB ET0                ; Enable Timer 0 Interrupt

   RET

; The LED is off during the high section and on during the low section of each cycle

; The Flag F0 is used to remember whether we are timing tlow or thigh.

TIMER_0_INTERRUPT:

   JB F0, HIGH_DONE        ; If F0 is set then we just finished the high section of the

LOW_DONE:                  ;   cycle so Jump to HIGH_DONE

   SETB F0               ; Make F0=1 to indicate start of high section

   SETB P2.3             ; Turn off LED

   MOV TH0, R7                ; Load high byte of timer with R7 (our pulse width control value)

   CLR TF0               ; Clear the Timer 0 interrupt flag

   RETI                ; Return from Interrupt to where the program came from

HIGH_DONE:

   CLR F0                ; Make F0=0 to indicate start of low section

   CLR P2.3              ; Turn on LED
```

```
  MOV A, #0FFH            ; Move FFH (255) to A
  CLR C                   ; Clear C (the carry bit) so it does not affect the subtraction
  SUBB A, R7              ; Subtract R7 from A. A = 255 - R7.
  MOV TH0, A              ; so the value loaded into TH0 + R7 = 255
  CLR TF0                 ; Clear the Timer 0 interrupt flag
  RETI                    ; Return from Interrupt to where the program came from
START:                    ;main program (on power up, program starts at this point)
  ACALL INITIALIZE        ;set up control registers
LOOP:
  MOV R7,P1               ; set pulse width control to dim
  AJMP LOOP               ;go to LOOP(always jump back to point labeled LOOP)
END                       ;end program
```

Appendix A

Net list and Schematic

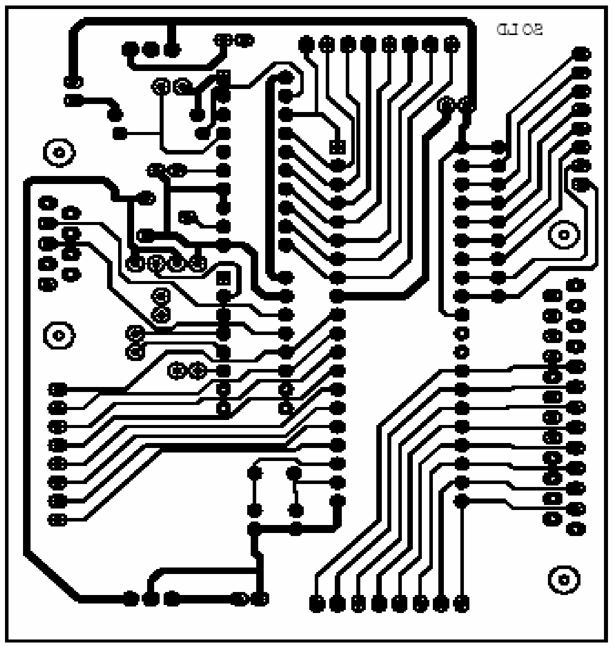
```
NETS

+5V        = C8/1     RN1/9     SW1/2     U1/31     U1/40

             U6/2;

+5VMAX     = C6/2     C7/1      R3/1      U3/20     U4/16

             U5/2;

GND        = C1/2     C2/2      C3/1      C5/1      C7/2

             J5/2     J6/2      J7/2      P1/5      P3/18

             P3/19    P3/20     P3/22     P3/23     P3/24

             P3/25    R1/2      R3/3      U1/20     U3/1

             U3/7     U3/8      U3/10     U4/15     U5/3

             U6/3;

N1         = C1/1     R2/1      U3/4;

N2         = R2/2     U3/19;

N3         = C3/2     U1/18     Y2/1;

N4         = C2/1     U1/19     Y2/2;

N5         = C5/2     U4/6;

N6         = C9/2     U4/3;

N7         = C6/1     U4/2;

N8         = C4/1     U4/4;

N9         = C9/1     U4/1;

N10        = C4/2     U4/5;

N11        = P1/2     U4/14;
```

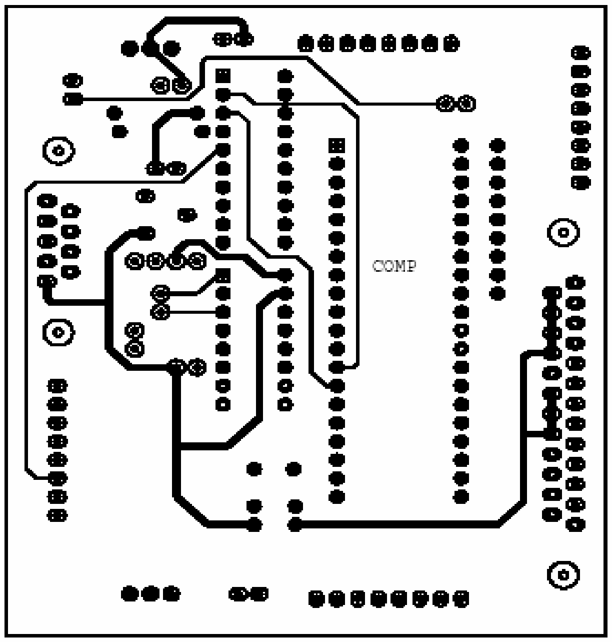
```
N12       = P1/3    U4/13;

N14       = J1/2    RN1/7    U1/38;

N15       = J1/8    RN1/1    U1/32;

N16       = J1/4    RN1/5    U1/36;

N17       = J1/5    RN1/4    U1/35;

N18       = J1/7    RN1/2    U1/33;

N19       = J1/3    RN1/6    U1/37;

N20       = J1/1    RN1/8    U1/39;

N21       = J1/6    RN1/3    U1/34;

N22       = C8/2    R1/1     SW1/1    U1/9;

P10       = J2/1    U1/1     U3/18;

P11       = J2/2    U1/2     U3/17;

P12       = J2/3    U1/3     U3/16;

P13       = J2/4    U1/4     U3/15;

P14       = J2/5    U1/5     U3/14;

P15       = J2/6    U1/6     U3/13;

P16       = J2/7    U1/7     U3/12;

P17       = J2/8    U1/8     U3/11;

P20       = J3/1    P3/2     U1/21;

P21       = J3/2    P3/3     U1/22;

P22       = J3/3    P3/4     U1/23;

P23       = J3/4    P3/5     U1/24;

P24       = J3/5    P3/6     U1/25;

P25       = J3/6    P3/7     U1/26;

P26       = J3/7    P3/8     U1/27;
```

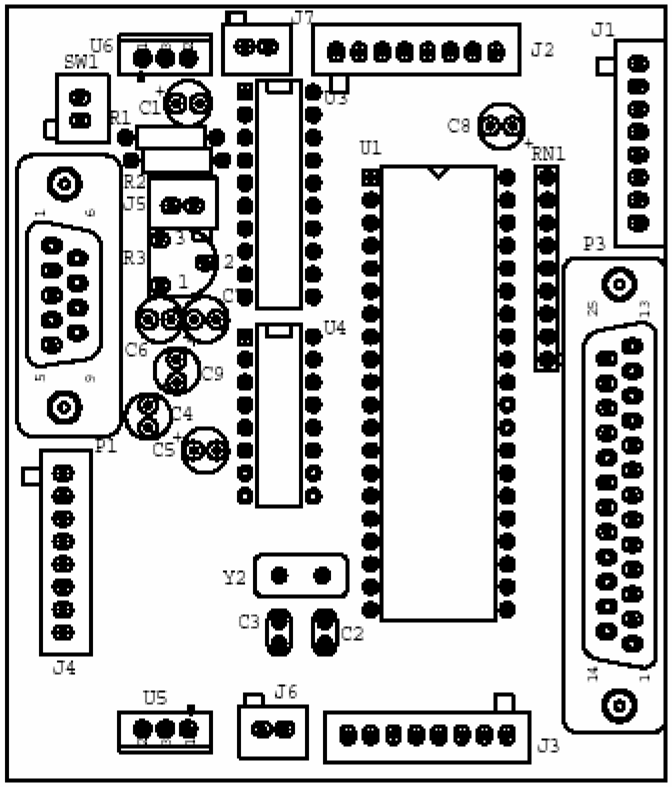
```
P27          = J3/8       P3/9       U1/28;

P30          = J4/1       U1/10      U4/12;

P31          = J4/2       U1/11      U4/11;

P32          = J4/3       U1/12;

P33          = J4/4       U1/13      U3/2;

P34          = J4/5       U1/14      U3/3;

P35          = J4/6       U1/15      U3/5;

P36          = J4/7       U1/16;

P37          = J4/8       U1/17;

PSCIN          = J7/1       U6/1;

PSMIN          = J6/1       U5/1;

X1           = J5/1       U3/6;

X2           = R3/2       U3/9;


--------------------------------------------------------------------

                              STATISTICS

                     Number of Nets     :  52

                     Number of Nodes    :  170

--------------------------------------------------------------------
```

Appendix B

Placement and Routing






J7
U6
SW1
J2
J1
C1
R1
U3
C8
U1
RN1
R2
J5
R3
P3
U4
C6
C9
C4
C5
P1
Y2
C3
C2
J4
U5
J6
J3