

## 8051 Interfacing and Applications

### Objectives:

At the end of this chapter, we will be able to:

- List the different devices that can be interfaced with 8051
- Understand the working principle.
- Develop the following applications using assembly and C
  - Stepper motor interface
  - DC motor interfacing and PWM
  - Digital-to-Analog interfacing
  - Analog-to-Digital interfacing
  - LCD interface
  - Keyboard interface

This chapter basically gives an insight into the study of different interfacing listed above. Further we will also study and understand their operation that is the working principle. We will further discuss on how to develop these interfaces using assembly and C.

### 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 1 below.

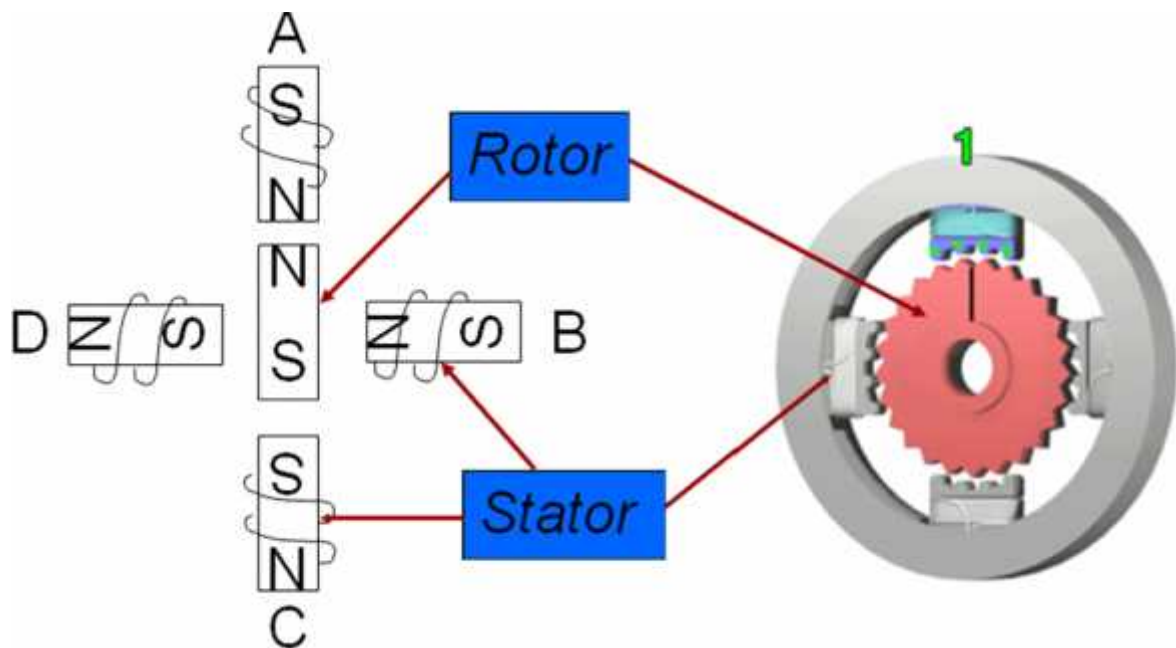


Figure 1: Structure of stepper motor

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 =  $360^\circ / \text{step angle}$

Steps per second =  $(\text{rpm} \times \text{steps per revolution}) / 60$

Example: step angle =  $2^\circ$

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

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

Clockwise (indicated by a downward arrow) and Counter-clockwise (indicated by an upward arrow) directions are shown relative to the table.

##### One Phase ON

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

Clockwise (indicated by a downward arrow) and Counter-clockwise (indicated by an upward arrow) directions are shown relative to the table.

#### Half Step (8 – sequence):

The sequence is tabulated as below:

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

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

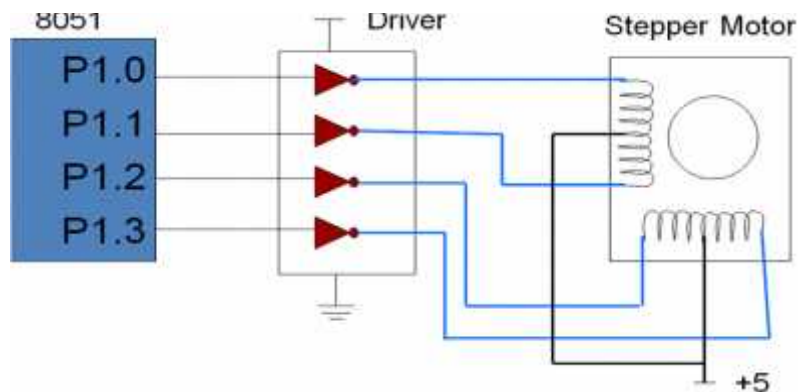


Figure 2: 8051 interface to stepper motor

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++);
}

```

Example 6: Problem definition is same as example 2.

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);
            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++);
}
```

## DC Motor Interfacing with 8051:

The DC motor is another widely used device that translates electrical pulses into mechanical movement. Motor has 2 leads +ve and -ve, connecting them to a DC voltage supply moves the motor in one direction. On reversing the polarity rotates the motor in the reverse direction. Basic difference between Stepper and DC motor is stepper motor moves in steps while DC motor moves continuously. Another difference is with stepper motor the number of steps can be counted while it is not possible in DC motor. Maximum speed of a DC motor is indicated in rpm. The rpm is either with no load it is few thousands to tens of thousands or with load rpm decreases with increase in load.

Voltage and current rating : Nominal voltage is the voltage for a motor under normal condition. It ranges from 1V to 150V. As voltage increases, rpm goes up. Current rating is the current consumption when the nominal voltage is applied with no load that is 25mA to a few amperes. As load increases, rpm increases, unless voltage or current increases implies torque increases. With fixed voltage, as load increases, power consumption of a DC motor is increased.

### Unidirectional Control:

Figure 3 shows the rotation of the DC motor in clockwise and anticlockwise direction.

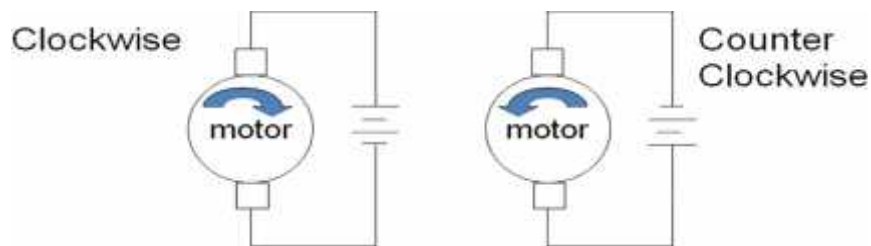
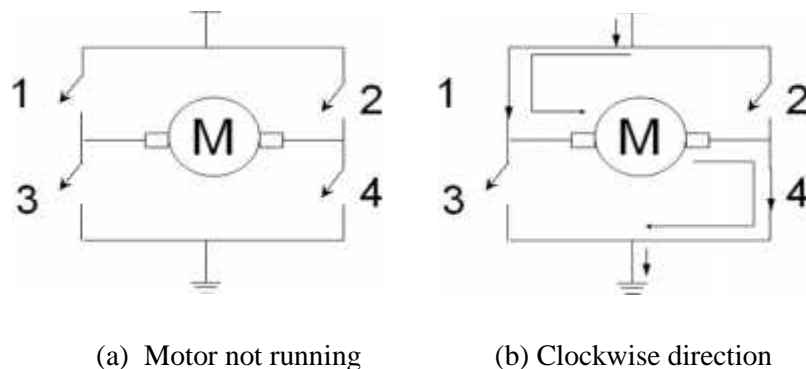


Figure 3: DC motor rotation

### Bidirectional Control:



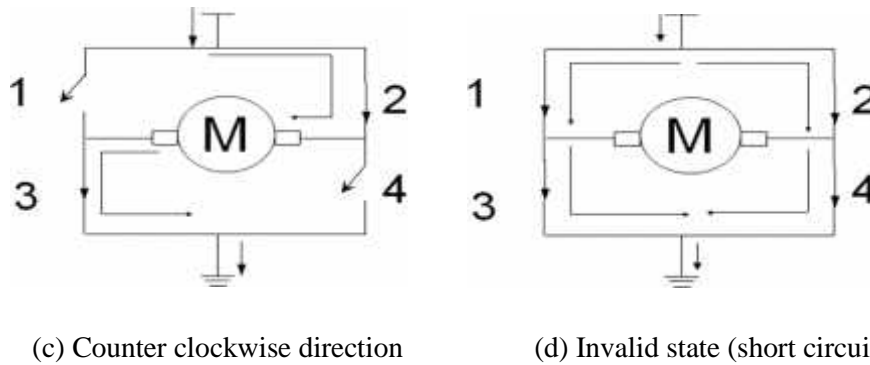


Figure 4: H-Bridge Motor Configuration

Figure 4 shows the H-Bridge motor configuration. It consists of four switches and based on the closing and opening of these switches the motor either rotates in clockwise or anti-clockwise direction.

As seen in figure 4a, all the switches are open hence the motor is not running. In b, turning of the motor is in one direction when the switches 1 and 4 are closed that is clockwise direction.

Similarly, in c the switches 2 and 3 are closed so the motor rotates in anticlockwise direction, while in figure 4d all the switches are closed which indicates a invalid state or a short circuit.

The interfacing diagram of 8051 to bidirectional motor control can be referred to fig 17-18 from text prescribed.

Example 6: A switch is connected to pin P2.7. Write an ALP to monitor the status of the SW. If SW = 0, DC motor moves clockwise and if SW = 1, DC motor moves anticlockwise.

Program:

```

ORG 0000H
CLR P1.0
CLR P1.1
CLR P1.2
CLR P1.3
SETB P2.7
MONITOR: JNB P2.7, CLOCK
          SETB P1.0
          CLR P1.1
          CLR P1.2
          SETB P1.3
          SJMP MONITOR
CLOCK:   CLR P1.0
          SETB P1.1
          SETB P1.2
          CLR P1.3
          SJMP MONITOR
END

```



## Pulse Width Modulation (PWM):

The speed of the motor depends on 3 parameters: load, voltage and current. For a given load, we can maintain a steady speed by using PWM. By changing the width of the pulse applied to DC motor, power provided can either be increased or decreased. Though voltage has fixed amplitude, has a variable duty cycle. The wider the pulse, higher the speed obtained. One of the reasons as to why dc motor are referred over ac is, the ability to control the speed of the DC motor using PWM. The speed of the ac motor is dictated by the ac frequency of voltage applied to the motor and is generally fixed. Hence, speed of the AC motors cannot be controlled when load is increased.

Figure 5 below shows the pulse width modulation comparison.

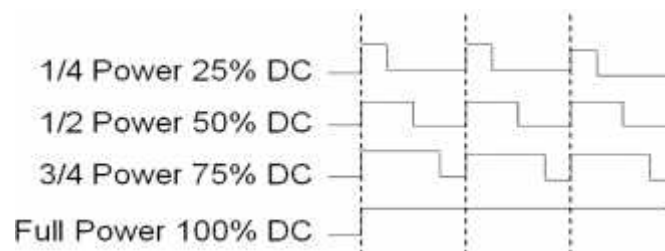


Figure 5: PWM comparison

Example 7 and 8 are the 8051 C version of the programs written earlier.

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

Program:

```
#include <reg51.h>
sbit SW =P2^7;
sbit Enable = P1^0;
sbit MTR_1 = P1^1;
sbit MTR_2 = P1^2;
void main ( )
{
    SW=1;
    Enable = 0;
    MTR_1=0;
    MTR_2=0;
while( )
{
    Enable =1;
    if( SW==1)
    {
        MTR_1=1;
        MTR_2=0;
    }
    else
    {
        MTR_1=0;
        MTR_2=1;
    }
}}
```

Example 8: A switch is connected to pin P2.7. Write an C to monitor the status of the SW. If SW = 0, DC motor moves 50% duty cycle pulse and if SW = 1, DC motor moves with 25% duty cycle pulse.

Program:

```
#include <reg51.h>
sbit SW = P2^7;
sbit MTR = P1^0;
void main ( )
{
    SW=1;
    MTR=0;
while( )
{
    if( SW==1)
    {
        MTR=1;
        Msdelay(25);
        MTR=0;
        Msdelay(75);
    }
    else
    {
        MTR=1;
        Msdelay(50);
        MTR=0;
        Msdelay(50);
    }
}
}
```

The interfacing diagrams for the above examples can be referred to the text.

### **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  $2^n$ , where n is the number of data bit inputs.

### DAC0808:

The digital inputs are converted to current  $I_{out}$ , and by connecting a resistor to the  $I_{out}$  pin, we can convert the result to voltage. The total current  $I_{out}$  is a function of the binary numbers at the D0-D7 inputs of the DAC0808 and the reference current  $I_{ref}$ , and is as follows:

$$I_{out} = I_{ref} \left( \frac{D_7}{2} + \frac{D_6}{4} + \frac{D_5}{8} + \frac{D_4}{16} + \frac{D_3}{32} + \frac{D_2}{64} + \frac{D_1}{128} + \frac{D_0}{256} \right)$$

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 6 below.

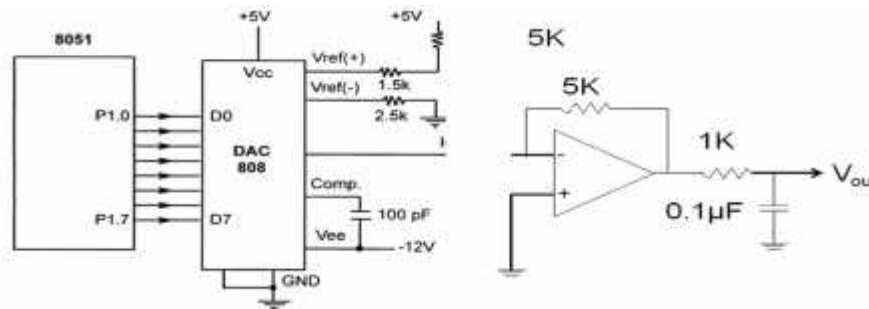


Figure 6: 8051 connection to DAC0808

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.

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

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.

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

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.

0804

### Pin Description of ADC804:

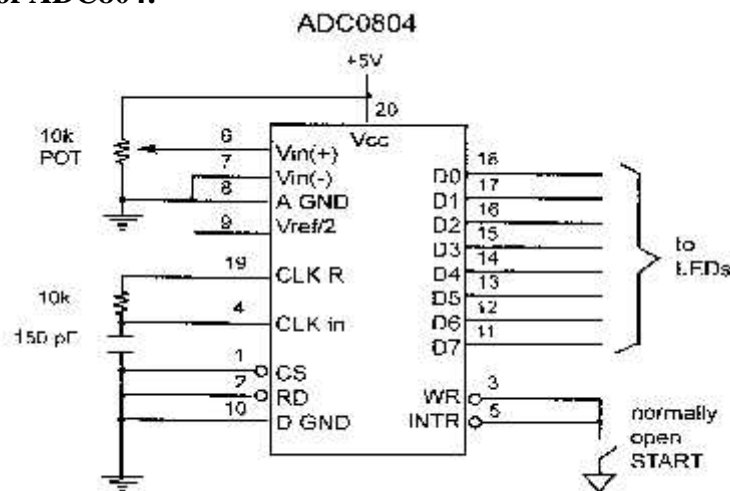


Figure 7: Pin out of ADC0804

- **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:

$$f = \frac{1}{1.1RC}$$

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

Vref/2(v)	Vin(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

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

- D<sub>out</sub> = digital data output (in decimal),
  - V<sub>in</sub> = 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 V<sub>in</sub> and digital ground is connected to the ground of the V<sub>cc</sub> pin. To isolate the analog V<sub>in</sub> signal from transient voltages caused by digital switching of the output D0 – D7. This contributes to the accuracy of the digital data output.
  - **V<sub>in</sub>(+) & V<sub>in</sub>(-):** Differential analog inputs where V<sub>in</sub> = V<sub>in</sub> (+) – V<sub>in</sub> (-). V<sub>in</sub> (-) is connected to ground and V<sub>in</sub> (+) 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 V<sub>in</sub> 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

Figure 8 shows the read and write timing for ADC804. Figure 9 and 10 shows the self clocking with the RC component for frequency and the external frequency connected to XTAL2 of 8051.

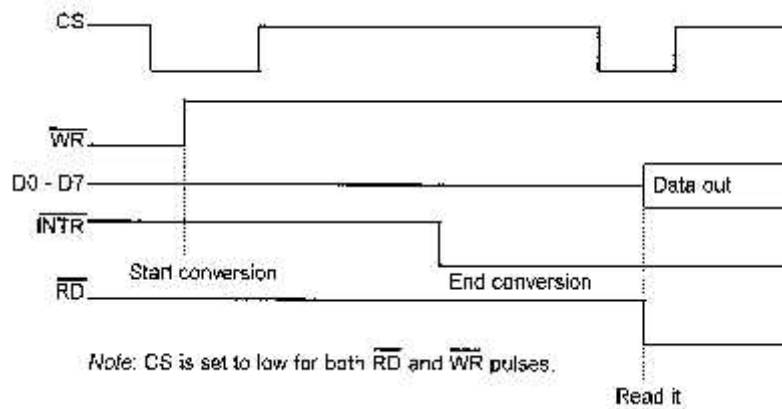


Figure 8: Read and Write timing for ADC804

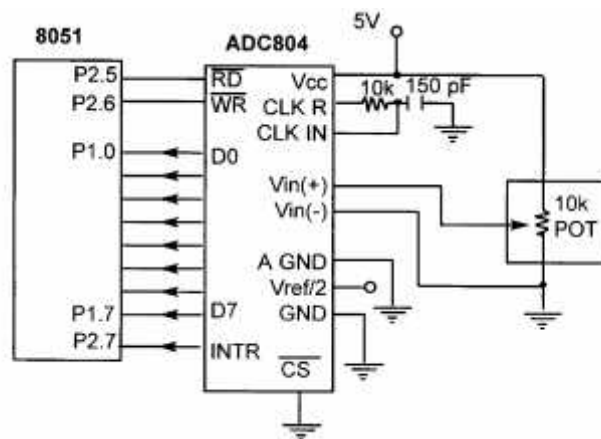


Figure 9: 8051 Connection to ADC804 with Self-clocking

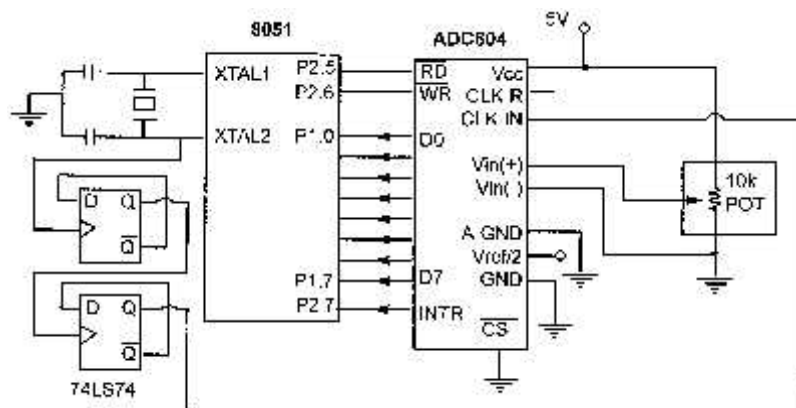


Figure 10: 8051 Connection to ADC804 with Clock from XTAL2 of 8051

Now let us see how we write assembly as well as C program for the interfacing diagram shown in figure 10.

**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 11 below. (Explanation can be done as is with ADC0804).



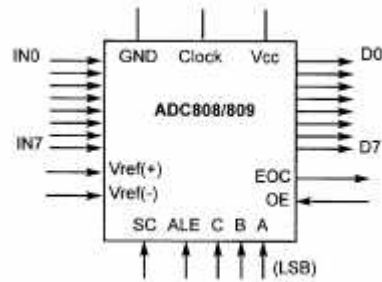


Figure 11: Pin out of ADC0808

### Steps to Program ADC0808/809

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.

Let us write an assembly and C program for the interfacing of 8051 to ADC0808 as shown in figure 12 below.(Figure 12 can be referred from the text prescribed.)

Programming ADC0808/809 in assembly

```

        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

```

Note: replace the assembly instructions with equivalent C statements for programming ADC0808 in C

## 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

**LCD 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
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 figure 14 and 15.

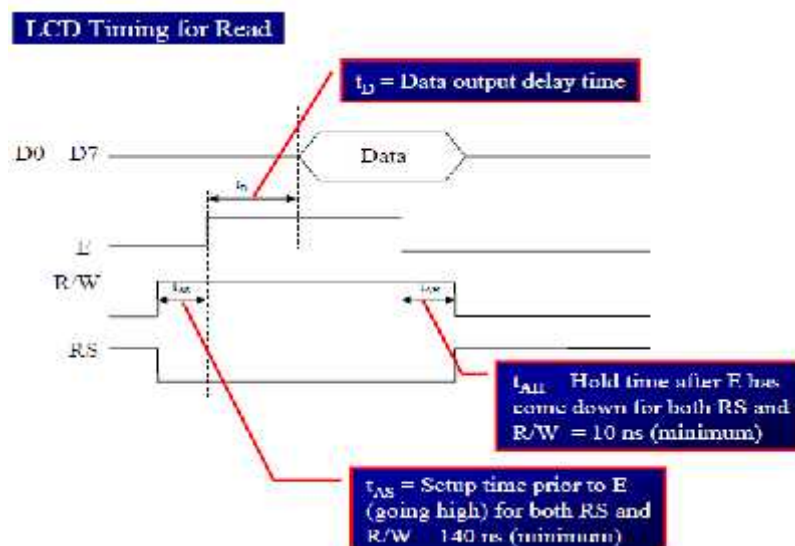


Figure 14: LCD timing for read

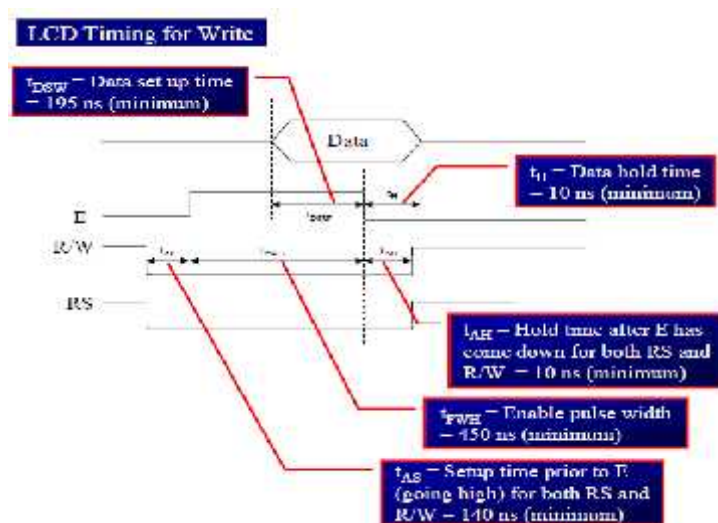


Figure 15: LCD timing for write

### 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. This is shown in the code below. The interfacing diagram of LCD to 8051 is as shown in the figure 16.

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

```

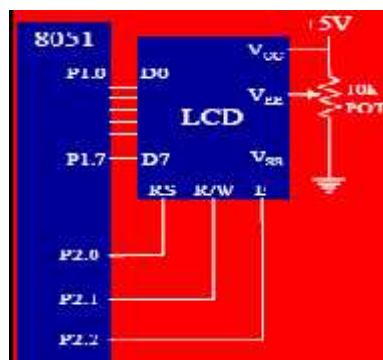


Figure 16: 8051 Connection to LCD

**Sending Data/ Commands to LCDs checking the Busy Flag**

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++);
}

```

### Keyboard Interfacing:

Keyboards are organized in a matrix of rows and columns. The CPU accesses both rows and columns through ports. Therefore, with two 8-bit ports, an 8 x 8 matrix of keys can be connected to a microprocessor. When a key is pressed, a row and a column make a contact. Otherwise, there is no connection between rows and columns. A 4x4 matrix connected to two ports. The rows are connected to an output port and the columns are connected to an input port.

#### Scanning and Identifying the Key:

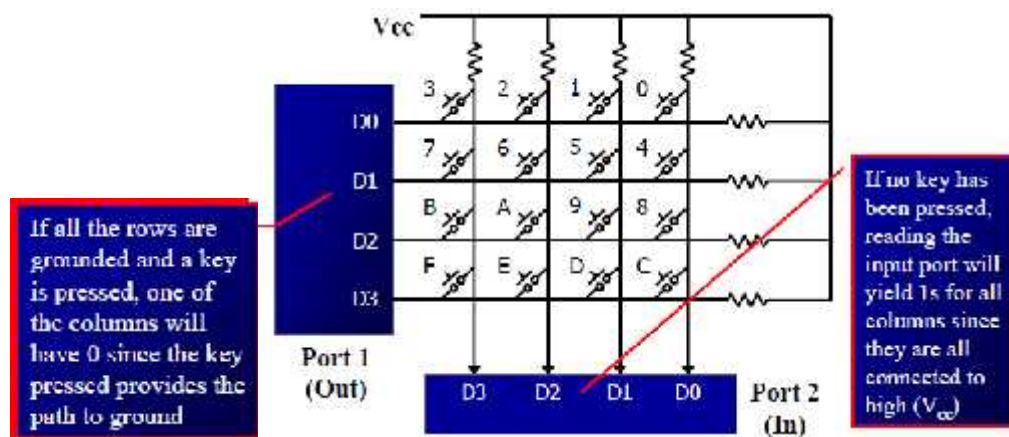


Figure 17: A 4X4 matrix keyboard

It is the function of the microcontroller to scan the keyboard continuously to detect and identify the key pressed

- To detect a pressed key, the microcontroller grounds all rows by providing 0 to the output latch, then it reads the columns
- If the data read from columns is D3 – D0 =1111, no key has been pressed and the process continues till key press is detected

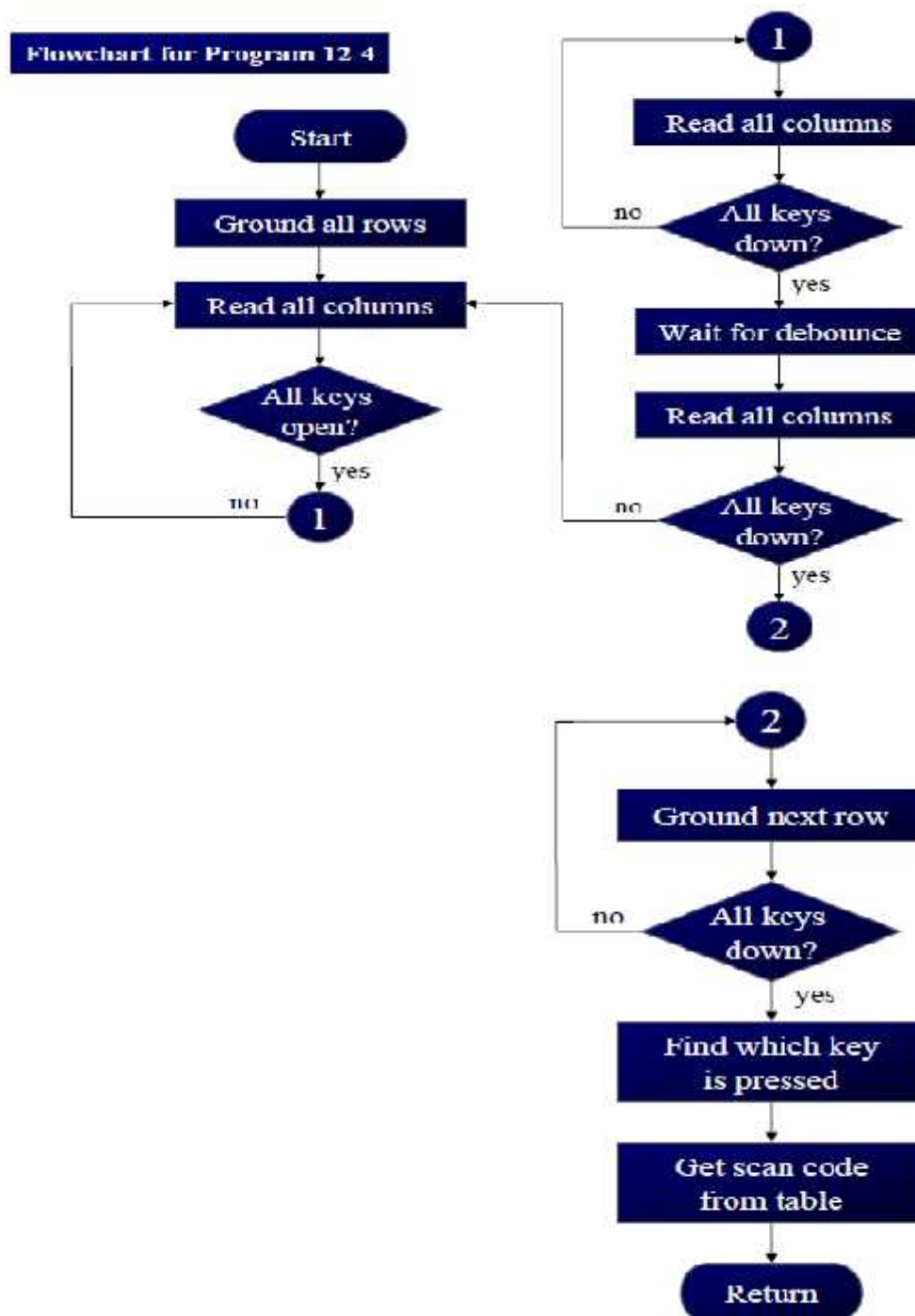


- If one of the column bits has a zero, this means that a key press has occurred. For example, if  $D3 - D0 = 1101$ , this means that a key in the D1 column has been pressed. After detecting a key press, the microcontroller will go through the process of identifying the key.
- Starting with the top row, the microcontroller grounds it by providing a low to row D0 only. It reads the columns; if the data read is all 1s, no key in that row is activated and the process is moved to the next row.
- It grounds the next row, reads the columns, and checks for any zero. This process continues until the row is identified.
- After identification of the row in which the key has been pressed, find out which column the pressed key belongs to.

**Algorithm for detection and identification of key activation goes through the following stages:**

1. To make sure that the preceding key has been released, 0s are output to all rows at once, and the columns are read and checked repeatedly until all the columns are high.
  - When all columns are found to be high, the program waits for a short amount of time before it goes to the next stage of waiting for a key to be pressed.
2. To see if any key is pressed, the columns are scanned over and over in an infinite loop until one of them has a 0 on it.
  - Remember that the output latches connected to rows still have their initial zeros (provided in stage 1), making them grounded.
  - After the key press detection, it waits 20 ms for the bounce and then scans the columns again.
    - (a) It ensures that the first key press detection was not an erroneous one due to a spike noise.
    - (b) The key press. If after the 20-ms delay the key is still pressed, it goes back into the loop to detect a real key press.
3. To detect which row key press belongs to, it grounds one row at a time, reading the columns each time.
  - If it finds that all columns are high, this means that the key press cannot belong to that row. Therefore, it grounds the next row and continues until it finds the row the key press belongs to.
  - Upon finding the row that the key press belongs to, it sets up the starting address for the look-up table holding the scan codes (or ASCII) for that row.
4. To identify the key press, it rotates the column bits, one bit at a time, into the carry flag and checks to see if it is low.
  - Upon finding the zero, it pulls out the ASCII code for that key from the look-up table.
  - otherwise, it increments the pointer to point to the next element of the look-up table.

The flowchart for the above algorithm is as shown below:



Note: The assembly as well as the C program can be written in accordance to the algorithm of the flowchart shown.

## **Summary**

This chapter gives the details of six different devices that can be interfaced to 8051. These are widely used in many applications. Initially, we discussed about the stepper motor, giving the details on the working, sending sequence and hence writing assembly and C program. In continuation to that we also learnt how to interface DC motor, and DC motors with PWM. The chapter also covers the study of devices such as DAC, parallel ADC and serial ADC, LCD and Keyboard along with the interfacing of these devices to 8051. We further, studied how to write assembly and C program for all the above said interfaces which will help in developing applications.