Module – 5

8051 Interrupts and Interfacing Applications

Interrupt is one of the most important and powerful concepts and features in microcontroller/processor applications. Almost all the real world and real time systems built around microcontrollers and microprocessors make use of interrupts.

What is an Interrupt?

An interrupt refer to a notification, communicated to the controller, by a hardware device or software, on receipt of which controller momentarily stops and responds to the interrupt. Whenever an interrupt occurs the controller completes the execution of the current instruction and starts the execution of an **Interrupt Service Routine** (ISR) or Interrupt Handler.

ISR is a piece of code that tells the processor or controller what to do when the interrupt occurs. After the execution of ISR, controller returns back to the instruction it has jumped from (before the interrupt was received). The interrupts can be either **hardware interrupts** or **software interrupts**.

Need of interrupts

An application built around microcontrollers generally has the following structure. It takes input from devices like keypad, ADC etc; processes the input using certain algorithm; and generates an output which is either displayed using devices like seven segment, LCD or used further to operate other devices like motors etc. In such designs, controllers interact with the inbuilt devices like timers and other interfaced peripherals like sensors, serial port etc. The programmer needs to monitor their status regularly like whether the sensor is giving output, whether a signal has been received or transmitted, whether timer has finished counting, or if an interfaced device needs service from the controller, and so on. This state of continuous monitoring is known as polling.

In polling, the microcontroller keeps checking the status of other devices; and while doing so it does no other operation and consumes all its processing time for monitoring. This problem can be addressed by using interrupts. In interrupt method, the controller responds to only when an interruption occurs. Thus in interrupt method, controller is not required to regularly monitor the status (flags, signals etc.) of interfaced and inbuilt devices.

To understand the difference better, consider the following. The polling method is very much similar to a salesperson. The salesman goes door-to-door requesting to buy its product or service. Like controller keeps monitoring the flags or signals one by one for all devices and caters to whichever needs its service. Interrupt, on the other hand, is very similar to a shopkeeper. Whosoever needs a service or product goes to him and apprises him of his/her needs. In our case, when the flags or signals are received, they notify the controller that they need its service.

Hardware & Software Interrupt

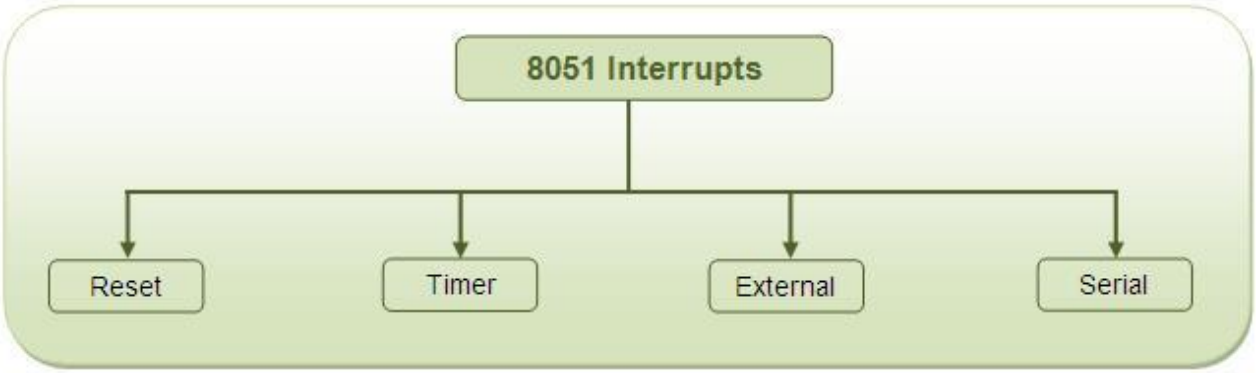
The interrupts in a controller can be either hardware or software. If the interrupts are generated by the controller's inbuilt devices, like timer interrupts; or by the interfaced devices, they are called the hardware interrupts. If the interrupts are generated by a piece of code, they are termed as software interrupts.

Multiple interrupts

What would happen if multiple interrupts are received by a microcontroller at the same instant? In such a case, the controller assigns priorities to the interrupts. Thus the interrupt with the highest priority is served first. However the priority of interrupts can be changed configuring the appropriate registers in the code.

8051 Interrupts

The 8051 controller has six hardware interrupts of which five are available to the programmer. These are as follows:



1. RESET interrupt – This is also known as Power on Reset (POR). When the RESET interrupt is received, the controller restarts executing code from 0000H location. This is an interrupt which is not available to or, better to say, need not be available to the programmer.

2. Timer interrupts – Each Timer is associated with a Timer interrupt. A timer interrupt notifies the microcontroller that the corresponding Timer has finished counting.

3. External interrupts – There are two external interrupts EX0 and EX1 to serve external devices. Both these interrupts are active low. In AT89C51, P3.2 (INT0) and P3.3 (INT1) pins are available for external interrupts 0 and 1 respectively. An external interrupt notifies the microcontroller that an external device needs its service.

4. Serial interrupt – This interrupt is used for serial communication. When enabled, it notifies the controller whether a byte has been received or transmitted.

How is an interrupt serviced?

Every interrupt is assigned a fixed memory area inside the processor/controller. The Interrupt Vector Table (IVT) holds the starting address of the memory area assigned to it (corresponding to every interrupt).

The interrupt vector table (IVT) for AT89C51 interrupts is as follows :

Interrupt	ROM Location (Hex)	Pin	Flag clearing
Reset	0000	9	Auto
External interrupt 0	0003	12	Auto
Timer interrupt 0	000B	–	Auto
External interrupt 1	0013	13	Auto
Timer interrupt 1	001B	–	Auto
Serial COM interrupt	0023	–	Programmer clears it

When an interrupt is received, the controller stops after executing the current instruction. It transfers the content of program counter into stack. It also stores the current status of the interrupts internally but not on stack. After this, it jumps to the memory location specified by **Interrupt Vector Table** (IVT). After that the code written on that memory area gets executed. This code is known as the Interrupt Service Routine (ISR) or interrupt handler. ISR is a code written by the programmer to handle or service the interrupt.

Programming Interrupts

While programming interrupts, first thing to do is to specify the microcontroller which interrupts must be served. This is done by configuring the Interrupt Enable (IE) register which enables or disables the various available interrupts. The Interrupt Enable register has following bits to enable/disable the hardware interrupts of the 8051 controller.

IE :	EA	-	ET2	ES	ET1	EX1	ET0	EX0

Bit Values of IE Register of 8051 Microcontroller

To enable any of the interrupts, first the EA bit must be set to 1. After that the bits corresponding to the desired interrupts are enabled. ET0, ET1 and ET2 bits are used to enable the Timer Interrupts 0, 1 and 2, respectively. In AT89C51, there are only two timers, so ET2 is not used. EX0 and EX1 are used to enable the external interrupts 0 and 1. ES is used for serial interrupt.

EA bit acts as a lock bit. If any of the interrupt bits are enabled but EA bit is not set, the interrupt will not function. By default all the interrupts are in disabled mode.

Note that the IE register is bit addressable and individual interrupt bits can also be accessed.
Note that the IE register is bit addressable and individual interrupt bits can also be accessed.
For example –
IE = 0x81; enables External Interrupt0 (EX0)
IE = 0x88; enables Serial Interrupt

Setting the bits of IE register is necessary and sufficient to enable the interrupts. **Next step** is to specify the controller what to do when an interrupt occurs. This is done by writing a subroutine or function for the interrupt. This is the ISR and gets automatically called when an interrupt occurs. It is not required to call the Interrupt Subroutine explicitly in the code.

An important thing is that the definition of a subroutine must have the keyword **interrupt** followed by the interrupt number. A subroutine for a particular interrupt is identified by this number.

These subroutine numbers corresponding to different interrupts are tabulated below.

Number	Interrupt	Symbol
0	External0	EX0
1	Timer0	IT0
2	External1	EX1
3	Timer1	IT1
4	Serial	ES
5	Timer2	ET2

For example : Interrupt routine for Timer1

```
void ISR_timer1(void) interrupt 3
{
        <Body of ISR>
}
```

For example : Interrupt routine for External Interrupt0 (EX0)

```
void ISR_ex0(void) interrupt 0
{
        <Body of ISR>
}
```

Note that the interrupt subroutines always have void return type. They never return a value.

Programming Timer Interrupts

1. Programming Timer Interrupts

The timer interrupts IT0 and IT1 are related to Timers 0 and 1, respectively. (Please refer 8051 Timers for details on Timer registers and modes.) The interrupt programming for timers involves following steps :

1. Configure TMOD register to select timer(s) and its/their mode.
2. Load initial values in THx and TLx for mode 0 and 1; or in THx only for mode 2.
3. Enable Timer Interrupt by configuring bits of IE register.
4. Start timer by setting timer run bit TRx.
5. Write subroutine for Timer Interrupt. The interrupt number is 1 for Timer0 and 3 for Timer1.

6. Note that it is not required to clear timer flag TFx.
7. To stop the timer, clear TRx in the end of subroutine. Otherwise it will restart from 0000H in case of modes 0 or 1 and from initial values in case of mode 2.
8. If the Timer has to run again and again, it is required to reload initial values within the routine itself (in case of mode 0 and 1). Otherwise after one cycle timer will start counting from 0000H.

Example code

Timer interrupt to blink an LED; Time delay in mode1 using interrupt method

```
// Use of Timer mode0 for blinking LED using interrupt method
// XTAL frequency 11.0592MHz
#include<reg51.h>
sbit LED = P1^0;              //LED connected to D0 of port 1

void timer(void) interrupt 1              //interrupt no. 1 for Timer 0
{
        led=~led;                         //toggle LED on interrupt
        TH0=0xFC;                         // initial values loaded to timer
        TL0=0x66;
}
main()
{
        TMOD = 0x01;                      // mode1 of Timer0
        TH0 = 0xFC;                       // initial values loaded to timer
        TL0 = 0x66;
        IE = 0x82;                        // enable interrupt
        TR0 = 1;              //start timer
        while(1);             // do nothing
}
```

Programming External Interrupts

2. Programming External Interrupts

The external interrupts are the interrupts received from the (external) devices interfaced with the microcontroller. They are received at INTx pins of the controller. These can be level triggered or edge triggered. In level triggered, interrupt is enabled for a low at INTx pin; while in case of edge triggering, interrupt is enabled for a high to low transition at INTx pin. The edge or level trigger is decided by the TCON register. The TCON register has following bits:

TCON	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Bit Values of TCON Register of 8051 Microcontroller

Setting the IT0 and IT1 bits make the external interrupt 0 and 1 edge triggered respectively. By default these bits are cleared and so external interrupt is level triggered.

Note: For a level trigger interrupt, the INTx pin must remain low until the start of the ISR and should return to high before the end of ISR. If the low at INTx pin goes high before the start of ISR, interrupt will not be generated. Also if the INTx pin remains low even after the end of ISR, the interrupt will be generated once again. This is the reason why level trigger interrupt (low) at INTx pin must be four machine cycles long and not greater than or smaller than this.

Following are the steps for using external interrupt:

1. Enable external interrupt by configuring IE register.
2. Write routine for external interrupt. The interrupt number is 0 for EX0 and 2 for EX1 respectively.
3.

Example 5.1

```
//Level trigger external interrupt

void main()
{
        IE = 0x81;
        while(1);
}
void ISR_ex0(void) interrupt 0
{
        <body of interrupt>
}
```

Example 5.2

```
//Edge trigger external interrupt
void main()
{
        IE = 0x84;
        IT1 = 1;
        while(1);
}
void ISR_ex1(void) interrupt 2
{
        <body of interrupt>
}
```

Programming Serial Interrupt

To use the serial interrupt the ES bit along with the EA bit is set. Whenever one byte of data is sent or received, the serial interrupt is generated and the TI or RI flag goes high. Here, the TI or RI flag needs to be cleared explicitly in the interrupt routine (written for the Serial Interrupt).

The programming of the Serial Interrupt involves the following steps:

1. Enable the Serial Interrupt (configure the IE register).
2. Configure SCON register.
3. Write routine or function for the Serial Interrupt. The interrupt number is 4.
4. Clear the RI or TI flag within the routine.

Example 5.3

Send 'A' from serial port with the use of interrupt

```
// Sending 'A' through serial port with interrupt
// XTAL frequency 11.0592MHz
void main()
{
        TMOD = 0x20;
        TH1 = -1;
        SCON = 0x50;
        TR1 = 1;
        IE = 0x90;
        while(1);
}
void ISR_sc(void) interrupt 4
{
        if(TI==1)
        {
                SBUF = 'A';
                TI = 0;
        }
        else
                RI = 0;
}
```

Example 5.4

```
// Receive data from serial port through interrupt
// XTAL frequency 11.0592MHz
void main()
{
        TMOD = 0x20;
        TH1 = -1;
        SCON = 0x50;
        TR1 = 1;
        IE = 0x90;
        while(1);
}
void ISR_sc(void) interrupt 4
{
        unsigned char val;
        if(TI==1)
        {
                TI = 0;
        }
        else
        {
                val = SBUF;
                RI = 0;
        }
}
```

Stepper Motor Interfacing:

Stepper motor is a widely used device that translates electrical pulses into mechanical movement. Stepper motor is used in applications such as; disk drives, dot matrix printer, robotics etc,. The construction of the motor is as shown in figure below.

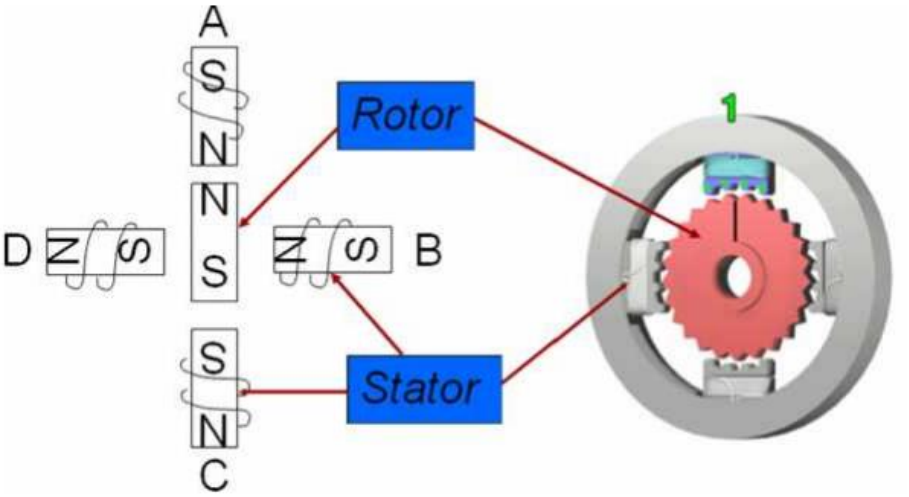


Figure: Structure of stepper motor

It has a permanent magnet rotor called the shaft which is surrounded by a stator. Commonly used stepper motors have four stator windings that are paired with a center – tapped common. Such motors are called as four-phase or unipolar stepper motor. The stator is a magnet over which the electric coil is wound. One end of the coil are connected commonly either to ground or +5V. The other end is provided with a fixed sequence such that the motor rotates in a particular direction. Stepper motor shaft moves in a fixed repeatable increment, which allows one to move it to a precise position. Direction of the rotation is dictated by the stator poles. Stator poles are determined by the current sent through the wire coils.

Step angle: Step angle is defined as the minimum degree of rotation with a single step.
No of steps per revolution = 360° / step angle Steps per second = (rpm x steps per revolution) / 60
Example: step angle = 2° No of steps per revolution = 180

Switching Sequence of Motor: As discussed earlier the coils need to be energized for the rotation. This can be done by sending a bits sequence to one end of the coil while the other end is commonly connected. The bit sequence sent can make either one phase ON or two phase ON for a full step sequence or it can be a combination of one and two phase ON for half step sequence. Both are tabulated below.

Full Step: Two Phase ON

Clockwise ↓

Step #	A	B	C	D
1	1	0	0	1
2	1	1	0	0
3	0	1	1	0
4	0	0	1	1

↑ Counter-clockwise

One Phase ON

Clockwise ↓

Step #	A	B	C	D
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1

↑ Counter-clockwise

Half Step (8 – sequence): The sequence is tabulated as below:

Clk ↓

Step #	A	B	C	D
1	1	0	0	1
2	1	0	0	0
3	1	1	0	0
4	0	1	0	0
5	0	1	1	0
6	0	0	1	0
7	0	0	1	1
8	0	0	0	1

↑ Anti-Clk

8051 Connection to Stepper Motor: (explanation of the diagram can be done)

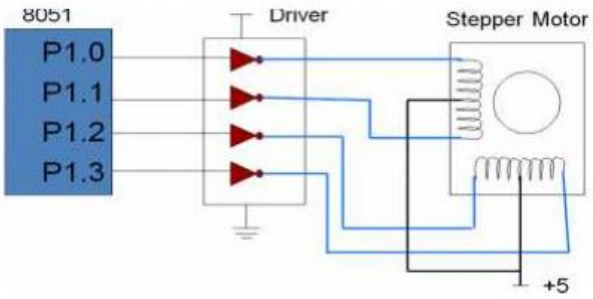


Figure: 8051 interface to stepper motor

The following example 1 to example 6 shown below will elaborate on the discussion done above:

Example 1: Write an ALP to rotate the stepper motor clockwise / anticlockwise continuously with full step sequence.

Program:

```
        MOV   A, #66H
BACK:   MOV   P1, A
        RR    A
        ACALL DELAY
        SJMP  BACK
DELAY:  MOV   R1, #100
UP1:    MOV   R2, #50
UP:     DJNZ  R2, UP
        DJNZ  R1, UP1
        RET
```

Note: motor to rotate in anticlockwise use instruction RL A instead of RR A

Example 2: A switch is connected to pin P2.7. Write an ALP to monitor the status of the SW. If SW = 0, motor moves clockwise and if SW = 1, motor moves anticlockwise.

Program:

```
        ORG 0000H
        SETB P2.7
        MOV A, #66H
        MOV P1,A
TURN: JNB P2.7, CW
        RL A
        ACALL DELAY
        MOV P1,A
        SJMP TURN
CW:   RR A
        ACALL DELAY
        MOV P1,A
        SJMP TURN
DELAY: as previous example
```

Example 3: Write an ALP to rotate a motor 90° clockwise. Step angle of motor is 2°.

Solution:
Step angle = 2°
Steps per revolution = 180
No of rotor teeth = 45
For 90° rotation the no of steps is 45

Program:

```
        ORG 0000H
        MOV A, #66H
        MOV R0, #45
BACK: RR A
        MOV P1, A
        ACALL DELAY
        DJNZ R0, BACK
        END
```

Example 4: Rotate the stepper motor continuously clockwise using half-step 8-step sequence. Say the sequence is in ROM locations.

Program:

```
        ORG 0000H
START: MOV R0, #08
        MOV DPTR, #HALFSTEP
RPT:    CLR A
        MOVC A, @A+DPTR
        MOV P1, A
        ACALL DELAY
        INC DPTR
        DJNZ R0, RPT
        SJMP START
       ORG 0200H
HALFSTEP DB 09, 08, 0CH, 04, 06, 02, 03, 01
        END
```

Programming Stepper Motor with 8051 C

The following examples 5 and 6 will show the programming of stepper motor using 8051 C.

Example 5: Problem definition is same as example 1.

Program:

```
#include <reg51.h>
void main ()
{
        while (1)
        {
          P1=0x66;
          MSDELAY (200);
          P1=0x33;
          MSDELAY (200);
          P1=0x99;
          MSDELAY (200);
          P1=0xCC;
          MSDELAY (200);
        }
}
void MSDELAY (unsigned char value)
{
        unsigned int x,y;
        for(x=0;x<1275;x++)
        for(y=0;y<value;y++);
}
```

Program:

```
#include <reg51.h>
sbit SW=P2^7;
void main ()
{
       SW=1;
        while (1)
        {
          if(SW==0){
           P1=0x66;
           MSDELAY (100);
           P1=0x33;
           MSDELAY (100);
           P1=0x99;
           MSDELAY (100);
           P1=0xCC;
           MSDELAY (100);
        }
        else {
           P1=0x66;
           MSDELAY (100);
           P1=0xCC;
```

```
            MSDELAY (100);
        }
        else {
            P1=0x66;
            MSDELAY (100);
            P1=0xCC;
            MSDELAY (100);
            P1=0x99;
            MSDELAY (100);
            P1=0x33;
            MSDELAY (100);

}
void MSDELAY (unsigned char value)
{
        unsigned int x,y;
        for(x=0;x<1275;x++)
        for(y=0;y<value;y++);
}
```

Digital-to-Analog (DAC) converter:

The DAC is a device widely used to convert digital pulses to analog signals. In this section we will discuss the basics of interfacing a DAC to 8051. The two method of creating a DAC is binary weighted and R/2R ladder. The Binary Weighted DAC, which contains one resistor or current source for each bit of the DAC connected to a summing point. These precise voltages or currents sum to the correct output value. This is one of the fastest conversion methods but suffers from poor accuracy because of the high precision required for each individual voltage or current. Such high- precision resistors and current-sources are expensive, so this type of converter is usually limited to 8-bit resolution or less.

The R-2R ladder DAC, which is a binary weighted DAC that uses a repeating cascaded structure of resistor values R and 2R. This improves the precision due to the relative ease of producing equal valued matched resistors (or current sources). However, wide converters perform slowly due to increasingly large RC-constants for each added R-2R link.

The first criterion for judging a DAC is its resolution, which is a function of the number of binary inputs. The common ones are 8, 10, and 12 bits. The number of data bit inputs decides the resolution of the DAC since the number of analog output levels is equal to 2n, where n is the number of data bit inputs. DAC0808: The digital inputs are converter to current Iout, and by connecting a resistor to the Iout pin, we can convert the result to voltage. The total current Iout is a function of the binary numbers at the D0-D7 inputs of the DAC0808 and the reference current Iref , and is as follows:

Usually reference current is 2mA. Ideally we connect the output pin to a resistor, convert this current to voltage, and monitor the output on the scope. But this can cause inaccuracy; hence an opamp is used to convert the output current to voltage. The 8051 connection to DAC0808 is as shown in the figure below.

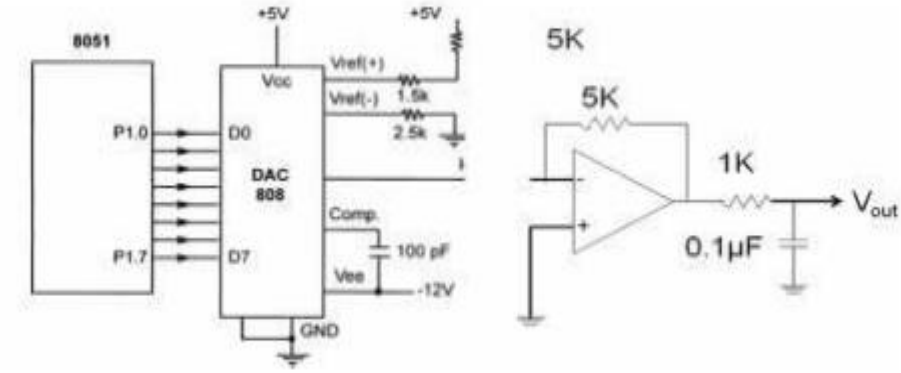


Figure: 8051 connection to DAC0808

The following examples 9, 10 and 11 will show the generation of waveforms using DAC0808.

Example 9: Write an ALP to generate a triangular waveform.

Program:

```
        MOV A, #00H
INCR:   MOV P1, A
        INC A
        CJNE A, #255, INCR
DECR:   MOV P1, A
        DEC A
        CJNE A, #00, DECR
        SJMP INCR
        END
```

Example 10: Write an ALP to generate a sine waveform.

$V_{out} = 5V(1+\sin\theta)$

Solution: Calculate the decimal values for every 10 degree of the sine wave. These values can be maintained in a table and simply the values can be sent to port P1. The sinewave can be observed on the CRO.

Program:

```
        ORG 0000H
AGAIN:  MOV DPTR, #SINETABLE
        MOV R3, #COUNT
UP:     CLR A
        MOVC A, @A+DPTR
        MOV P1, A
        INC DPTR
        DJNZ R3, UP
        SJMP AGAIN
        ORG 0300H
SINETABLE DB 128, 192, 238, 255, 238, 192, 128, 64, 17, 0, 17, 64, 128
        END
```

Note: to get a better wave regenerate the values of the table per 2 degree.

Example 10: Write a C program to generate a sine waveform.

$V_{out} = 5V(1+\sin\theta)$

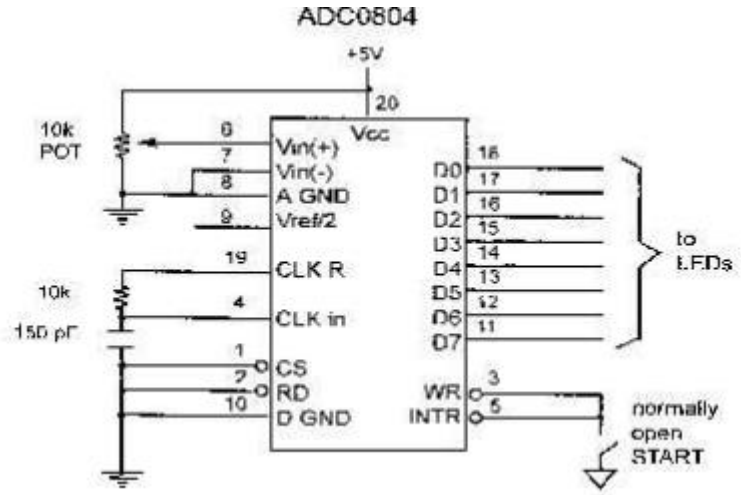
Program:

```
#include<reg51.h>
sfr dacdata=P1;
void main( )
{
        unsigned char sinetable[12]={ 128, 192, 238, 255, 238, 192,
                                      128, 64, 17, 0, 17, 64};
        unsigned char x;
        while (1)
        {
                for(x=0;x<12;x++)
                {
                   dacdata = sinetable[x];
                }
        }
}
```

Analog-to-digital converter (ADC) interfacing:

ADCs (analog-to-digital converters) are among the most widely used devices for data acquisition. A physical quantity, like temperature, pressure, humidity, and velocity, etc., is converted to electrical (voltage, current) signals using a device called a transducer, or sensor We need an analog-to-digital converter to translate the analog signals to digital numbers, so microcontroller can read them. ADC804 chip: ADC804 IC is an analog-to-digital converter. It works with +5 volts and has a resolution of 8 bits. Conversion time is another major factor in judging an ADC. Conversion time is defined as the time it takes the ADC to convert the analog input to a digital (binary) number. In ADC804 conversion time varies depending on the clocking signals applied to CLK R and CLK IN pins, but it cannot be faster than 110μs.

Pin Description of ADC804:



CLK IN and CLK R: CLK IN is an input pin connected to an external clock source. To use the internal clock generator (also called self-clocking), CLK IN and CLK R pins are connected to a capacitor and a resistor and the clock frequency is determined by:
Typical values are R = 10K ohms and C =150pF. We get f = 606 kHz and the conversion time is 110μs.
Vref/2 : It is used for the reference voltage. If this pin is open (not connected), the analog input voltage is in the range of 0 to 5 volts (the same as the Vcc pin). If the analog input range needs to be 0 to 4 volts, Vref/2 is connected to 2 volts. Step size is the smallest change can be discerned by an ADC

Vref/2 Relation to Vin Range

$V_{ref}/2$(v)	V_{in}(V)	Step Size (mV)
Not connected*	0 to 5	5/256=19.53
2.0	0 to 4	4/255=15.62
1.5	0 to 3	3/256=11.71
1.28	0 to 2.56	2.56/256=10
1.0	0 to 2	2/256=7.81
0.5	0 to 1	1/256=3.90

D0-D7: The digital data output pins. These are tri-state buffered. The converted data is accessed only when CS =0 and RD is forced low. To calculate the output voltage, use the following formula

$$D_{out} = \frac{V_{in}}{step\ size}$$

Dout = digital data output (in decimal),
Vin = analog voltage, and
step size (resolution) is the smallest change

Analog ground and digital ground: Analog ground is connected to the ground of the analog Vin and digital ground is connected to the ground of the Vcc pin. To isolate the analog Vin signal from transient voltages caused by digital switching of the output D0 – D7. This contributes to the accuracy of the digital data output.

Vin(+) & Vin(-): Differential analog inputs where Vin = Vin (+) – Vin (-). Vin (-) is connected to ground and Vin (+) is used as the analog input to be converted.

RD: Is "output enable" a high-to-low RD pulse is used to get the 8-bit converted data out of ADC804.

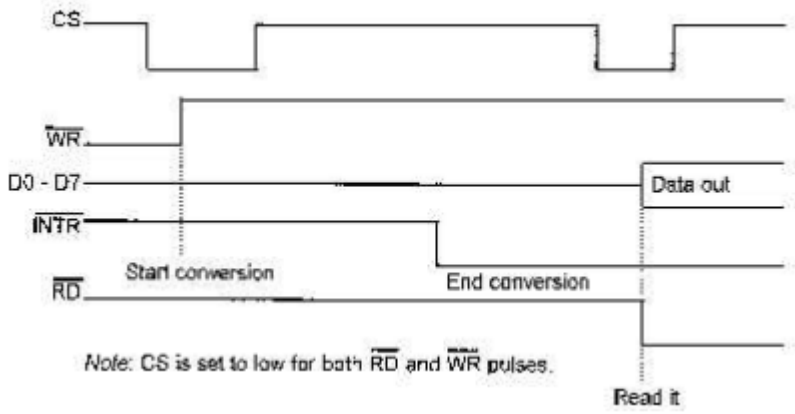
INTR: It is "end of conversion" When the conversion is finished, it goes low to signal the CPU that the converted data is ready to be picked up.

WR: It is "start conversion" When WR makes a low-to-high transition, ADC804 starts converting the analog input value of Vin to an 8- bit digital number.

CS: It is an active low input used to activate ADC804.

The following steps must be followed for data conversion by the ADC804 chip:
1. Make CS= 0 and send a L-to-H pulse to pin WR to start conversion.
2. Monitor the INTR pin, if high keep polling but if low, conversion is complete, go to next step.
3. Make CS= 0 and send a H-to-L pulse to pin RD to get the data out

The following figure shows the read and write timing for ADC804.



The following figures shows the self-clocking with the RC component for frequency and the external frequency connected to XTAL2 of 8051.

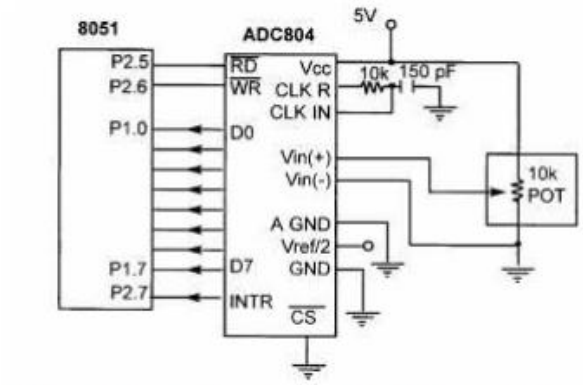


Figure : 8051 Connection to ADC0804 with Self-clocking

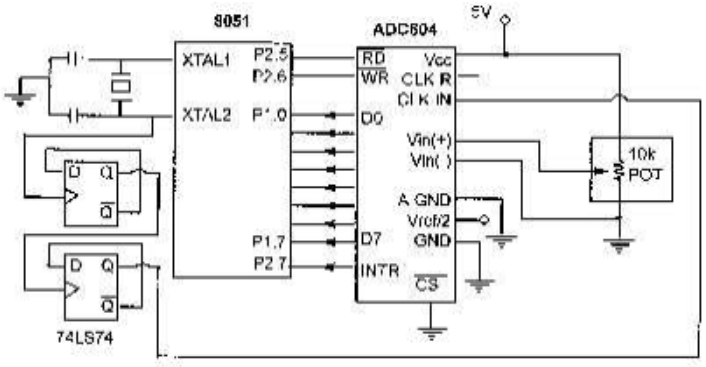


Figure : 8051 Connection to ADC0804 with Clock from XTAL2 of 8051

Programming ADC0804 in assembly

```
        MYDATA EQU P1
        MOV P1, #0FFH
        SETB P2.7
BACK:   CLR P2.6
        SETB P2.6
HERE:   JB P2.7, HERE
        CLR P2.5
        MOV A, MYDATA
        SETB P2.5
        SJMP BACK
```

Programming ADC0804 in C

```
#include<reg51.h>
Sbit RD=P2^5;
Sbit WR=P2^6;
Sbit INTR=P2^7;
Sfr Mydata=P1;
Void main ( )
{
        Unsigned char value;
        Mydata =0xFF;
        INTR=1;
        RD=1;
        WR=1;
        While (1)
        {
         WR=0;
         WR=1;
          While (INTR == 1);
         RD=0;
         Value =Mydata;
         RD=1;
        }
}
```

ADC0808/0809 chip: ADC808 has 8 analog inputs. It allows us to monitor up to 8 different transducers using only single chip. The chip has 8-bit data output just like the ADC804. The 8 analog input channels are multiplexed and selected according to the values given to the three address pins, A, B, and C. that is; if CBA=000, CH0 is selected; CBA=011, CH3 is selected and so on. The pin details of ADC0808 are as shown in the figure below. (Explanation can be done as is with ADC0804).

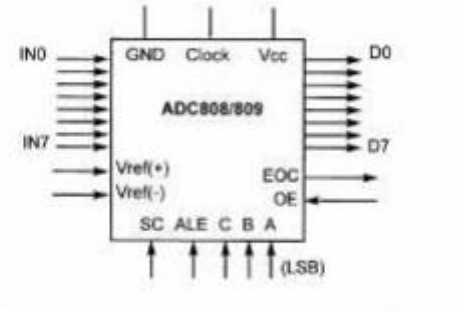


Figure : Pin out of ADC0808

Steps to Program ADC0808/0809

1. Select an analog channel by providing bits to A, B, and C addresses.

2. Activate the ALE pin. It needs an L-to-H pulse to latch in the address.

3. Activate SC (start conversion) by an H-to-L pulse to initiate conversion.

4. Monitor EOC (end of conversion) to see whether conversion is finished.

5. Activate OE (output enable) to read data out of the ADC chip. An H-to-L pulse to the OE pin will bring digital data out of the chip.

```
          MYDATA EQU P1
          ORG 0000H
          MOV MYDATA, #0FFH
          SETB P2.7
          CLR P2.4
          CLR P2.6
          CLR P2.5
BACK:     CLR P2.0
          CLR P2.1
          SETB P2.2
          ACALL DELAY
          SETB P2.4
          ACALL DELAY
          SETB P2.6
          ACALL DELAY
          CLR P2.4
          CLR P2.6
HERE:     JB P2.7, HERE
HERE1:    JNB P2.7, HERE1
          SETB P2.5
          ACALL DELAY
          MOV A, MYDATA
          CLR P2.5
          SJMP BACK
```

LCD Interfacing:

LCD is finding widespread use replacing LEDs for the following reasons: The declining prices of LCD The ability to display numbers, characters, and graphics Incorporation of a refreshing controller into the LCD, thereby relieving the CPU of the task of refreshing the LCD Ease of programming for characters and graphics.

Pin Description:

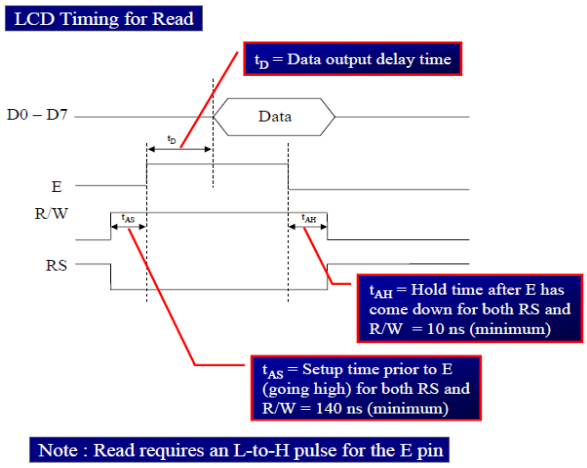
Pin	Symbol	I/O	Description
1	V_{SS}	--	Ground
2	V_{CC}	--	+5V power supply
3	V_{EE}	--	Power supply to control contrast
4	RS	I	RS=0 to select command register, RS=1 to select data register
5	R/W	I	R/W=0 for write, R/W=1 for read
6	E	I/O	Enable
7	DB0	I/O	The 8-bit data bus
8	DB1	I/O	The 8-bit data bus
9	DB2	I/O	The 8-bit data bus
10	DB3	I/O	The 8-bit data bus
11	DB4	I/O	The 8-bit data bus
12	DB5	I/O	The 8-bit data bus
13	DB6	I/O	The 8-bit data bus
14	DB7	I/O	The 8-bit data bus

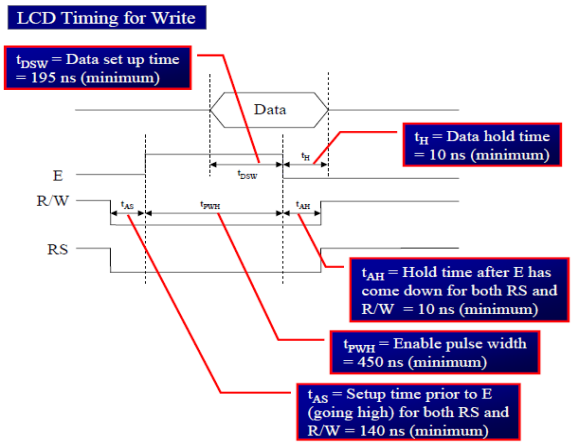
- Send displayed information or instruction command codes to the LCD
- Read the contents of the LCD's internal registers

Command codes

Code (Hex)	Command to LCD Instruction Register
1	Clear display screen
2	Return home
4	Decrement cursor (shift cursor to left)
6	Increment cursor (shift cursor to right)
5	Shift display right
7	Shift display left
8	Display off, cursor off
A	Display off, cursor on
C	Display on, cursor off
E	Display on, cursor blinking
F	Display on, cursor blinking
10	Shift cursor position to left
14	Shift cursor position to right
18	Shift the entire display to the left
1C	Shift the entire display to the right
80	Force cursor to beginning to 1st line
C0	Force cursor to beginning to 2nd line
38	2 lines and 5x7 matrix

LCD timing diagram for reading and writing is as shown in the below figures.





Sending Data/ Commands to LCDs with Time Delay: To send any of the commands to the LCD, make pin RS=0. For data, make RS=1. Then send a high-to-low pulse to the E pin to enable the internal latch of the LCD.

Example 11: Write an ALP to initialize the LCD and display message "YES". Say the command to be given is :38H (2 lines ,5x7 matrix), 0EH (LCD on, cursor on), 01H (clear LCD), 06H (shift cursor right), 86H (cursor: line 1, pos. 6)

Program:

```
;calls a time delay before sending next data/command ;P1.0-P1.7 are connected to
LCD data pins D0-D7 ;P2.0 is connected to RS pin of LCD ;P2.1 is connected to
R/W pin of LCD ;P2.2 is connected to E pin of LCD
ORG 0H
        MOV A,#38H          ;INIT. LCD 2 LINES, 5X7 MATRIX
        ACALL COMNWRT       ;call command subroutine
        ACALL DELAY         ;give LCD some time
        MOV A,#0EH          ;display on, cursor on
        ACALL COMNWRT       ;call command subroutine
        ACALL DELAY          ;give LCD some time
        MOV A,#01           ;clear LCD
        ACALL COMNWRT       ;call command subroutine
        ACALL DELAY         ;give LCD some time
        MOV A,#06H          ;shift cursor right
        ACALL COMNWRT        ;call command subroutine
        ACALL DELAY         ;give LCD some time
        MOV A,#86H          ;cursor at line 1, pos. 6
        ACALL COMNWRT       ;call command subroutine
        ACALL DELAY         ;give LCD some time
```

```
        MOV A,#'Y'          ;display letter Y
        ACALL DATAWRT       ;call display subroutine
        ACALL DELAY         ;give LCD some time
        MOV A,#'E'          ;display letter E
        ACALL DATAWRT       ;call display subroutine
        ACALL DELAY         ;give LCD some time
        MOV A,#'S'          ;display letter S
        ACALL DATAWRT       ;call display subroutine
AGAIN: SJMP AGAIN           ;stay here

COMNWRT:                    ;send command to LCD
        MOV P1,A            ;copy reg A to port 1
        CLR P2.0            ;RS=0 for command
        CLR P2.1            ;R/W=0 for write
        SETB P2.2           ;E=1 for high pulse
        ACALL DELAY         ;give LCD some time
        CLR P2.2            ;E=0 for H-to-L pulse
RET

DATAWRT:                    ;write data to LCD
        MOV P1,A            ;copy reg A to port 1
        SETB P2.0           ;RS=1 for data
        CLR P2.1            ;R/W=0 for write
        SETB P2.2           ;E=1 for high pulse
        ACALL DELAY         ;give LCD some time
        CLR P2.2            ;E=0 for H-to-L pulse
RET

DELAY:
        MOV R3,#50          ;50 or higher for fast CPUs
HERE2: MOV R4,#255          ;R4 = 255
HERE: DJNZ R4,HERE          ;stay until R4 becomes 0
        DJNZ R3,HERE2
RET
END
```

Example 12: Modify example 11, to check for the busy flag (D7=>P1.7), then send the command and hence display message "NO".

```
;Check busy flag before sending data, command to LCD;p1=data pin ;P2.0 connected
to RS pin ;P2.1 connected to R/W pin ;P2.2 connected to E pin
ORG 0H
        MOV A,#38H              ;init. LCD 2 lines ,5x7 matrix
        ACALL COMMAND           ;issue command
        MOV A,#0EH              ;LCD on, cursor on
        ACALL COMMAND           ;issue command
        MOV A,#01H              ;clear LCD command
        ACALL COMMAND           ;issue command
        MOV A,#06H              ;shift cursor right
        ACALL COMMAND           issue command
        MOV A,#86H              ;cursor: line 1, pos. 6
        ACALL COMMAND           ;command subroutine
        MOV A,#'N'              ;display letter N
        ACALL DATA_DISPLAY
        MOV A,#'O'               ;display letter O
        ACALL DATA_DISPLAY
HERE:SJMP HERE          ;STAY HERE

COMMAND:
        ACALL READY             ;is LCD ready?
        MOV P1,A                 ;issue command code
        CLR P2.0                ;RS=0 for command
        CLR P2.1                ;R/W=0 to write to LCD
        SETB P2.2               ;E=1 for H-to-L pulse
        CLR P2.2                ;E=0,latch in

RET

DATA_DISPLAY:
        ACALL READY             ;is LCD ready?
        MOV P1,A                ;issue data
        SETB P2.0               ;RS=1 for data
        CLR P2.1                ;R/W =0 to write to LCD
        SETB P2.2               ;E=1 for H-to-L pulse
        CLR P2.2                ;E=0,latch in
RET

READY:
        SETB P1.7                ;make P1.7 input port
        CLR P2.0                 ;RS=0 access command reg
        SETB P2.1               ;R/W=1 read command reg ;
 BACK:SETB P2.2                 ;E=1 for H-to-L pulse
        CLR P2.2                ;E=0 H-to-L pulse
        JB P1.7,BACK            ;stay until busy flag=0
RET
END
```

Programming LCD in C

Example 13: Write an 8051 C program to send letters 'P', 'I', and 'C' to the LCD using the busy flag method.

Solution:

```
#include <reg51.h>
sfr ldata = 0x90;                    //P1=LCD data pins
sbit rs = P2^0;
sbit rw = P2^1;
sbit en = P2^2;
sbit busy = P1^7;
void main(){
        lcdcmd(0x38);
        lcdcmd(0x0E);
        lcdcmd(0x01);
        lcdcmd(0x06);
        lcdcmd(0x86);                //line 1, position 6
        lcddata('P');
        lcddata('I');
        lcddata('C');
}

void lcdcmd(unsigned char value){
        lcdready();                  //check the LCD busy flag
        ldata = value;               //put the value on the pins
        rs = 0;
        rw = 0;
        en = 1;                       //strobe the enable pin
        MSDelay(1);
        en = 0;
        return;
}
void lcddata(unsigned char value){
        lcdready();                  //check the LCD busy flag
        ldata = value;               //put the value on the pins
        rs = 1;
        rw = 0;
        en = 1;                      //strobe the enable pin
        MSDelay(1);
        en = 0;
        return;
}
```

```
void lcdready(){
        busy = 1;                   //make the busy pin at input
        rs = 0;
        rw = 1;
        while(busy==1){             //wait here for busy flag
        en = 0;                     //strobe the enable pin
        MSDelay(1);
        en = 1;
}
}
void Msdelay(unsigned int itime){
        unsigned int i, j;
        for(i=0;i<itime;i++)
        for(j=0;j<1275;j++);
}
```

****