(73)

32116.

PUNE INSTITUTE OF COMPUTER TECHNOLOGY, PUNE - 411043

Department of Electronics & Telecommunication Engineering

CLASS: T.E. E &TC Engg. COURSE: MC

EXPT. NO.: 4 DATE: 15/09/23

TITLE: Interfacing Stepper motor with 89C51

PROBLEM STATEMENT:

A.) Write a program to interface stepper motor to 8051 and rotate it through clockwise /anti clockwise direction continuously.

B.) Draw the complete interfacing diagram & calculate the count to be loaded for rotating the motor through 90 degrees. The given step angle is 18 degrees.

OBJECTIVE:

a. To understand the working principle of stepper motor
b. To understand half step and full step concept
c. To implement the stepper motor interfacing using development board and rotate the motor clockwise and anticlockwise.

S/W PACKAGES and H/W USED:

Keil IDE, SST Programmer, Universal Microcontroller Kit, Stepper motor (1A, __° step angle)

THEORY

1. **Stepper motor:**

A stepper motor is an electromechanical device which converts electrical pulses into discrete mechanical movements. The shaft or spindle of a stepper motor rotates in discrete step increments when electrical command pulses are applied to it in the proper sequence. The motors rotation has several direct relationships to these applied input pulses. The sequence of the applied pulses is directly related to the direction of motor shafts rotation. The speed of the motor shafts rotation is directly related to the frequency of the input pulses and the length of rotation is directly related to the number of input pulses applied.

1.1 Stepper Motor Types

There are three basic stepper motor types.

Permanent Magnet Stepper Motor:

Permanent magnet motors use a permanent magnet (PM) in the rotor and operate on the attraction or repulsion between the rotor PM and the stator electromagnets.

Variable Reluctance Stepper Motor:

Variable reluctance (VR) motors have a plain iron rotor and operate based on the principle that minimum reluctance occurs with minimum gap, hence the rotor points are attracted toward the stator magnet poles.

Hybrid Synchronous Stepper Motor:

Hybrid stepper motors are named because they use a combination of permanent magnet (PM) and variable reluctance (VR) techniques to achieve maximum power in a small package size.

1.2 Operation of Stepper Motor:

Stepper motors consist of a permanent magnetic rotating shaft, called the rotor, and electromagnets on the stationary portion that surrounds the motor, called the stator. Fig7.1 illustrates one complete rotation of a stepper motor. At position 1, we can see that the rotor is beginning at the upper electromagnet, which is currently active (has voltage applied to it). To move the rotor clockwise (CW), the upper electromagnet is deactivated, and the right electromagnet is activated, causing the rotor to move 90 degrees CW, aligning itself with the active magnet. This process is repeated in the same manner at the south and west electromagnets until we once again reach the starting position.

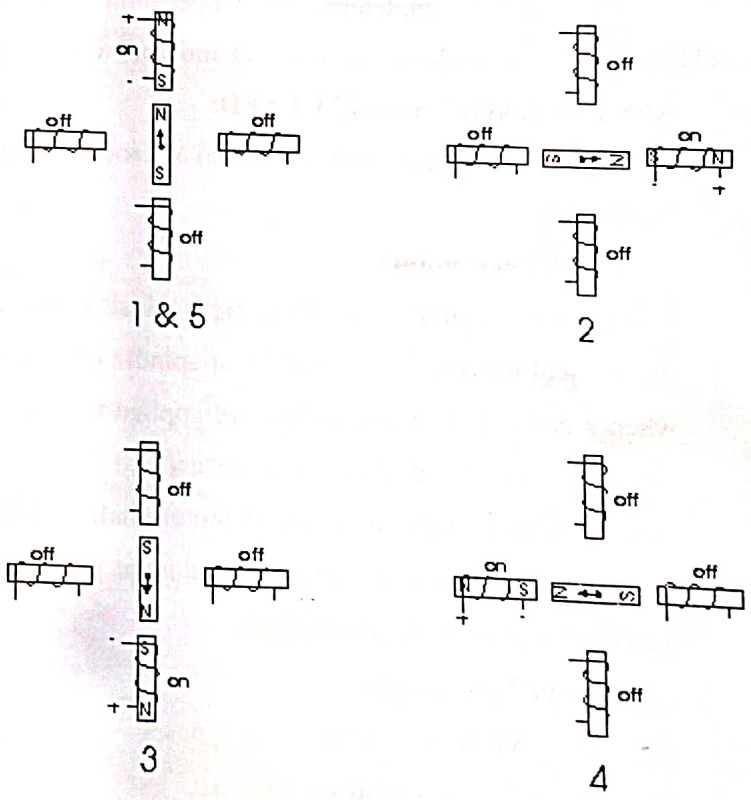


Fig 1 Excitation of rotor with one stator coil energized

In the above example, we used a motor with a resolution of 90 degrees or demonstration purposes. In reality, this would not be a very practical motor tor most applications. The average stepper motor's resolution -- the number of degrees rotated per pulse -- is much higher than this. For

example, a motor with a resolution of 5 degrees would move its rotor 5 degrees per step, thereby requiring 72 pulses (steps) to complete a full 360-degree rotation.

You may double the resolution of some motors by a process known as "half-stepping". Instead of switching the next electromagnet in the rotation on one at a time, with half stepping you turn on both electromagnets, causing an equal attraction between, thereby doubling the resolution. As you can see in Fig7.2, in the first position only the upper electromagnet is active, and the rotor is drawn completely to it. In position 2, both the top and right electromagnets are active, causing the rotor to position itself between the two active poles. Finally, in position 3, the top magnet is deactivated, and the rotor is drawn all the way right. This process can then be repeated for the entire rotation.

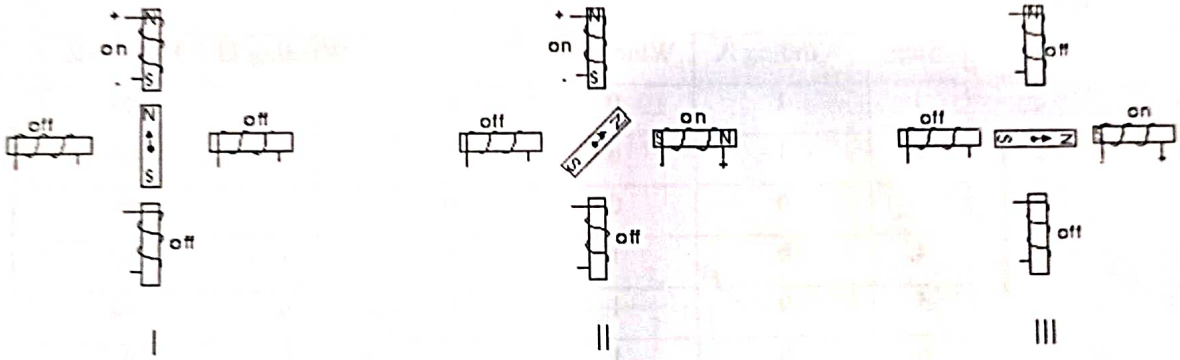


Fig 2 Excitation of rotor with two stator coils energized.

1.3 Step angle, Half step and Full Step Excitation:

Step angle is the minimum degree of rotation associated with a single excitation. The total number of steps needed to complete one revolution (360°) is known as step per revolution.

Ex: If the step angle is 1.8°; then 200 excitation pulses will be required to complete one revolution.

The excitation sequences are of two types:

- 4 step switching sequence/ Full step excitation.
- 8 Step sequence/ Half step sequence

The 4-step sequence repeats the excitation of windings after every 4th pulse while the 8 step sequence repeats after 8 steps.

The following table shows the 4 step excitation sequences for clockwise and anti-clockwise rotation.

Step	Winding A	Winding B	Winding C	Winding D	Hex Code
1	1	0	0	1	09
2	0	1	0	1	05
3	0	1	1	0	06
4	1	0	1	0	0A

Clockwise ↓ ; ↑ Counter Clock.

The following table shows the 8 step excitation sequences for clockwise and anti-clockwise rotation.

Step	Winding A	Winding B	Winding C	Winding D	Hex Code
1	1	0	0	0	08
2	1	0	0	1	09
3	0	0	0	1	01
4	0	1	0	1	05
5	0	1	0	0	04
6	0	1	1	0	06
7	0	0	1	0	02
8	1	0	1	0	0A

Clockwise ↓ ; ↑ Counter Clock.

1.4 Calculation:

Formula to calculate count value:

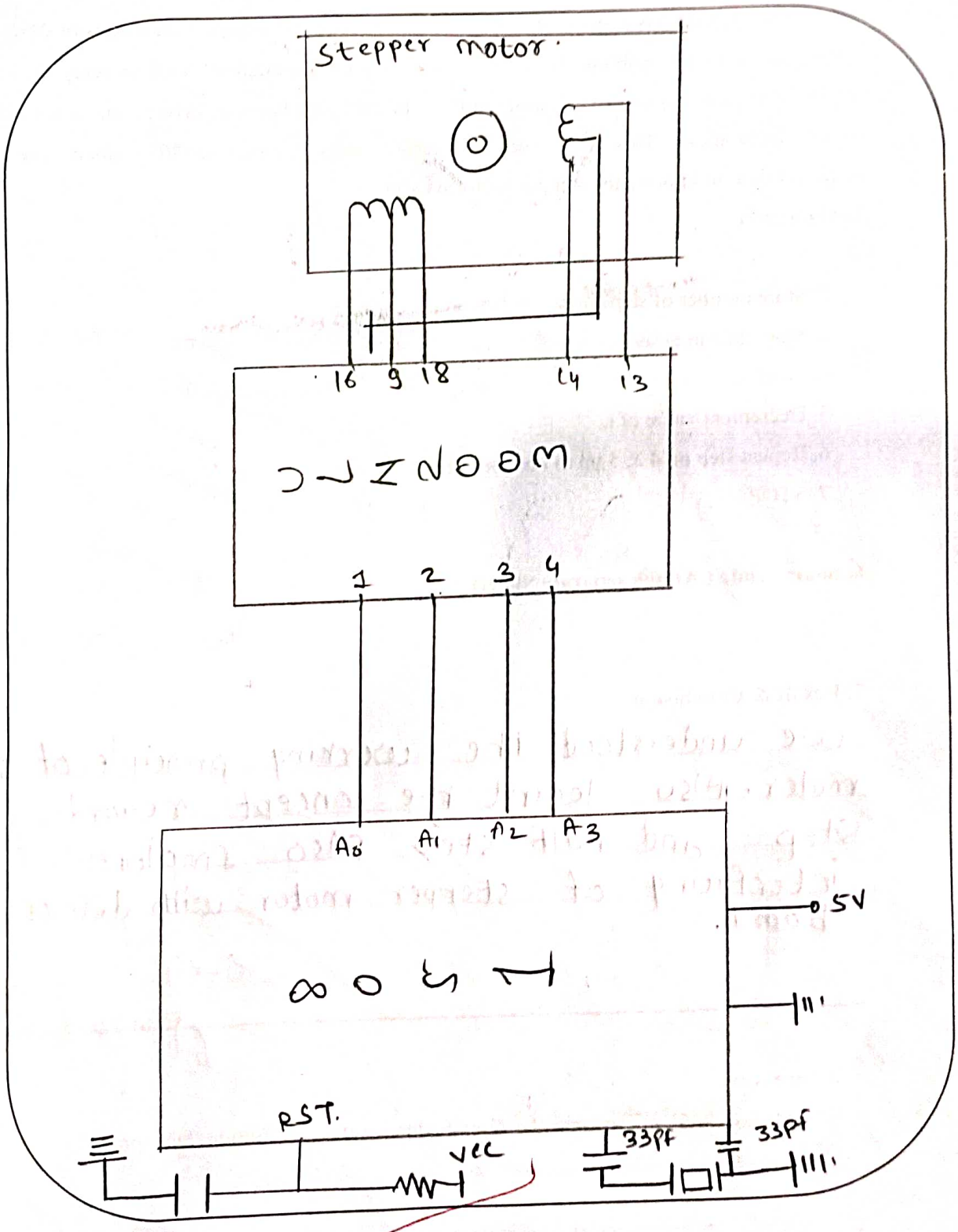
Count value= Degree expected / Step angle x No. of Steps required

count value $= \frac{90}{1.8 \times 4} = 12.5^0$

step angle. (expected degree) $= \frac{360}{4} = 90^\circ$.

∴ Stepper motor will take $12.5 \times 4 = 50$ steps

2. Interfacing Diagram



4. The Driver Circuit:

The ULN2003 is a monolithic IC consists of seven NPN darlington transistor pairs with high voltage and current capability. It is commonly used for applications such as relay drivers, motor, display drivers, led lamp drivers, logic buffers, line drivers, hammer drivers and other high voltage current applications. This IC is used to interface stepper motor to 8051 since the ports lack sufficient current to drive the stepper motor winding.

5. Algorithm:

1. START
2. Store number of steps in i.
3. Store data in array.
4. Send data to port P1
5. Decrement value of i.
6. Repeat step no 4 & 5 up to i become zero.
7. STOP

6. Source code (Attach separate Sheet)

7. Result & Conclusion

we understood the working principle of stepper motor. Also learnt the concept regarding full step and half step. Also Implemented the interfacing of stepper motor with development board.

sen

8. References:

a. Mazidi, 8051 microcontroller & embedded system 3rd Edition, Pearson

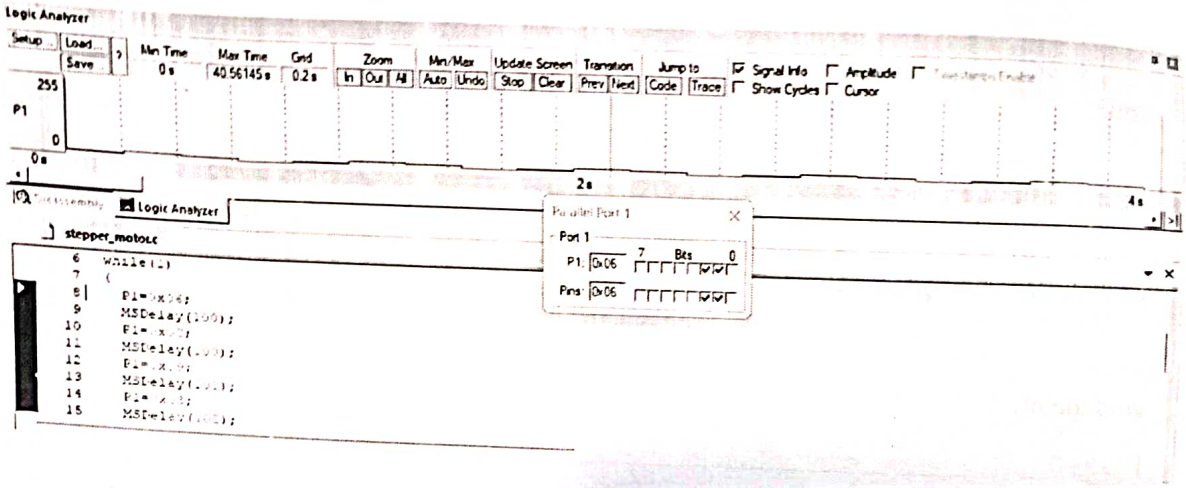
b. Datasheet of 8051 microcontroller

1) Rotating Stepper motor clockwise continuously

Code:

```
#include<reg51.h>
void MSDelay(unsigned int);

void main()
{
        while(1)
        {
                P1=0x06;
                MSDelay(100);
                P1=0x0C;
                MSDelay(100);
                P1=0x09;
                MSDelay(100);
                P1=0x03;
                MSDelay(100);
        }
}

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

Output:



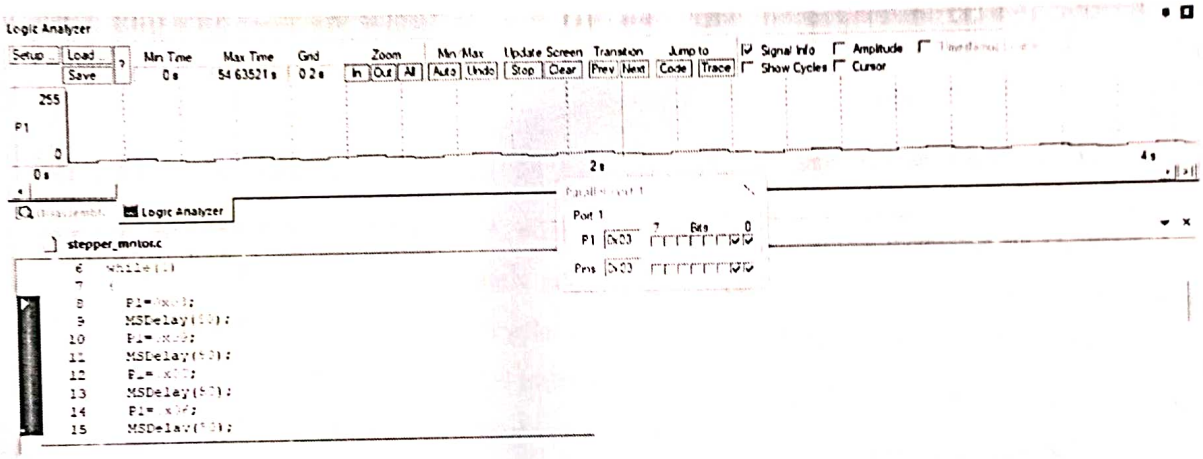
2) Rotating Stepper motor anti-clockwise continuously

Code:

```
#include<reg51.h>
void MSDelay(unsigned int);

void main()
{
        while(1)
        {
                P1=0x03;
                MSDelay(50);
                P1=0x09;
                MSDelay(50);
                P1=0x0C;
                MSDelay(50);
                P1=0x06;
                MSDelay(50);
        }
}
```

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

Output:



3) Rotating Stepper motor through 90 Degrees.

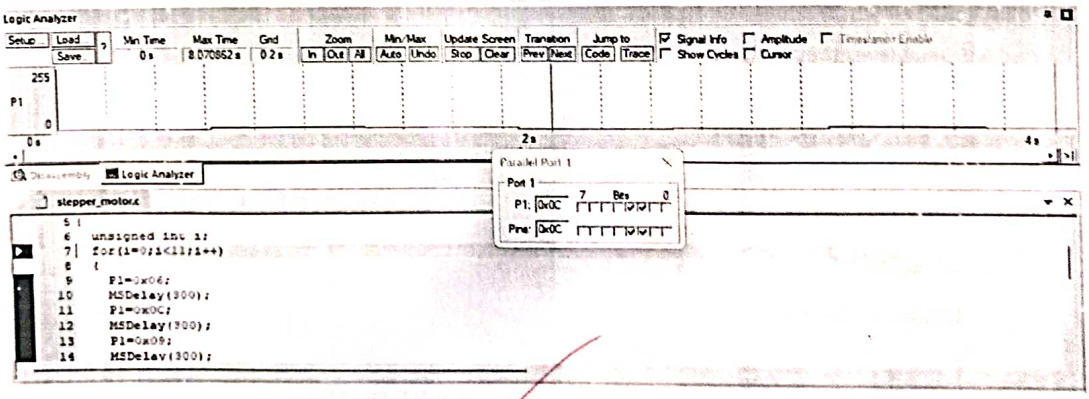
Code:

```
#include<reg51.h>
void MSDelay(unsigned int);

void main()
{
        unsigned int i;
        for(i=0;i<11;i++)
        {
                P1=0x06;
```

Batch :- LS

```
                MSDelay(300);
                P1=0x0C;
                MSDelay(300);
                P1=0x09;
                MSDelay(300);
                P1=0x03;
                MSDelay(300);
        }
        while(1)
        {
                P1=0x00;
        }
}

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

Output:



CLASS : T.E. E &TC Engg.

EXPT. NO.: 5

COURSE: MC

DATE: 22/09/23

TITLE: Interfacing Push buttons, LEDs, Relay & Buzzer to PIC Microcontroller

PROBLEM STATEMENT:

Interface Push buttons, LEDs, Relay and Buzzer to PIC Microcontroller. Write a program in Embedded C to interact with peripherals as follows.

a. LED's start chasing from left to right and turn ON Relay, buzzer whenever pushbutton 1 is pressed.

b. LED's start chasing from right to left and turn OFF Relay, buzzer whenever pushbutton 2 is pressed.

OBJECTIVE:

a) To understand the PORT Structure of PIC Microcontroller.

b) To study the SFRs to control the PORT Pins.

c) To interface common peripherals like pushbuttons, LEDs, relay.

d) To understand the use of MPLAB IDE and C18 Compiler.

e) To write a simple program in Embedded C.

S/W PACKAGES AND H/W USED:

MPLAB IDE, C18 Compiler, Explore PIC Development Board

REFERENCES:

a. Mazidi, PIC microcontroller & embedded system 3rd Edition, Pearson

b. Datasheet of PIC18F4550 / PIC18F4520 / PIC18F458, www.microchip.com

c. ExplorePIC Board Manual

THEORY

Depending on the device selected, there are up to five general purpose I/O ports available on PIC18F Microcontroller devices. Some pins of the I/O ports are multiplexed with an alternate function from the peripheral features on the device. In general, when a peripheral is enabled, that pin may not be used as a general purpose I/O pin.

1. Some common Features of the I/O Ports

- Up to 70 bi-directional I/O pins ○ Some multiplexed with peripheral functions
- High drive capability ○ 25mA source/sink capability
- Direct, single cycle bit manipulation
- 4kV ESD protection diodes ○ Based on human body model □ After reset:
  - ○ Digital I/O default to Input (Hi-Z) ○ Analog capable pins default to analog

2. SFR Associated with I/O Port

Each port has three registers for its operation, figure 1.1 and figure 1.2 show the functioning of each register:

- **TRIS register** (Data Direction register): To select PORT pin as input or output. All port pins are input by default. Whenever a bit in the TRISx register is a 0, the corresponding bit in PORTx is an output. If the bit in TRISx is a 1, the corresponding bit in PORTx is an input.

  For example: TRISD=0x00 defines PORTD as output port. If single port pin has to be used say RB1; TRISBbits.TRISB1=1 will configure it as input pin.
- **PORT register** reads the levels on the pins of the device.
- **LAT register** (output latch): The data latch (LAT register) is useful for read-modify-write operations on the value that the I/O pins are driving.

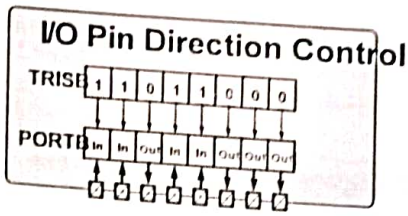


Figure 1.1: TRISx Register

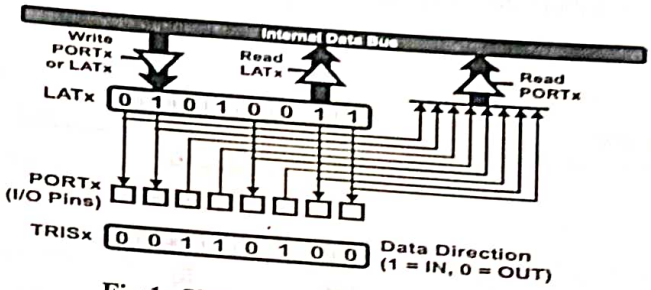


Fig 1: SFRs associated with I/O Port

2.1 Register Map:

SFR	Description	Access	Reset Value	Address
PORT A : RA6 – RA0				
TRISA	PORTA Direction Register	Read/Write	0x7F	0xF92
PORTA	PORTA Read/Write Register	Read/Write	unknown	0xF80
LATA	PORTA Output Latch Register	Read/Write	unknown	0xF89
PORT B: RB7 - RB0				
TRISB	PORTB Direction Register	Read/Write	0xFF	0xF93
PORTB	PORTB Read/Write Register	Read/Write	unknown	0xF81
LATB	PORTB Output Latch Register	Read/Write	unknown	0xF8A
PORT C: RC7 - RC0				
TRISC	PORTC Direction Register	Read/Write	0xFF	0xF94
PORTC	PORTC Read/Write Register	Read/Write	unknown	0xF82
LATC	PORTC Output Latch Register	Read/Write	unknown	0Xf8B
PORT D: RD7 - RD0				
TRISD	PORTD Direction Register	Read/Write	0xFF	0xF95
PORTD	PORTD Read/Write Register	Read/Write	unknown	0xF83

LATD	PORTD Output Latch Register	Read/Write	unknown	0xF8C
PORT D: RE2 – RE1				
TRISE	PORTE Direction Register	Read/Write	0x07	0xF96
PORTE	PORTE Read/Write Register	Read/Write	unknown	0xF84
LATE	PORTE Output Latch Register	Read/Write	unknown	0xF8D

2.2 Port Pin Diagram

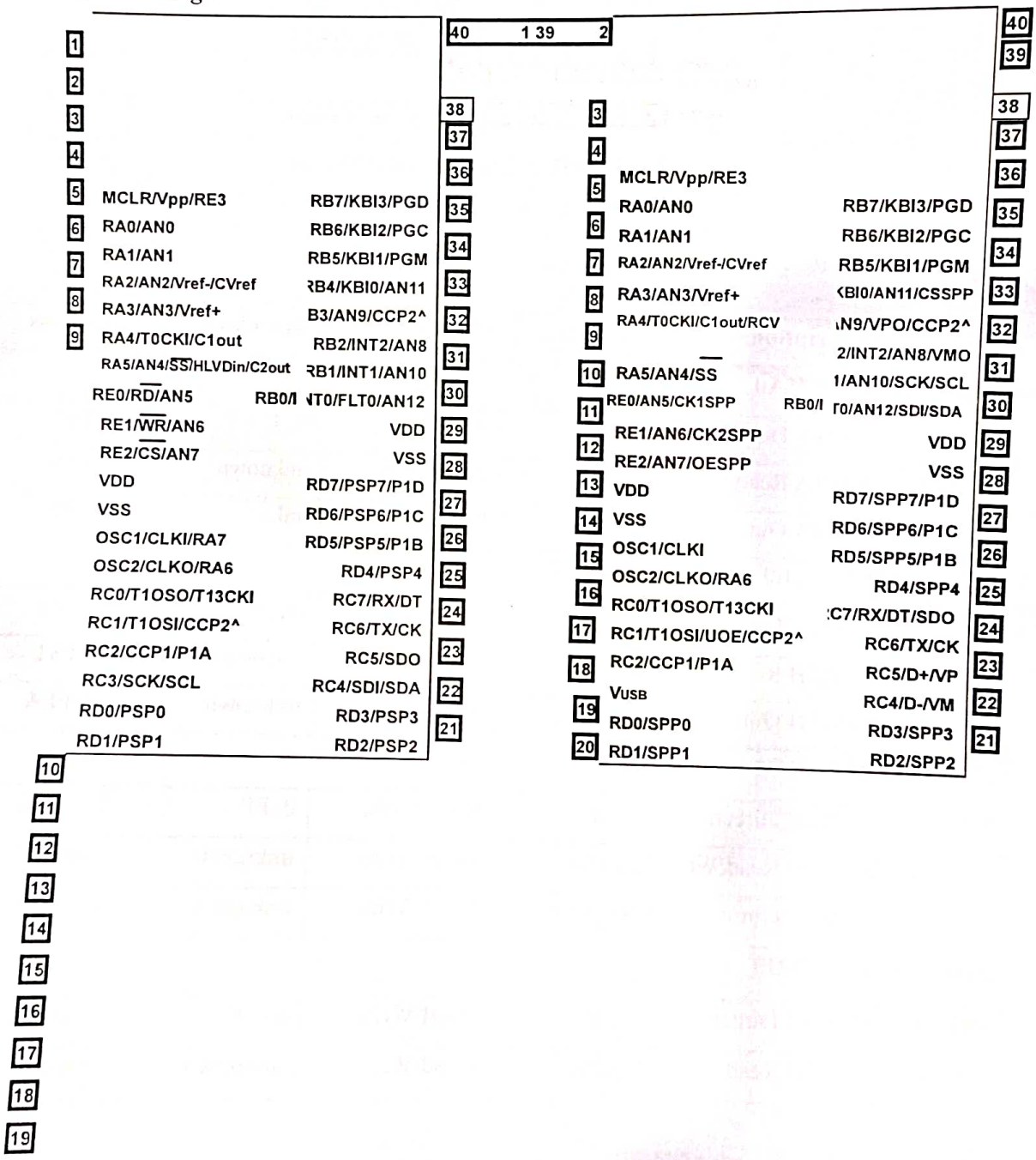


Fig 2 PIC18F4520 Pins

Fig 3 PIC18F4550 Pins

3. Generic I/O Port structure:

All the ports of PIC18 are bidirectional and identical. They all have the following four components in their structure as shown in figure 1.3.

1. DATA LATCH
2. OUTPUT DRIVER
3. INPUT BUFFER
4. TRIS LATCH

The PIC18 Ports have both the latch and buffer. Therefore, when reading the ports there are two possibilities:

1. Reading the input pin
2. Reading the latch

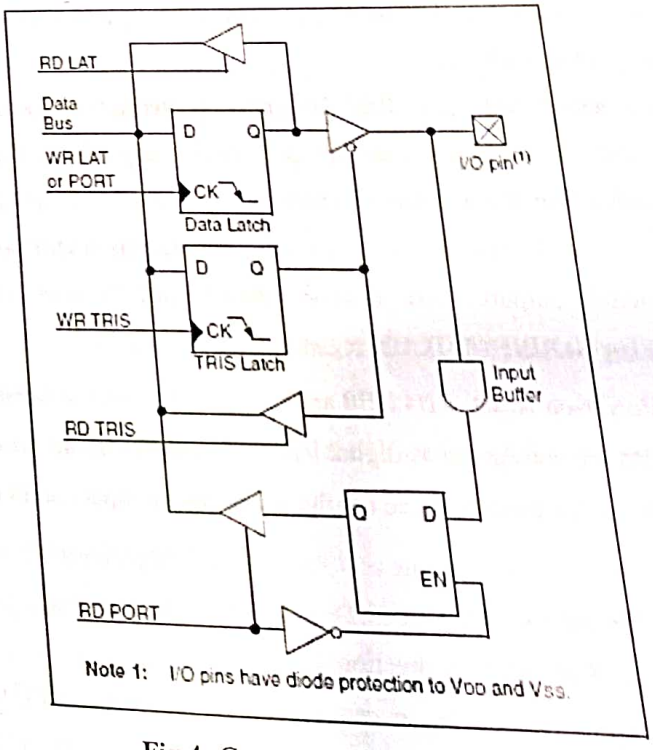


Fig 4: Generic I/O Port Operation

3.1 PORTA, TRISA and LATA Registers

- PORTA is an 8-bit wide, bidirectional port.

- The RA4 pin is multiplexed with the Timer0 module clock input to become the RA4/T0CKI pin.
- The RA6 pin is multiplexed with the main oscillator pin; it is enabled as an oscillator or I/O pin by the selection of the main oscillator in Configuration Register.
- Several PORTA pins are multiplexed with analog inputs, the analog VREF+ and VREF- inputs and the comparator voltage reference output.
- The operation of pins RA5 and RA3:RA0 as A/D converter inputs is selected by clearing/setting the control bits in the ADCON1 register.
- On a Power-on Reset, RA5 and RA3:RA0 are configured as analog inputs and read as '0'. RA4 is configured as a digital input.

3.2 PORTB, TRISB and LATB Registers

- PORTB is an 8-bit wide, bidirectional port.
- Each of the PORTB pins has a weak internal pull-up. A single control bit can turn on all the pull-ups. This is performed by clearing bit RBPU (INTCON2 register). The weak pull-up is automatically turned off when the port pin is configured as an output. The pull-ups are disabled on a Power-on Reset.
- Four of the PORTB pins (RB7:RB4) have an interrupt on change feature. Only pins configured as inputs can cause this interrupt to occur (i.e., any RB7:RB4 pin configured as an output is excluded from the interrupt on change comparison). The input pins (of RB7:RB4) are compared with the old value latched on the last read of PORTB. The "mismatch" outputs of RB7:RB4 are ORed together to generate the RB Port Change Interrupt with Flag bit RBIF (INTCON register).
- On a Power-on Reset, RB4:RB0 are configured as analog inputs by default and read as '0'; RB7:RB5 are configured as digital inputs. By programming the configuration bit, PBADEN, RB4:RB0 will alternatively be configured as digital inputs on POR.

3.3 PORTC, TRISC and LATC Registers

- PORTC is an 8-bit wide, bidirectional port.
- PORTC is multiplexed with several peripheral functions. PORTC is primarily multiplexed with serial communication modules, including the EUSART, MSSP module and the USB module.
- PORTC pins have Schmitt Trigger input buffers.

- When enabling peripheral functions, care should be taken in defining TRIS bits for each PORTC pin.
- Some peripherals override the TRIS bit to make a pin an output, while other peripherals override the TRIS bit to make a pin an input.
- The user should refer to the corresponding peripheral section for the correct TRIS bit settings.
- The pin override value is not loaded into the TRIS register.
- This allows read-modify-write of the TRIS register, without concern due to peripheral overrides.
- In PIC18F2455/2550/4455/4550 devices, the RC3 pin is not implemented.
- On a Power-on Reset, these pins are configured as digital inputs.

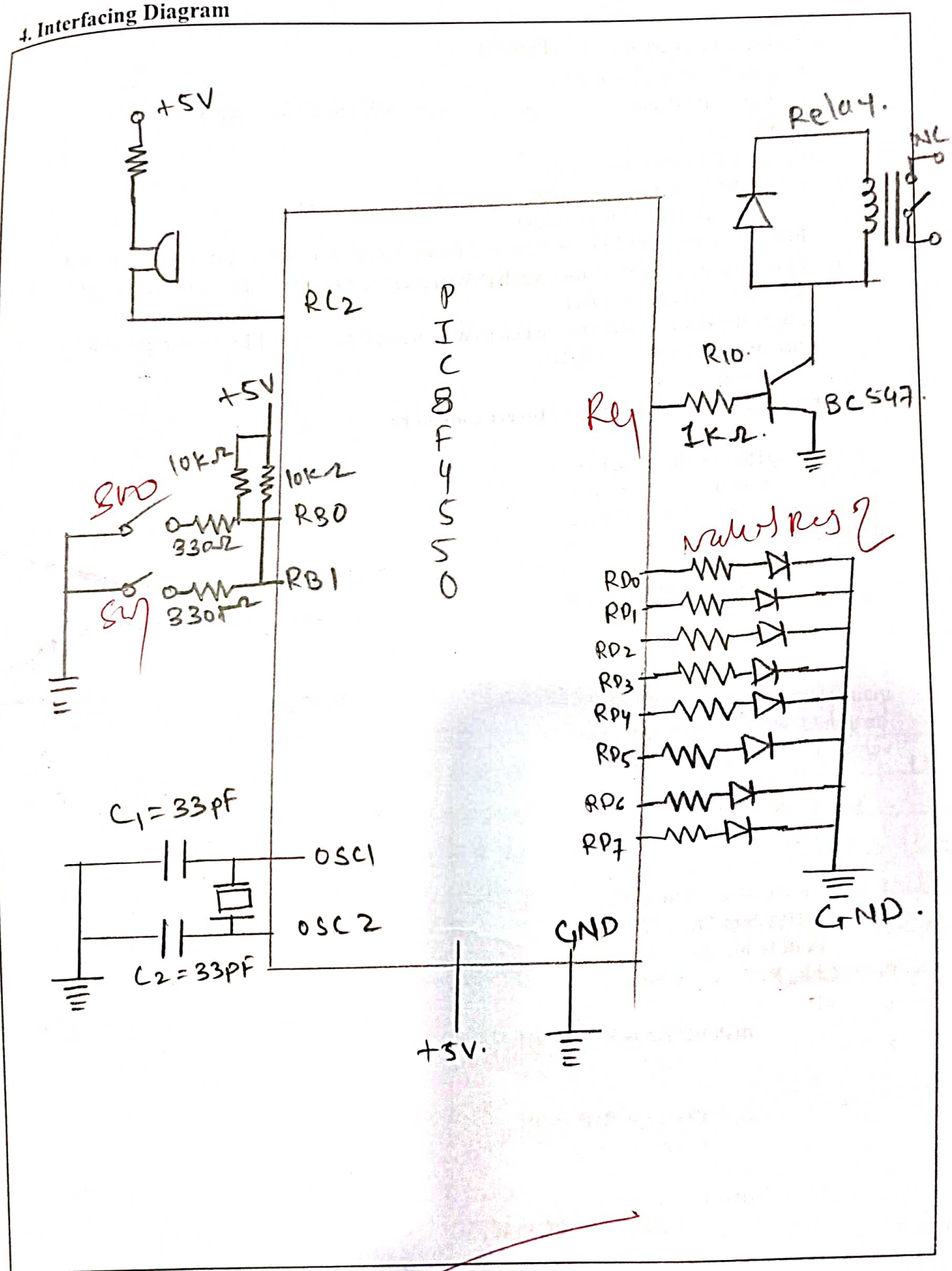
3.4 PORTD, TRISD and LATD Registers

- PORTD is an 8-bit wide, bidirectional port.
- All pins on PORTD are implemented with Schmitt Trigger input buffers.
- Three of the PORTD pins are multiplexed with outputs P1B, P1C and P1D of the enhanced CCP module.
- PORTD can also be configured as an 8-bit wide microprocessor port (Parallel Slave Port (PSP) or Streaming Parallel Port (SPP)). In this mode, the input buffers are TTL.
- On a Power-on Reset, these pins are configured as digital inputs.

3.5 PORTE, TRISE and LATE Registers

- For 40/44-pin devices, PORTE is a 4-bit wide port.
- Three pins (RE0/RD/AN5, RE1/WR/AN6 and RE2/CS/AN7) are individually configurable as inputs or outputs.
- These pins have Schmitt Trigger input buffers. When selected as an analog input, these pins will read as '0's.
- The fourth pin of PORTE (MCLR/VPP/RE3) is an input only pin. Its operation is controlled by the MCLRE configuration bit. When selected as a port pin (MCLRE = 0), it functions as a digital input only pin; as such, it does not have TRIS or LAT bits associated with its operation. Otherwise, it functions as the device's Master Clear input. In either configuration, RE3 also functions as the programming voltage input during programming.
- For 28-pin devices, PORTE is only available when Master Clear functionality is disabled (MCLRE = 0). In these cases, PORTE is a single bit, input only port comprised of RE3 only.

4. Interfacing Diagram



5. Algorithm

1. Activate the internal pull on PORTB
2. Disable the all analog inputs
3. Configure RB0 and RB1 as input for sensing SW1 and SW0 respectively
4. Configure
    a. RC1 (relay) as output
    b. RC2 (buzzer) as output
    c. PORTD (LED) as output
5. Initialize value for LEDs, Buzzer and Relay. Keep them off on power ON or on RESET.
6. Check the status of RB1 for switch (SW0) press. Chase the LEDs from left to right on PORTD, turn on the Buzzer and Relay.
7. Check the status of RB0 for switch (SW1) press. Chase the LEDs from right to left on PORTD, turn off the Buzzer and Relay.

6. Source code (Attach Print out / Insert code here)

```
#include <p18f4550.h> #include
"vector_relocate.h"              void
msdelay(unsigned   int   itime){
unsigned int i,j;
        for(i=0;i<itime;i++){
                for(j=0;j<710;j++){
                }
        }
}        void
main(){
 unsigned int i;  unsigned char
val=0; INTCON2bits.RBPU = 0;
        ADCON1=0x0F;
        TRISD = 0x00;
        TRISBbits.TRISB0=1;
        TRISBbits.TRISB1=1;
        TRISCbits.TRISC1=0;
        TRISCbits.TRISC2=0;
        PORTCbits.RC1=0;
PORTCbits.RC2=0;   while(1)
        {
                if(PORTBbits.RB0==0){
                        val=1;
                }
                if(PORTBbits.RB1==0){
                        val=2;
                }
                if(val==1){
                        PORTCbits.RC1=1;
```

```
                PORTCbits.RC2=1;
            PORTD = 0x80;
        msdelay(100);
        for(i=0;i<8;i++){
        PORTD=PORTD>>1;
        msdelay(100);
        }
    }
        if(val==2){
        PORTCbits.RC1=0;
        PORTCbits.RC2=0;
        PORTD = 0x01;
        msdelay(100);
        for(i=0;i<8;i++){
        PORTD=PORTD<<1;
        msdelay(100);
        }
    }
}


}
```

7. Result (Output snaps) & Conclusion

In this experiment we implemented PIC18F4550 microcontroller interfacing with LED, buzzer. and relay by the power of PIC microcontroller we can achieve enhanced functionality and it's multifunctional port pins allow us to interface multiple components at same time.

Sen
13/1/23 (C)

```
//Expt.1: Interfacing LEDs, Switches, Buzzer and Relay
//Includes
#include <p18f4550.h>                    //Include Controller specific .h
#include "vector_relocate.h"     //Vector Remapping for USB HID Bootloader

//Declarations
#define lrbit    PORTBbits.RB1    //SW0 interfaced to RB1
#define rlbit    PORTBbits.RB0    //SW1 interfaced to RB0
#define buzzer   PORTCbits.RC2    //Buzzer interfaced to RC2
#define relay    PORTDbits.RD7    //Relay interfaced to RC1

//Function Prototypes
void msdelay (unsigned int time);//Function for delay

//Start of Program Code
void main()                                          //Main Program
{
        unsigned char i,val=0;          //Variable to latch the switch condition
    INTCON2bits.RBPU=0;                 //To Activate the internal pull on PORTB
        ADCON1 = 0x0F;                          //To disable the all analog inputs


        TRISBbits.RB0=1;                //To configure RB0 as input for sensing SW
        TRISBbits.RB1=1;                //To configure RB1 as input for sensing SW0

        TRISDbits.TRISD7=0;             //To configure RC1 (relay) as output
        TRISCbits.TRISC2=0;                     //To configure RC2 (buzzer) as
output
        TRISA = 0x00;                           //To configure PORTD (LED) as
output


        PORTA = 0x00;                   //Initial Value for LED
        buzzer = 0;                             //Initial Value for Buzzer
        relay = 0;                              //Initial Value for Relay

while (1)                                       //While loop for repeated
operation
        {
        if (lrbit==0)                   //To check whether SW0 is pressed
                val = 1;                        // Latch the status of
switch SW0
        if (rlbit==0)                   //To check whether SW1 is pressed
                val = 2;                        // Latch the status of
switch SW1

       if (val == 1)
          {
               buzzer = 1;
               relay = 1;
       PORTA = 0x20;
```

```
            msdelay(50);
        for(i=0;i<8;i++)
          {
                  PORTA = PORTA >>1;
           msdelay(50);
          }       //Shift left by 1 bit

                                  // Make the MSB bit equal to 1
             }
          if (val == 2)
             {
                  buzzer = 0;
                  relay = 0;
                  PORTA = 0x01;
           msdelay(50);
                  for(i=0;i<8;i++)
 {
                  PORTA = PORTA <<1;
    msdelay(50);
}       //Shift left by 1 bit    // Make the MSB bit equal to 1
             }
          }
}
Program                                                          //End of the

//Function Definitions
void msdelay (unsigned int time)//Function for delay
{
unsigned int i, j;
 for (i = 0; i < time; i++)
          for (j = 0; j < 710; j++);     //Calibrated for a 1 ms delay in MPLAB
}
```

32116

PUNE INSTITUTE OF COMPUTER TECHNOLOGY, PUNE - 411043

Department of Electronics & Telecommunication Engineering

CLASS : T.E. E &TC Engg. **COURSE: MC**
EXPT. NO.: 6 **DATE:** 06/10/23

TITLE : Generation of Square wave using timer with interrupt.

PROBLEM STATEMENT:

Write a program to generate a square wave of 10 Hz. Use timer0 interrupt for generation of delay.

OBJECTIVE:

a. To understand the basic concepts of Timer and Counter
b. To study in detail Timer0 of PIC Microcontroller
c. To Study interrupt structure of PIC Microcontroller
d. To use timer interrupt and its related SFRs.

S/W PACKAGES AND H/W USED:

MPLAB IDE, C18 Compiler, DSO, Universal Board of PIC18F and PICkit3

REFERENCES:

a. Mazidi, PIC microcontroller & embedded system 3rd Edition ,Pearson
b. Datasheet of PIC18F4550 / PIC18F4520 / PIC18F458, www.microchip.com

THEORY

1. Timer

The microcontroller oscillator uses quartz crystal for its operation. Even though it is not the simplest solution, there are many reasons to use it. The frequency of such oscillator is precisely defined and very stable, so that pulses it generates are always of the same width, which makes them ideal for time measurement. Such oscillators are also used in quartz watches. If it is necessary to measure time between two events, it is sufficient to count up pulses generated by this oscillator. This is exactly what the timer does. Most programs use these miniature electronic 'stopwatches. These are commonly 8- or 16-bit SFRs the contents of which are automatically incremented by each coming pulse. Once a register is completely loaded and overflowed, an interrupt may be generated!

If the timer uses an internal quartz oscillator for its operation, then it can be used to measure time between two events (if the register value is A at the moment measurement starts, and B at the moment it terminates, then the elapsed time is equal to the result of subtraction B - A). If registers use pulses coming from external source, then such a timer is turned into a counter.

1.1 Using a pre scaler in timer operation:

A pre scaler is an electronic device used to reduce frequency by a predetermined factor. In order to generate one pulse on its output, it is necessary to bring 1, 2, 4 or more pulses on its input. Most microcontrollers have one or more pre scaler built in and their division rate may be changed from within the program. The pre scaler is used when it is necessary to measure longer periods of time.

2. Counters

If the timer receives pulses from the microcontroller input pin, then it turns into a counter. Obviously, it is the same electronic circuit able to operate in two different modes. The only difference is that in this case, the pulses to be counted come over the microcontroller input pin and their duration (width) is mostly undefined. This is why they cannot be used for time measurement, but for other purposes such as counting products on an assembly line, number of axis rotation, passengers etc. (depending on sensor in use).

3. Timers / Counters in PIC 18F4550 Microcontroller

There are 4 timers on chip in PIC microcontroller. These timers can also be used as counters when external pulses are applied. The timers are programmable, and sometimes share with other peripheral devices. These are named as TMR0, TMR1, TMR2 and TMR3.

Parameter	Timer 0	Timer 1 & 3	Timer 2 & 4
Size of timer register	8-bit or 16-bit	16-bit	8-bit
Clock Source (Internal)	Fosc/4	Fosc/4	Fosc/4
Clock Source (External)	T0CKI Pin	T13CKI Pin / T1OSC	None
Clock Scaling (Prescaler)	Prescaler 8-bits (1:2 -> 1:256)	Prescaler 2-bits (1:1 -> 1:8)	Prescaler (1:1, 1:4, 1:8) Postscaler (1:1 -> 1:16)
Interrupt Event	On Overflow	On Overflow	TMR Reg matches with PR2

Table 1: Comparison of Timers in PIC Microcontroller

4. Timer0 Module in PIC Microcontroller

The Timer0 module incorporates the following features:

- Software selectable operation as a timer or counter in both 8-bit or 16-bit modes

- Readable and writable registers
- Dedicated 8-bit, software programmable prescaler
- Selectable clock source (internal or external)
- Edge select for external clock
- Interrupt on overflow

The T0CON register controls all aspects of the module's operation, including the prescale selection. It is both readable and writable. A simplified block diagram of the Timer0 module in 8-bit mode is shown in Figure 1.1. Figure 1.2 shows a simplified block diagram of the Timer0 module in 16-bit mode.

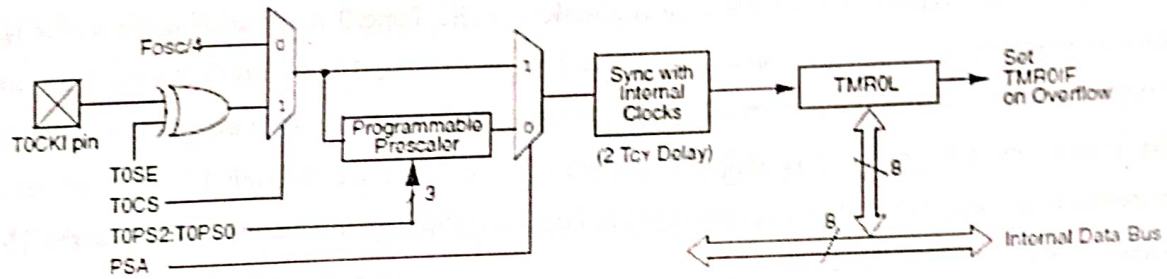


Fig 1: Block Diagram of Time0 in 8-bit

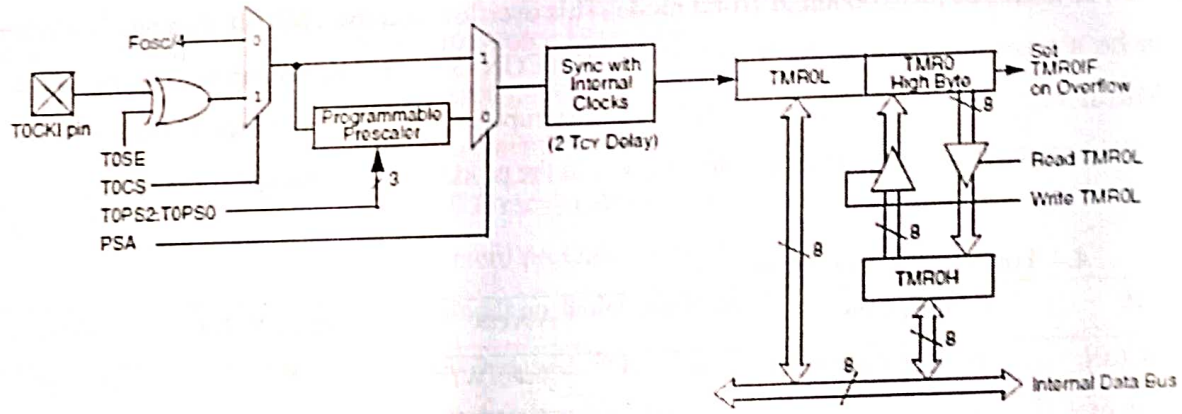


Fig 2: Block Diagram of Time0 in 16-bit

4.1 Timer0 Operation

Timer0 can operate as either a timer or a counter; the mode is selected by clearing the T0CS bit (T0CON<5>). In Timer mode, the module increments on every clock by default unless a different prescaler value is selected. If the TMR0 register is written to, the increment is inhibited for the

following two instruction cycles. The user can work around this by writing an adjusted value to the TMR0 register.

The Counter mode is selected by setting the T0CS bit (= 1). In Counter mode, Timer0 increments either on every rising or falling edge of pin RA4/T0CKI. The incrementing edge is determined by the Timer0 Source Edge Select bit, T0SE (T0CON<4>); clearing this bit selects the rising edge. An external clock source can be used to drive Timer0; however, it must meet certain requirements to ensure that the external clock can be synchronized with the internal phase clock (TOSC). There is a delay between synchronization and the onset of incrementing the timer/counter.

4.2 Pre scaler

An 8-bit counter is available as a pre scaler for the Timer0 module. The pre scaler is not directly readable or writable; its value is set by the PSA and T0PS2:T0PS0 bits (T0CON<3:0>) which determine the pre scaler assignment and pre scale ratio. Clearing the PSA bit assigns the pre scaler to the Timer0 module. When it is assigned, pre scale values from 1:2 through 1:256, in power-of-2 increments are selectable. When assigned to the Timer0 module, all instructions writing to the TMR0 register clear the pre scaler count.

4.3 Timer0 Interrupt

The TMR0 interrupt is generated when the TMR0 register overflows from FFh to 00h in 8-bit mode, or from FFFFh to 0000h in 16-bit mode. This overflow sets the TMR0IF flag bit. The interrupt can be masked by clearing the TMR0IE bit (INTCON<5>). Before reenabling the interrupt, the TMR0IF bit must be cleared in software by the Interrupt Service Routine. Since Timer0 is shut down in Sleep mode, the TMR0 interrupt cannot awaken the processor from Sleep.

4.4 Timer0 Register Map:

SFR	Description	Access	Reset Value	Address
T0CON	Timer0 Control Register	Read/Write	0xFF	0xFD5
TMR0L	Timer0 Register Lower Byte	Read/Write	UNKNOWN	0xFD6
TMR0H	Timer0 Register Higher Byte	Read/Write	0x00	0xFD7
INTCON	Interrupt Control Register	Read/Write	0x00	0xFF2
INTCON2	Interrupt Control Register 2	Read/Write	0xFF	0xFF1

4.4. Register (SFR) Description

4.4.1 TMR0L : Used in 8-bit and 16bit mode. The register holds the current count value which is updated by clock source. User must write initial value.

4.4.2 TMR0H : Used only in 16-bit mode. The register holds the higher byte current count value which is updated by clock source.

4.4.3 T0CON: Timer0 Control Register

D7	D6	D5	D4	D3	D2	D1	D0
TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0

Bit No.	Control Bit	Description
Bit 7	TMR0ON	Timer0 On/Off Control bit 1 = Enables Timer0, 0 = Stops Timer0
Bit 6	T08BIT	Timer0 8-Bit/16-Bit Control bit 1 = Timer0 is configured as an 8-bit timer/counter 0 = Timer0 is configured as a 16-bit timer/counter
Bit 5	T0CS	Timer0 Clock Source Select bit 1= Transition on T0CKI pin 0 = Internal instruction cycle clock (CLKO)
Bit 4	T0SE	Timer0 Source Edge Select bit 1 = Increment on high-to-low transition on T0CKI pin 0 = Increment on low-to-high transition on T0CKI pin
Bit 3	PSA	Timer0 Prescaler Assignment bit 1 = TImer0 prescaler is NOT assigned. 0 = Timer0 prescaler is assigned.
Bit 2-0	T0PS2:T0PS0	Timer0 Prescaler Select bits 111 = 1:256 Prescale value 110 = 1:128 Prescale value 101 = 1:64 Prescale value 100 = 1:32 Prescale value 011 = 1:16 Prescale value

		010 = 1:8 Prescale value 001 = 1:4 Prescale value 000 = 1:2 Prescale value

NOTE: All bits are applicable to this exercise.

4.4.4 INTCON: Interrupt Control Register

D7	D6	D5	D4	D3	D2	D1	D0
GIE/GIEH	PEIE/ GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF

Bit No.	Control Bit	Description
Bit 7	GIE/GIEH	**Global Interrupt Enable bit When IPEN = 0:** **1 = Enables all unmasked interrupts** **0 = Disables all interrupts** When IPEN = 1: **1** = Enables all high priority interrupts 0 = Disables all high priority interrupts
Bit 6	PEIE/GIEL	Peripheral Interrupt Enable bit
Bit 5	TMR0IE	**TMR0 Overflow Interrupt Enable bit** **1 = Enables the TMR0 overflow interrupt** **0 = Disables the TMR0 overflow interrupt**
Bit 4	INT0IE	INT0 External Interrupt Enable bit 1 = Enables the INT0 external interrupt 0 = Disables the INT0 external interrupt
Bit 3	RBIE	RB Port Change Interrupt Enable bit 1 = Enables the RB port change interrupt 0 = Disables the RB port change interrupt
Bit 2	TMR0IF	**TMR0 Overflow Interrupt Flag bit** **1 = TMR0 register has overflowed (must be cleared in software) 0 = TMR0 register did not overflow**
Bit 1	INT0IF	INT0 External Interrupt Flag bit 1 = The INT0 external interrupt occurred (must be cleared in software) 0 = The INT0 external interrupt did not occur
Bit 0	RBIF	RB Port Change Interrupt Flag bit(1) 1 = At least one of the RB7:RB4 pins changed state (must be cleared in software) 0 = None of the RB7:RB4 pins have changed state

NOTE: Only highlighted bits are applicable to this exercise.

5. Time delay generation using timer:

The following steps are taken to generate a time delay using polling method

i. Load the proper value in T0CON indicating which timer mode, clock source, prescaler assignment.

ii. Load the registers TMR0H first and then TMR0L with initial count values. Delay generated is Depends upon the initial count value. iii. Start the timer by setting TMR0ON bit in T0CON.

iv. Keep the monitoring the timer flag (TMR0IF).

v. Get out of the loop when TMR0IF becomes high.

vi. Stop the timer. vii. Clear the TMR0IF flag for the next round.

viii. Go back to the step ii to load TMR0H and TMR0L values.

The following steps are taken to generate a time delay using interrupt method:

i. Load the proper value in T0CON indicating which timer mode, clock source, prescaler assignment.

ii. Load the registers TMR0H first and then TMR0L with initial count values. Delay generated is Depends upon the initial count value. iii. Enable the Timer0 Interrupt and Global Interrupt using INTCON Register iv. Start the timer by setting TMR0ON bit in T0CON. v. Write the ISR at Interrupt vector 0x1008.

   a. In ISR Clear the TMR0IF flag for the next round.

   b. In ISR reload TMR0H and TMR0L values.

The size of the time delay depends on following factors, (a) the crystal frequency and (b) the timer's 16-bit register (c) the prescaler value. The largest delay is achieved by the making both TMR0H and TMR0L zero and using maximum prescaler value i.e. 1:256.

For Fosc = 48 MHz, Therefore Tp= 4/ Fosc. (The Fosc is internally divided by 4)

Formula for delay calculations using timer0 is:

(a) Without Prescaler

Time Delay (Td) = (65536 - NNNNN) x Time Period T_p

(b) With Prescaler (1:256)

Time Delay (Td) = (65536 - NNNNN) x (Timer Period T_p x Prescaler Value)

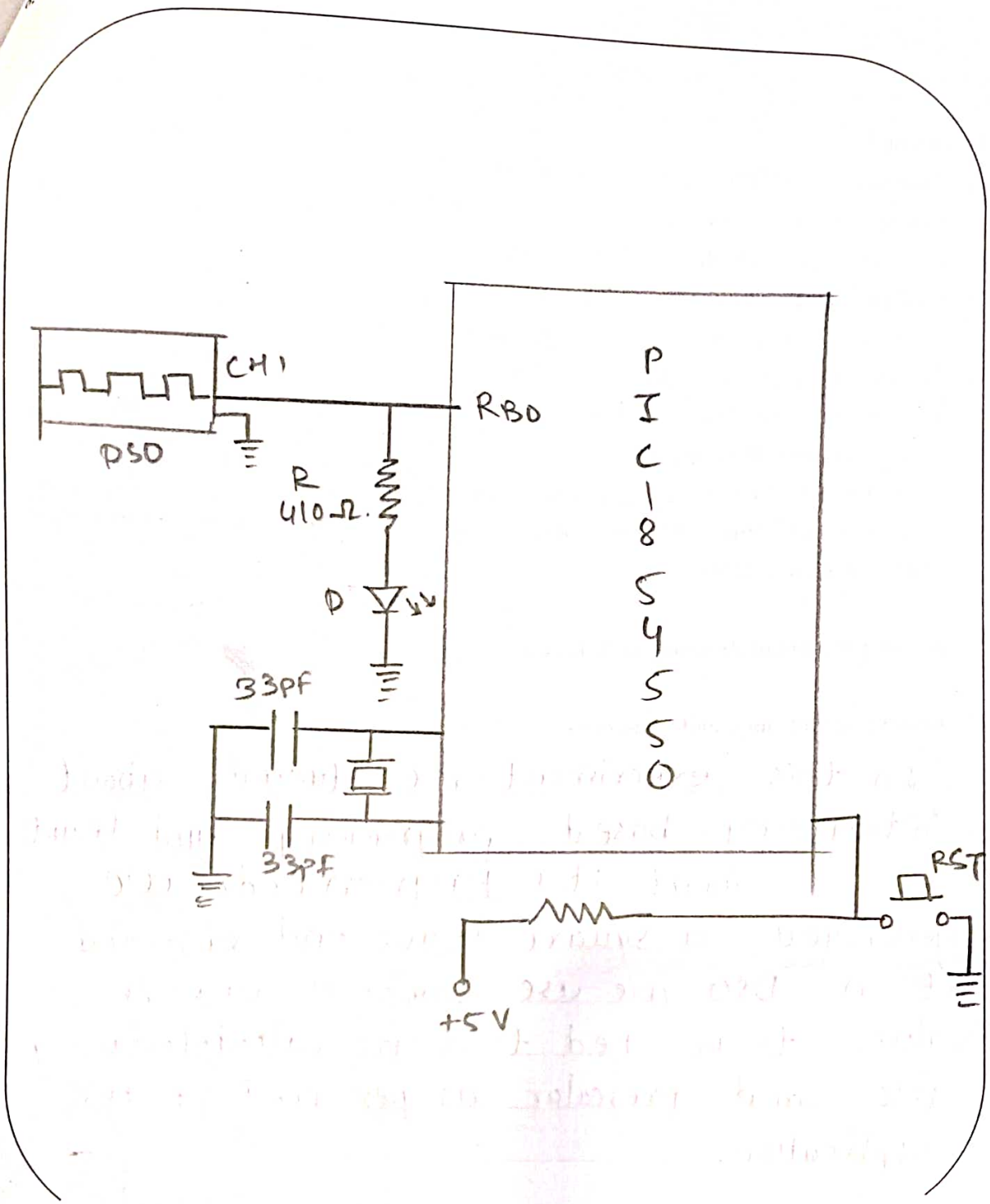
5.1 Finding the Values to be loaded into timer for desired delay: (a) Without Prescaler

- Divide the desired time delay **Td** by T_p to get n
- Perform 65536 – n. where n is the decimal value from step 1.
- Convert the result of step 2 to hexadecimal, where yyxx is the initial hex value to be loaded into the timer's registers.
- Set TMR0L = xx and TMR0H = yy. **(b) With Prescaler**
- Multiply Timer period T_F by prescaler value to Teq

  $T_{eq} = T_F$ * **Prescaler vale**
- Divide the desired time delay **Td** by T_{eq} to get n n = Td/Teq
- Perform 65536 – n. where n is the decimal value from above.
- Convert the result of above step to hexadecimal, where yyxx is the initial hex value to be loaded into the timer's registers.
- Set TMR0L = xx and TMR0H = yy.

5.2 Calculation for generating square wave of 10Hz.

Interfacing Diagram



7. Algorithm

1. Configure Port pin RB0 as output and initial value 0.
2. Load the value 0x01 in T0CON indicating which timer in 16-bit mode, clock source Fosc/4, prescaler assignment with 1:16.
3. Load the registers TMR0H first and then TMR0L with calculated initial count values.
4. Enable the Timer0 Interrupt and Global Interrupt using INTCON Register
5. Start the timer by setting TMR0ON bit in T0CON.
6. Write the ISR at Interrupt vector 0x1008 which contains following.
    a. Toggle the PORT pin RB0
    b. Clear the TMR0IF flag for the next round.
    c. Reload TMR0H and TMR0L values.
7. Check the result on DSO.
8.

8. **Attach print out of Program (code here):**

9. **Result (Output snap) and Conclusion:**

In this experiment we studied about interfacing - based programming and learnt more about it's program code. We generated a square wave and observed it on DSO, we use timer 0 and the values to be fed to it are calculated using fosc and prescalar as per need of the application

Seen 20/1/25

***

```
*************/
//This program demonstrates the use of timer0 to generate 10Hz frequency by   */
interrrupt method     //
/*********************/
#include <p18f4550.h>


void timer_isr(void);
void delay_ms(unsigned int);

extern void _startup (void);
#pragma code RESET_INTERRUPT_VECTOR = 0x1000

void reset (void)
{
        _asm
goto _startup
_endasm
}
#pragma code

#pragma code HIGH_INTERRUPT_VECTOR = 0x1008
void high_ISR (void)
{
_asm

goto timer_isr
_endasm    //The program is relocated to execute the interrupt routine timer_iser

}
#pragma code


// This function is executed as soon as the timer interrupt is generated due to
timer overflow
#pragma interrupt timer_isr
void timer_isr(void)
{
TMR0H = 0X6D;                          // Reloading the timer values after overflow
TMR0L = 0X26;
PORTBbits.RB0 = ~PORTBbits.RB0;                          //Toggle the PORTB led
outputs RB0 - RB3
  INTCONbits.TMR0IF = 0;              //Resetting the timer overflow interrupt flag
}


void main()
{
    INTCON2bits.RBPU=0;   //To Activate the internal pull on PORTB
ADCON1 = 0x0F;
```

```
 TRISBbits.TRISB0 = 0;
 PORTBbits.RB0=0;
 T0CON = 0x03; //Set the timer to 16-bit mode,internal instruction cycle clock,1:256
 prescaler
   TMR0H = 0x00;                // Reset Timer0 to 0x48E5 TO MAKE DELAY OF 1 SECOND
   TMR0L = 0x00;
 INTCONbits.GIE = 1; // Global interrupt enabled
 INTCONbits.TMR0IE = 1; // TMR0 interrupt enabled
   T0CONbits.TMR0ON = 1; // Start timer0
 while(1);
 }


 void delay_ms(unsigned int time)
 {
  unsigned int i,j;
   for (i=0;i<time;i++)
 for (j=0; j<710;j++);
 }
```

PUNE INSTITUTE OF COMPUTER TECHNOLOGY, PUNE - 411043

Department of Electronics & Telecommunication Engineering

CLASS : T.E. E &TC Engg. **COURSE: MC**

EXPT. NO.: 7 **DATE:** 13/10/23

TITLE: LCD interfacing with PIC18F4550

PROBLEM STATEMENT:

Interface 16*2 LCD module to PIC microcontroller and write a program in Embedded C to display characters on 16x2 LCD display.

OBJECTIVE:

a. To understand the working of LCD module.
b. To study the LCD section of PIC Microcontroller.
c. To interface LCD module to PIC Microcontroller

S/W PACKAGES AND H/W USED:

MPLAB IDE, C18 Compiler, PIC KIT3, Universal Development board

REFERENCES:

a. Mazidi, PIC microcontroller & embedded system 3rd Edition, Pearson
b. Datasheet of PIC18F4550 / PIC18F4520 / PIC18F458, www.microchip.com

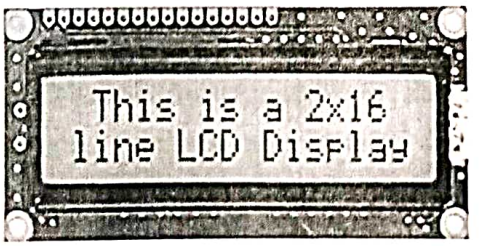
THEORY:

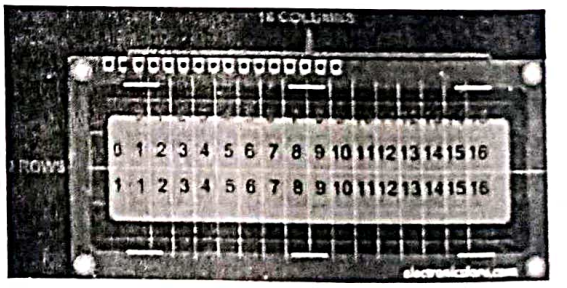
LCDs (Liquid Crystal Displays) are used for displaying status or parameters in embedded systems. LCD 16x2 is a 16-pin device which has 8 data pins (D0-D7) and 3 control pins (RS, RW, EN). The remaining 5 pins are for supply and backlight for the LCD.

The control pins help us configure the LCD in command mode or data mode. They also help configure read mode or write mode and also when to read or write.

LCD 16x2 can be used in 4-bit mode or 8-bit mode depending on the requirement of the application. In order to use it, we need to send certain commands to the LCD in command mode and once the LCD is configured according to our need, we can send the required data in data mode.

Basics of LCD:





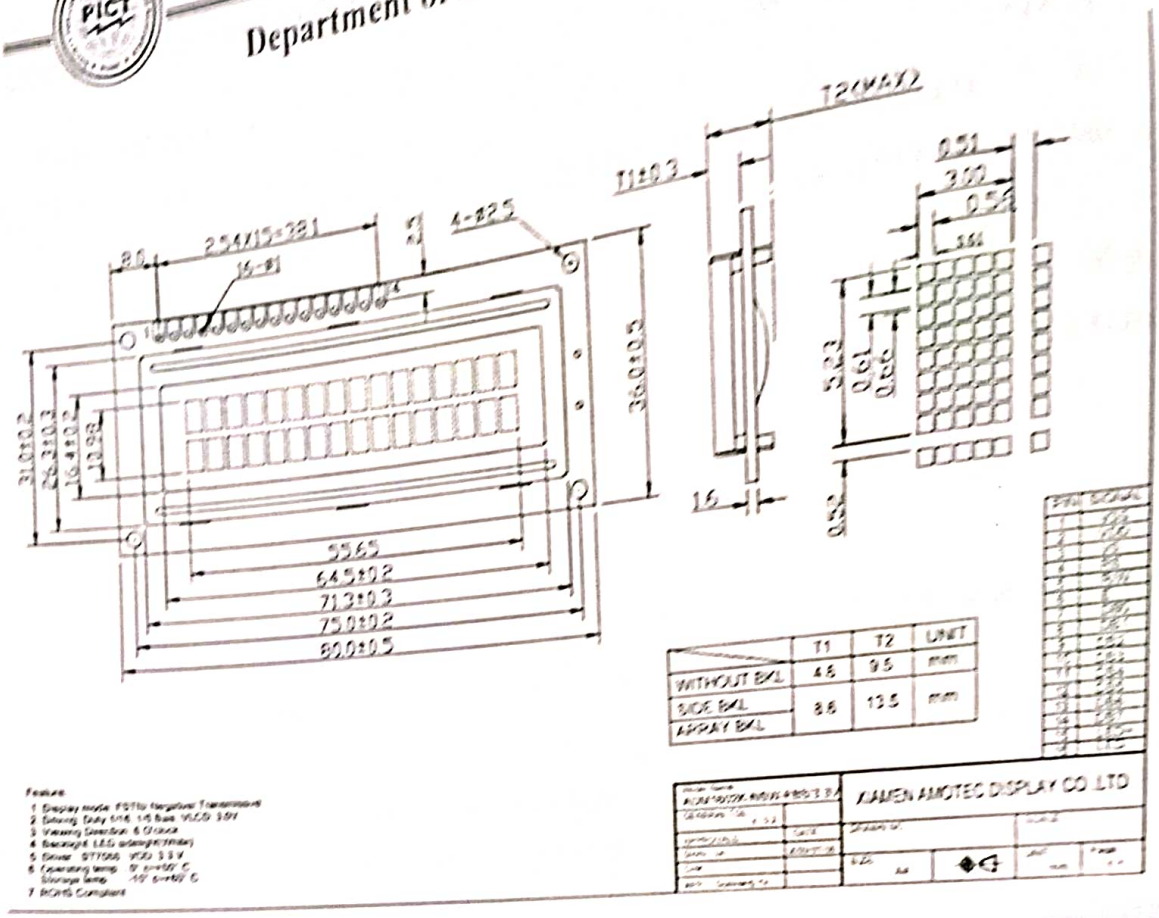


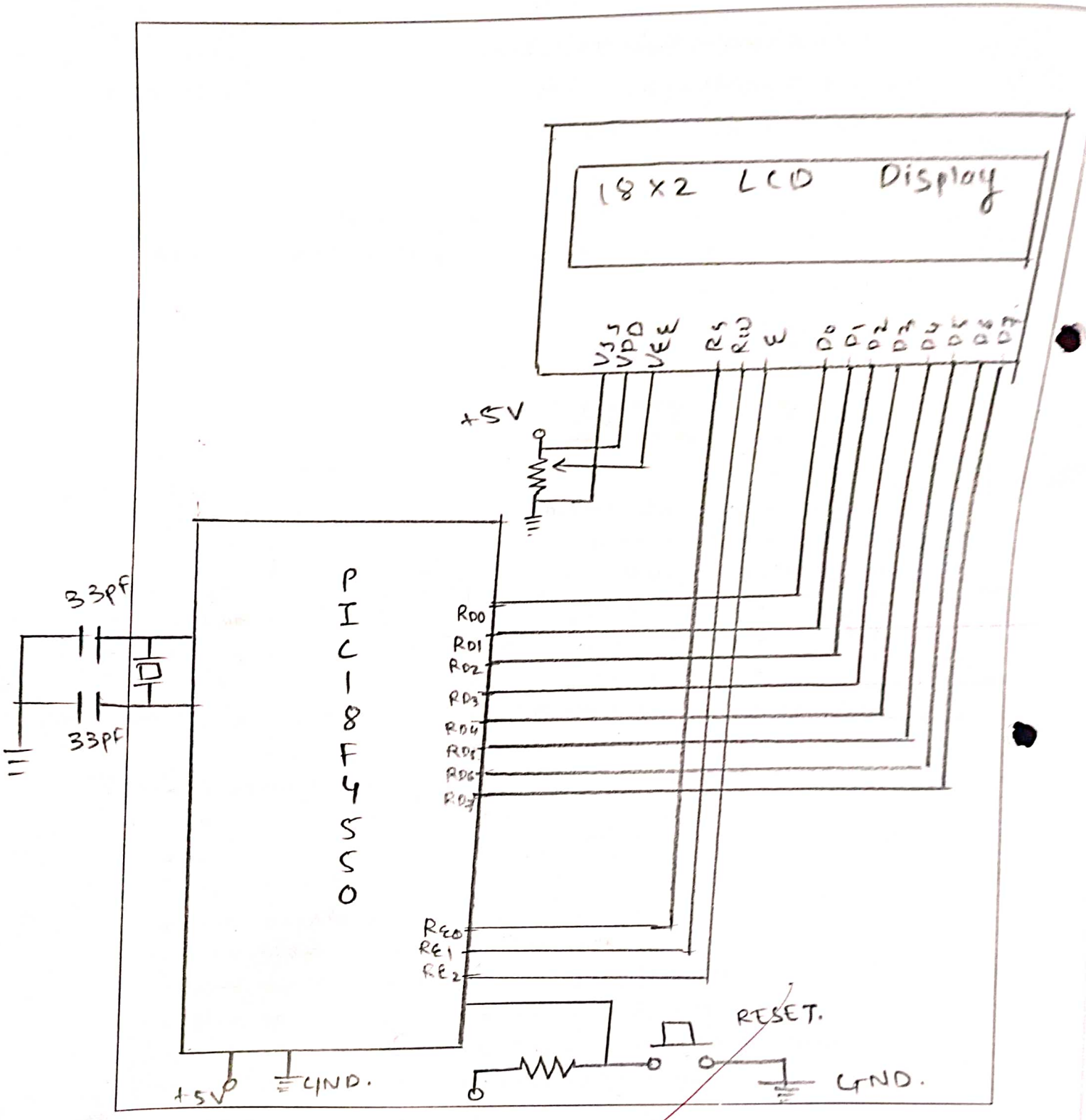
Fig 1

- It has two internal registers:
  - Command register: It is used to store command for example clear display, display ON, cursor ON, select LCD 16*2
  - Data register: It is used t store ASCII value of character that is to be display on LCD screen.
- To select these registers, RS (Register Select) terminal is used. When RS=0, command register is selected and when RS=1, data register is selected.
- If we want to write command in command register RS=0 and when we want data, RS=1 means ASCII value of character which we want to display on LCD.
- R/W – Read write signal (W is active low): Generally, we perform write operation. When this signal is 0, then write operation is performed means we can write command or data in command or data register. When this pin is 1, can read from LCD.
- EN (Enable) - Whenever we want to latch a data,then place data or command on data bus that D0 – D7, after that need to send EN signal. EN signal for write operation is high to low pulse means make this pin high and after some delay make it low.
- Command for LCD

- 38H – It is used to select LCD of 2 rows and 5*7 matrix display (to display each character).
- 0EH – It is used to display ON and cursor ON
- 01H – It is used to clear the display. Before displaying any data on LCD, we can send this command to clear the previous data.

- Command for LCD's character address
  - Each character has address starts from 80H (first row) and C0H for second row.
  - To display character at a particular position, need to send address of character in command register.
- Flow chart for LCD interfacing:
  - Steps for command
    - Place command on port
    - Make RS=0 (command)
    - Make R/W=0 (write)
    - Make enable 1 and then 0
    - Call delay (10ms)
  - Steps for data (ASCII)
    - Place data on port
    - Make RS=1 (data)
    - Make R/W=0 (write)
    - Make enable 1 and then 0
    - Call delay (ms)

5. Interfacing diagram



6. Source code (Attach Print out / Insert code here)

7. Result:

Sr.No.	Vin(Volts)	Result on ADC
1.		
2.		
3.		
4.		
5.		

8. Conclusion:

We studied about interfacing of LCD with PIC18F4550. LCD was observed to have 3 control pins and 8 data pins(D_0 to D_7) which can be connected to LCD control pins. Hence the desired message can be displayed on LCD by passing appropriate LCD commands.

Sen

27/1/23

B 11

```
//Expt.2: LCD Interfacing
//Includes
#include <p18f4550.h>
#include "vector_relocate.h"

//Declarations
#define LCD_DATA    PORTD                       //LCD data port to PORTD
#define ctrl        PORTE                       //LCD control port to PORTE
#define rs          PORTEbits.RE0       //register select signal to RE0
#define rw          PORTEbits.RE1       //read/write signal to RE1
#define en          PORTEbits.RE2       //enable signal to RE2

//Function Prototypes
void init_LCD(void);                                        //Function to initialise
the LCD
void LCD_command(unsigned char cmd);    //Function to pass command to the LCD
void LCD_data(unsigned char data);              //Function to write character to
the LCD
void LCD_write_string(static char *str);//Function to write string to the LCD
void msdelay (unsigned int time);               //Function to generate delay

//Start of Main Program
void main(void)
{
     char var1[] = "aaaaa";//Declare message to be displayed
     char var2[] = "aaaaaa";

     ADCON1 = 0x0F;                         //Configuring the PORTE pins as digital I/O

     TRISD = 0x00;              //Configuring PORTD as output
     TRISE = 0x00;                      //Configuring PORTE as output

     init_LCD();                // call function to initialise of LCD
     msdelay(50);               // delay of 50 mili seconds

     LCD_write_string(var1);//Display message on first line
     msdelay(15);

     LCD_command(0xC0);         // initiate cursor to second line
     LCD_write_string(var2);//Display message on second line

     while (1);                             //Loop here
}                                                       //End of Main

//Function Definitions
void msdelay (unsigned int time) //Function to generate delay
{
unsigned int i, j;
  for (i = 0; i < time; i++)
        for (j = 0; j < 710; j++);//Calibrated for a 1 ms delay in MPLAB
```

```
}


void init_LCD(void)             // Function to initialise the LCD
{
    LCD_command(0x38);      // initialization of 16X2 LCD in 8bit mode
    msdelay(15);
    LCD_command(0x01);      // clear LCD
    msdelay(15);
    LCD_command(0x0C);      // cursor off
    msdelay(15);
    LCD_command(0x80);      // go to first line and 0th position
    msdelay(15);
}

void LCD_command(unsigned char cmd) //Function to pass command to the LCD
{
    LCD_DATA = cmd;             //Send data on LCD data bus
    rs = 0;                             //RS = 0 since command to LCD
    rw = 0;                             //RW = 0 since writing to LCD
    en = 1;                             //Generate High to low pulse on EN
    msdelay(15);
    en = 0;
}

void LCD_data(unsigned char data)//Function to write data to the LCD
{
    LCD_DATA = data;    //Send data on LCD data bus
    rs = 1;                             //RS = 1 since data to LCD
    rw = 0;                             //RW = 0 since writing to LCD
    en = 1;                             //Generate High to low pulse on EN
        msdelay(15);
    en = 0;
}
//Function to write string to LCD
void LCD_write_string(static char *str)
{
    int i = 0;
    while (str[i] != 0)
    {
        LCD_data(str[i]);       // sending data on LCD byte by byte
        msdelay(15);
        i++;
    }
}
```

32116

PUNE INSTITUTE OF COMPUTER TECHNOLOGY, PUNE – 411043

Department of Electronics & Telecommunication Engineering

CLASS : T.E. E &TC Engg. **COURSE: MC**

EXPT. NO.: 8 **DATE:** 20|10|23

TITLE: Generation of PWM Signal

PROBLEM STATEMENT:

Interface DC Motor (9V – 12V; 200 – 300 rpm) to PIC microcontroller. Write a program in Embedded C to control the speed of DC motor using on-chip PWM Module of PIC Microcontroller.

OBJECTIVE:

a. To understand the working of PWM.
b. To study the on-chip CCP Module (PWM section) of PIC Microcontroller.
c. To interface DC motor, via Driver IC L293D, to PIC Microcontroller
d. Apply the PWM signal to control the speed of DC Motor

S/W PACKAGES AND H/W USED:

MPLAB IDE, C18 Compiler, Universal Board of PIC18F and DC Motor

REFERENCES:

a. Mazidi, PIC microcontroller & embedded system 3rd Edition, Pearson
b. Datasheet of PIC18F4550 / PIC18F4520 / PIC18F458, www.microchip.com
c. Datasheet of L293D, www.ti.com

THEORY:

PWM Technique:

1. CCP Module of PIC

Depending upon the device selected there are 1 – 4 CCP (Capture/Compare/PWM) modules in PIC microcontroller. Each module contains a 16-bit register, which can operate as a 16-bit Capture register, a 16-bit Compare register or a PWM Master/Slave Duty Cycle register.

Each Capture/Compare/PWM module is associated with a control register (generically, CCPxCON) and a data register (CCPRx). The data register, in turn, is comprised of two 8-bit registers: CCPRxL (low byte) and CCPRxH (high byte). All registers are both readable and writable.

The CCP modules utilize Timers 1, 2 or 3, depending on the mode selected. Timer1 and Timer3 are available to modules in Capture or Compare modes, while Timer2 is available for modules in PWM mode. The pin assignment for CCP2 (Capture input, Compare and PWM output) can change, based on device configuration. The CCP2MX Configuration bit determines which pin

CCP2 is multiplexed to. By default, it is assigned to RC1 (CCP2MX = 1). If the Configuration bit is cleared, CCP2 is multiplexed with RB3. Changing the pin assignment of CCP2 does not automatically change any requirements for configuring the port pin. Users must always verify that the appropriate TRIS register is configured correctly for CCP2 operation, regardless of where it is located.

1.1 Capture Mode Applications

- Event arrival time recording
- Period measurement
- Pulse-width measurement
- Interrupt generation
- Event counting
- Time reference
- Duty cycle measurement

1.2 Compare Mode Applications

- Generate …
  - Single pulse
  - Train of pulses
  - Periodic waveform
- Start ADC conversion
- Time reference

2. PWM Mode of CCP Module

In Pulse-Width Modulation (PWM) mode, the CCPx pin produces up to a 10-bit resolution PWM output. Since the CCP2 pin is multiplexed with a PORTB or PORTC data latch, the appropriate TRIS bit must be cleared to make the CCP2 pin an output. A PWM output has a time base (period) and a time that the output stays high (duty cycle). The frequency of the PWM is the inverse of the period (1/period).

The simplified block diagram of the CCP1 configured to generate PWM is shown in Figure 1.1, with example waveforms in Figure 1.2. The block diagram for CCP2 is similar to CCP1. It can be seen that Figure 1.1 contains CCPR1L, Timer 2, comparator and the PR2 register working together. Although not shown, Timer 2 is still driven from the on-chip oscillator, through its own

prescaler. The comparator output drives an R–S flip-flop. When the value in PR2 equals Timer 2, then the comparator clears the timer and sets the flip-flop, whose output goes high. This is seen in Figure 1.2. This action sets the PWM period.

Having established the PWM period, let us consider how the pulse width is determined. A second Compare register arrangement is introduced to do this. This is made up of the CCPR1H register, plus a second comparator. As the logic of the diagram shows, every time this comparator finds equal input values, it resets the output flip-flop, clearing the output to zero. It is this comparator that determines the pulse width. Again, this is shown in Figure 1.2. To change the pulse width, the programmer writes to the CCPR1L register, which acts as a buffer. Its value is transferred to CCPR1H only when a PWM cycle is complete, to avoid output errors in the process. The block diagram is made more complex because three of the registers are 'stretched', to make them potentially 10-bit instead of 8-bit. This increases the resolution. CCPR1L uses two bits of the CCP1CON register. CCPR1H is extended with an internal 2-bit latch, while the extension to Timer 2 is as described in Note 1 of Figure 1.1. Because of these two extra bits, in its 10-bit version it is effectively clocked direct from the internal oscillator signal, undivided. If the prescaler is used, then it acts on this frequency, not the usual Fosc/4. Notice, however, that the PR2 register remains at eight bits. This means that the PWM period has only an 8-bit equivalent resolution.

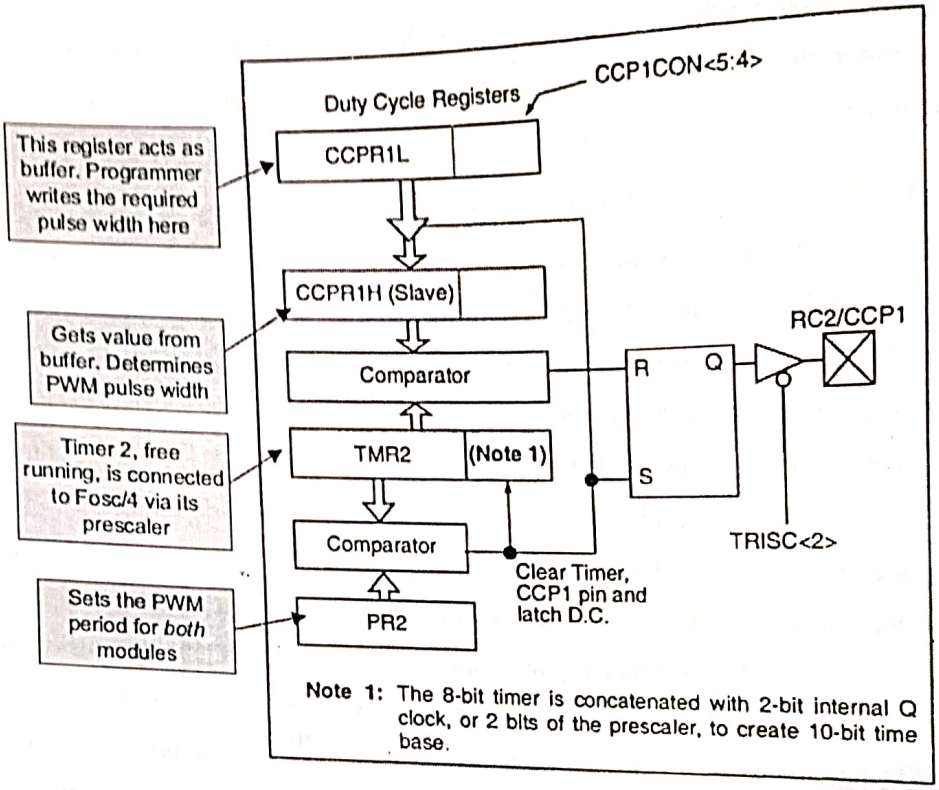


Fig1: PWM Block Diagram

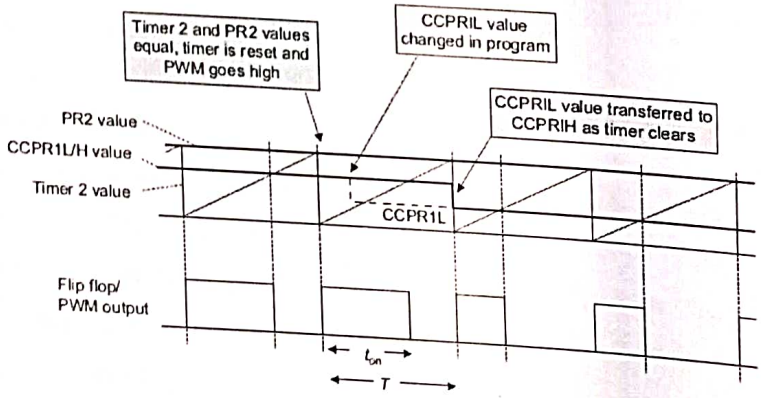


Fig 2: PWM output waveform

2.1 PWM Period and Duty Cycle

The PWM period T is determined by the interaction of the PR2 register and the eight bits of Timer 2. It may be calculated as follows:

PWM Period = T_{pwm} = (PR2 + 1) x (Timer 2 input Clock)

PWM Period = T_{pwm} = (PR2 + 1) x (4 * Tosc * TMR2 Prescaler value)

Therefore to find the PR2 for desired PWM Period

Above equation can be represented in terms of frequency as below.

Where,

Timer 2 input Clock = 4 * Tosc * TMR2 Prescaler value

Fosc = Input Crystal Oscillator Frequency

Tosc = input clock period = 1/Fosc

Fpwm = PWM frequency

Tpwm = PWM Period = 1/Fpwm

The PWM pulse width **Ton** is determined by the interaction of the extended CCPR1H Register (all 10 bits of it) and the extended (10-bit) Timer 2. It may be calculated as follows:

Ton = (Pulse width register) x (PWM Timer input clock period)

Where, 'PWM timer input clock period' is the period of the clock input to the extended Timer 2 and 'pulse width register' is the value in the extended CCPR1L register.

Hence,

Ton = (CCPR1L: CCP1CON<5:4>) x (Tosc * TMR2 Prescaler value)

Note that there is not here a factor of four with the Tosc term, as we are using 10-bits of Timer 2. Rearranging the above equation, to find value of pulse width register **(CCPR1L: CCP1CON<5:4>).**

Above equation can be represented in terms of frequency and percentage Duty cycle (%DCpwm) as below.

Note: Choose $TMR2_{PRE}$ to ensure that PR2 is in the range of 0 to 255 for the desired PWM frequency and CCPR1L:CCP1CON<5:4> is in the range of 0 to 1023 for the desired PWM duty cycle.

2.2 Pin Description

Signal	Pin No.	Symbol
$CCP2^{(1)}$	16	$RC1/T1OSCI/CCP2^{(1)}$
CCP1	17	RC2/CCP1/P1A
$CCP2^{(1)}$	36	$RB3/AN9/CCP2^{(1)}$
Note 1: RB3 is the alternate pin for CCP2 multiplexing.		

2.3 PWM Register Map:

SFR	Description	Access	Reset Value	Address
CCP1CON	Standard CCP1 Control Reg.	Read/Write	0x00	0xFBD
CCPR1L	CCP1 Register low byte	Read/Write	unknown	0xFBE
CCPR1H	CCP1 Register high byte	No access	unknown	0xFBF
CCP2CON	Standard CCP2 Control Reg.	Read/Write	0x00	0xFBA
CCPR2L	CCP2 Register high byte	Read/Write	unknown	0xFBB
CCPR2H	CCP2 Register high byte	No access	unknown	0XFBC
T2CON	Timer 2 Control Register	Read/Write	0x00	0xFCA
TMR2	Timer 2 Register	Read/Write	0x00	0xFCC
PR2	Timer 2 Period register	Read/Write	0xFF	0xFCB

2.4 PWM Mode Register Description

2.4.1 Standard CPPx Control Register (CCPxCON – CCP1CON/CCP2CON)

D7	D6	D5	D4	D3	D2	D1	D0
---	----	DC1B1	DC1B0	CCP1M3	CCP1M2	CCP1M1	CCP1M0

Bit No.	Control Bit	Description
Bit 7 – 6	Unimplemented	Read as '0'
Bit 5 – 4	DCxB1:DCxB0	**PWM Duty Cycle Bit 1 and Bit 0 for CCPx Module** Unused in Capture mode & Compare mode: **PWM mode:** These bits are the two LSbs (bit 1 and bit 0) of the 10-bit PWM duty cycle. The eight MSbs of the duty cycle are found in CCPR1L.
Bit 3 – 0	CCPxM3:CCPxM0	**CCPx Module Mode Select bits** 0000 = CCP disabled (resets CCPx module) 0001 = Reserved 0010 = Compare mode: toggle output on match (CCPxIF bit is set) 0011 = Reserved 0100 = Capture mode: every falling edge 0101 = Capture mode: every rising edge 0110 = Capture mode: every 4th rising edge 0111 = Capture mode: every 16th rising edge 1000 = Compare mode: initialize CCPx pin low; on compare match, force CCPx pin high (CCPxIF bit is set) 1001 = Compare mode: initialize CCPx pin high; on compare match, force CCPx pin low (CCPxIF bit is set) 1010 = Compare mode: generate software interrupt on compare match (CCPxIF bit is set, CCPx pin reflects I/O state) 1011 = Compare mode: trigger special event, reset timer, start A/D conversion on CCP2 match (CCPxIF bit is set)

		11xx = PWM mode

NOTE: Only highlighted bits are applicable to this exercise.

2.4.2 Timer 2 Control Register

D7	D6	D5	D4	D3	D2	D1	D0
--	T2OUTPS3	T2OUTPS2	T2OUTPS1	T2OUTPS0	TMR2ON	T2CKPS1	T2CKPS0

Bit No.	Control Bit	Description
Bit 7	Unimplemented	Read as '0'
Bit 6 - 3	T2OUTPS3:T2OUTPS0 :	Timer2 Output Postscaler Select bits (1:1 to 1:16) 0000 = 1:1 Postscaler 0001 = 1:2 Postscaler • • 1111 = 1:16 Postscaler
Bit 2	TMR2ON:	Timer2 On bit 1 = Timer2 is on, 0 = Timer2 is off
Bit 1 - 0	T2CKPS1:T2CKPS0	Timer2 Clock Prescaler Select bits 00 = Prescaler is 1:1 01 = Prescaler is 1:4 1x = Prescaler is 1:16

3. Timer 2 Module

The Timer2 module timer incorporates the following features:

- 8-bit timer and period registers (TMR2 and PR2, respectively)
- Readable and writable (both registers)
- Software programmable prescaler (1:1, 1:4 and 1:16)
- Software programmable Postscaler (1:1 through 1:16)
- Interrupt on TMR2 to PR2 match
- Optional use as the shift clock for the MSSP module

The module is controlled through the T2CON register which enables or disables the timer and configures the prescaler and Postscaler. Timer2 can be shut off by clearing control bit, TMR2ON

(T2CON<2>), to minimize power consumption. A simplified block diagram of the module is shown in Figure 9.3.

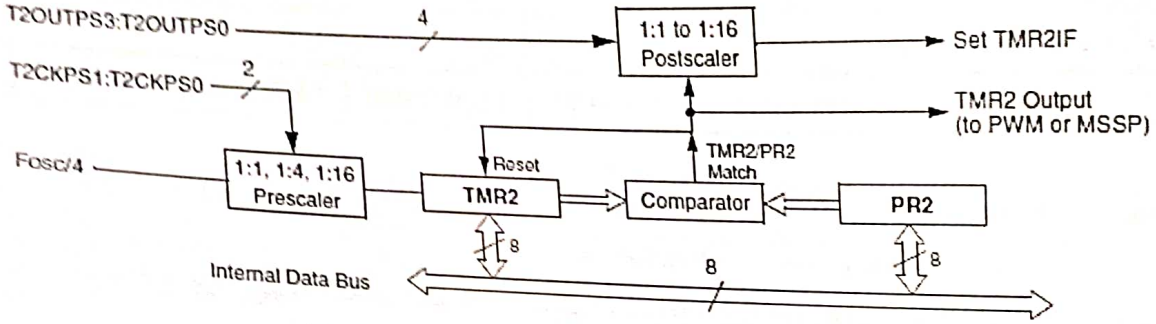


Fig 3: Simplified Block Diagram of Timer2 Module

3.1 Timer2 Operation

In normal operation, TMR2 is incremented from 00h on each clock (FOSC/4). A 2-bit counter/prescaler on the clock input gives direct input, divide-by-4 and divide-by- 16 prescale options. These are selected by the prescaler control bits, T2CKPS1:T2CKPS0 (T2CON<1:0>). The value of TMR2 is compared to that of the period register, PR2, on each clock cycle. When the two values match, the comparator generates a match signal as the timer output. This signal also resets the value of TMR2 to 00h on the next cycle and drives the output counter/ Postscaler.

The TMR2 and PR2 registers are both directly readable and writable. The TMR2 register is cleared on any device Reset, while the PR2 register initializes at FFh. TMR2 is not cleared when T2CON is written. Both the prescaler and postscaler counters are cleared on the following events:

- a write to the TMR2 register
- a write to the T2CON register
- any device Reset (Power-on Reset, MCLR Reset, WDT Reset or Brown-out)

3.2 Timer2 Interrupt

Timer2 also can generate an optional device interrupt. The Timer2 output signal (TMR2 to PR2 match) provides the input for the 4-bit output counter/postscaler. This counter generates the TMR2 match interrupt flag which is latched in TMR2IF (PIR1<1>). The interrupt is enabled by setting the TMR2 Match Interrupt Enable bit, TMR2IE (PIE1<1>). A range of 16 postscale options (from 1:1 through 1:16 inclusive) can be selected with the postscaler control bits, T2OUTPS3:T2OUTPS0 (T2CON<6:3>).

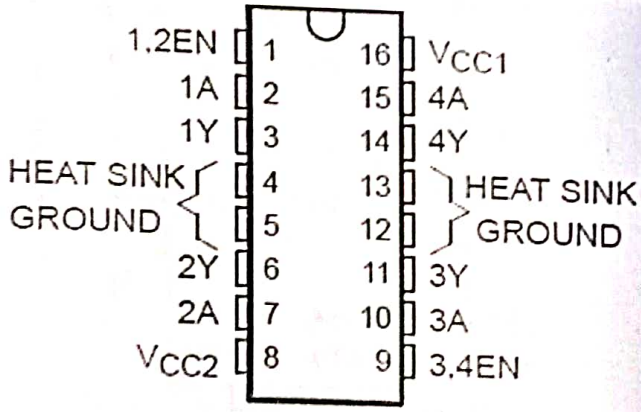
3.3 TMR2 Output

The unscaled output of TMR2 is available primarily to the CCP modules, where it is used as a time base for operations in PWM mode. Timer2 can be optionally used as the shift clock source for the MSSP module operating in SPI mode.

4. Quadruple half H Drivers (L293D)

The L293D is quadruple high-current half-H drivers. The L293D is designed to provide bidirectional drive currents of up to 600-mA at voltages from 4.5 V to 36 V. The device is designed to drive inductive loads such as relays, solenoids, dc and bipolar stepping motors, as well as other high-current/high-voltage loads in positive-supply applications. All inputs are TTL compatible. Each output is a complete totem-pole drive circuit, with a Darlington transistor sink and a pseudo-Darlington source.

Drivers are enabled in pairs, with drivers 1 and 2 enabled by 1,2EN and drivers 3 and 4 enabled by 3,4EN as shown in pin diagram and function table. When an enable input is high, the associated drivers are enabled, and their outputs are active and in phase with their inputs. When the enable input is low, those drivers are disabled, and their outputs are off and in the high-impedance state. With the proper data inputs, each pair of drivers forms a full-H (or bridge) reversible drive suitable for solenoid or motor applications.



FUNCTION TABLE
(each driver)

INPUTS†		OUTPUT
A	EN	Y
H	H	H
L	H	L
X	L	Z

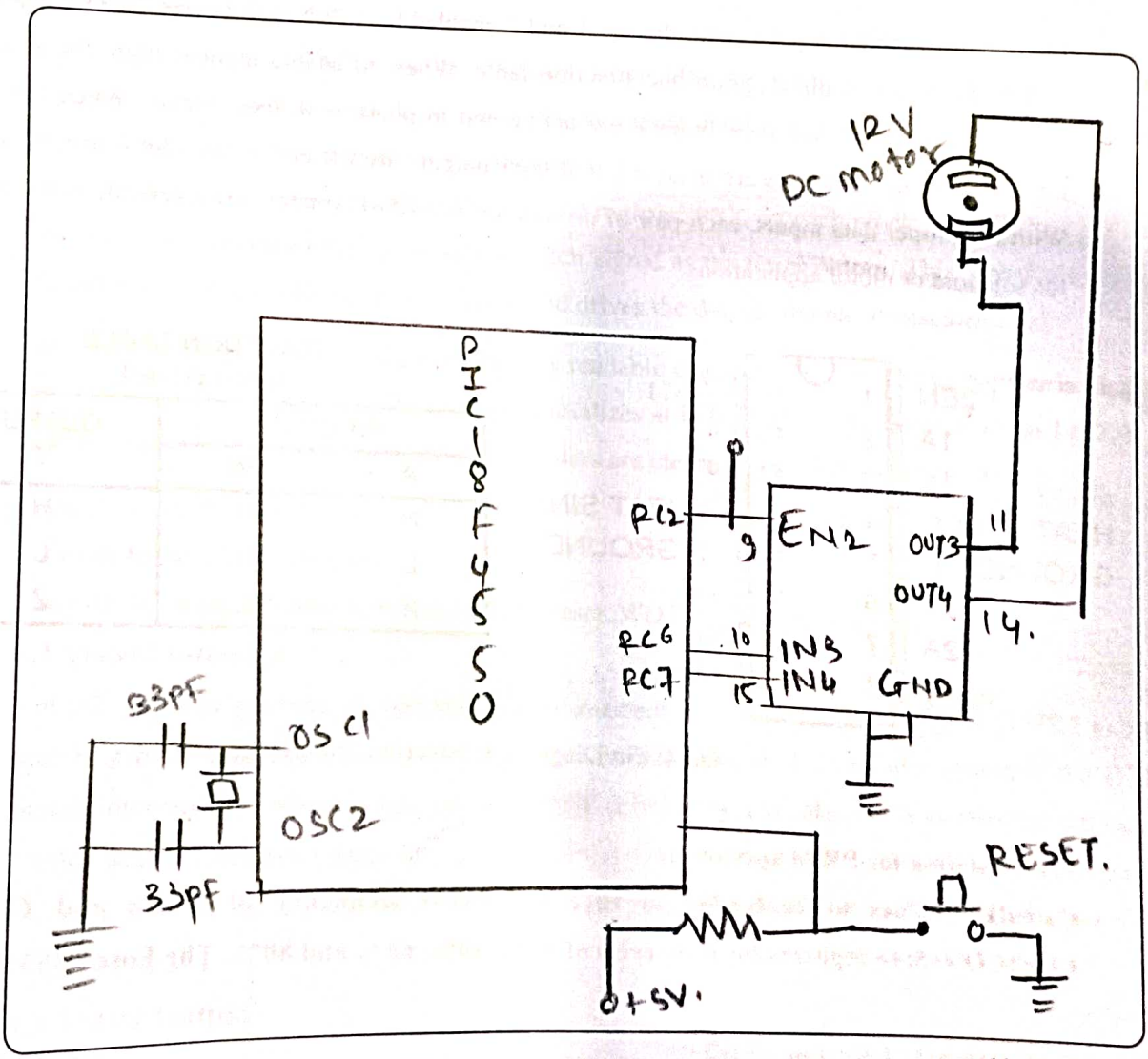
Fig 4: Pin Diagram & Function Table

5. Calculation for PWM operation:

Calculate values to be loaded in PR2 for PWM frequency of 4KHz and CCPR1L: CCP1CON<5:4> registers for duty cycle of 20%, 40%, 60% and 80%. The Fosc= 48MHz.

	Duty Cycle	CCPR1L Value	CCP1CON<5:4> DC1B1:DC1B0
Fosc = 48 MHz PWM = 4 Khz PR2 =	20%		
	40%		
	60%		
	80%		
	100%		

6. Interfacing diagram



7. Algorithm

The following steps should be taken when configuring the CCP module for PWM operation:

1. Set the PWM period by writing to the PR2register.
2. Set the PWM duty cycle by writing to the CCPR1L register and CCP1CON<5:4> bits.
3. Make the CCP1 pin an output by clearing the appropriate TRIS bit.
4. Make the RD5 and RD6 as output to TURN ON the Motor.
5. Set the TMR2 prescale value, then enable Timer2 by writing to T2CON.
6. Configure the CCP1 module for PWM operation.

8. Source code:

9. Result & Conclusion:

In this experiment we studied the interfacing of DC motor with PIC18F4550 with help of L293D motor driver. We varied the duty cycle and observed it's output/ effect on intensity of LED as well as speed of DC motor.

Sen

27/1/23

B17

```
 // program to generate 4 kHz PWM waveform
#include<p18f4520.h>
#include "vector_relocate.h"

void myMsDelay (unsigned int time);

void main()
{
        TRISCbits.TRISC2 = 0 ;              // Set PORTC, 2 as output
    TRISD=0;
        PR2 = 0XBB;

    CCP1CON = 0x0C;                        // Configure CCP1CON as explained above.
        T2CON = 0x07;

        PORTDbits.RD5 = 1 ;  // anticolockwise
        PORTDbits.RD6 = 0 ;
   while(1)
        {

                CCPR1L = 0xBB;  //0b10111011//187
                CCP1CONbits.DC1B0=0;
                CCP1CONbits.DC1B1=0;
                myMsDelay(200);


                CCPR1L = 149;
                CCP1CONbits.DC1B0=0;
                CCP1CONbits.DC1B1=1;
                myMsDelay(10);

                CCPR1L = 112;
                CCP1CONbits.DC1B0=1;
                CCP1CONbits.DC1B1=0;
                myMsDelay(100);

                CCPR1L = 0x4A;
                CCP1CONbits.DC1B0=1;
                CCP1CONbits.DC1B1=1;
                myMsDelay(5);

                CCPR1L = 0x25;
                CCP1CONbits.DC1B0=0;
                CCP1CONbits.DC1B1=1;
                myMsDelay(10);


        }
```

```
}

void myMsDelay (unsigned int time)
{
        unsigned int i, j;
        for (i = 0; i < time; i++)
                for (j = 0; j <665; j++);/*Calibrated for a 1 ms delay in MPLAB*/

}
```

```
//*****************************************************************
//  Program for PWM Generation using PIC18F4550.
//  PWM output          :       RC2
//*****************************************************************
#include <p18f4550.h>
#include "vector_relocate.h"

void myMsDelay (unsigned int time)      // Definition of delay subroutine
{
        unsigned int i, j;
        for (i = 0; i < time; i++)              // Loop for itime
                for (j = 0; j < 710; j++);      // Calibrated for a 1 ms delay in
MPLAB
}

void main()
{
        TRISCbits.TRISC2 = 0 ;  // Set PORTC, RC2 as output (CCP1)
    TRISDbits.TRISD5 = 0 ;      // Set PORTD, RD5 as output (DCM IN2)
        TRISDbits.TRISD6 = 0 ;  // Set PORTD, RD6 as output (DCM IN1)
        PR2 =     187;          // set PWM Frequency 4KHz
    CCP1CON = 0x0C;          // Configure CCP1CON as PWM mode.
        T2CON = 0x07;           //Start timer 2 with prescaler 1:16
        PORTDbits.RD6 = 1;      // Turn ON the Motor
    PORTDbits.RD5 = 0;
  while(1)      // Endless Loop
        {
                // ----------Duty Cycle 80%-----------
                CCP1CONbits.DC1B0 = 0;
                CCP1CONbits.DC1B1 = 0;
                CCPR1L = 0x96;
                myMsDelay(2000);
                // -----------------------------------
                // ----------Duty Cycle 60%-----------
                CCP1CONbits.DC1B0 = 0;
                CCP1CONbits.DC1B1 = 1;
                CCPR1L = 0x70;
                myMsDelay(2000);
                // -----------------------------------
                // ----------Duty Cycle 40%-----------
                CCP1CONbits.DC1B0 = 0;
                CCP1CONbits.DC1B1 = 0;
                CCPR1L = 0x4B;
                myMsDelay(2000);
                // -----------------------------------
                // ----------Duty Cycle 20%-----------
                CCP1CONbits.DC1B0 = 0;
                CCP1CONbits.DC1B1 = 1;
                CCPR1L = 0x25;
                myMsDelay(2000);
```

CLASS : T.E. E &TC **SUBJECT: MC**

EXPT. NO.: 09 **DATE:** 27|0|23

TITLE: On-chip ADC Programming

PROBLEM STATEMENT:

Interface analog voltage 0 – 5v to PIC microcontroller and Write a program in Embedded C to convert analog voltage into digital using on-chip ADC Module. Display the Digital result in BCD / Voltage form on 16x2 LCD display.

OBJECTIVE:

a. To understand the working of A/D converter.

b. To study the on-chip ADC section of PIC Microcontroller.

c. To interface analog input to PIC Microcontroller

S/W PACKAGES AND H/W USED:

MPLAB IDE, C18 Compiler, PIC KIT3, Universal Development board

REFERENCES:

a. Mazidi, PIC microcontroller & embedded system 3rd Edition ,Pearson

b. Datasheet of PIC18F4550 / PIC18F4520 / PIC18F458, www.microchip.com

THEORY:

1. ADC Devices:

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). 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). 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.

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.

Some of the major characteristics of ADC are a) resolution b) conversion time c) reference voltage (Vref).

1.1 Resolution

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. This is

shown in Table 1. Although the resolution for ADC chip is designed at the time of its design and cannot b changed, we can control the step size with the help of reference voltage (Vref).

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

Table 1: Resolution verses step size for ADC.

1.2 Conversion time

Conversion time is defined as the time it takes the ADC to convert the analog input to digital (binary) number. The conversion time is dictated by the clock source connected to the ADC in addition to the method used for data conversion and technology used in the fabrication of the ADC chip.

In addition to conversion time, acquisition time is another major factor in judging an ADC. *Acquisition time* is defined as the time it takes to sample the analog voltage using sample and hold circuit. Sampled analog input is applied to actual conversion unit of ADC.

1.3 Reference Voltage (Vref)

The reference voltage along with resolution of the ADC chip, gives us the step size.

Step size = Vref / 2^n where n= no. of bits

For example if the analog input range needs to be 0 to 3 Volts, Vref is connected to 3 Volts. That gives 3V/1024 = 2.92 mV step size for 10-bit ADC.

Vref (V)	Vin (V)	Step Size (mV)
5.00	0 to 5	5/1024 = 4.88
2.56	0 to 2.56	2.56/1024 = 2.5
1.024	0 to 1.024	1.024/1024 = 1

Table 2: Relation between Vref, Vin and step size

2. 10-Bit Analog-To-Digital Converter (A/D) Module of PIC

The Analog-to-Digital (A/D) converter module has 13 for the 40/44-pin devices. This module allows conversion of an analog input signal to a corresponding 10-bit digital number.

2.2 Pin Description

Signal	Pin No.	Symbol
Channel 0 (AN0)	2	RA0/AN0
Channel 1 (AN1)	3	RA1/AN1
Channel 2 (AN2)	4	RA2/AN2/Vref-/CVref
Channel 3 (AN3)	5	RA3/AN3/Vref+
Channel 4 (AN4)	7	RA5/AN4/SS/HLVDIN/C2OUT
Channel 5 (AN5)	8	RE0/AN5/CK1SPP
Channel 6 (AN6)	9	RE1/AN6/CK2SPP
Channel 7 (AN7)	10	RE2/AN7/OESPP
Channel 8 (AN8)	35	RB2/AN8/INT2/VMO
Channel 9 (AN9)	36	RB3/AN9/VPO
Channel 10 (AN10)	34	RB1/AN10/INT1/SCK/SCL
Channel 11 (AN11)	37	RB4/AN11/KBIO/CSSPP
Channel 12 (AN12)	33	RB0/AN12/INT0/FLT0/SDI/SDA

2.3 ADC Register Map:

SFR	Description	Access	Reset Value	Address
ADCON0	A/D Control Register 0	Read/Write	0x00	0xFC2
ADCON1	A/D Control Register 1	Read/Write	0x00	0xFC1
ADCON2	A/D Control Register 2	Read/Write	0x00	0xFC0
ADRESH	A/D Result High Register	Read	unknown	0xFC4
ADRESL	A/D Result Low Register	Read	unknown	0xFC3

2.4 ADC Register Description

2.4.1 A/D Control Register 0 (ADCON0)

D7	D6	D5	D4	D3	D2	D1	D0
–	–	CHS3	CHS2	CHS1	CHS0	GO/*DO	ADON

Bit No.	Control Bit	Description	
Bit 7 - 6	Unimplemented	Read as '0'	
Bit 5 - 2	CHS3:CHS0	Analog Channel Select bits	
		0000 = Channel 0 (AN0)	0001 = Channel 1 (AN1)
		0010 = Channel 2 (AN2)	0011 = Channel 3 (AN3)
		0100 = Channel 4 (AN4)	0101 = Channel 5 (AN5)
		0110 = Channel 6 (AN6)	0111 = Channel 7 (AN7)
		1000 = Channel 8 (AN8)	1001 = Channel 9 (AN9)
		1010 = Channel 10 (AN10)	1011 = Channel 11 (AN11)
		1100 = Channel 12 (AN12)	1101 = Unimplemented
		1110 = Unimplemented	1111 = Unimplemented
Bit 1	GO/DONE	A/D Conversion Status bit 1 = A/D conversion in progress; 0 = A/D Idle	
Bit 0	ADON	A/D On bit **1 = A/D converter module is enabled** 0 = A/D converter module is disabled	

NOTE: Only highlighted bits are applicable to this exercise.

2.4.2 A/D Control Register 1 (ADCON1)

D7	D6	D5	D4	D3	D2	D1	D0
--	--	VCFG1	VCFG0	PCFG3	PCFG2	PCFG1	PCFG0

Bit No.	Control Bit	Description
Bit 7 - 6	Unimplemented	Read as '0'
Bit 5	VCFG0	Voltage Reference Configuration bit (VREF- source) 1 = VREF- (AN2); **0 = VSS**
Bit 4	VCFG0	Voltage Reference Configuration bit (VREF+ source) 1 = VREF+ (AN3); **0 = VDD**
bit 3-0	PCFG3:PCFG0	A/D Port Configuration Control bits shown in table 8.3 **1110 : Analog channel AN0**

NOTE: Only highlighted bits are applicable to this exercise.

PCFG3: PCFG0	AN12	AN11	AN10	AN9	AN8	AN7(2)	AN6(2)	AN5(2)	AN4	AN3	AN2	AN1	AN0
0000(1)	A	A	A	A	A	A	A	A	A	A	A	A	A
0001	A	A	A	A	A	A	A	A	A	A	A	A	A
0010	A	A	A	A	A	A	A	A	A	A	A	A	A
0011	D	A	A	A	A	A	A	A	A	A	A	A	A
0100	D	D	A	A	A	A	A	A	A	A	A	A	A
0101	D	D	D	A	A	A	A	A	A	A	A	A	A
0110	D	D	D	D	A	A	A	A	A	A	A	A	A
0111(1)	D	D	D	D	D	A	A	A	A	A	A	A	A
1000	D	D	D	D	D	D	A	A	A	A	A	A	A
1001	D	D	D	D	D	D	D	A	A	A	A	A	A
1010	D	D	D	D	D	D	D	D	A	A	A	A	A
1011	D	D	D	D	D	D	D	D	D	A	A	A	A
1100	D	D	D	D	D	D	D	D	D	D	A	A	A
1101	D	D	D	D	D	D	D	D	D	D	D	A	A
1110	D	D	D	D	D	D	D	D	D	D	D	D	A
1111	D	D	D	D	D	D	D	D	D	D	D	D	D

A = Analog input D = Digital I/O

Table 3: PCFG3:PCFG0: A/D Port Configuration Control bits:

2.4.3 A/D Control Register 2 (ADCON2)

D7	D6	D5	D4	D3	D2	D1	D0
ADFM	--	ACQT2	ACQT1	ACQT0	ADCS2	ADCS1	ADCS0

Bit No.	Control Bit	Description
Bit 7	ADFM	**A/D Result Format Select bit** **1 = Right justified;** 0 = Left justified
Bit 6	Un	**Unimplemented** Read as '0'
Bit 5–3	ACQT2:ACQT0	**A/D Acquisition Time Select bits** 000=0 TAD; **001=2 TAD**; 010=4 TAD; 011 =6 TAD 100=8 TAD; 101=12 TAD; 110=16 TAD; 111 =20 TAD
bit 2–0	ADCS2:ADCS0	**A/D Conversion Clock Select bits** 000=FOSC/2; 001=FOSC/8; 010=FOSC/32; 011=FRC* 100=FOSC/4; 101=FOSC/16; **110=FOSC/64**; 111=FRC* FRC* - clock derived from A/D RC oscillator

NOTE: Only highlighted bits are applicable to this exercise.

2.4.3.1 A/D Result Format Select bit (ADFM):

We need right justified data. Therefore ADFM = '1'.

If you write 1 then result is **right justified** as shown below;

15	14	13	12	11	10	9	8	ADRESH
—	—	—	—	—	—			
7	6	5	4	3	2	1	0	ADRESL

If you write 0 then result is **left justified** as shown below;

15	14	13	12	11	10	9	8	ADRESH
7	6	5	4	3	2	1	0	ADRESL
		—	—	—	—	—	—	

2.4.3.2 A/D Conversion Clock Select bits (ADCS2:ADCS0)

The A/D conversion time per bit is defined as TAD. The A/D conversion requires 11 TAD per 10-bit conversion. The source of the A/D conversion clock is software selectable. There are seven possible options for TAD as shown in Table 4.For correct A/D conversions, the A/D conversion clock (TAD) must be as short as possible but greater than the minimum TAD. Table 8.4 shows the resultant TAD times derived from the device operating frequencies and the A/D clock source selected.

Minimum TAD = 0.7 us (to be maintained as per datasheet)

Therefore for 10bit ADC

Typical conversion time = 11 TAD = 11 * 0.7 us = 7.7 us

AD Clock Source (TAD)		**Maximum Device Frequency**
Operation	**ADCS2:ADCS0**	**PIC18FXXXX**
2 TOSC	000	2.86 MHz
4 TOSC	100	5.71 MHz
8 TOSC	001	11.43 MHz
16 TOSC	101	22.86 MHz
32 TOSC	010	40.00 MHz
64 TOSC	110	40.00MHz
RC(3)	x11	1.00 MHz

Table 4: TAD vs. DEVICE OPERATING FREQUENCIES

24.3.3 A/D Acquisition Time Select bits (ACQT2:ACQT0)

Figure 1 shows that the acquisition time (TACQ) is dependent on amplifier settling time, capacitor charging time and temperature coefficient.

TACQ = TAMP + TC + TCOFF

where,

TAMP : Amplifier Settling Time = 0.2 us (Typical value)

TC : Holding Capacitor Charging Time

= -(CHOLD)(RIC + RSS + RS) ln(1/2048) µs

= -(25 pF) (1 kΩ + 2 kΩ + 2.5 kΩ) ln(0.0004883) µs (Typical value)

= 1.05 µs

TCOFF : Temperature Coefficient = (Temp – 25°C) (0.02 µs/°C)

= (85°C – 25°C)(0.02 µs/°C) (Typical temp)

= 1.2 µs

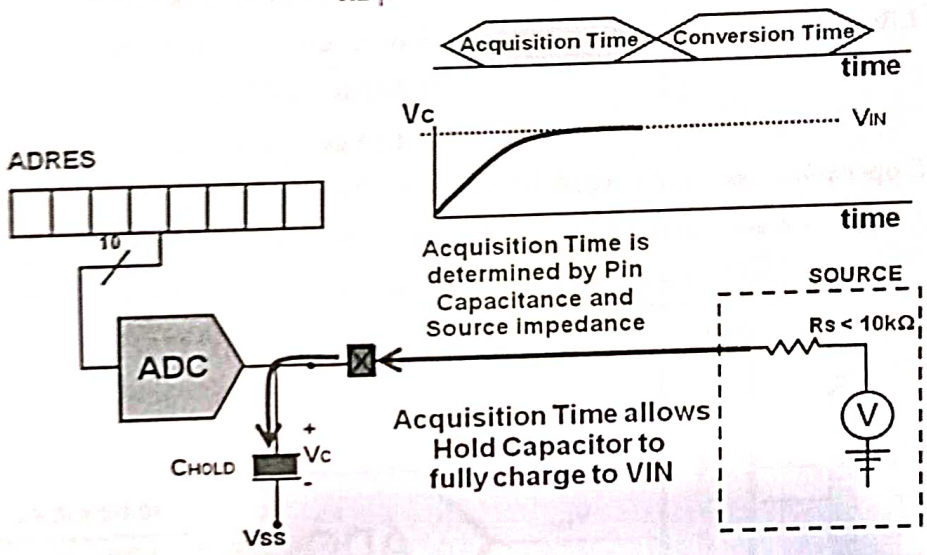


Figure 1: Acquisition time of ADC module

Therefore minimum **calculated acquisition time** is

TACQ = 0.2 µs + 1.05 µs + 1.2 µs = 2.45 µs

Derived A/D acquisition time = n x TAD

where n is depends on A/D Acquisition Time Select bits (ACQT2:ACQT0).

Select A/D Acquisition Time Select bits (ACQT2:ACQT0) such that

Derived A/D acquisition time ≥ Calculated acquisition time TACQ

2.5 Calculations

a. For Fosc = 48 MHz

Tosc = 1/48MHz = 20.83ns

Selected ADC clock source **ADCS2:ADCS0 = 110** *i.e Fosc/64*

TAD = 64 x Tosc = 64 x 20.83ns

TAD = 1.33 us >> minimum TAD (0.7 us)

Conversion Time = 11 x TAD = 11 x 1.33 us

= 1.47 us

Selected A/D Acquisition Time Select bits **ACQT2:ACQT0 = 001** *i.e n= 2*

Derived A/D acquisition time = n x TAD = 2 x TAD

= 2 x 1.33 us

= 2.66 us

Derived A/D acquisition time >> calculated acquisition time (2.4 us)

Effective conversion time = acquisition time + conversion time

= 2.66us + 1.47 us

= 4.13 us

2.6 ADC operation

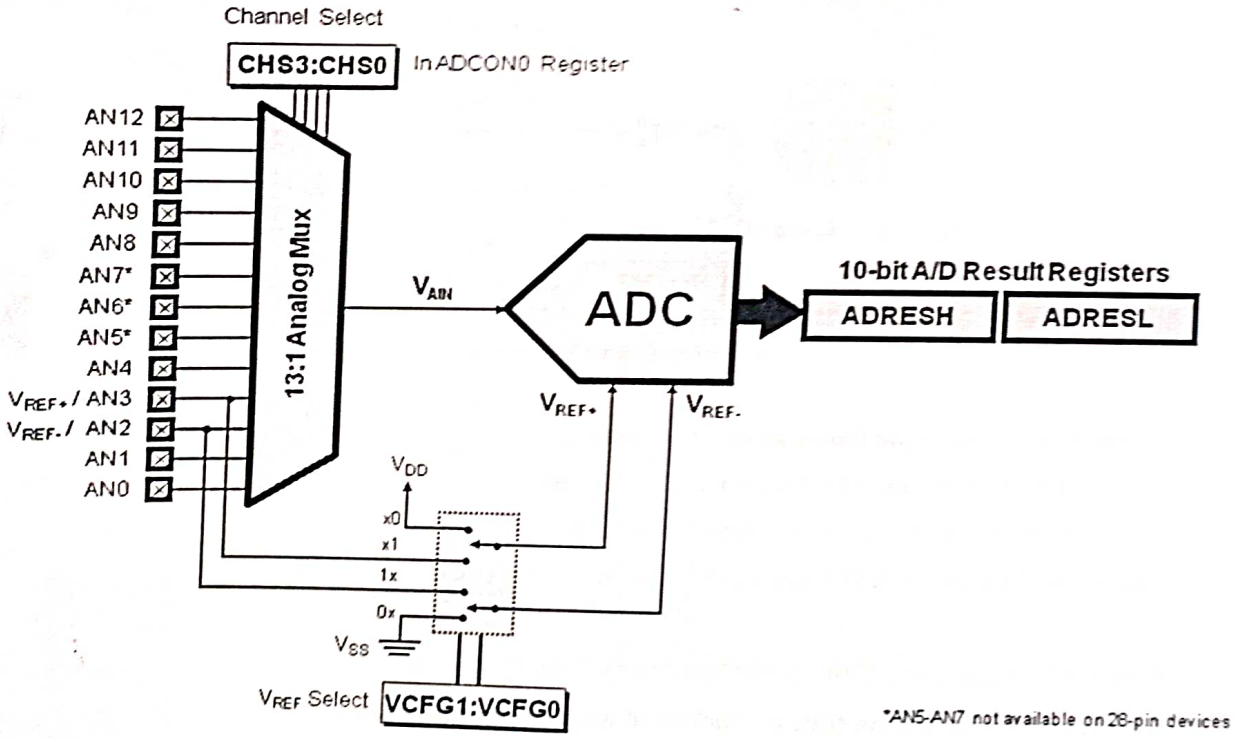


Figure 2: Block Diagram of on-chip ADC Module

The following steps should be followed to perform an A/D conversion:

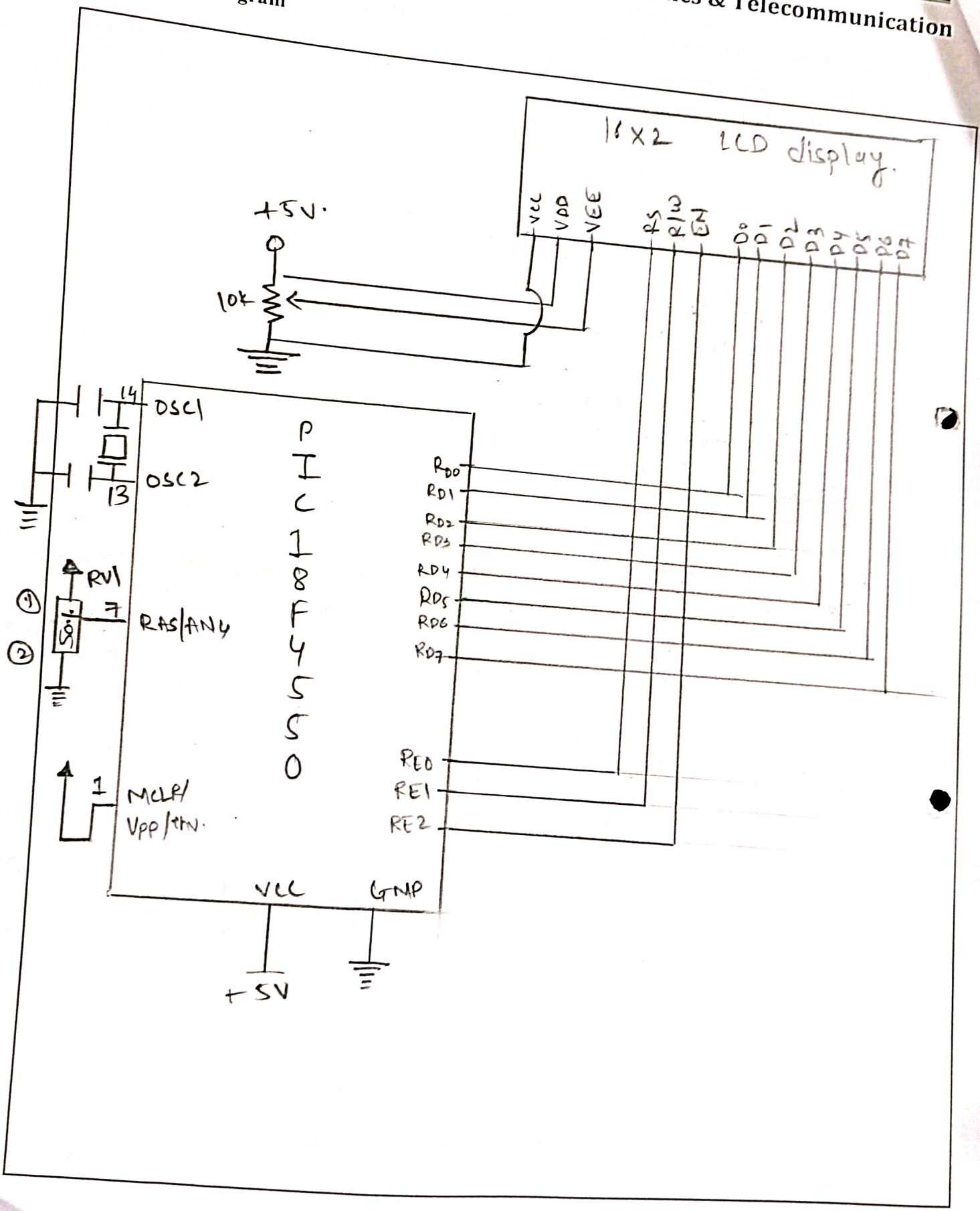
1. Configure the A/D module:

- Configure analog pins, voltage reference and digital I/O (ADCON1)
- Select A/D input channel (ADCON0)
- Select A/D acquisition time (ADCON2)
- Select A/D conversion clock (ADCON2)
- Turn on A/D module (ADCON0)

2. Configure A/D interrupt (if desired):
   - Clear ADIF bit
   - Set ADIE bit
   - Set GIE bit
3. Wait the required acquisition time (if required).
4. Start conversion:
   - Set GO/DONE bit (ADCON0 register)
5. Wait for A/D conversion to complete, by either:
   - Polling for the GO/DONE bit to be cleared

   OR
   - Waiting for the A/D interrupt
6. Read A/D Result registers (ADRESH:ADRESL); clear bit ADIF, if required.
7. For next conversion, go to step 1 or step 2, as required. The A/D conversion time per bit is defined as TAD. A minimum wait of 3 TAD is required before the next acquisition starts.

5. Interfacing diagram



6. Source code (Attach Print out)

7. References:

a. Mazidi, PIC microcontroller & embedded system 3rd Edition ,Pearson
b. Datasheet of PIC18F4550 / PIC18F4520 / PIC18F458,
www.microchip.com
c. ExplorePIC Board Manual

8. Result:

Sr.No.	Vin(Volts)	Result on ADC
1.	0	0
2.	1	205
3.	2	410
4.	3	615
5.	5	1023.

Conclusion:

We studied about working of ADC along with on-chip section on the PIC18F4550 interfacing of analog voltage was conducted to observe different voltage values on the LCD display, using the on-chip memory.

Sem
27/11/24

```
#include<p18f4550.h>
#include"vector_relocate.h"

#define LCD_DATA    PORTD                   //LCD data port
port                                                        //LCD signal
#define en          PORTEbits.RE2       // enable signal
#define rw          PORTEbits.RE1       // read/write signal
#define rs          PORTEbits.RE0      // register select signal

void LCD_cmd(unsigned char cmd);
void myMsDelay (unsigned int time)
{
        unsigned int i, j;
        for (i = 0; i < time; i++)
                for (j = 0; j < 665; j++);
}

void init_LCD(void)
{
    LCD_cmd(0x38);      // initialization of 16X2 LCD in 8bit mode
    myMsDelay(15);

    LCD_cmd(0x01);      // clear LCD
    myMsDelay(15);

    LCD_cmd(0x0E);      // cursor off
    myMsDelay(15);

    LCD_cmd(0x80);      // ---8 go to first line and --0 is for 0th position
    myMsDelay(15);

          // ---8 go to first line and --0 is for 0th position

}
//Function to pass command to the LCD
void LCD_cmd(unsigned char cmd)
{
    LCD_DATA = cmd;
    rs = 0;
    rw = 0;
    en = 1;
    myMsDelay(15);
    en = 0;
    myMsDelay(15);

}

//Function to write data to the LCD
void LCD_write(unsigned char data)
```

```
{
    LCD_DATA = data;
    rs = 1;
    rw = 0;
    en = 1;
    myMsDelay(15);
    en = 0;
    myMsDelay(15);

}

void main(void)
{
 unsigned int val[4],ADC_Result=0,var;
unsigned char i,str[]="Result:";


 TRISD = 0x00;          //Configuring PORTD as output

TRISE=0;
TRISA=0xFF;
init_LCD();
// ADC Initialization
 ADCON1=0x0A;   // Reference as VDD & VSS, AN0 set as analog pins
 ADCON2=0b10010110; // Result is right Justified
                                          //Acquisition Time 4TAD
                                          //ADC Clk FOSC/64
 ADCON0=0X09; //Turn ON ADC module

LCD_cmd(0x80);
for(i=0;str[i]!='\0';i++)
{
LCD_write(str[i]);
myMsDelay(200);
}
 while(1)
 {
  ADCON0bits.GO=1;
  while(ADCON0bits.GO==1);
 var=((unsigned int)ADRESH) << 8;
  ADC_Result=var+ADRESL;


for(i=0;i<4;i++)
{
val[i]=ADC_Result%0x0A;
val[i]=val[i]+0x30;
ADC_Result=ADC_Result/0x0A;
}
```

```
LCD_cmd(0x87);
LCD_write(val[3]);
LCD_write(val[2]);
LCD_write(val[1]);
LCD_write(val[0]);



//myMsDelay(500);
 }
}
```