MODULE-5: Interfacing

Structure

5.1 Interfacing: LCD interfacing,

5.2 Keyboard interfacing.

5.3 ADC, DAC and sensor interfacing: ADC 0808 interfacing to 8051,

5.4 Serial ADC Max1112 ADC interfacing to 8051,

5.5 DAC interfacing, Sensor interfacing and signal conditioning.

5.6 Motor control: Relay, PWM, DC and stepper motor: Relays and opt isolators,

5.7 stepper motor interfacing,

5.8 DC motor interfacing and PWM.

5.9 8051 interfacing with 8255: Programming the 8255, 8255 interfacing, C programming for 8255

Objectives

1. To Interface 8051 with real-world devices such as LCDs and keyboards, ADC, DAC chips and sensors.
2. To Interface 8031/51 with external memories, 8255 chip to add ports and relays, optisolators and motors.

5.1 Interfacing: LCD interfacing

LCD operation

In recent years the LCD is finding widespread use replacing LEDs (seven-segment LEDs or other multi segment LEDs). This is due to the following reasons:

1. The declining prices of LCDs.
    1. The ability to display numbers, characters, and graphics. This is in contrast to LEDs, which are limited to numbers and a few characters.
    2. Incorporation of a refreshing controller into the LCD, thereby relieving the CPU of the task of refreshing the LCD. In contrast, the LED must be refreshed by the CPU (or in some other way) to keep displaying the data.
2. Ease of programming for characters and graphics.

LCD pin descriptions

The LCD discussed in this section has 14 pins. The function of each pin is given in Table 12-1. Figure 12-1 shows the pin positions for various LCDs.

^{V}CC> ^{V}SS> and ^{V}EE

While V_{cc} and V_{ss} provide +5V and ground, respectively, V_{EE} is used for controlling LCD contrast.

RS, register select

There are two very important registers inside the LCD. The RS pin is used for their selection as follows. If RS = 0, the instruction command code register is selected, allowing the user to send a command such as clear display, cursor at home, etc. If RS = 1 the data register is selected, allowing the user to send data to be displayed on the LCD.

R/W, read/write

R/W input allows the user to write information to the LCD or read information from it. R/W = 1 when reading; R/W = 0 when writing.

E, enable

The enable pin is used by the LCD to latch information presented to its data pins. When data is supplied to data pins, a high-to-low pulse must be applied to this pin in order for the LCD to latch in the data present at the data pins. This pulse must be a minimum of 450 ns wide.

DO-D7

1. The 8-bit data pins, DO – D7, are used to send information to the LCD or read the contents of the LCD's internal registers.
2. To display letters and numbers, we send ASCII codes for the letters A – Z, a – z, and numbers 0 – 9 to these pins while making RS = 1.
3. We also use RS = 0 to check the busy flag bit to see if the LCD is ready to receive information. The busy flag is D7 and can be read when R/W = 1 and RS = 0, as follows: if R/W= 1, RS = 0. When D7 = 1 (busy flag = 1), the LCD is busy taking care of internal operations and will not accept any new information. When D7 = 0, the LCD is ready to receive new information.

Note: It is recommended to check the busy flag before writing any data to the LCD.

Table 5.1: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
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 of 1st line
C0	Force cursor to beginning of 2nd line
38	2 lines and 5x7 matrix

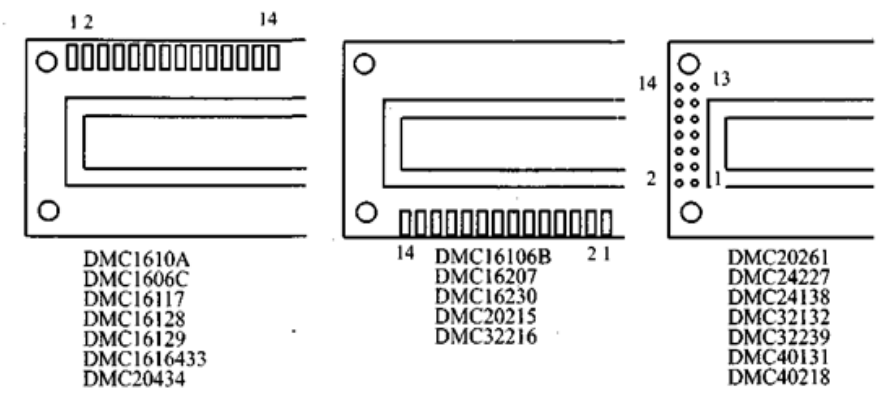


Figure 5.1. Pin Positions for Various LCDs from Optrex

Sending commands and data to LCDs with a 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.

Program 5-1: Communicating with LCD using a delay

```
;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,#84H          ;cursor at line 1,pos. 4
          ACALL  COMNWRT         ;call command subroutine
          ACALL  DELAY           ;give LCD some time
          MOV    A,#'N'          ;display letter N
          ACALL  DATAWRT         ;call display subroutine
          ACALL  DELAY           ;give LCD some time
          MOV    A,#'O'          ;display letter O
          ACALL  DATAWRT         ;call display subroutine
AGAIN:    SJMP   AGAIN           ;stay here
COMNWRT:                         ;send command to LCD
          MOV    P1,A            ;copy reg A to port1
          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 port1
          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
```

5.1.1 Sending code or data to the LCD with checking busy flag

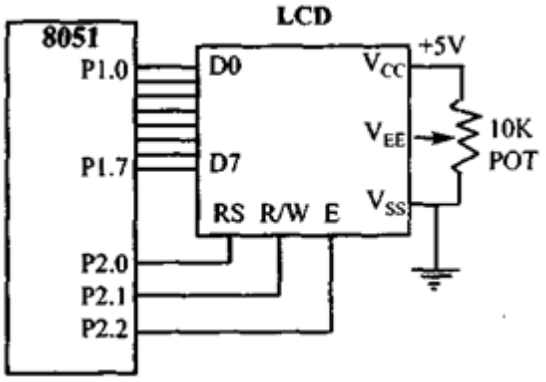
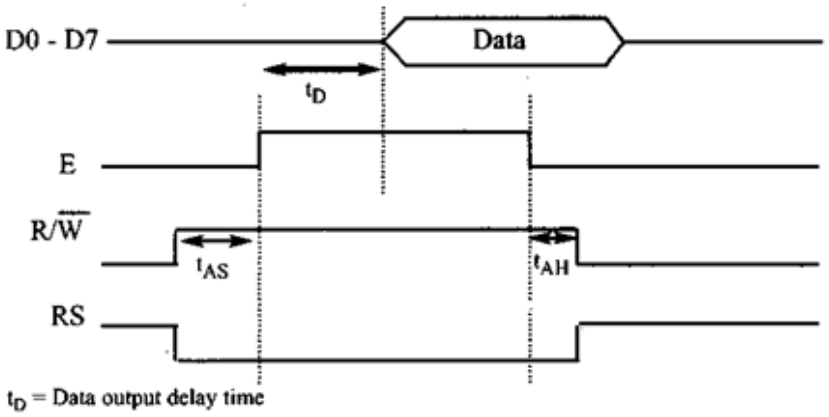


Figure 5.2: LCD Connections

Program 5- 2: Communicating with LCD using the busy flag

```
;Check busy flag before sending data, command to LCD
;P1=data pin,P2.0=RS,P2.1=R/W,P2.2=E pins
         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
          ACALL  DELAY          ;give LCD some time
          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
;read command reg and check busy flag
BACK:     CLR    P2.2           ;E=0 for L-to-H pulse
          ACALL  DELAY          ;give LCD some time
          SETB   P2.2           ;E=1 L-to-H pulse
          JB     P1.7,BACK      ;stay until busy flag=0
          RET
          END
```

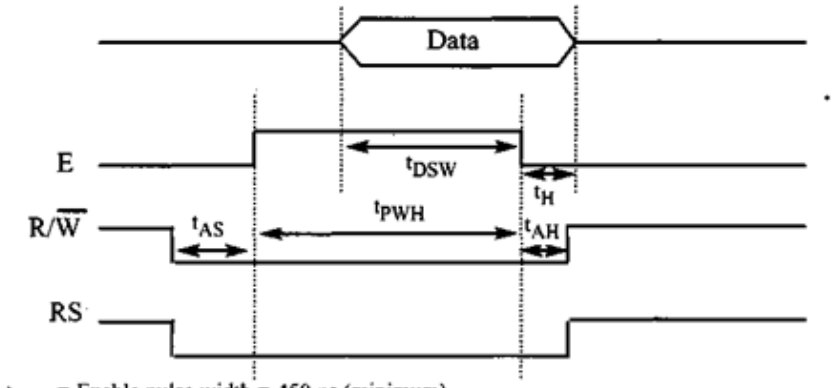


t_D = Data output delay time
t_{AS} = Setup time prior to E (going high) for both RS and R/W = 140 ns (minimum)
t_{AH} = Hold time after E has come down for both RS and R/W = 10 ns (minimum)

Note: Read requires an L-to-H pulse for the E pin.

Figure 5.3. LCD Timing for Read (L-to-H for E line)

Notice in the above program that the busy flag is D7 of the command register. To read the command register we make R/W = 1 and RS = 0, and a L-to-H pulse for the E pin will provide us the command register. After reading the command register, if bit D7 (the busy flag) is high, the LCD is busy and no information (command or data) should be issued to it. Only when D7 = 0 can we send data or commands to the LCD. Notice in this method that no time delays are used since we are checking the busy flag before issuing commands or data to the LCD. Contrast the Read and Write timing for the LCD. Note that the E line is negative-edge triggered for the write while it is positive-edge triggered for the read.



t_{PWH} = Enable pulse width = 450 ns (minimum)
t_{DSW} = Data setup time = 195 ns (minimum)
t_H = Data hold time = 10 ns (minimum)
t_{AS} = Setup time prior to E (going high) for both RS and R/W = 140 ns (minimum)
t_{AH} = Hold time after E has come down for both RS and R/W = 10 ns (minimum)

Figure 5.4. LCD Timing for Write (H-to-L for E line)

5.1.2 LCD data sheet

In the LCD, one can put data at any location. The following shows address locations and how they are accessed.

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	1	A	A	A	A	A	A	A

where AAAAAAA = 0000000 to 0100111 for line 1 and AAAAAAA – 1000000 to 1100111 for line 2.

Table 5.2: LCD addressing

	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
Line 1 (min)	1	0	0	0	0	0	0	0
Line 1 (max)	1	0	1	0	0	1	1	1
Line 2 (min)	1	1	0	0	0	0	0	0
Line 2 (max)	1	1	1	0	0	1	1	1

The upper address range can go as high as 0100111 for the 40-charac-ter-wide LCD, while for the 20-character-wide LCD it goes up to 010011 (19 decimal = 10011 binary). Notice that the upper range 0100111 (binary) = 39 decimal, which corresponds to locations 0 to 39 for the LCDs of 40×2 size.

16 x 2 LCD	80	81	82	83	84	85	86 through 8F
	C0	C1	C2	C3	C4	C5	C6 through CF
20 x 1 LCD	80	81	82	83	through 93		
20 x 2 LCD	80	81	82	83	through 93		
	C0	C1	C2	C3	through D3		
20 x 4 LCD	80	81	82	83	through 93		
	C0	C1	C2	C3	through D3		
	94	95	96	97	through A7		
	D4	D5	D6	D7	through E7		
40 x 2 LCD	80	81	82	83	through A7		
	C0	C1	C2	C3	through E7		

Note: All data is in hex.

Figure 5.5: Cursor Addresses for Some LCDs

Example 5-3

Write an 8051 C program to send letters ‘M’, ‘D’, and ‘£’ to the LCD using delays.

Solution:

```
#include <reg51.h>
sfr ldata = 0x90;                    //P1=LCD data pins (Fig. 12-2)
sbit rs = P2^0;
sbit rw = P2^1;
sbit en = P2^2;
void main()
  {
    lcdcmd(0x38);
    MSDelay(250);
    lcdcmd(0x0E);
    MSDelay(250);
    lcdcmd(0x01);
    MSDelay(250);
    lcdcmd(0x06);
    MSDelay(250);
    lcdcmd(0x86);                    //line 1, position 6
    MSDelay(250);
    lcddata('M');
    MSDelay(250);
    lcddata('D');
    MSDelay(250);
    lcddata('E');
  }

void lcdcmd(unsigned char value)
  {
    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)
  {
    ldata = value;                   // put the value on the pins
    rs = 1;
    rw = 0;
    en = 1;                          // strobe the enable pin
    MSDelay(1);
    en = 0;
    return;
  }

void MSDelay(unsigned int itime)
  {
    unsigned int i, j;
    for(i=0;i<itime;i++)
      for(j=0;j<1275;j++);
  }
```

Example 5-3

Repeat Example 5-3 using the busy flag method.

Solution:

```
#include <reg51.h>
sfr ldata = 0x90;                    //P1=LCD data pins (Fig. 12-2)
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('M');
    lcddata('D');
    lcddata('E');
  }

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 an input
    rs = 0;
    rw = 1;
    while(busy==1)                   //wait here for busy flag
      {
        en = 0;                      //strobe the enable pin
        MSDelay(1);
        en = 1;
      }
    return;
  }
```

```
void MSDelay(unsigned int itime)
  {
    unsigned int i, j;
    for(i=0;i<itime;i++)
      for(j=0;j<1275;j++);
  }
```

5.2 Keypad Interfacing

Interfacing the keyboard to the 8051

1. At the lowest level, 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.
2. When a key is pressed, a row and a column make a contact; otherwise, there is no connection between rows and columns. In IBM PC keyboards, a single microcontroller (consisting of a microprocessor, RAM and EPROM, and several ports all on a single chip) takes care of hardware and software interfacing of the keyboard.
3. In such systems, it is the function of programs stored in the EPROM of the microcontroller to scan the keys continuously, identify which one has been activated, and present it to the motherboard. In this section we look at the mechanism by which the 8051 scans and identifies the key.

5.2.1 Scanning and identifying the key

1. Figure5.5 shows a 4 x 4 matrix connected to two ports. The rows are connected to an output port and the columns are connected to an input port.
2. If no key has been pressed, reading the input port will yield 1 s for all columns since they are all connected to high (V_{cc}).
3. If all the rows are grounded and a key is pressed, one of the columns will have 0 since the key pressed provides the path to ground. It is the function of the microcontroller to scan the keyboard continuously to detect and identify the key pressed. How it is done is explained next.

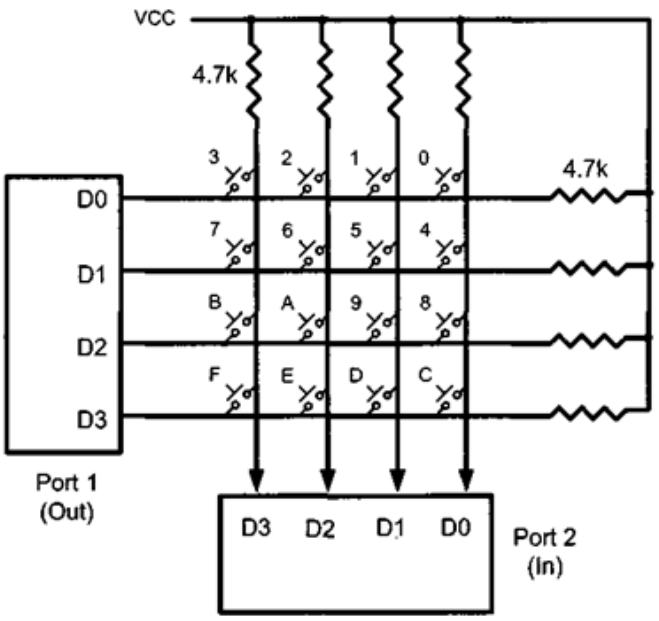


Figure 5.6: Matrix Keyboard Connection to Ports

Grounding rows and reading the columns

1. 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 the columns is D3 – DO = 1111, no key has been pressed and the process continues until a key press is detected.
2. However, if one of the column bits has a zero, this means that a key press has occurred. For example, if D3 – DO = 1101, this means that a key in the Dl column has been pressed.
3. After a key press is detected, 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 DO only; then it reads the columns.
4. If the data read is all Is, 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

Example 5-4

Identify the row and column of the pressed key for each of the following.

1. D3 – DO = 1110 for the row, D3 – DO = 1011 for the column
2. D3 – DO = 1101 for the row, D3 – DO = 0111 for the column

Solution:

The row and column can be used to identify the key.

1. The row belongs to DO and the column belongs to D2; therefore, key number 2 was pressed.
2. The row belongs to Dl and the column belongs to D3; therefore, key number 7 was pressed

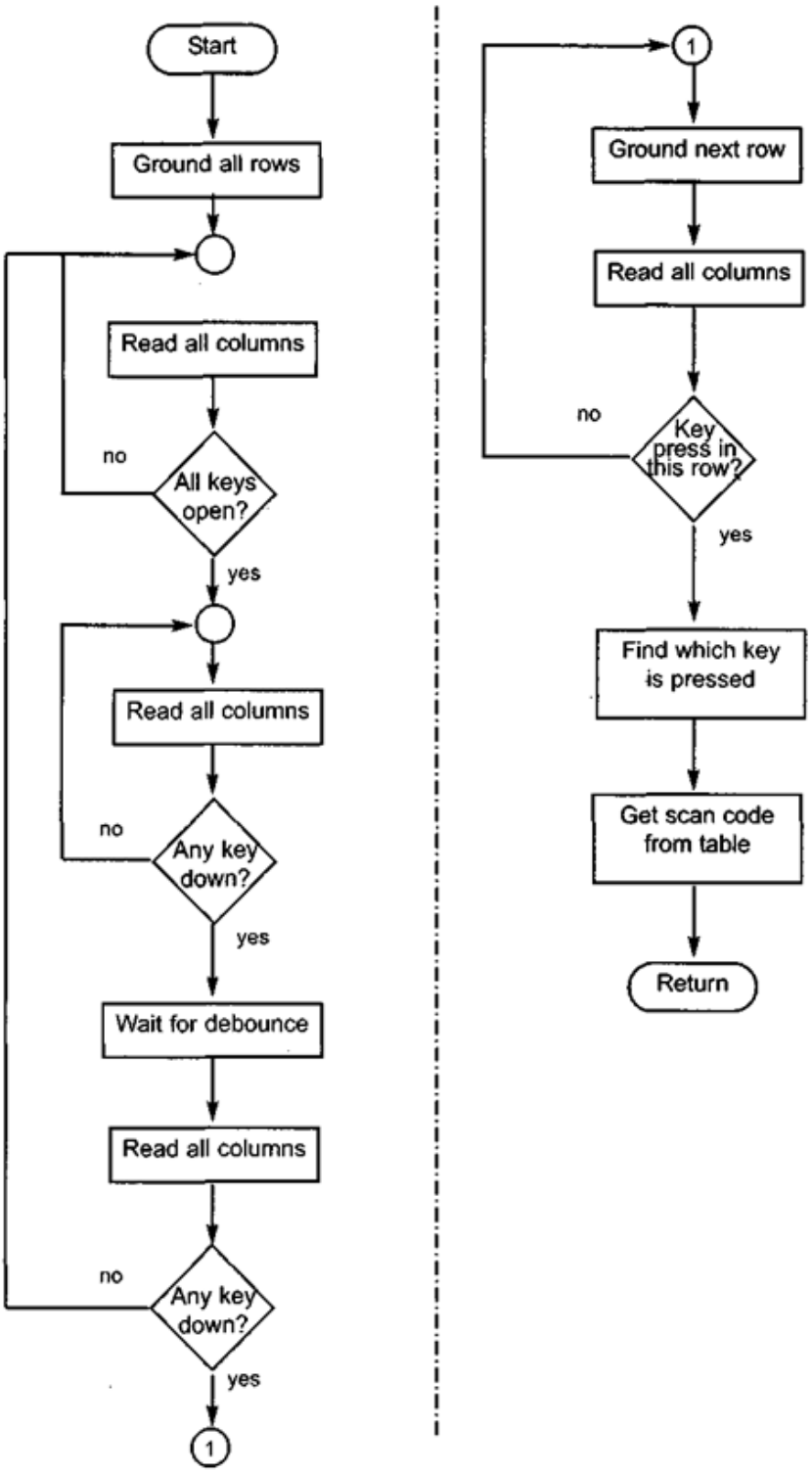


Figure 5.7: Flowchart for Program

Program goes through the following four major stages:

1. To make sure that the preceding key has been released, Os 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, the microcontroller waits 20 ms for the bounce and then scans the columns again. This serves two functions: (a) it ensures that the first key press detection was not an erroneous one due to a spike noise, and (b) the 20-ms delay prevents the same key press from being interpreted as a multiple key press. If after the 20Tins delay the key is still pressed, it goes to the next stage to detect which row it belongs to; otherwise, it goes back into the loop to detect a real key press.

3. To detect which row the key press belongs to, the microcontroller 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 the ASCII value) for that row and goes to the next stage to identify the key.

4. To identify the key press, the microcontroller 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.

Example 5-5 shows keypad programming in 8051 C.

```
;Keyboard subroutine. This program sends the ASCII code
;for pressed key to P0.1
;P1.0-P1.3 connected to rows P2.0-P2.3 connected to columns
        MOV   P2,#0FFH            ;make P2 an input port
K1:     MOV   P1,#0               ;ground all rows at once
        MOV   A,P2          ;read all col. ensure all keys open
        ANL   A,#00001111B        ;masked unused bits
        CJNE  A,#00001111B,K1     ;check til all keys released
  :     ACALL DELAY               ;call 20 ms delay
        MOV   A,P2                ;see if any key is pressed
        ANL   A,#00001111B        ;mask unused bits
        CJNE  A,#00001111B,OVER   ;key pressed, await closure
        SJMP  K2                  ;check if key pressed
OVER:   ACALL DELAY               ;wait 20 ms debounce time
        MOV   A,P2                ;check key closure
        ANL   A,#00001111B        ;mask unused bits
        CJNE  A,#00001111B,OVER1  ;key pressed, find row
        SJMP  K2                  ;if none, keep polling
OVER1:  MOV   P1,#11111110B       ;ground row 0
        MOV   A,P2                ;read all columns
        ANL   A,#00001111B        ;mask unused bits
        CJNE  A,#00001111B,ROW_0  ;key row 0, find the col.
        MOV   P1,#11111101B       ;ground row 1
        MOV   A,P2                ;read all columns
        ANL   A,#00001111B        ;mask unused bits
        CJNE  A,#00001111B,ROW_1  ;key row 1, find the col.
        MOV   P1,#11111011B       ;ground row 2
        MOV   A,P2                ;read all columns
        ANL   A,#00001111B        ;mask unused bits
        CJNE  A,#00001111B,ROW_2  ;key row 2, find the col.
        MOV   P1,#11110111B       ;ground row 3
        MOV   A,P2                ;read all columns
        ANL   A,#00001111B        ;mask unused bits
        CJNE  A,#00001111B,ROW_3  ;key row 3, find the col.
        LJMP  K2                  ;if none, false input, repeat

ROW_0:  MOV   DPTR,#KCODE0        ;set DPTR=start of row 0
        SJMP  FIND                ;find col. key belongs to
ROW_1:  MOV   DPTR,#KCODE1        ;set DPTR=start of row 1
        SJMP  FIND                ;find col. key belongs to
ROW_2:  MOV   DPTR,#KCODE2        ;set DPTR=start of row 2
        SJMP  FIND                ;find col. key belongs to
ROW_3:  MOV   DPTR,#KCODE3        ;set DPTR=start of row 3
FIND:   RRC   A                   ;see if any CY bit is low
        JNC   MATCH               ;if zero, get the ASCII code
        INC   DPTR                ;point to next col. address
```

```
        SJMP   FIND                ;keep searching
MATCH:  CLR    A                   ;set A=0 (match is found)
        MOVC   A,@A+DPTR           ;get ASCII code from table
        MOV    P0,A                ;display pressed key
        LJMP   K1
;ASCII LOOK-UP TABLE FOR EACH ROW
        ORG    300H
KCODE0: DB     '0','1','2','3'          ;ROW 0
KCODE1: DB     '4','5','6','7'          ;ROW 1
KCODE2: DB     '8','9','A','B'          ;ROW 2
KCODE3: DB     'C','D','E','F'          ;ROW 3
       END
```

Program 5-6: Keyboard Program

Solution:

```
#include <reg51.h>

#define COL   P2            //define ports for easier reading
#define ROW   P1

void MSDelay(unsigned int value);
void SerTX(unsigned char);

unsigned char keypad[4][4] =     {'0','1','2','3',
                                  '4','5','6','7',
                                  '8','9','A','B',
                                  'C','D','E','F'};

void main()
  {
    unsigned char colloc, rowloc;

    TMOD = 0x20;                 //timer 1, mode 2
    TH1 = -3;                    //9600 baud
    SCON = 0x50;                 //8-bit, 1 stop bit
    TR1 = 1;                     //start timer 1

  //keyboard routine. This sends the ASCII
  //code for pressed key to the serial port
  COL = 0xFF;                    //make P2 an input port
  while(1)                       //repeat forever
    {
    do
      {
      ROW = 0x00;                //ground all rows at once
      colloc = COL;              //read the columns
      colloc &= 0x0F;            //mask used bits
```

```
  } while(colloc != 0x0F);  //check until all keys released

do
  {
  do
    {
    MSDelay(20);            //call delay
    colloc = COL;           //see if any key is pressed
    colloc &= 0x0F;         //mask unused bits
    } while(colloc == 0x0F);//keep checking for keypress

  MSDelay(20);              //call delay for debounce
  colloc = COL;             //read columns
  colloc &= 0x0F;           //mask unused bits
  } while(colloc == 0x0F);  //wait for keypress

  while(1)
    {
      ROW = 0xFE;           //ground row 0
      colloc = COL;         //read columns
      colloc &= 0x0F;       //mask unused bits
      if(colloc != 0x0F)    //column detected
        {
          rowloc = 0;       //save row location
          break;            //exit while loop
        }

      ROW = 0xFD;           //ground row 1
      colloc = COL;         //read columns
      colloc &= 0x0F;       //mask unused bits
      if(colloc != 0x0F)    //column detected
        {
          rowloc = 1;       //save row location
          break;            //exit while loop
        }

      ROW = 0xFB;           //ground row 2
      colloc = COL;         //read columns
      colloc &= 0x0F;       //mask unused bits
      if(colloc != 0x0F)    //column detected
        {
          rowloc = 2;       //save row location
          break;            //exit while loop
        }

      ROW = 0xF7;           //ground row 3
      colloc = COL;         //read columns
      colloc &= 0x0F;       //mask unused bits
      rowloc = 3;           //save row location
      break;                //exit while loop
    }

  //check column and send result to the serial port
  if(colloc == 0x0E)
    SerTX(keypad[rowloc][0]);
  else if(colloc == 0x0D)
    SerTX(keypad[rowloc][1]);
  else if(colloc == 0x0B)
    SerTX(keypad[rowloc][2]);
```

```
        else
          SerTX(keypad[rowloc][3]);
        }

   }

void SerTX(unsigned char x)
   {
     SBUF = x;                  //place value in buffer
     while(TI==0);              //wait until transmitted
     TI = 0;                    //clear flag
   }

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

5.3 ADC, DAC and sensor interfacing: ADC 0808 interfacing to 8051

1. Analog to digital converters are among the most widely used devices for data acquisitions. Digital computers use binary (discrete) value but in physical world everything is analog (continuous).
2. A physical quantity is converted to electrical signals using device called transducer or also called as sensors. Sensors and many other natural quantities produce an output that is voltage (or current).
3. Therefore we need an analog - to - digital converter to translate the analog signal to digital numbers so that the microcontroller can read and process them. An ADC has an n bit resolution where n can be 8, 10, 16, 0r even 24 bits. The higher resolution ADC provides a smaller step size, where step size is smallest change that can be discerned by an ADC.

Table 5.3: Resolution vs. Step Size for ADC

n-bit	Number of steps	Step Size (mV)
8	256	5/256 = 19.53
10	1024	5/1024 = 4.88
12	4096	5/4096 = 1.2
16	65536	5/65536 = 0.076

In addition to resolution, 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 digital (binary) number. The ADC chips are either parallel or serial. In parallel ADC, we have 8 or more pins dedicated to bring out the binary data, but in serial ADC we have only one pin for data out.

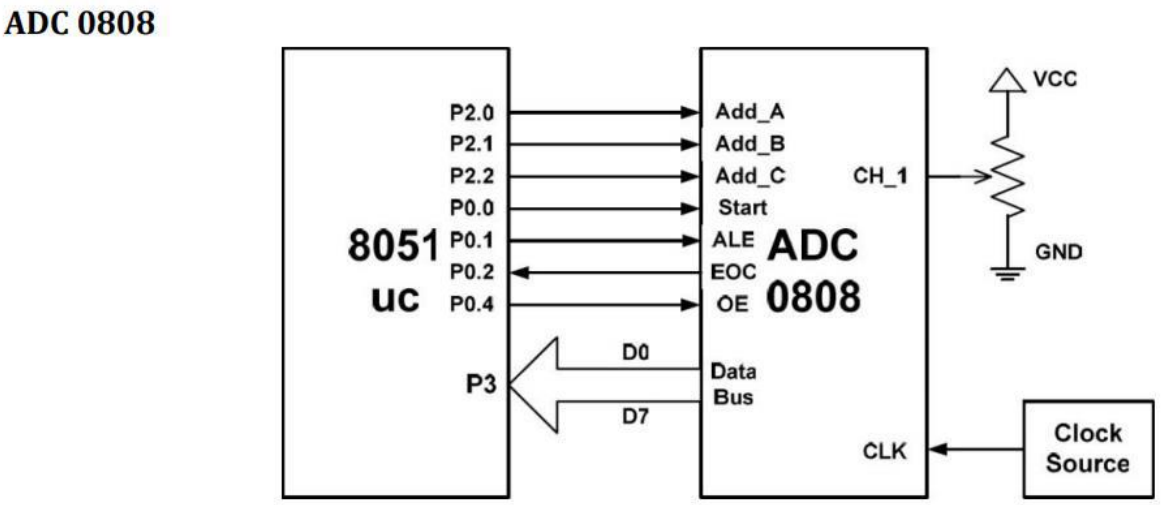


Figure 5.8: ADC interfacing

ADC0808 has 8 analog inputs. ADC0808 allows us to monitor up to 8 different analog inputs using only a single chip. ADC0808 has an 8-bit data output. The 8 analog inputs channels are multiplexed and selected according to table given below using three address pins, A, B, and C.

a) CS

Chip select is an active low input used to activate the ADC0804 chip. To access the ADC0804, this pin must be low.

b) RD (read)

This is an input signal and is active low. The ADC converts the analog input to its binary equivalent and holds it in an internal register. RD is used to get the converted data out of the ADC0804 chip. When CS = 0, if a high-to-low pulse is applied to the RD pin, the 8-bit digital output shows up at the DO – D7 data pins. The RD pin is also referred to as output enable (OE).

c) WR (write; a better name might be "start conversion")

This is an active low input used to inform the ADC0804 to start the conversion process. If CS = 0 when WR makes a low-to-high transition, the ADC0804 starts converting the analog input value of V_{in} to an 8-bit digital number.

d) CLK IN and CLK R

CLK IN is an input pin connected to an external clock source when an external clock is used for timing. However, the 804 has an internal clock generator. To use the internal clock generator (also called self-clocking) of the ADC0804, the CLK IN and CLK R pins are connected to a capacitor and a resistor.

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

Typical values are R = 10K ohms and C = 150 pF. Substituting in the above equation, we get/= 606 kHz. In that case, the conversion time is 110 us.

e) INTR (interrupt; a better name might be "end of conversion")

This is an output pin and is active low. It is a normally high pin and when the conversion is finished, it goes low to signal the CPU that the converted data is ready to be picked up. After INTR goes low, we make CS = 0 and send a high-to-low pulse to the RD pin to get the data out of the ADC0804 chip.

f) V_{in} (+) and V_{in} (-)

These are the differential analog inputs where $Vj_n = Vj_n (+) – Vj_n (-)$. Often the Vj_n (-) pin is connected to ground and the Vj_n (+) pin is used as the analog input to be converted to digital.

g) Vcc

This is the +5 volt power supply. It is also used as a reference voltage when the $V_{ref}/2$ input (pin 9) is open (not connected). This is discussed next.

h) $V_{ref}/2$

Pin 9 is an input voltage used for the reference voltage. If this pin is open (not connected), the analog input voltage for the ADC0804 is in the range of 0 to 5 volts (the same as the V_{cc} pin).

i) DO-D7

DO – D7 (D7 is the MSB) are the digital data output pins since ADC0804 is a parallel ADC chip. These are tri-state buffered and the converted data is accessed only when CS = 0 and RD is forced low. To calculate the output voltage, use the following formula.

j) Analog ground and digital ground

These are the input pins providing the ground for both the analog signal and the digital signal. Analog ground is connected to the ground of the analog V_{in} while digital ground is connected to the ground of the V_{cc} pin. The reason that we have two ground pins is to isolate the analog V_{in} signal from transient voltages caused by digital switching of the output DO – D7. Such isolation contributes to the accuracy of the digital data output. In our discussion, both are connected to the same ground; however, in the real world of data acquisition the analog and digital grounds are handled separately.

From this discussion we conclude that the following steps must be followed for data conversion by the ADC0804 chip.

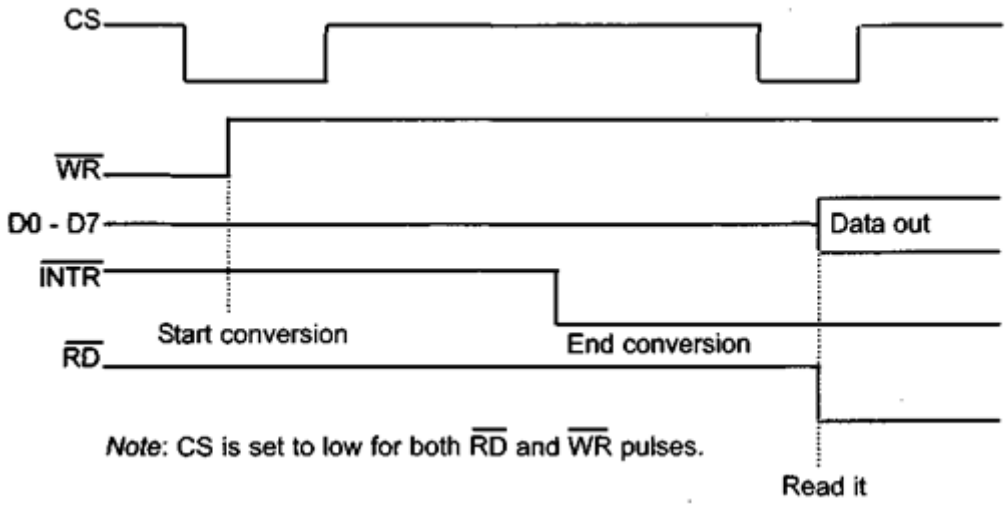


Figure 5.9: Read and Write Timing for ADC0804

5.4 Serial ADC Max1112 ADC interfacing to 8051

The MAX1112 is an 8-bit serial ADC chip with 8 channels of analog input. It has a single dout to bring out the digital data after it has been converted. It is compatible with a popular SPI and Microwire serial standard. The following are descriptions of the MAX1112 pins.

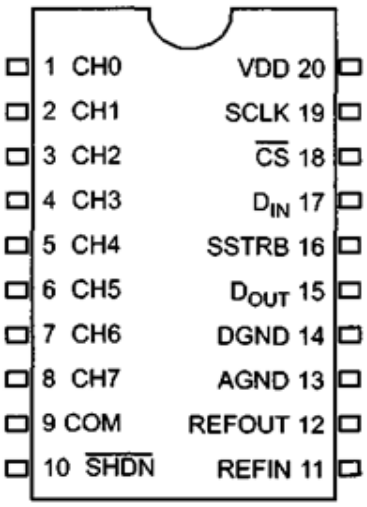


Figure 5.10: MAX 1112 Chip

a) CHO – CH7

CHO – CH7 are 8 channels of the analog inputs. In the single-ended mode, each of the channels can be used for an analog input where the COM pin is used as a ground reference for all the channels. In single-ended mode, 8 channels of input allow us to read 8 different analog inputs. We select the input channel by sending in the control byte via the DIN pin. In differential mode, we have 4 sets of 2-channel differentials. CHO and CHI go together, and CH2 -CH3, and so on.

b) COM

Ground reference for the analog input in single-ended mode.

c)CS

Chip select is an active low input used to select the MAX1112 chip.

d)SCLK

Serial clock input. SCLK is used to bring data out and send in the control byte, one bit at a time.

e) dout

Serial data out. The digital data is clocked out one bit at a time on the H-to-L edge (falling edge) of SCLK.

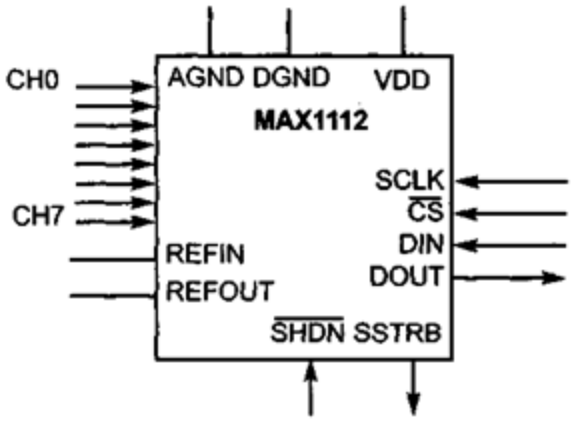


Figure 5.11: MAX1112 Serial ADC Block Diagram

f)din

Serial data in the control byte is clocked in one bit at a time on the L-to-H edge (rising edge) of SCLK.

g)SSTRB

Serial strobe output. In internal clock mode this indicates end-of-conversion. It goes high when the conversion is complete.

h) vdd

v_dd is the +5 volt power supply.

i) AGND, DGND (analog ground and digital ground)

Both are input pins providing ground for both the analog and the digital signals.

j)SHDN

Shutdown is an input and is normally not connected (or is connected to V_{DD}). If low, the ADC is shut down to save power. This is shut down by hardware. The control byte causes shutdown by software. *REFIN*

Reference voltage input. This voltage dictates the step size. *REFOUT*

Internal Reference Generator output. A 1µF bypass capacitor is placed between this pin and AGND.

k) MAX1112 control byte

The MAX1112 chip has 8 channels of analog inputs that are selected using a control byte. The control byte is fed into the MAX1112 serially one bit at a time via the D_{IN} pin with the help of SCLK. The control byte must be sent in with the MSB (most significant bit) going in first.

Start	SEL2	SLE1	SEL0	UN/BIP	SGL/DF	PD1	PD0

Start The MSB (D7) must be high to define the beginning of the control byte. It must be sent in first.

SEL2	SEL1	SEL0	**CHANNEL SELECTION** (SINGLE-ENDED MODE)
0	0	0	CHAN0
0	0	1	CHAN1
0	1	0	CHAN2
0	1	1	CHAN3
1	0	0	CHAN4
1	0	1	CHAN5
1	1	0	CHAN6
1	1	1	CHAN7

UNI/BIP 1 = unipolar: Digital data output is binary 00 - FFH.
0 = bipolar: Digital data output is in 2's complement.

SGL/DIF 1 = single-ended: 8 channels of single-ended with COM as reference
0 = differential: Two channels (eg., CH0 - CH1) are differential.

PD1 1 = fully operational
0 = power-down: Power down to save power using software.

PD0 1 = external clock mode: The conversion speed is dictated by SCLK.
0 = internal clock mode: The conversion speed is dictated internally, and the SSTRB pin goes high to indicate end-of-conversion (EOC).

Figure 5.12:. MAX1112 Control Byte

l) REFIN voltage and step size

The step size for the MAX1112 depends on the voltage connected to the REFIN pin. In unipolar mode, with V_{DD} = 5 V, we get 4.096 V for full-scale if the REFIN pin is connected to the AGND with a 1-fiF capacitor. That gives us a 16-mV step size since 4.096 V/256 = 16mV. To get a 10-mV step size, we need to connect the REFIN pin to a 2.56 V external voltage source, since 2.56 V/256 = 10 mV. According to the MAX1112 data sheet, the external reference voltage must be between 1 V and V_{DD}. Notice the lower limit for the reference voltage.

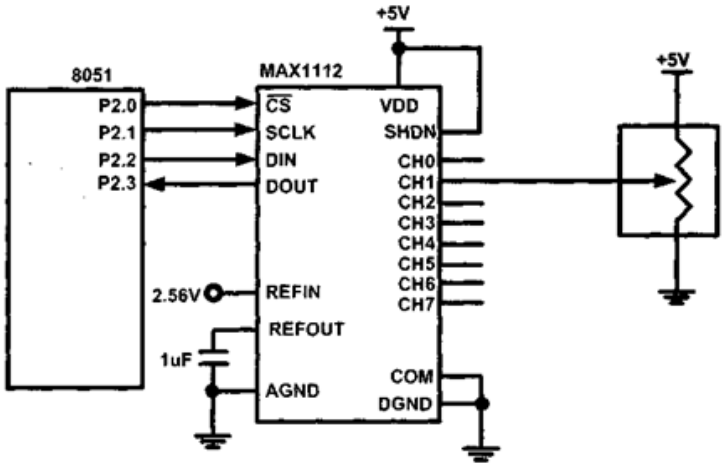


Figure 5.13: 8051 Connection to MAX1112 for 2nd Channel

Example 5-7

Find the MAX1112 control byte for (a) CHO, and (b) CH3. Assume single-ended, unipolar, internal clock, and fully operational modes.

Solution:

From Figure 13-14, we have the following:

(a) 10001110 (8E in hex) (b) 10111110 (BE in hex)

5.4.1 Start conversion and end of conversion for MAX1112

1. When the last bit of the control byte, PDO, is sent in, the conversion starts, and SSTRB goes low. The end-of-conversion state is indicated by SSTRB going high, which happens 55 us after PDO is clocked in.

2. We can either wait 55 µs, or monitor SSTRB before we get the digital data out of the ADC chip. Next we show how to get digital data out of the MAX1112.

Reading out digital data

The 8-bit converted digital data is brought out of the MAX1112.8-bit digital data is read out one bit at a time with the MSB (D7) coming out first. The SSTRB goes high to indicate that the conversion is finished. According to the MAX1112 data sheet, "after SSTRB goes high, the second falling edge of SCLK produces the MSB" of converted data at the $D0_{UT}$ pin. In other words, we need 9 pulses to get data out. To bring data out, CS must be low.

5.4.2 MAX1112 program in Assembly

The following is Assembly code for reading out digital data in the MAX1112:

```
        CS    BIT   P2.0
        SCLK  BIT   P2.1
        DIN   BIT   P2.2
        DOUT  BIT   P2.3

        SETB  DOUT        ;make it an input
        CLR   CS          ;CS=0
        SETB  SCLK
        ACALL DELAY       ;need delay for DS89C4x0
        CLR   SCLK        ;first H-to-L
        ACALL DLELAY      ;read data out on 2ND H-to-L
        CLR A
        MOV   R3,#8       ;
H2:     SETB  SCLK        ;
        ACALL DELAY       ;need delay for DS89C4x0
        CLR   SCLK        ;H-to-L pulse to get bit out
        ACALL DELAY ;
        MOV   C,DOUT      ;move bit to CY flag
        RLC   A           ;bring in the bit
        DJNZ  R3,H2       ;repeat for all 8 bits
        SETB  CS          ;CS=1
        MOV   P1,A        ;send converted data to P1
```

```
//C Code for reading data in MAX1112
#include <reg51.h>
sbit CS = P2^0;
sbit SCLK = P2^1;
sbit DIN = P2^2;
sbit DOUT = P2^3;
sbit LSBRA = ACC^0;
void main(void)
  {
    unsigned char x;
    CS=0;                    //select max1112
    SCLK=1;                  //an extra H-to-L pulse
    Delay();
    SCLK=0;
    Delay();
    for(x=0; x<8; x++)       //get all 8 bits
     {
       SCLK=1;
       Delay();
       SCLK=0;
       Delay()
       LSBRA=DOUT;           //bring in bit from DOUT
                             //pin to D0 of Reg A
       ACC = ACC << 1;       //keep shifting data
                             //for all 8 bits
     }
    CS=1;                    //deselect ADC
    P1=ACC;                  //display data on P1
  }
```

5.4.3 MAX1112 program in C

//The following program selects **the channel and** //reads ADC data

```
#include <reg51.h>
sbit CS    = P2^0;
sbit SCLK  = P2^1;
sbit DIN   = P2^2;
sbit DOUT  = P2^3;
sbit MSBRA = ACC^7;
sbit LSBRA = ACC^0;

void main(void)
  {
    unsigned char conbyte=0x9E;       //Chan 1
    unsigned char x;
    while(1)
     {
      ACC=conbyte;          //select the channel
      CS=0;
      for(x=0; x<8; x++)
       {
        SCLK=0;
        DIN=MSBRA;          //send D7 of Reg A to Din
        Delay();
        SCLK=1;             //latch in the bit
        Delay();
        ACC = ACC << 1;     //next bit
       }
      CS=1;                 //deselect MAX1112
      SCLK=0;               //Make SCLK low during conversion
      CS=0;                 //read the data
      SCLK=1;               //an extra H-to-L pulse
      Delay();
      SCLK=0;               //get all 8 bits
      Delay();
      for(x=0; x<8; x++)
       {
        SCLK=1;
        Delay();
        SCLK=0;
        Delay()
        LSBRA=DOUT;         //bring in bit from DOUT
                            //pin to D0 of Reg A


       ACC = ACC << 1;    //keep shifting data
                          //for all 8 bits
      }
     CS=1;                //deselect ADC
     P1=ACC;              //display data on P1
}
```

5.5 DAC interfacing, Sensor interfacing and signal conditioning

Digital-to-analog (DAC) converter

1. The digital-to-analog converter (DAC) is a device widely used to convert digital pulses to analog signals. In this section we discuss the basics of interfacing a DAC to the 8051.

2. Recall from your digital electronics book the two methods of creating a DAC: binary weighted and R/2R ladder. The vast majority of integrated circuit DACs, including the MC1408 (DAC0808) used in this section use the R/2R method since it can achieve a much higher degree of precision.
3. The first criterion for judging a DAC is its resolution, which is a function of the number of binary inputs.
4. 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″, where *n* is the number of data bit inputs.
5. Therefore, an 8-input DAC such as the DAC0808 provides 256 discrete voltage (or current) levels of output. Similarly, the 12-bit DAC provides 4096 discrete voltage levels. The total current provided by the I_{out} pin is a function of the binary numbers at the DO – D7 inputs of the DAC0808 and the reference current (I_{ref}), and is as follows:

$$I_{out} = I_{ref}\left(\frac{D7}{2} + \frac{D6}{4} + \frac{D5}{8} + \frac{D4}{16} + \frac{D3}{32} + \frac{D2}{64} + \frac{D1}{128} + \frac{D0}{256}\right)$$

Where DO is the LSB, D7 is the MSB for the inputs, and I_{ref} is the input current that must be applied to pin 14. The I_{ref} current is generally set to 2.0 mA.

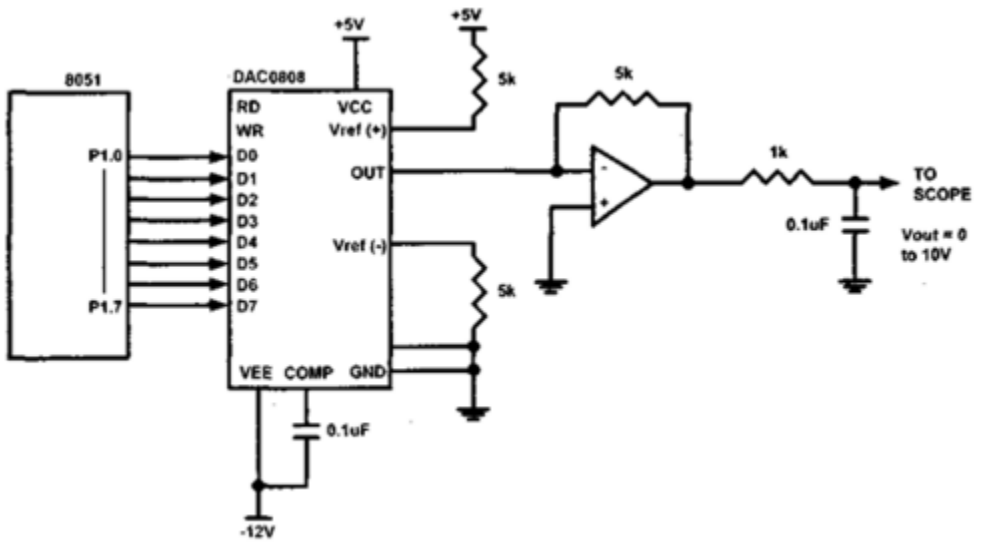


Figure 5.14: 8051 Connection to DAC808

Generating a sine wave

1. To generate a sine wave, we first need a table whose values represent the magnitude of the sine of angles between 0 and 360 degrees. The values for the sine function vary from -1.0 to +1.0 for 0- to 360-degree angles.
2. Therefore, the table values are integer numbers representing the voltage magnitude for the sine of theta. This method ensures that only integer numbers are output to the DAC by the 8051 microcontroller.
3. Table shows the angles, the sine values, the voltage magnitudes, and the integer values representing the voltage magnitude for each angle (with 30-degree increments). Here we assumed the full-scale voltage of 10 V for DAC output.
4. Full-scale output of the DAC is achieved when all the data inputs of the DAC are high. Therefore, to achieve the full-scale 10 V output, we use the following equation.

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

Table 5.4: Angle vs. Voltage Magnitude for Sine Wave

Angle 9 (degrees)	Sin 0	V_{out} (Voltage Magnitude)	Values Sent to DAC (decimal) 5 V + (5 V X sin 6) (Voltage Mag. X 25.6)
0	0	5	128
30	0.5	7.5	192
60	0.866	9.33	238
90	1.0	10	255
120	0.866	9.33	238
150	0.5	7.5	192
180	0	5	128
210	-0.5	2.5	64
240	-0.866	0.669	17
270	-1.0	0	0
300	-0.866	0.669	17
330	-0.5	2.5	64
360	0	5	128

Example 5-8

Verify the values given for the following angles: (a) 30° (b) 60°.

Solution:

(a) $V_{out} = 5\ V + (5\ V \times \sin\theta) = 5\ V + 5 \times \sin 30° = 5\ V + 5 \times 0.5 = 7.5\ V$
DAC input values = 7.5 V × 25.6 = 192 (decimal)

(b) $V_{out} = 5\ V + (5\ V \times \sin\theta) = 5\ V + 5 \times \sin 60° = 5\ V + 5 \times 0.866 = 9.33\ V$
DAC input values = 9.33 V × 25.6 = 238 (decimal)

Example 5-9

DAC interfacing in C

```
#include <reg51.h>
sfr DACDATA = P1;
void main()
  {
    unsigned char WAVEVALUE[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 = WAVEVALUE[x];
       }
      }
  }
```

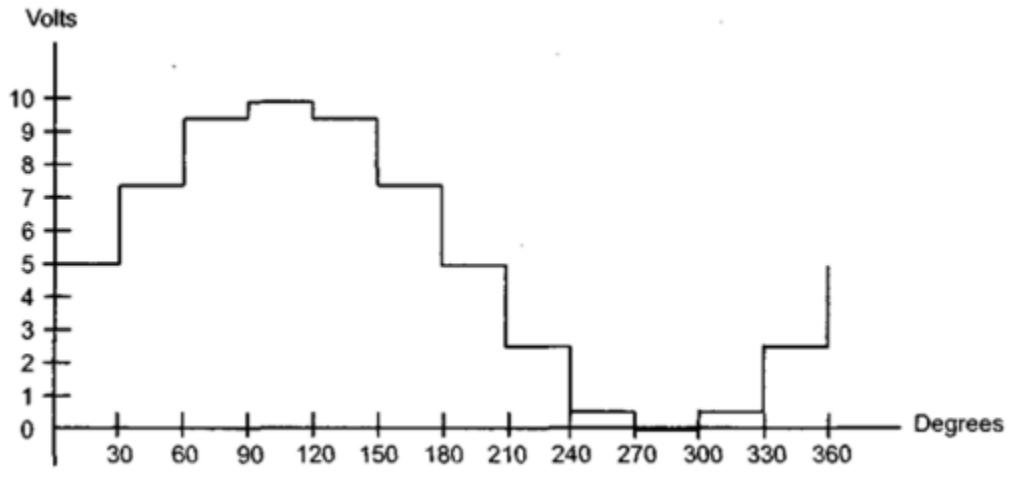


Figure 5.15: Angle vs. Voltage Magnitude for Sine Wave

5.7 Sensor interfacing and signal conditioning

a) Temperature sensors

1. *Transducers* convert physical data such as temperature, light intensity, flow, and speed to electrical signals. Depending on the transducer, the output produced is in the form of voltage, current, resistance, or capacitance. For example, temperature is converted to electrical signals using a transducer called a *thermistor*.
2. A thermistor responds to temperature change by changing resistance, but its response is not linear.
3. The complexity associated with writing software for such nonlinear devices has led many manufacturers to market a linear temperature sensor. Simple and widely used linear temperature sensors include the LM34 and LM35 series from National Semiconductor Corp.

b) LM35

The LM35 series sensors are precision integrated-circuit temperature sensors whose output voltage is linearly proportional to the Celsius (centigrade) temperature. The LM35 requires no external calibration since it is internally calibrated. It outputs 10 mV for each degree of centigrade temperature.

Table 5.5: LM35 Temperature Sensor Series Selection Guide

Part Temperature Range Accuracy	Output Scale
LM35A -55 C to +150 C +1.0 C	IOmV/C
LM35 -55 C to +150 C +1.5 C	10 mV/C
LM35CA -40 C to +110 C +1.0 C	IOmV/C
LM35C -40 C to +1 IOC +1.5 C	IOmV/C
LM35D 0 C to +100 C +2.0 C	IOmV/C

Note: **Temperature range is in degrees Celsius.**

c) **Signal conditioning and interfacing the LM35 to the 8051**

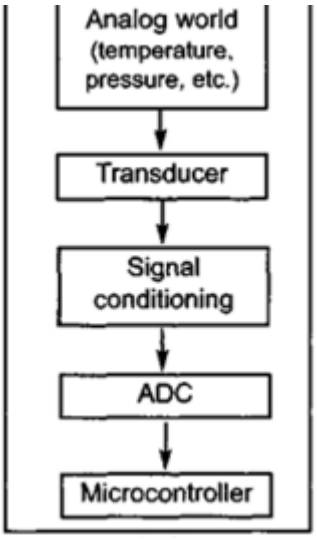


Figure 5.16: Signal conditioning

1. Signal conditioning is widely used in the world of data acquisition. The most common transducers produce an output in the form of voltage, current, charge, capacitance, and resistance.
2. However, we need to convert these signals to voltage in order to send input to an A- to-D converter. This conversion (modification) is commonly called *signal conditioning*. Signal conditioning can be a current-to-voltage conversion or a signal amplification.
3. For example, the thermistor changes resistance with temperature. The change of resistance must be translated into voltages in order to be of any use to an ADC. Look at the case of connecting an LM35 to an ADC0848. Since the ADC0848 has 8-bit resolution with a maximum of 256 (2^8) steps and the LM35 (or LM34) produces 10 mV for every degree of temperature change, we can condition V_{in} of the ADC0848 to produce a V_{out} of 2560 mV (2.56 V) for full-scale output.
4. Therefore, in order to produce the full-scale V_{out} of 2.56 V for the ADC0848, we need to set V_{ref} = 2.56. This makes V_{out} of the ADC0848 correspond directly to the temperature as monitored by the LM35

Table 5.6: Temperature vs. V_{out} for ADC0848

Temp. (C)	V_{in} (mV)	V_{out} (D7 - D0)
0	0	0000 0000
1	10	0000 0001
2	20	0000 0010
3	30	0000 0011
10	100	0000 1010
30	300	0001 1110

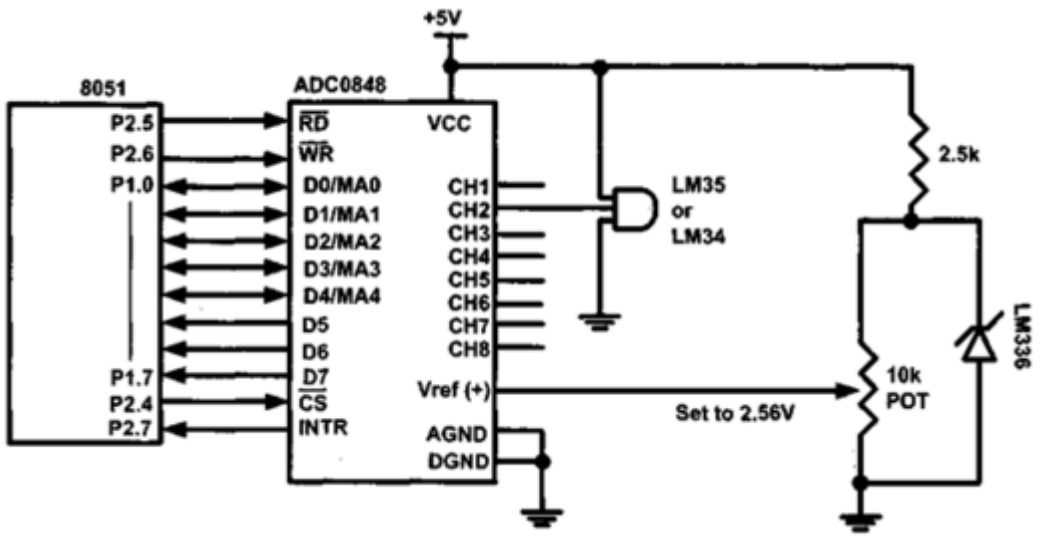


Figure 5.17: 8051 Connection to ADC0848 and Temperature Sensor

Reading and displaying temperature

5.6 Motor control: Relay, PWM, DC and stepper motor: Relays and opt isolators

Electromechanical relays

1. A *relay* is an electrically controllable switch widely used in industrial controls, automobiles, and appliances. It allows the isolation of two separate sections of a system with two different voltage sources. For example, a +5V system can be isolated from a 120V system by placing a relay between them. One such relay is called an electromechanical (or electromagnetic) relay (EMR)
2. The EMRs have three components: the coil, spring, and contacts. In Figure 17-1, a digital +5V on the left side can control a 12V motor on the right side without any physical contact between them.
3. When current flows through the coil, a magnetic field is created around the coil (the coil is energized), which causes the armature to be attracted to the coil. The

armature's contact acts like a switch and closes or opens the circuit. When the coil is not energized, a spring pulls the armature to its normal state of open or closed.

4. In the block diagram for electromechanically relays (EMR) we do not show the spring, but it does exist internally. There are all types of relays for all kinds of applications. In choosing a relay the following characteristics need to be considered:

The maximum DC/AC voltage and current that can be handled by the contacts. This is in the range of a few volts to hundreds of volts, while the current can be from a few amps to 40A or more, depending on the relay. Notice the difference between this voltage/current specification and the voltage/current needed for energizing the coil. The fact that one can use such a small amount of voltage/current on one side to handle a large amount of voltage/current on the other side is what makes relays so widely used in industrial controls.

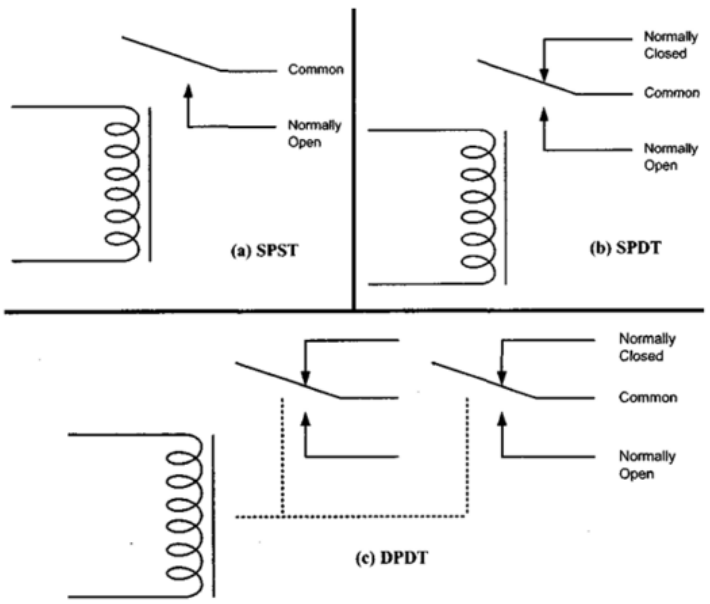


Figure 5.18: Relay Diagrams

5.6.1 Driving a relay

Digital systems and microcontroller pins lack sufficient current to drive the relay. While the relay's coil needs around 10 mA to be energized, the microcontroller's pin can provide a maximum of 1-2 mA current. For this reason, we place a driver, such as the ULN2803, or a power transistor between the microcontroller and the relay

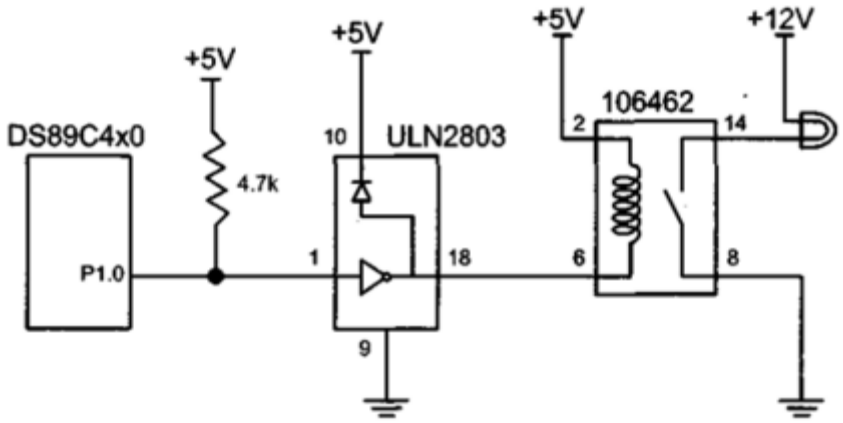


Figure 5.19: DS89C4xO Connection to Relay

5.6.2 Solid-state relay

1. Another widely used relay is the solid-state relay. In this relay, there is no coil, spring, or mechanical contact switch.
2. The entire relay is made out of semiconductor materials. Because no mechanical parts are involved in solid-state relays, their switching response time is much faster than that of electromechanical relays.
3. Another problem with the electromechanical relay is its life expectancy. The life cycle for the electromechanical relay can vary from a few hundred thousands to few million operations.
4. Wear and tear on the contact points can cause the relay to malfunction after a while. Solid-state relays have no such limitations. Extremely low input current and small packaging make solid-state relays ideal for microprocessor and logic control switching.
5. They are widely used in controlling pumps, solenoids, alarms, and other power applications. Some solid-state relays have a phase control option, which is ideal for motor-speed control and light-dimming applications.

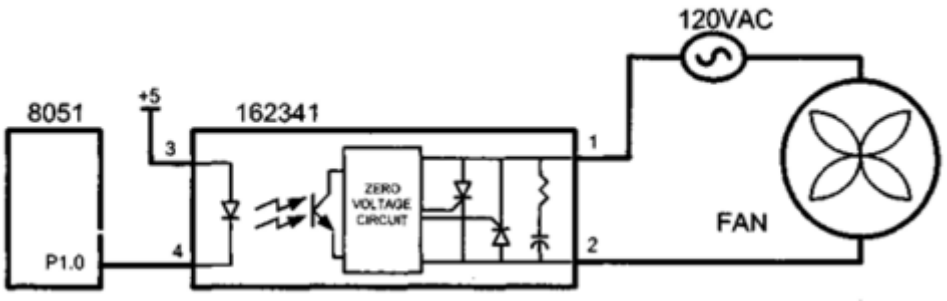


Figure 5.20:8051 Connection to a Solid-State Relay

5.6.3 Reed switch

Another popular switch is the reed switch. When the reed switch is placed in a magnetic field, the contact is closed. When the magnetic field is removed, the contact is forced open by its spring. The reed switch is ideal for moist and marine environments where it can be submerged in fuel or water. They are also widely used in dirty and dusty atmospheres since they are tightly sealed.

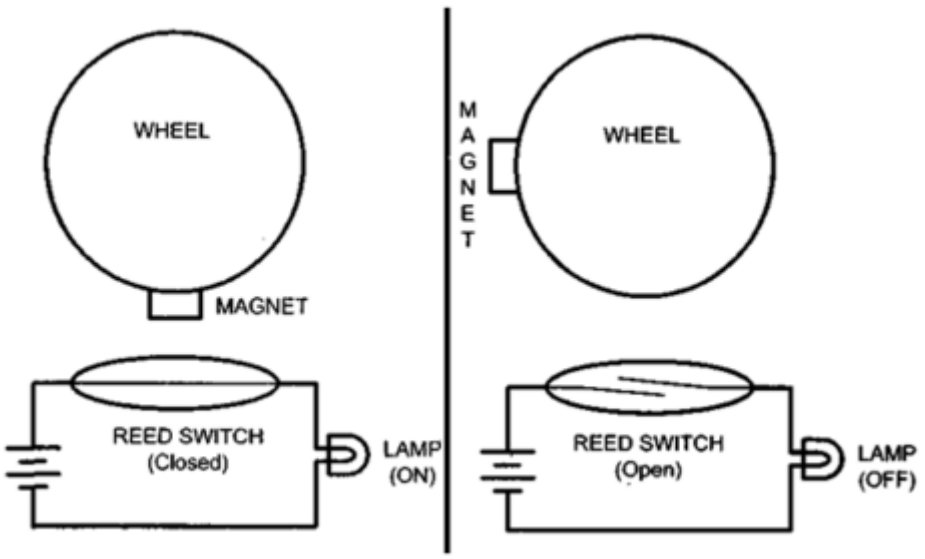


Figure 5.21. Reed Switch and Magnet Combination

Optoisolator

1. In some applications we use an optoisolator (also called optocoupler) to isolate two parts of a system. An example is driving a motor. Motors can produce what is called back EMF, a high voltage spike produced by a sudden change of current as indicated in the V = Ldi/dt formula.
2. In situations such as printed circuit board design, we can reduce the effect of this unwanted voltage spike (called ground bounce) by using decoupling capacitors (see Appendix C).
3. In systems that have inductors (coil winding), such as motors, decoupling capacitor or a diode will not do the job. In such cases we use optoisolators. An optoisolator has an LED (light-emitting diode) transmitter and a photosensor receiver, separated from each other by a gap.
4. When current flows through the diode, it transmits a signal light across the gap and the receiver produces the same signal with the same phase but a different current and amplitude.
5. Optoisolators are also widely used in communication equipment such as modems. This allows a computer to be connected to a telephone line without risk of damage from power surges.
6. The gap between the transmitter and receiver of optoisolators prevents the electrical current surge from reaching the system.

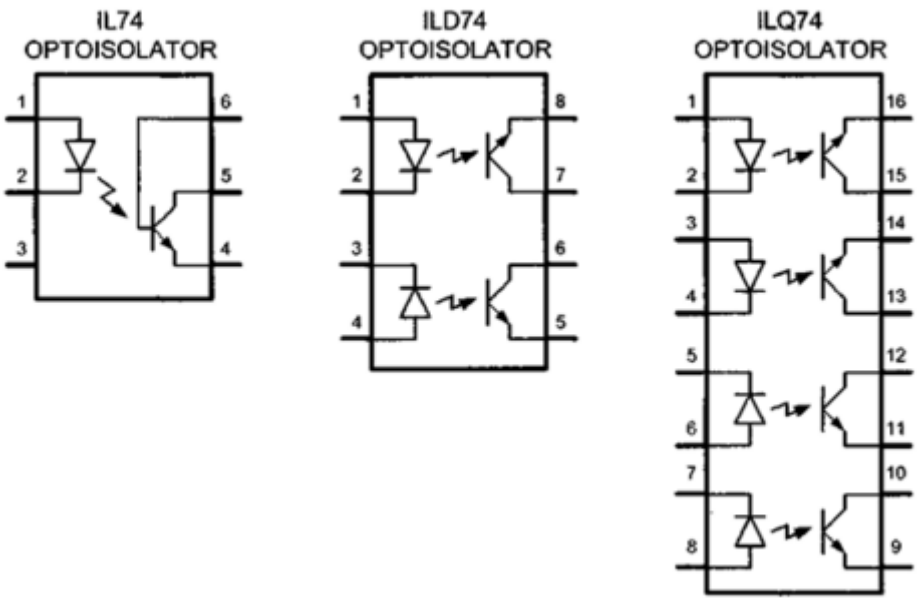


Figure 5.22: Opt isolator Package Examples

5.7 Stepper Motor Interfacing

Stepper motors

1. A *stepper motor* is a widely used device that translates electrical pulses into mechanical movement. In applications such as disk drives, dot matrix printers, and robotics, the stepper motor is used for position control. Stepper motors commonly have a permanent magnet *rotor* (also called the *shaft)* surrounded by a *stator*
2. There are also steppers called variable reluctance *stepper motors* that do not have a PM rotor.
3. The most common stepper motors have four stator windings that are paired with a center-tapped common
4. This type of stepper motor is commonly referred to as *a. four-phase* or unipolar stepper motor.

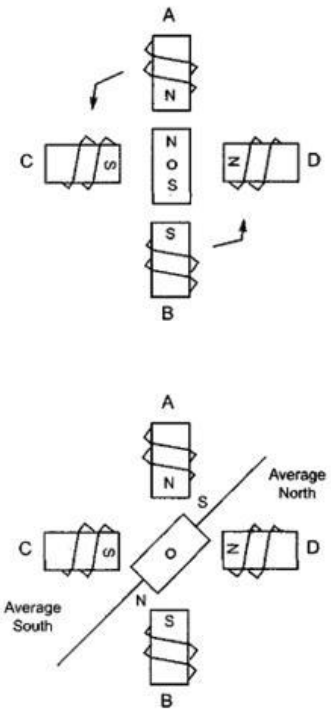


Figure 5.23: Stepper Motor alignment

Table 5-7 Normal 4-Step Sequence

Clockwise	Step #	Winding A	Winding B	Winding C	Winding D	Counter-clockwise
↓	1	1	0	0	1	↑
	2	1	1	0	0	
	3	0	1	1	0	
	4	0	0	1	1	

5.7.1 Step angle

Table 5-8 Stepper Motor Step angles

Step Angle	Steps per Revolution
0.72	500
1.8	200
2.0	180
2.5	144
5.0	72
7.5	48
15	24

1. How much movement is associated with a single step? This depends on the internal construction of the motor, in particular the number of teeth on the stator and the rotor. The *step angle* is the minimum degree of rotation associated with a single step. Various motors have different step angles.
2. The term *steps per revolution.* This is the total number of steps needed to rotate one complete rotation or 360 degrees (e.g., 180 steps x 2 degrees = 360).
3. It must be noted that perhaps contrary to one's initial impression, a stepper motor does not need more terminal leads for the stator to achieve smaller steps. All the stepper motors discussed in this section have 4 leads for the stator winding and 2 COM wires for the center tap. Although some manufacturers set aside only one lead for the common signal instead of two, they always have 4 leads for the stators. Next we discuss some associated terminology in order to understand the stepper motor further.

a) Steps per second and rpm relation

The relation between rpm (revolutions per minute), steps per revolution, and steps per second is as follows.

b) The four-step sequence and number of teeth on rotor

The switching sequence is called the 4-step switching sequence since after four steps the same two windings will be "ON" How much movement is associated with these four steps? After completing every four steps, the rotor moves only one tooth pitch. Therefore, in a stepper motor with 200 steps per revolution, the rotor has 50 teeth since 4×50 = 200 steps are needed to complete one revolution. This leads to the conclusion that the minimum step angle is always a function of the number of teeth on the rotor. In other words, the smaller the step angle, the more teeth the rotor passes.

Table 5-9: Half-Step 8-Step Sequence

Clockwise ↓ / Counter-clockwise ↑

Step #	Winding A	Winding B	Winding C	Winding 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

Motor speed

The motor speed, measured in steps per second (steps/s), is a function of the switching rate. By changing the length of the time delay loop, we can achieve various rotation speeds.

Holding torque

The following is a definition of holding torque: "With the motor shaft at standstill or zero rpm condition, the amount of torque, from an external source, required to break away the shaft from its holding position. This is measured with rated voltage and current applied to the motor." The unit of torque is ounce-inch (or kg-cm).

Wave drive 4-step sequence

In addition to the 8-step and the 4-step sequences discussed earlier, there is another sequence called the wave drive 4-step sequence.

5.7.2 Unipolar versus bipolar stepper motor interface

1. There are three common types of stepper motor interfacing: universal, unipolar, and bipolar.
2. They can be identified by the number of connections to the motor. A universal stepper motor has eight, while the unipolar has six and the bipolar has four.
3. The universal stepper motor can be configured for all three modes, while the unipolar can be either unipolar or bipolar. Obviously the bipolar cannot be configured for universal nor unipolar mode.\
4. **Unipolar stepper motors** can be controlled using the basic interfacing ,whereas the bipolar stepper requires H-Bridge circuitry. Bipolar stepper motors require a higher operational current than the unipolar; the advantage of this is a higher holding torque.

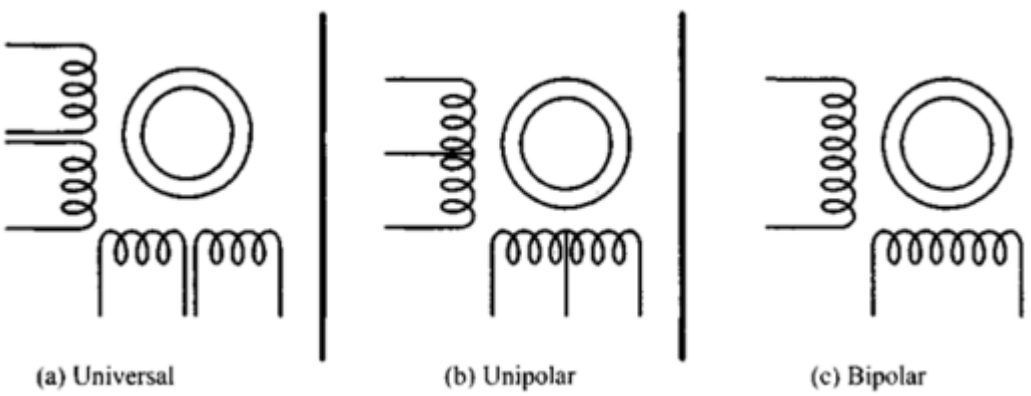


Figure 5.24: Common Stepper Motor Types

Example 5-10

A switch is connected to pin P2.7. Write a program to monitor the status of SW and perform the following:

1. If SW = 0, the stepper motor moves clockwise.
2. If SW = 1, the stepper motor moves counterclockwise.

Solution:

```
      ORG    0H              ;starting address
MAIN: SETB   P2.7            ;make an input
      MOV    A, #66H         ;starting phase value
      MOV    P1, A           ;send value to port
TURN:
      JNB    P2.7, CW        ;check switch result
      RR     A               ;rotate right
      ACALL  DELAY           ;call delay
      MOV    P1,A            ;write value to port
      SJMP   TURN            ;repeat
CW:   RL     A               ;rotate left
      ACALL  DELAY           ;call delay
      MOV    P1,A            ;write value to port
      SJMP   TURN            ;repeat

DELAY:
      MOV    R2, #100
H1:   MOV    R3, #255
H2:   DJNZ   R3, H2
      DJNZ   R2,H1
      RET
      END
```

Example 5-11

A switch is connected to pin P2.7. Write a C program to monitor the status of SW and perform the following:

1. If SW = 0, the stepper motor moves clockwise.
2. If SW = 1, the stepper motor moves counterclockwise.

```
#include <reg.h>
sbit SW=P2^7;

void main()
  {
    SW = 1;
    while(1)
      {
        if(SW == 0)
          {
            P1 = 0x66;
            MSDelay(100);
            P1 = 0xCC;
            MSDelay(100);
            P1 = 0x99;
            MSDelay(100);
            P1 = 0x33;
            MSDelay(100);
          }
        else
          {
            P1 = 0x66;
            MSDelay(100);


            P1 = 0x33;
            MSDelay(100);
            P1 = 0x99;
            MSDelay(100);
            P1 = 0xCC;
            MSDelay(100);
          }
      }
  }

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

5.8 DC Motor Interfacing

1. A direct current (DC) motor is another widely used device that translates electrical pulses into mechanical movement. In the DC motor we have only + and – leads. Connecting them to a DC voltage source moves the motor in one direction.
2. By reversing the polarity, the DC motor will move in the opposite direction. One can easily experiment with the DC motor.
3. For example, small fans used in many motherboards to cool the CPU are run by DC motors. By connecting their leads to the + and – voltage source, the DC motor moves.

4. While a stepper motor moves in steps of 1 to 15 degrees, the DC motor moves continuously. In a stepper motor, if we know the starting position we can easily count the number of steps the motor has moved and calculate the final position of the motor. This is not possible in a DC motor.
5. The maximum speed of a DC motor is indicated in rpm and is given in the data sheet. The DC motor has two rpm: no-load and loaded. The manufacturer's data sheet gives the no-load rpm. The no-load rpm can be from a few thousand to tens of thousands. The rpm is reduced when moving a load and it decreases as the load is increased. For example, a drill turning a screw has a much lower rpm speed than when it is in the no-load situation. DC motors also have voltage and current ratings.
6. The nominal voltage is the voltage for that motor under normal conditions, and can be from 1 to 150V, depending on the motor. As we increase the voltage, the rpm goes up. The current rating is the current consumption when the nominal voltage is applied with no load, and can be from 25mA to a few amps.
7. As the load increases, the rpm is decreased, unless the current or voltage provided to the motor is increased, which in turn increases the torque.
8. With a fixed voltage, as the load increases, the current (power) consumption of a DC motor is increased. If we overload the motor it will stall, and that can damage the motor due to the heat generated by high current consumption.

5.8.1 Unidirection Control

The DC motor rotation for clockwise (CW) and counterclockwise (CCW) rotations.

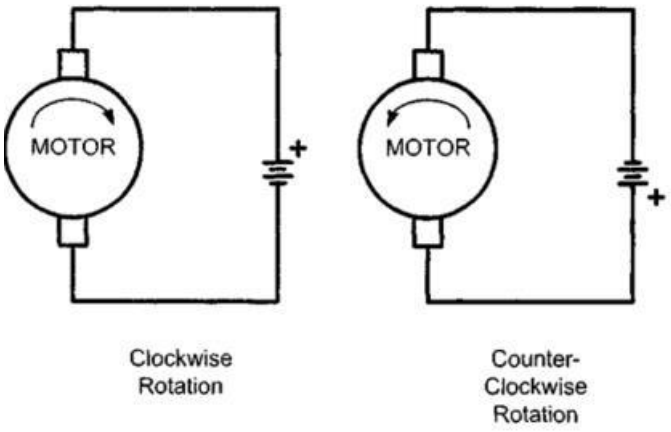


Figure 5.25:. DC Motor Rotation (Permanent Magnet Field)

5.8.2 Bidirectional control

With the help of relays or some specially designed chips we can change the direction of the DC motor rotation.

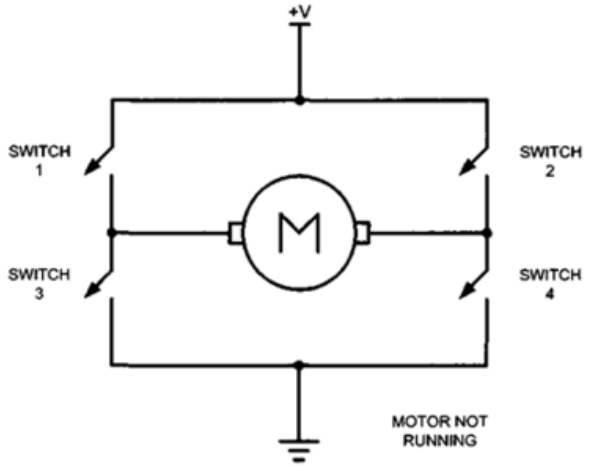


Figure 5.26:H-Bridge Motor Configuration

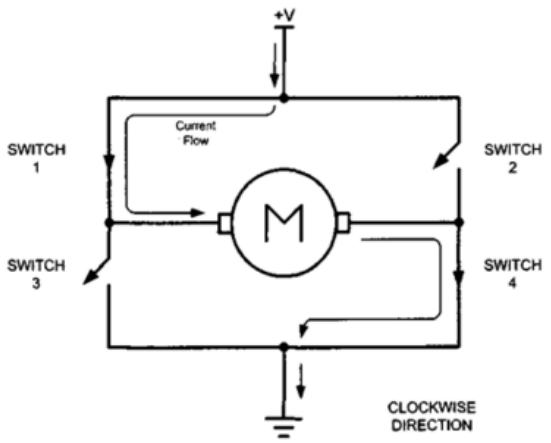


Figure 5.27: H-Bridge Motor Clockwise Configuration

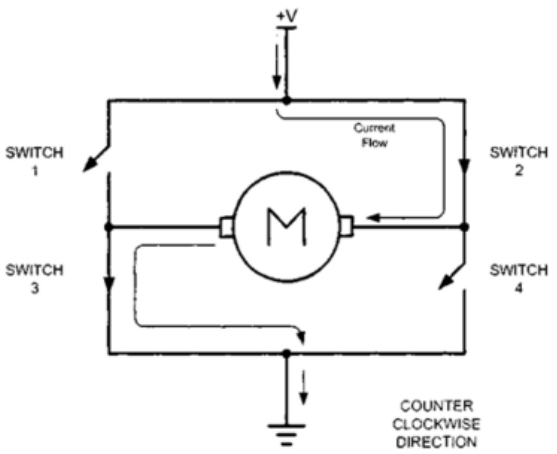


Figure 5-28: H-Bridge Motor Counter clockwise Configuration

5.8.3 Pulse width modulation (PWM)

1. The speed of the motor depends on three factors: (a) load, (b) voltage, and (c) current. For a given fixed load we can maintain a steady speed by using a method called *pulse width modulation* (PWM).
2. By changing (modulating) the width of the pulse applied to the DC motor we can increase or decrease the amount of power provided to the motor, thereby increasing or decreasing the motor speed. Notice that, although the voltage has a fixed amplitude, it has a variable duty cycle.
3. That means the wider the pulse, the higher the speed. PWM is so widely used in DC motor control that some microcontrollers come with the PWM circuitry embedded in the chip. In such microcontrollers all we have to do is load the proper registers with the values of the high and low portions of the desired pulse, and the rest is taken care by the microcontroller.
4. This allows the microcontroller to do other things. For microcontrollers without PWM circuitry, we must create the various duty cycle pulses using software, which prevents the microcontroller from doing other things.
5. The ability to control the speed of the DC motor using PWM is one reason that DC motors are preferable over AC motors.
6. AC motor speed is dictated by the AC frequency of the voltage applied to the motor and the frequency is generally fixed. As a result, we cannot control the speed of the AC motor when the load is increased. As was shown earlier, \\e can also change the DC motor's direction and torque.

Example 5-12

Refer to the figure in this example. Write a program to monitor the status of the switch and perform the following:

1. If P2.7 = 1, the DC motor moves with 25% duty cycle pulse.
2. If P2.7 = 0, the DC motor moves with 50% duty cycle pulse.

```
      ORG 0H
MAIN:
      CLR P1.0                    ;turn off motor
      SETB P2.7
MONITOR:
      JNB P2.7, FIFTYPERCENT
      SETB P1.0                   ;high portion of pulse
      MOV R5, #25
      ACALL DELAY
      CLR P1.0                    ;low portion of pulse
      MOV R5, #75
      ACALL DELAY
      SJMP MONITOR
FIFTYPERCENT:
      SETB P1.0                   ;high portion of pulse
      MOV R5, #50
      ACALL DELAY
      CLR P1.0                    ;low portion of pulse
      MOV R5, #50
      ACALL DELAY
      SJMP MONITOR
DELAY:
H1:   MOV R2, #100
H2:   MOV R3, #255
H3:   DJNZ R3, H3
      DJNZ R2, H2
      DJNZ R5, H1
      RET
      END
```

Example 5-13

Two switches are connected to pins P2.0 and P2.1. Write a C program to monitor the status of both switches and perform the following: SW2(P2.7) SW1(P2.6)

1. DC motor moves slowly (25% duty cycle).
2. DC motor moves moderately (50% duty cycle).
3. DC motor moves fast (75% duty cycle).
4. DC motor moves very fast (100% duty cycle).

```
#include <reg51.h>
sbit MTR = P1^0;
void MSDelay(unsigned int value);

void main()
  {
  unsigned char z;
  P2 = 0xFF;
  z = P2;
  z = z & 0x03;
  MTR = 0;
  while(1)
    {
     switch(z)
       {
       case(0):
         {
           MTR = 1;
           MSDelay(25);
           MTR = 0;
           MSDelay(75);
           break;
         }
       case(1):
         {
           MTR = 1;
           MSDelay(50);
           MTR = 0;
           MSDelay(50);
           break;
         }
       case(2):
         {
           MTR = 1;
           MSDelay(75);
           MTR = 0;
           MSDelay(25);
           break;
         }
       default:
         MTR = 1;
    }
  }
```

5.9 8051 interfacing with 8255: Programming the 8255, 8255 interfacing, C programming for 8255

In this section we study the 8255 chip, one of the most widely used I/O chips. We first describe its features and then show the connection between the 8031/51 and 8255 chips.

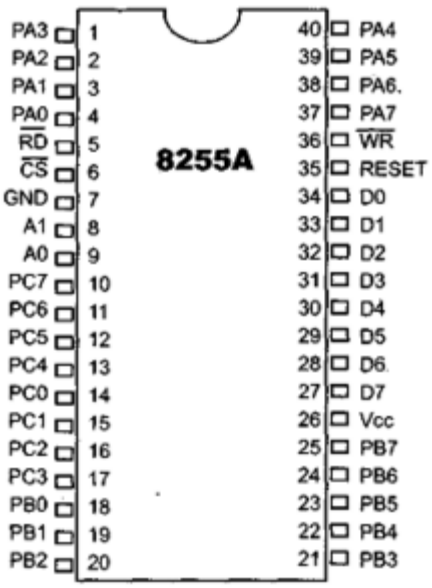


Figure 5-29: 8255 Pin

8255 features

It has three separately accessible ports. The ports are each 8-bit, and are named A, B, and C. The individual ports of the 8255 can be programmed to be input or output, and can be changed dynamically. In addition, 8255 ports, have handshaking capability, thereby allowing interface with devices that also have handshaking signals, such as printers. The handshaking capability of the 8255 is not discussed since it is no longer used.

a) PAO – PA7

The 8-bit port A can be programmed as all input, or as all output, or all bits as bidirectional input/output.

b) PBO – PB7

The 8-bit port B can be programmed as all input or as all output. Port B cannot be used as a bidirectional port.

c) PCO-PC7

This 8-bit port C can be all input or all output. It can also be split into two parts, CU (upper bits PC4 – PC7) and CL (lower bits PCO – PC3). Each can be used for input or output.

d) RD and WR

These two active-low control signals are inputs to the 8255. The RD and WR signals from the 8031/51 are connected to these inputs.

e) DO – D7 data pin

The data pins of the 8255 are connected to the data pins of the microcontroller allowing it to send data back and forth between the controller and the 8255 chip.

RESET

This is an active-high signal input into the 8255 used to clear the control register. When RESET is activated, all ports are initialized as input ports. In many designs this pin is connected to the RESET output of the system bus or grounded to make it inactive. Like all 1C input pins, it should not be left unconnected.

5.9.1 Mode selection of the 8255

While ports A, B, and C are used to input or output data, it is the control register that must be programmed to select the operation mode of the three ports. The ports of the 8255 can be programmed in any of the following modes.

1. Mode 0, simple I/O mode. In this mode, any of the ports A, B, CL, and CU can be programmed as input or output. In this mode, all bits are out or all are in. In other words, there is no such thing as single-bit control as we have seen in PO – P3 of the 8051. Since the vast majority of applications involving the 8255 use this simple I/O mode, we will concentrate on this mode in this chapter.
2. Mode 1. In this mode, ports A and B can be used as input or output ports with handshaking capabilities. Handshaking signals are provided by the bits of port C.
3. Mode 2. In this mode, port A can be used as a bidirectional I/O port with hand shaking capabilities whose signals are provided by port C. Port B can be used either in simple I/O mode or handshaking mode 1.
4. BSR (bit set/reset) mode. In this mode, only the individual bits of port C can be programmed.

Example 5-14

Find the control word of the 8255 for the following configurations:

1. All the ports of A, B, and C are output ports (mode 0).
2. PA = in, PB = out, PCL = out, and PCH = out.

Solution:

(a) 1000 0000 = 80H (b) 1001 0000 = 90H

The 8255 chip is programmed in any of the 4 modes mentioned earlier by sending a byte (Intel calls it a control word) to the control register of the 8255. We must first find the port

addresses assigned to each of ports A, B, C, and the control register. This is called *mapping* the I/O port.

the 8255 is connected to an 8031/51 as if it is RAM memory. Notice the use of RD and WR signals. This method of connecting an I/O chip to a CPU is called *memory-mapped I/O,* since it is mapped into memory space. In other words, we use memory space to access I/O devices. For this reason we use instructions such as MOVX to access the 8255. In Chapter 14 we used MOVX to access RAM and ROM. For an 8255 connected to the 8031/51 we must also use the MOVX instruction to communicate with it.

Outcomes

At the end of the module the students will be able to:

CO5: Analyse **Interfacing of** 8051 Microcontroller for different I/O devices and 8255 Microcontroller. [**L4, MODULE 5]**