DYP DPU

Dr. D. Y. Patil Institute of Technology, Pimpri, Pune

Department of E&TC Engineering

Microcontroller – TE (2019 Course) (Semester I)

Course: Microcontroller Lab Manual

Academic year 2025-2026

Department of Electronics & Telecommunication Engineering

T.E. – (Semester I)

Course: Microcontrollers

LAB MANUAL

INDEX

Department of Electronics & Telecommunication Engineering

Course – Microcontrollers
Academic Year 2025-26
Semester 1

EXPERIMENT NO. - 1

TITLE	: Simple programs on Memory transfer
AIM/ OBJECTIVE	: Embedded C Programs on Memory Transfer
HARDWARE/SOFTWARE USED	: Keil uvision3

Theory:

Brief Introduction of Microcontroller:

Microcontrollers are essentially microprocessors (CPU) with on chip program and data memory along with ports and data memory along with ports, timers, serial communication features, etc., all in a single –packaged IC. This integration makes the microcontrollers power efficient, compact and more reliable. MCS-48 was the first microcontroller designed by Intel, which was extensively, used in IBM PC keyboards. MCS- 51 was the next microcontroller designed by Intel and made available by 1980.

The MCS-51 family of microcontrollers was designed around Harvard architecture, which treats program memory and data memory separately. Presently, many manufacturers offer their own unique microcontrollers of 8-, 16- or 32 bit capabilities. These are available in various types of packages, like TQFP or dual-in line Package (DIP), etc. Most microcontrollers offer some protection against unwanted software piracy. Power management also forms an important feature of microcontrollers, as; in general, these are powered from battery sources.

Numbering Convention of MCS-51 Microcontrollers

8	X	Y	Z

X

0	PROM
7	EPROM
9	FLASH

Y

3	No on-chip program memory
5	On chip program memory

Z

	Program Memory	Data memory	Timer/counter
1	4k	128 bytes	2
2	8k	256 bytes	3
4	16k	256 bytes	5

8051 → X=0 (PROM), Y=5(on chip program memory) and Z=1 (Program memory =4K, data memory=128 bytes and timer/counter =2)

8751→ X=7 (EPROM), Y=5(on chip program memory) and Z=1 (Program memory =4K, data memory=128 bytes and timer/counter =2)

AT89C51-12PC → AT for Atmel, 9 before C indicates FLASH type ROM; C before the 51 stands for CMOS, which has a low power consumption, ; 5 indicate on-chip program memory, 1 indicates program memory =4K, data memory=128 bytes and timer/counter =2 ; "12" indicates 12 MHz ; "P" is for plastic DIP package ; "C" is for commercial.

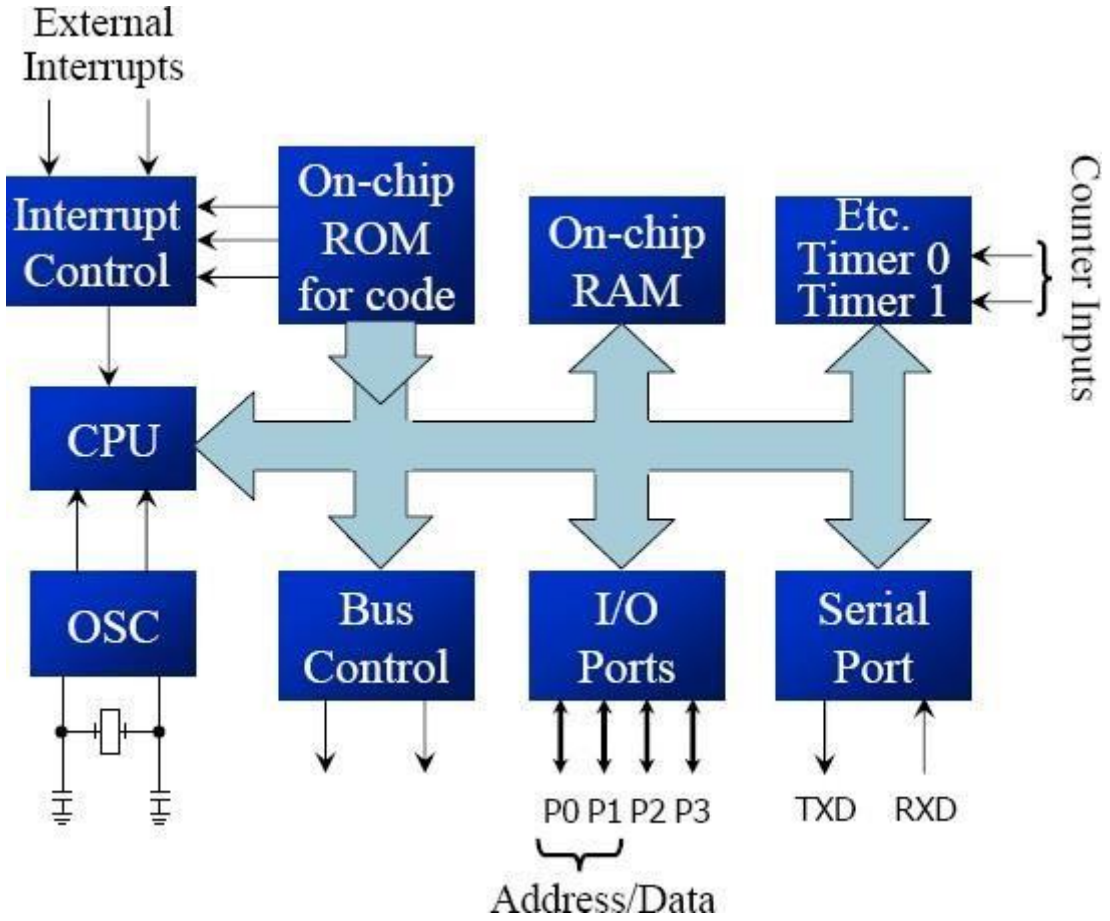
Microcontroller

The fixed amount of on-chip ROM, RAM and number of I/O ports makes them ideal for many applications in which cost and space are critical In many applications, the space it takes, the power it consumes, and the price per unit are much more critical considerations than the computing power.

- 8-bit microcontrollers
  - Motorola's 6811
  - Intel's 8051
  - Zilog's Z8
  - Microchip's PIC

- Intel introduced 8051, referred as MCS-51, in 1981
  - The 8051 is an 8-bit processor
  - The CPU can work on only 8 bits of data at a time
  - The 8051 had 128 bytes of RAM
  - 4K bytes of on-chip ROM
  - Two timers
  - One serial port
  - Four I/O ports, each 8 bits wide
  - 6 interrupt sources

The 8051 became widely popular after allowing other manufactures to make and market any flavor of the 8051, but remaining code-compatible.

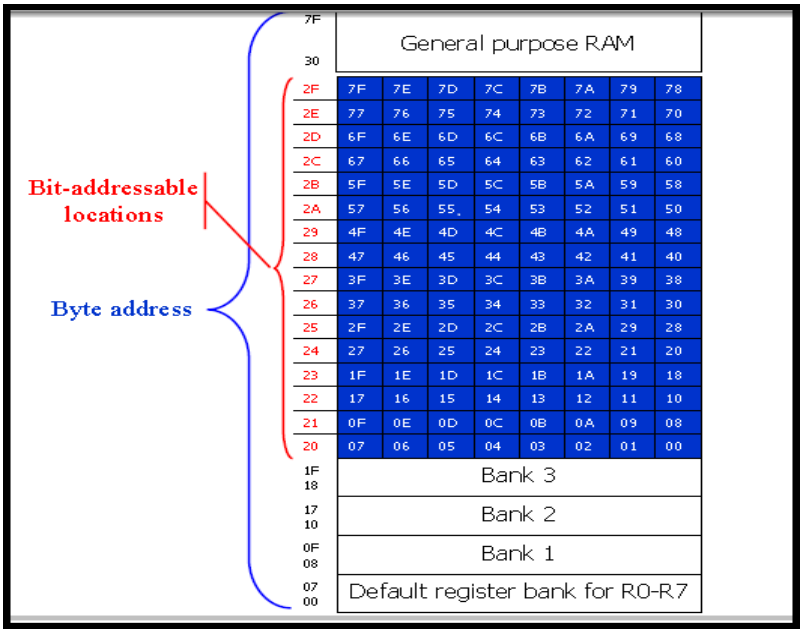


- Feature 8051 8052 8031:-

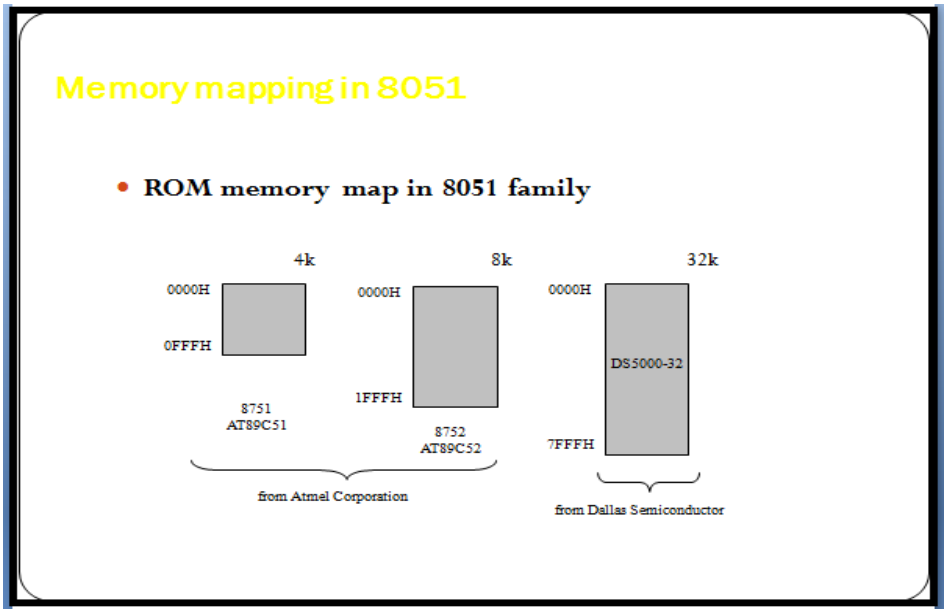
	8051	8052	8031
Interrupt sources	6	8	6
Serial port	1	1	1
I/O pins	32	32	32
Timers	2	3	2
RAM (bytes)	128	256	128
ROM (on-chip program space in bytes)	4K	8K	0K

- Registers in Microcontroller:

The most widely used registers A (Accumulator) For all arithmetic and logic instructions B, R0, R1, R2, R3, R4, R5, R6, R7 DPTR (data pointer), and PC (program counter)



(Internal RAM of 8051)



*C program to add two 8 bit numbers

```
#include<reg51.h>
void main(void)
{
  unsigned char x,y,z;
  x=0x0A;
  y=0x05;
  P1=0x00;
  z=x+y;
  P1=z;
}
```

*C program to subtract two 8 bit numbers

```
#include<reg51.h>
void main(void)
{
  unsigned char x,y,z;
  x=0x08;
  y=0x04;
  P1=0x00;
  z=x-y;
  P1=z;
}
```

*C program to multiply two 8 bit numbers

```
#include<reg51.h>
void main(void)
{
  unsigned char x,y,z;
  x=0x10;
  y=0x02;
  P1=0x00;
  z=x*y;
  P1=z;
}
```

*C program to divide two 8 bit numbers

```
#include<reg51.h>
void main(void)
{
  unsigned char x,y,z;
  x=0x10;
  y=0x02;
  P1=0x00;
  z=x/y;
  P1=z;
   }
```

*C program to data transfer

```
#include <reg51.h>
void main(void)
{
    unsigned char mynum[]= "012345ABCD";
    unsigned char z;
    for(z=0; z<=9; z++) P1=
    mynum[z];
}
```

CONCLUSION:

Course – Microcontrollers
Academic Year 2025-26

Semester 1

EXPERIMENT NO. - 2

TITLE	: Parallel port interacting of LEDS
AIM/ OBJECTIVE	: Parallel port interacting of LEDS—Different programs (flashing, Counter, BCD, HEX, Display of Characteristic) using 8051 Microcontroller HARDWARE USED: Microcontroller kit, Power supply, connecting wires.
SOFTWARE USED	: Keil µVision 4, Flash Magic.

Theory:

LEDs are by far the most widely used means of taking output. They find huge application as indicators during experimentations to check the validity of results at different stages. They are very cheap and easily available in a variety of shape, size and colours. LEDs are connected to the port of the microcontroller. LEDs need approximately 10mA current to flow through them in order to glow at maximum intensity. However the output of the controller is not sufficient enough to drive the LEDs, so if the positive leg of the LED is connected to the pin and the negative to ground as shown in the figure, the LED will not glow at full illumination. To overcome this problem LEDs are connected in the reverse order and they run on negative logic i.e., whenever 1 is given on any pin of the port, the LED will switch off and when logic 0 is provided the LED will glow at full intensity. The circuit of LED interfacing with microcontroller is as shown below:

Program:

a) The program for blinking of 8 LED's interfaced with port P1 of 8051.

```
#include <reg51.h>
void delay();
void main(void)
{
 while(1)  //INFINITE LOOP

{
   P1=0x0F0;
  delay();
  P1=0x0F;
delay();
 }
}
void delay()
{
TMOD=0x01;
TH0=0XFF;
TL0=0x0A;
TR0=1;
While (TF0==0);
TR0=0;
TF0=0;
}
```

b) The program for toggling of 8 LED's interfaced with port P1 of 8051.

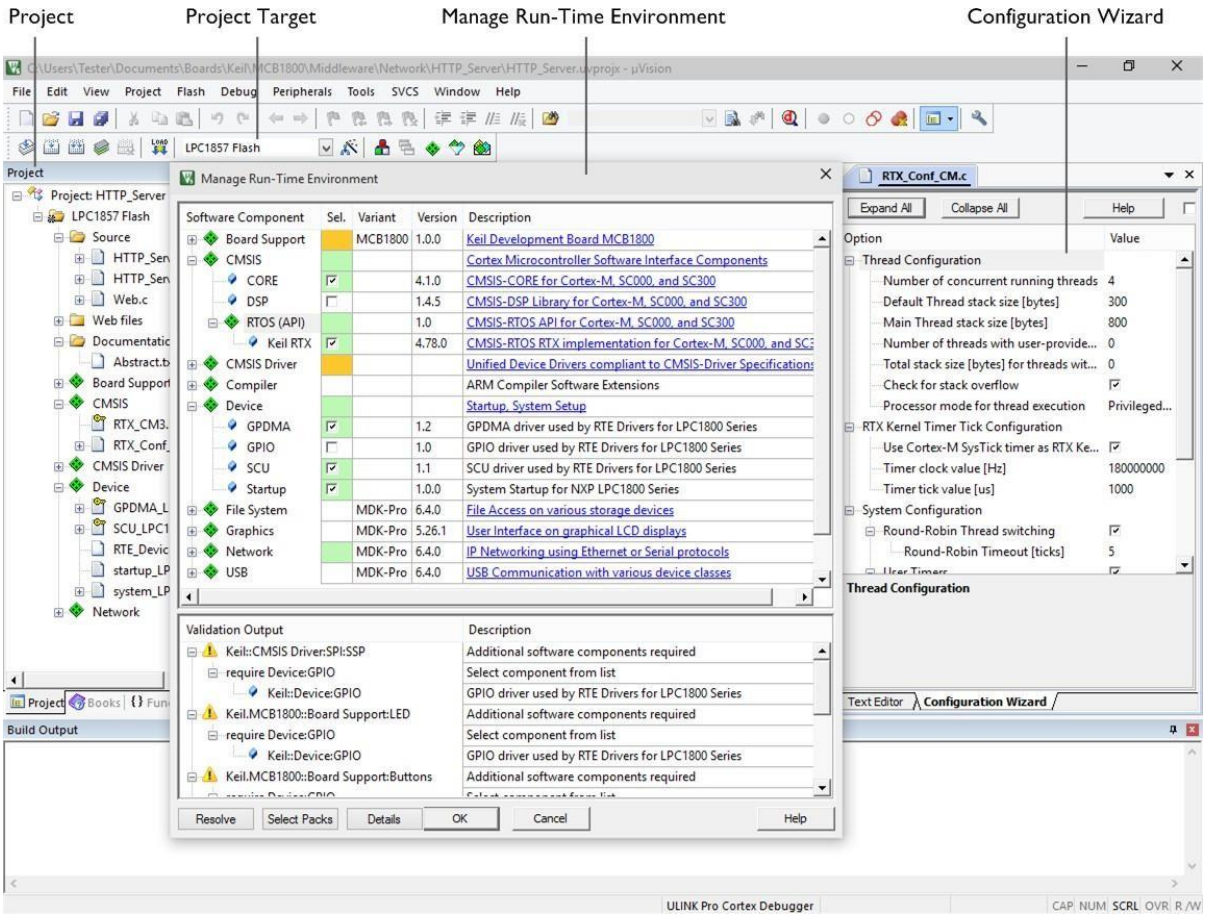
```
#include <reg51.h>
void delay();
void main(void)
{
 while(1)  //INFINITE LOOP

{
   P1=0xAA;
  delay();
  P1=0x00;
delay();
 }
}
void delay()
{
TMOD=0x01;
TH0=0XFF;
TL0=0x0A;
TR0=1;
While (TF0==0);
TR0=0;
TF0=0;
}
```

WORKING OF SOFTWARE:

1. Keil µVision:

The µVision IDE combines project management, run-time environment, build facilities, source code editing, and program debugging in a single powerful environment. µVision is easy-to-use and accelerates your embedded software development. µVision supports multiple screens and allows you to create individual window layouts anywhere on the visual surface.



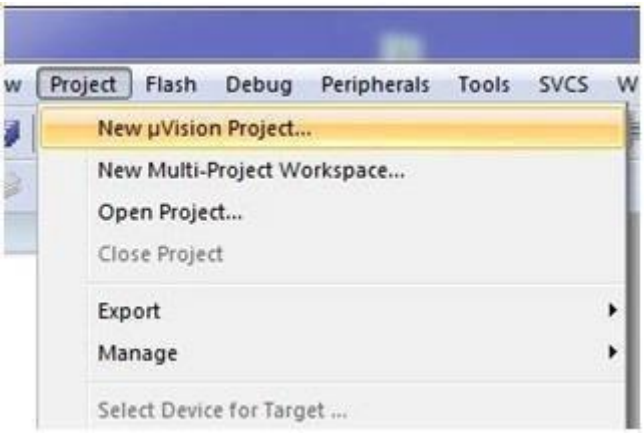
Procedure of using Keil:

Step 1: Download the Keil Uvision IDE

For learning purposes, you can try the evaluation version of Keil which has code limitation of 2K bytes. You must download the C51 version for programming on 8051 microcontroller architecture.

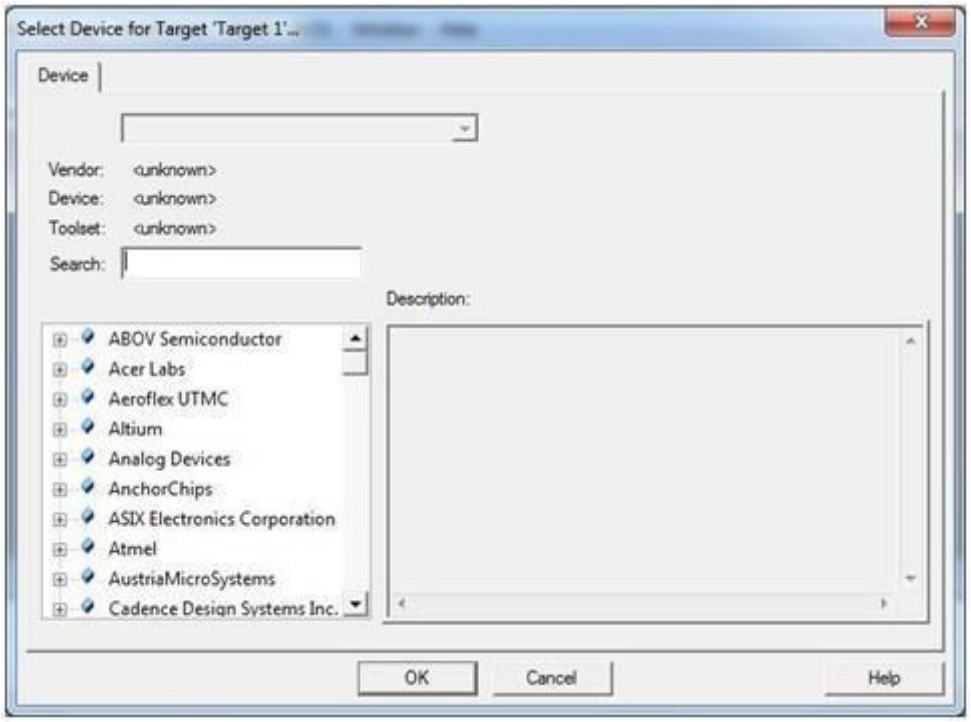
Step 2: To initiate the programming you must create a project using the keil Uvision IDE

The option to create a new project will be available under the project tab in the toolbar. Next, you have to store the project in a folder and give a suitable name to it.



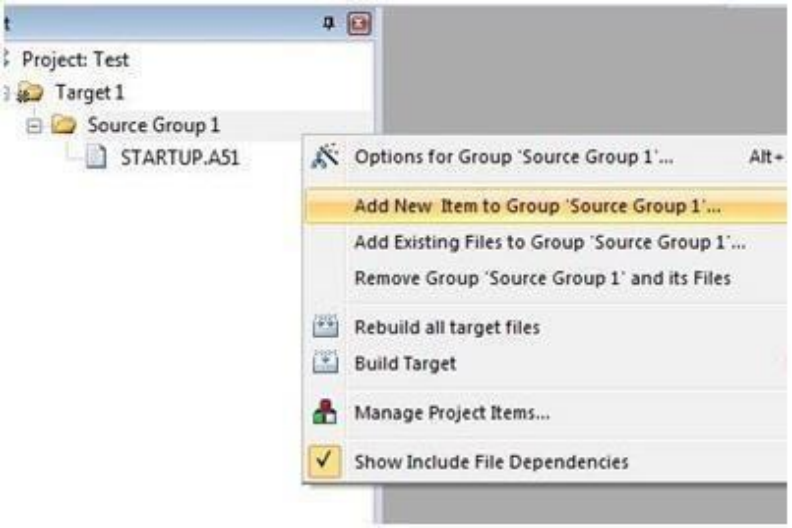
Step 3: Selecting the type of device you are working with

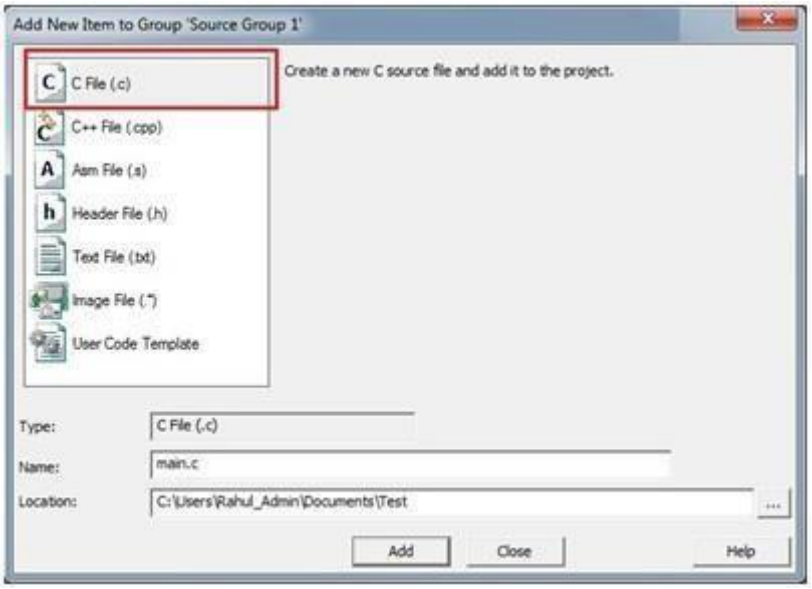
The device selection dialog provides you with the option to select the 8051 derivatives for which you want to develop the program. If you are not sure about your device you can refer to the description of the devices which is displayed on the left pane of the dialog. Accordingly, select your device and click OK to confirm.



Step 4: Adding C files to your project

You must add C file to your project before you begin coding. This can be done by right-clicking on your project from the project pane and selection "Add a new item to source group 1". In the next dialog box, you will be given with the choice on what type of file you want to add such as C, C++, header, text etc. Since here we are dealing with Embedded C programming, select C File (.c) option. Provide the necessary name, location and click on add.



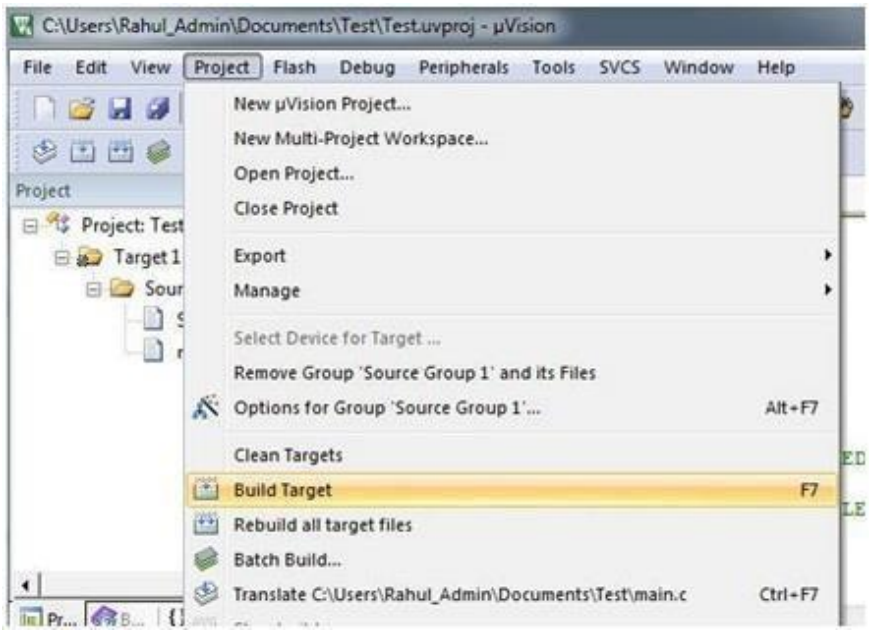


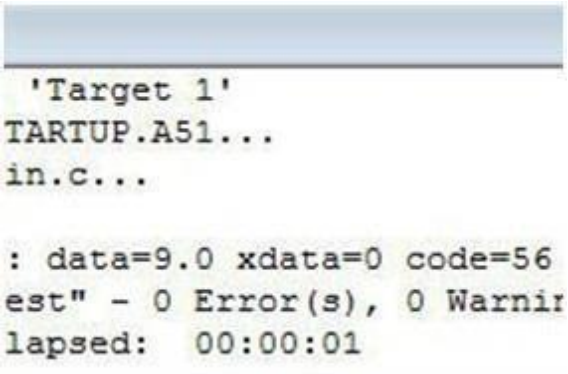
Step 5: Coding in C

The main part has finally arrived, so now you can go along with programming in C with your respective microcontroller.

Step 6 : Compiling and building the C project using Keil Uvision IDE

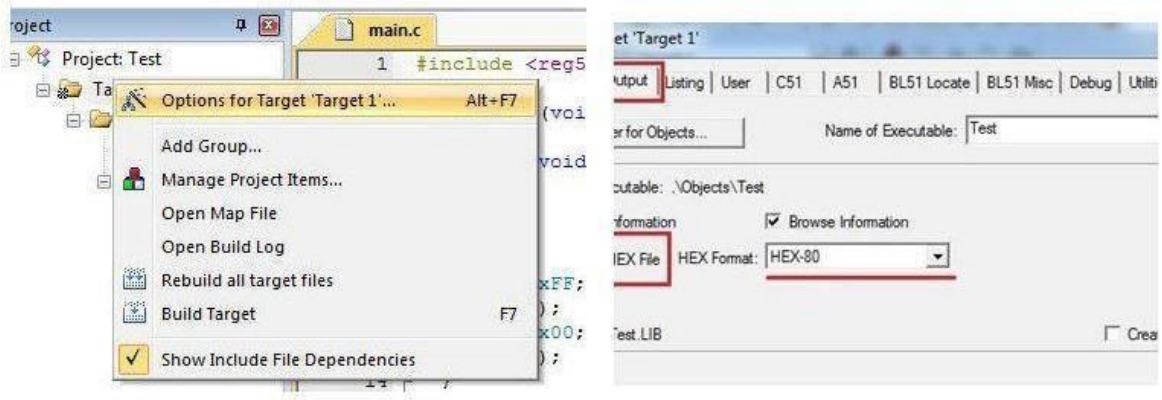
In order to build recently created C program go to Project tab and click on Build Target on the menu bar. An alternate way to do this is by pressing the F7 key. If the code that you have written is correct, the code will successfully compile without any errors. You can check your output in the Build Output pane.





Step 7: Generating the hex file using Keil Uvision IDE

The code you compiled cannot be directly fed to the microcontroller, it is not possible. For that purpose, we have to generate the hex code for your respective file.
In order to generate the hex code, right click on the 'Target 1' folder and select options for target 'Target 1'. Select the Output tab in the target 'Target 1' dialog box. Make sure Create Hex File option is checked and the HEX format should be HEX-80. Click OK.
Again rebuild your project by pressing F7. Your required hex file would have been generated with the same as your project in the Objects folder.
If you wish you can also view your hex code by using a notepad.



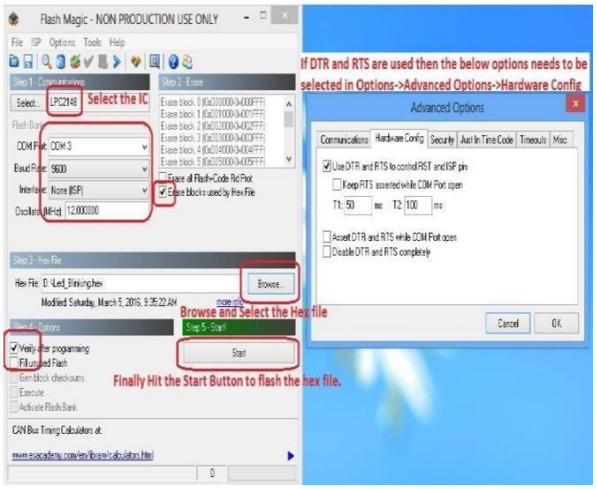
Step 8: Burning the hex code into 8051 microcontroller

In order to burn the hex code to your microcontroller, there are two ways which are specific to the device you are working with. Some devices, for example, P89V51 they have their own built-in boot loader and you can burn the hex code directly through the serial port. Mostly you will require a specific programmer for your 8051 microcontroller through which you can easily upload your hex code by connecting the programmer via normal USB port.

2. Flash Magic

Flash Magic is a PC tool for programming flash based microcontrollers from NXP using a serial or Ethernet protocol while in the target hardware.
Procedure for using Flash Magic:

1. Select the IC from Select Menu.
2. Select the COM Port. Check the device manager for detected Com port.
3. Select Baud rate from 9600-115200
4. Select None ISP Option.
5. Oscillator Freq 12.000000(12 MHz).
6. Check the Erase blocks used by Hex file option
7. Browse and Select the hex file.
8. Check the Verify After Programming Option.
9. If DTR and RTS are used then go to Options->Advanced Options-> Hardware Configuration and select the Use DTR and RTS Option.
10. Hit the Start Button to flash the hex file.
11. Once the hex file is flashed, Reset the board. Now the controller should run your application code.



CONCLUSION:

Course – Microcontrollers
Academic Year 2025-26 -
Semester 1

EXPERIMENT NO. - 3

TITLE : Interfacing of Multiplexed 7-segment display (counting application)

AIM/ OBJECTIVE : Write program for Interfacing Seven Segment Display to 8051 microcontroller.

HARDWARE/ SOFTWARE USED: Keil software, SST flash, SST89E516RD2 microcontroller Trainer kit

Theory:

Seven segment displays internally consist of 8 LEDs. In these LEDs, 7 LEDs are used to indicate the digit 0 to 9 and a single LED is used for indicating decimal point. Generally, seven segments are two types, one is a common cathode and the other is a common anode.

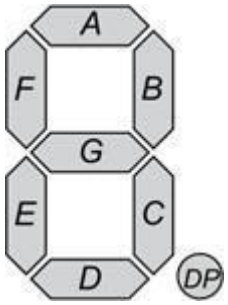


Fig.1 Seven segment

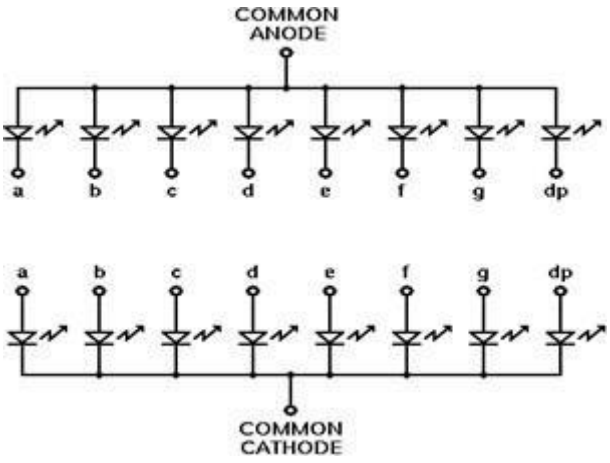


Fig.2 A common anode and Cathode LEDs

The common cathode, all the cathode of LEDs are tied together and labeled as com and the anode are left alone. In the common anode, seven-segment display all anodes are tied together and cathodes are left freely.

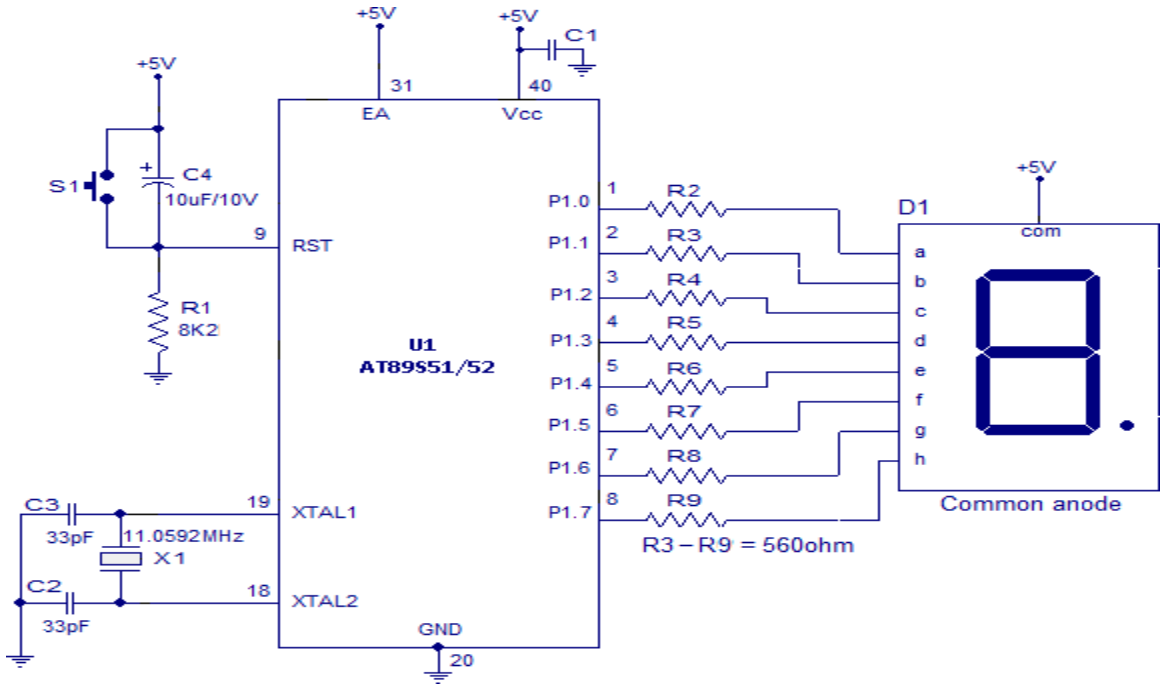
Seven Segment Truth Table for Common cathod:

Port P2	P2.7	P2.6	P2.5	P2.4	P2.3	P2.2	P2.1	P2.0	
Digit	0	G	F	E	D	C	B	A	Hex form
0	0	0	1	1	1	1	1	1	3F
1	0	0	0	0	0	1	1	0	06
2	0	1	0	1	1	0	1	1	5B
3	0	1	0	0	1	1	1	1	4F
4	0	1	1	0	0	1	1	0	66
5	0	1	1	0	1	1	0	1	6D
6	0	1	1	1	1	1	0	1	7D
7	0	0	0	0	0	1	1	1	07
8	0	1	1	1	1	1	1	1	7F
9	0	1	1	0	1	1	1	1	6F

Seven Segment Truth Table for Common anode:

Port P2	P2.7	P2.6	P2.5	P2.4	P2.3	P2.2	P2.1	P2.0	
Digit	0	G	F	E	D	C	B	A	Hex form
0	1	1	0	0	0	0	0	0	C0
1	1	1	1	1	1	0	0	1	F9
2	1	0	1	0	0	1	0	0	A4
3	1	0	1	1	0	0	0	0	B0
4	1	0	0	1	1	0	0	1	99
5	1	0	0	1	0	0	1	0	92
6	1	0	0	0	0	0	1	0	82
7	1	1	1	1	1	0	0	0	F8
8	1	0	0	0	0	0	0	0	80
9	1	0	0	1	00	0	0	0	90

Interfacing Diagram:



Procedure:

1. Open the Keil µVision4 or 3 Software

2. Click on the project –> click on new µvision project –> type project name –> Click on Save

3. Select the Target from SST family controller –> click OK–> click YES on a copy STARTUP.A51 to Project

4. Click on File –> click on the NEW document –> Save this document with the filename.C extension.

5. Type C program and save using filename.c

6. After this in Project window Right-click on Source group 1–>click on Add existing file on source group 1–> click filename.asm–> Click on Add.

7. After this click on Project –> click Build Target(or pressF7 key) –> on Build the output window shows 0 errors and 0 warning then project is completely built if not then remove the error then build it.

8. Click on 'Project' –> Click on 'option for target' –> click on 'Output'–>Tick on 'create Hex file'–>click on OK –> click on build

9. The hex file store at the location of your project in the object folder.

10. Use SST flash software to burn hex file in SST89E516RD2 microcontroller trainer kit.

CONCLUSION:

Course – Microcontrollers
Academic Year 2025-26

Semester 1

EXPERIMENT NO. - 4

TITLE : Interfacing of Stepper motor to 8051- software delay using Timer

AIM/ OBJECTIVE : Interfacing of Stepper motor to 8051.

HARDWARE/ SOFTWARE USED: Keil software, SST flash, SST89E516RD2 Microcontroller Trainer kit, Stepper Motor

Theory:

A stepper motor is a widely used device that translates electrical pulses into mechanical movement. In applications such as disk drivers, 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. The most common stator motors have four stator windings that are paired with a center tapped common. The center tap allows a change of current direction in each of two coils when a winding is grounded, thereby resulting in a polarity change of the stator. The stepper motor shaft moves in a fixed repeatable increment, which allows one to move it to a precise position. This repeatable fixed movement is possible as a result of basic magnetic theory where poles of the same polarity repel and opposite poles attract. The direction of the rotation is dictated by the stator poles. The stator poles are determined by the current sent through the wire coils. As the direction of the current is changed, the polarity is also changed causing the reverse motion of the rotor.

Table: 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	

This table shows a 2-phase, 4-step stepping sequence

Step angle: -

The step angle is the minimum degree of rotation associated with a single step. Various motors have different 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

Steps per second and rpm relation:-

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

The four –step sequence and number of teeth on rotor:-

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 x 50 = 200 steps are needed to complete one revolution. This leads to conclusion that 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.

For example a motor with a 2-degree step angle has following characteristics: Step angle: 2 degrees Steps per revolution: 180
Number of rotor teeth: 180/4= 45 Movement per 4-step sequence: 8 degrees

To allow for finer resolution, all stepper motors allow an 8-step sequence which is also called half stepping.

Table: Half-step 8-step sequence

	Step	Winding A	Winding B	Winding C	Winding D	
Clockwise ↓	1	1	0	0	1	Counter clockwise ↑
	2	1	0	0	0	
	3	1	1	0	1	
	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.

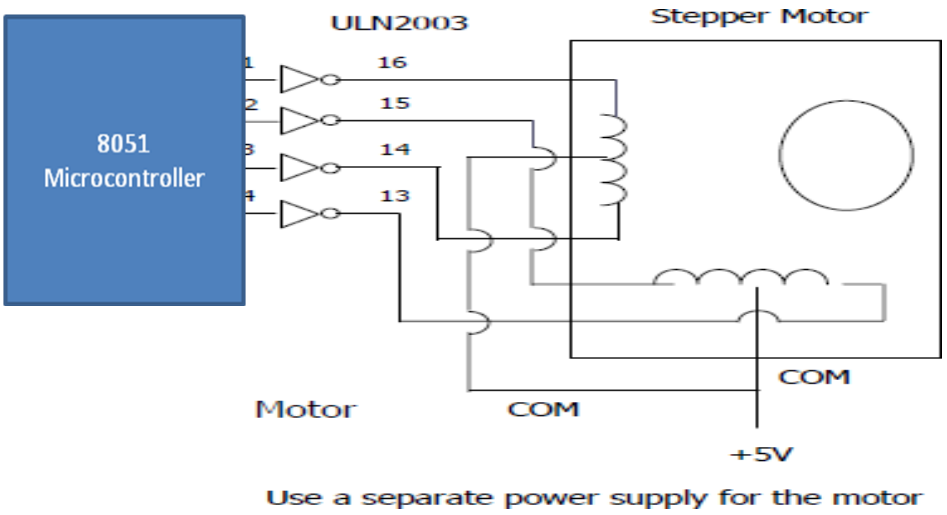
Holding torque: With the motor shaft at standstill or zero rpm condition, the amount of torque, from an external source required to break away the shaft from its holding position. This is measured with rated voltage and current applied to the motor.

Wave drives 4-step sequence: The 8-step sequence is the combination of the wave drive 4- step and normal 4-step sequences.

Unipolar versus bipolar stepper motor interface: There are three common types of stepper motor interfacing: universal, unipolar, and bipolar. 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. 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.

8051 connection to the stepper motor:-

Interfacing diagram:-



The 8051 lacks sufficient current to drive the stepper motor windings, we must use a driver such as the ULN 2003 to energize the stator. However if the transistors are used as drivers, we must also use diodes to take care of inductive current generated when the coil is turned off. One reason that using the ULN2003 is preferable to the use of transistors as drivers is that ULN2003 has an internal diode to take care of back EMF.

CONCLUSION:

Course – Microcontrollers
Academic Year 2025-26

Semester 1

EXPERIMENT NO. - 5

TITLE : A] Interfacing of LCD with 8051 microcontroller
: b] Interfacing of LCD with PIC 18FXXXX

AIM/ OBJECTIVE : LCD Interfacing with PIC18F4550 and with 8051 microcontroller

HARDWARE/ SOFTWARE USED: MPLAB IDE v7.51, Pic loader, Pic 18F4550 Trainer kit, Keil software, SST flash ,SST89E516RD2 microcontroller Trainer kit.

Theory:

INTRODUCTION TO LCD:-

Liquid crystal displays allow a better user interface with text messages to enter the instructions & get the response in the form of the text & know in a better manner what the machine is doing, including its diagnostic information. This also helps in fault findings and debugging. The main advantage of LCD displays is low power consumption and high speed with which the displayed information is updated. Hitachi 44780 LCD controller provides an easy solution to interface it with microcontrollers.12 microcontroller pins are needed for this interface.

LCD is finding widespread use replacing LEDs:-

- The declining prices of LCD
- The ability to display numbers, characters and graphics
- Incorporation of a refreshing controller
- into the LCD, thereby relieving the CPU of the task of refreshing the LCD
- Ease of programming for characters and graphics

THE LCD MODULE:-

The LCD has 16 pins in general, but usually 14 pins are of importance, pin number's 15 and 16 are used for backlight control.

LCD displays are available typically as 16x2, 20x1, 20x2 etc, along with LCD controller.16x2 means that 16 characters in each of the 2 lines can be stored. A standard LCD controller chip HD 44780U ca receives data from a microcontroller & communicates with the LCD. Figure shows the interfacing of LCD with microcontroller describing various pins whose description has been given in the table. There are 3 control lines & 8 (or 4 in case of 4- bit data) data lines. The three control signals are RS,E,R/W.

PIN DSICRIPTION OF LCD(16*2):

The most commonly used LCDs found in the market today are 1 Line, 2 Line or 4 Line LCDs which have only 1 controller and support at most of 80 characters, whereas LCD's supporting more than 80 characters make use of 2 HD44780 controllers.

Most LCDs with 1 controller has 14 Pins and LCDs with 2 controller has 16 Pins (two pins are extra in both for back-light LED connections). Pin description is shown in the table below.

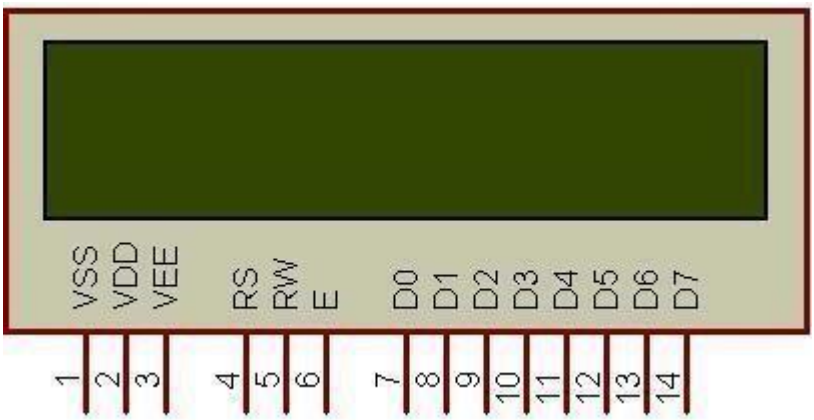
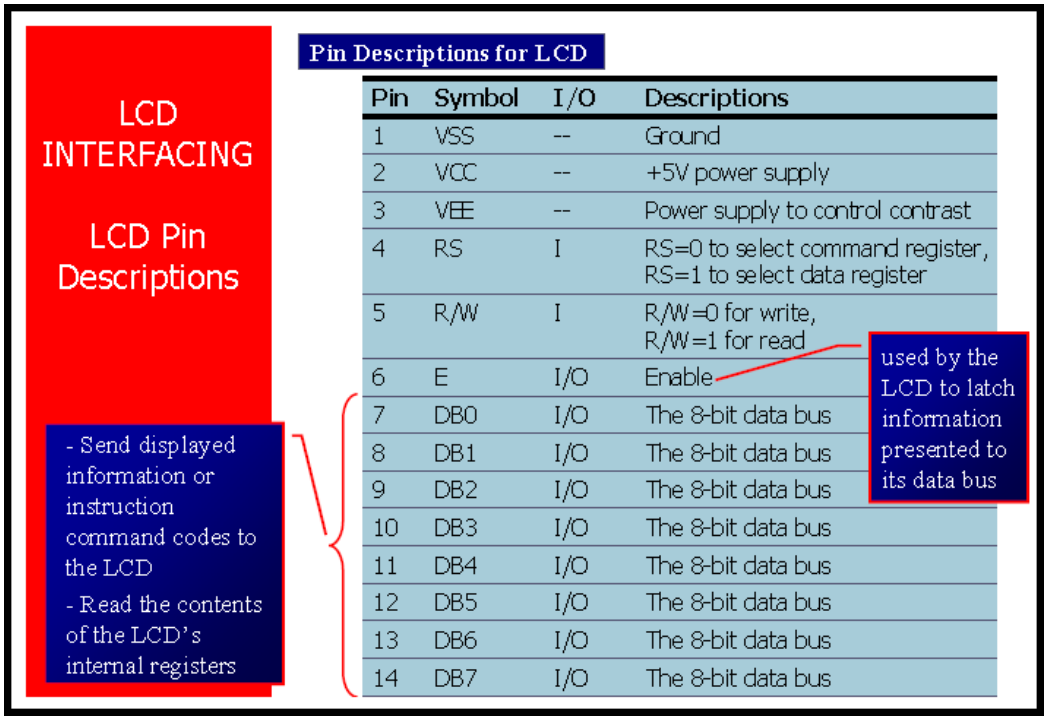


Figure 1: Character LCD type HD44780 Pin diagram



NOTE:-Pin number's 15 and 16 are used for backlight control for saving power it should be grounded.

'Enable-EN' 'register select-RS' & 'read/write-RW'.:-

EN line is to instruct the LCD that microcontroller is sending the data. This line is first made high by microcontroller, then the other control lines are RS &RW are defined & data is put on the data bus. After the data has been put on the data bus the EN is made low, means now the data is treated a command or instruction now to the data module. When RS line is high, the data is a text data to be displayed on the LCD.

Busy Flag (D7):-

The busy flag is D7 and can we read when RW=1 and RS=0, as follows; if RW=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. The read/write line RW, when made low, the information on the data bus is written to the LCD & when the RW line is high, it means that the data is being read from the LCD. It is very important to note that EN line must go from high to low for LCD to execute the instruction. The 8 bit data pins, D0-D7 are used to send the information to LCD or read the content of the LCD internal register. There are also instruction command codes that can be sent to the LCD to clear the display or force the cursor to the home position or blink the cursor. The most commonly used LCDs found in the market today are 1 Line, 2 Line or 4 Line LCDs which have only 1 controller and support at most of 80 characters, whereas LCD's supporting more than 80 characters make use of 2 HD44780 controllers. Most LCDs

with 1 controller has 14 Pins and LCDs with 2 controller has 16 Pins (two pins are extra in both for back-light LED connections). Pin description is shown in the table below.

Pin No.	Name	Description
Pin no. 1	D7	Data bus line 7 (MSB)
Pin no. 2	D6	Data bus line 6
Pin no. 3	D5	Data bus line 5
Pin no. 4	D4	Data bus line 4
Pin no. 5	D3	Data bus line 3
Pin no. 6	D2	Data bus line 2
Pin no. 7	D1	Data bus line 1
Pin no. 8	D0	Data bus line 0 (LSB)
Pin no. 9	EN1	Enable signal for row 0 and 1 (1^{st}controller)
Pin no. 10	R/W	0 = Write to LCD module 1 = Read from LCD module
Pin no. 11	RS	0 = Instruction input 1 = Data input
Pin no. 12	VEE	Contrast adjust
Pin no. 13	VSS	Power supply (GND)
Pin no. 14	VCC	Power supply (+5V)
Pin no. 15	EN2	Enable signal for row 2 and 3 (2^{nd}controller)
Pin no. 16	NC	Not Connected

LCD ADDRESSING for 16x2:-

16x2 LCD	80	81	82	83	84	85	86 through 8F
	C0	C1	C2	C3	C4	C5	C6 through CF

LCD COMMANDS CODES:-

LCD INTERFACING

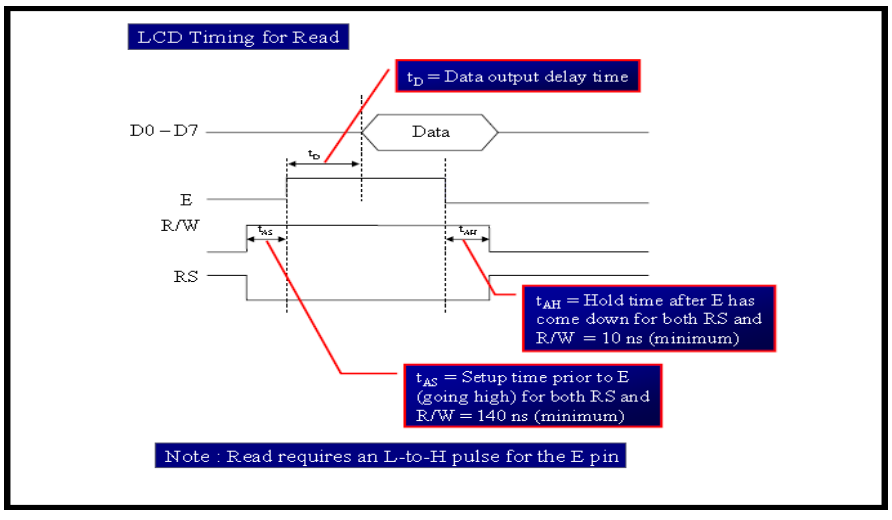
LCD Command Codes

LCD Command Codes

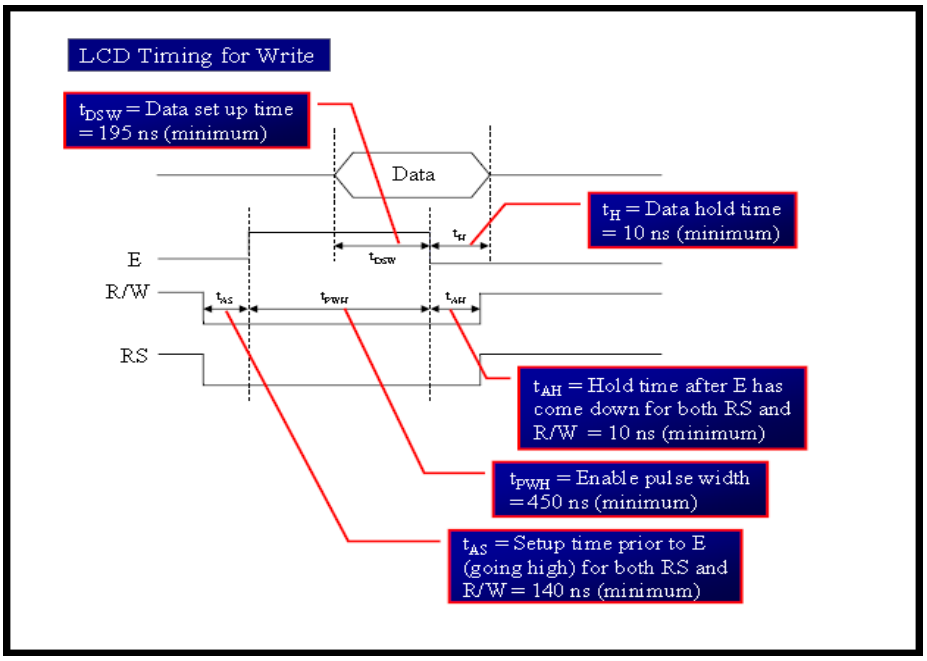
Code (Hex)	Command to LCD Instruction Register
1	Clear display screen
2	Return home
4	Decrement cursor (shift cursor to left)
6	Increment cursor (shift cursor to right)
5	Shift display right
7	Shift display left
8	Display off, cursor off
A	Display off, cursor on
C	Display on, cursor off
E	Display on,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

NOTE:-Generally used commands by any user : 38H,0EH,01H,06H,84H.

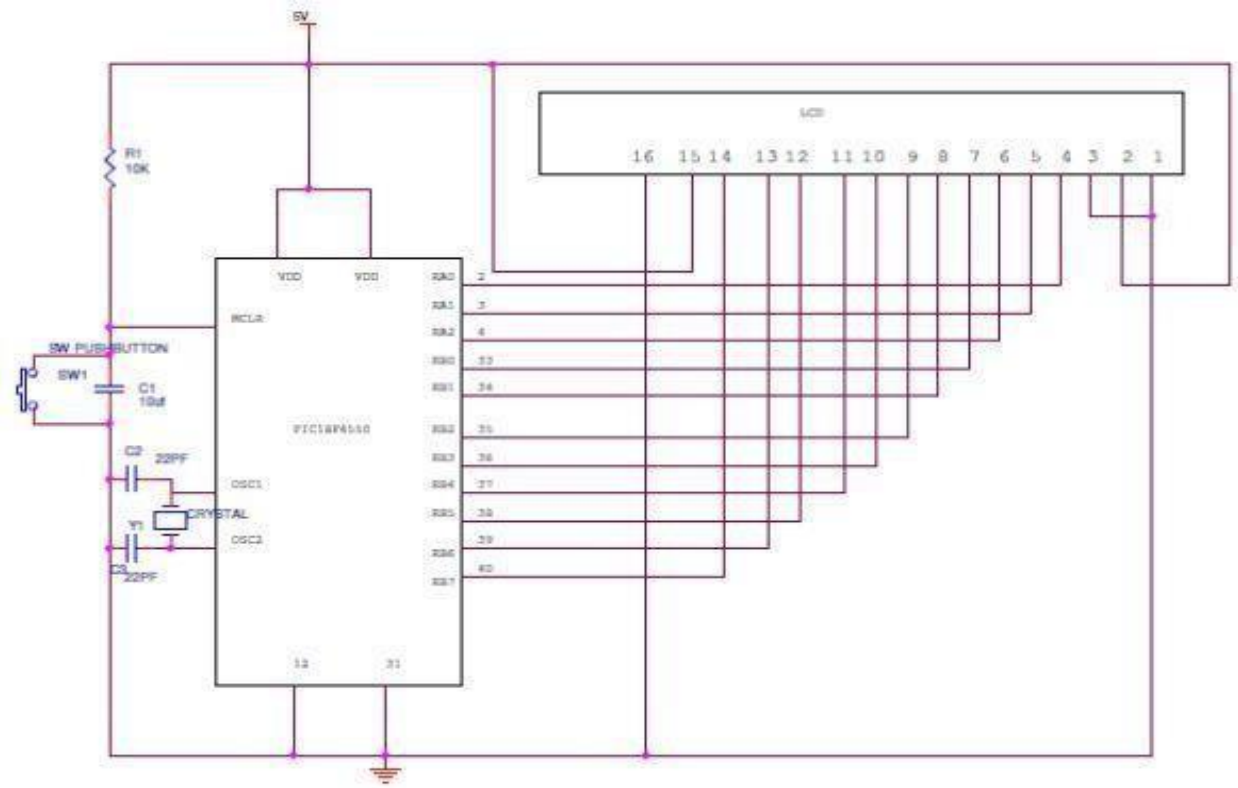
TIMING DIAGRAM FOR READ:-



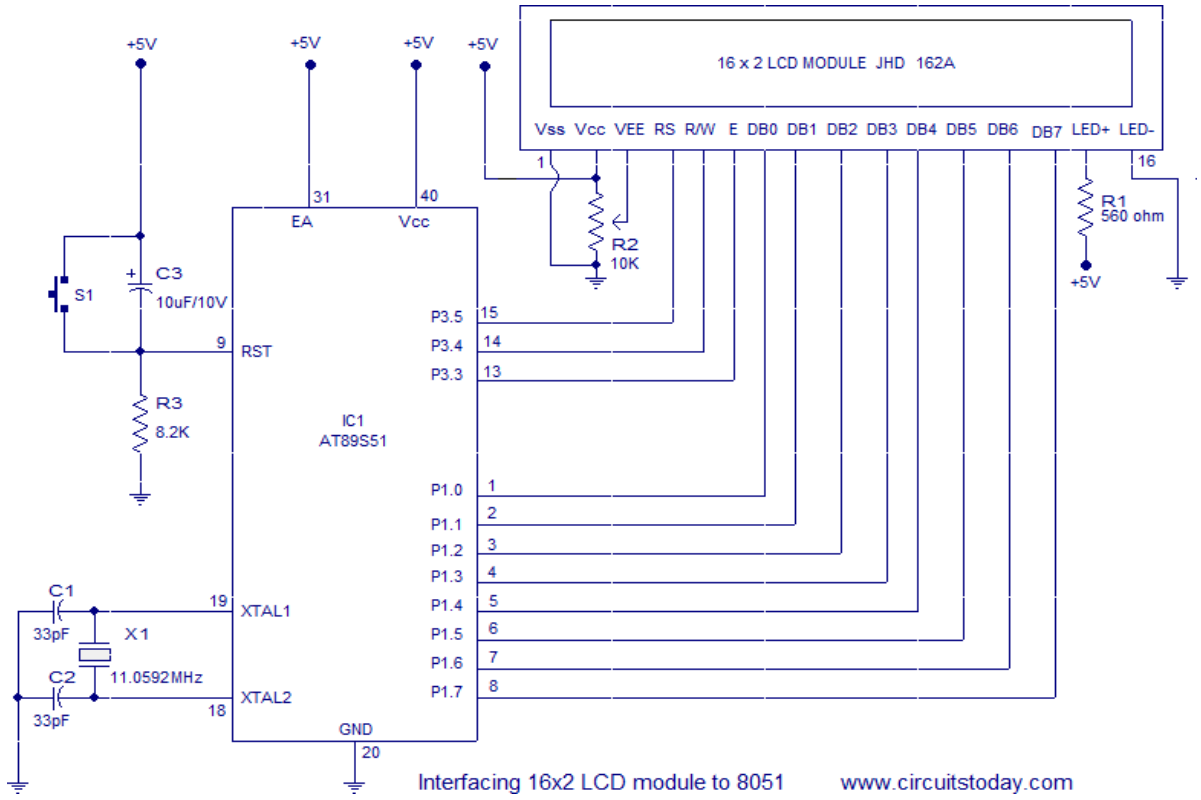
TIMING DIAGRAM FOR WRITE:-



Interfacing of LCD with PIC18F4550:



Interfacing of LCD with 8051 :



Programming Steps:

Before displaying anything on LCD, it needs to be configured with proper instructions. The following programming steps explain the procedure of configuring the LCD and display a character on it.

Step 1: Initialize the LCD.

The LCD must be initialized the by following pre-defined commands of character LCD.

- 0x38, to configure the LCD for 2-line, 5×7 font and 8-bit operation mode
- 0x0C, for Display On and Cursor Off
- 0x01, to Clear Display screen
- 0x06, to increment cursor
- 0x80, to set cursor position at first block of the first line of LCD.

The above set of commands is written in lcd_ ini() function of the adjoining code.

Step 2: Send the commands to LCD.

- Send the command byte to the port connected to LCD data pins
- RS=0, to select command register of LCD
- RW=0, to set the LCD in writing mode

- EN=1, a high to low pulse to latch command instruction
- Delay of 1ms
- EN=0

The above set of commands is written in lcd cmd(unsigned char) function.

Step 3: Send data to LCD.

- Send data at the port which connected to LCD data pins
- RS=1, register select to select data register of LCD
- RW=0, this set the LCD in writing mode
- EN=1, a high to low pulse to latch data
- Delay of 1ms
- EN=0

The lcddata(unsigned char) function has the above set of instructions.

Step 4: Display character on LCD.

The functions lcdcmd() and lcddata() are user-defined functions. They are used to send a character (E in this case) to be displayed on LCD.

```
lcdcmd(0x38);        // send command 0x38 to
LCD lcddata('E');    // send character E to LCD
```

CONCLUSION:

Course – Microcontrollers
Academic Year 2025-26

Semester 1

EXPERIMENT NO. - 6

TITLE : Write a program for interfacing button, LED, relay & buzzer as follows

AIM/ OBJECTIVE : Write a program for interfacing button, LED, relay & buzzer with PIC18F4550 as follows:

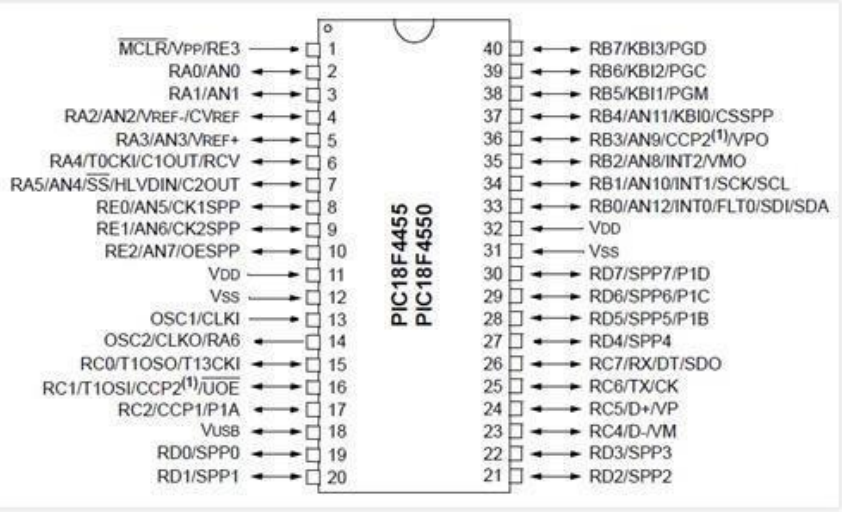
- When button 1 is pressed, relay and buzzer is turned off and LED's chase in right direction.
- When button 2 is pressed, relay and the buzzer is turned On and LED's chase in left direction.

HARDWARE/ SOFTWARE USED: MPLAB IDE v7.51, PIC Loader, PIC18F4550 Trainer Kit

Theory:

PIC is a family of Harvard architecture microcontrollers made by Microchip Technology. The PIC architecture is distinctively minimalist. It is characterized by the following features:

- Separate code and data spaces (Harvard architecture).
- A small number of fixed length instructions.
- Single-cycle execution (4 clock cycles).
- A single accumulator (W), the use of which (as source operand) is implied (i.e. is not encoded in the opcode).
- All RAM locations function as registers as both source and/or destination of math and other functions.
- 32 Kb Flash memories, 2K RAM, 256 bytes EEPROM.
- 2 comparators, 10 bit 13 channel ADC, 35 I/O Ports, 3 Timers, SPI and I2C supported.

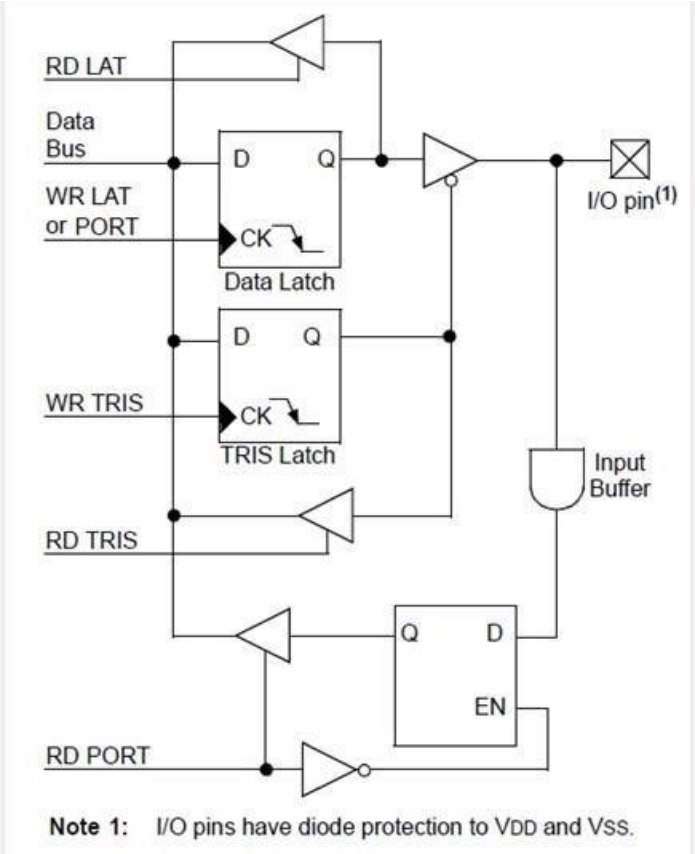


I/O Ports:

Depending on the device selected and features enabled, there are up to five ports available. 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.

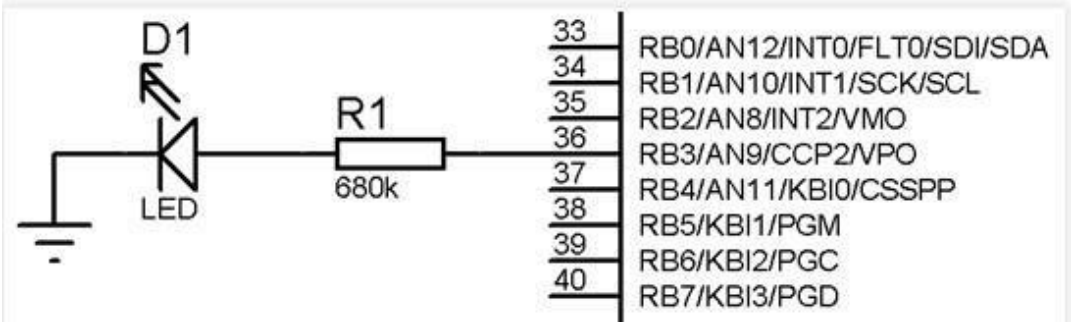
Each port has three registers for its operation. These registers are:

- TRIS register (data direction register)
- PORT register (reads the levels on the pins of the device)
- LAT register (output latch)
- The Data Latch register (LATA) is useful for read - modify-write operations on the value driven by the I/Opins.
- Reading the PORT A register reads the status of the pins; writing to it will write to the port latch.
- A simplified model of a generic I/O port, without the interfaces to other peripherals, is shown in Figure.



- Writing a '1' to the TRIS register configures the pin as input; whereas writing a '0' configures the pin as output.
- When configured as output, data can be written on the I/O pin using the LAT or the PORT register.
- When configured as input, the status of the I/O pin can be read using the PORT register.

Interfacing LED with microcontroller:



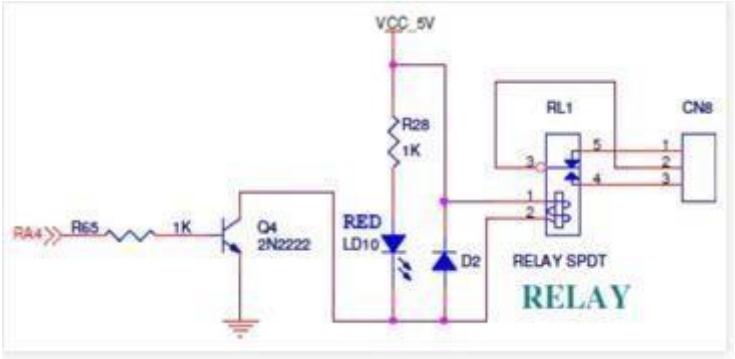
To drive the LED, following steps must be followed:

1. Configure the pin connected to the LED as output. This is done by writing 0 to the TRIS register. This can be done in 2 ways:
    a. Configuring entire PORTB as output:

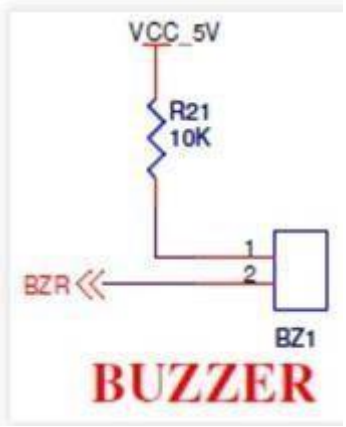
        TRISB = 0b00000000; or 0x00

    b. Configuring only the RB3 pin as output:
        TRISBbits.TRISB3 = 0; other pins remain unaffected.
2. Writing the data on the PORT pin to drive the LED ON or OFF. To turn the LED ON, we can follow one of these steps:
    a. LATB = 0b00001000; or 0x08 or
    b. PORTB= 0b00001000; or 0x08.
    c. LATBbits.LATB3 = 1; or
    d. PORTBbits.RB3 = 1;
3. To turn the LED OFF, we can follow one of these steps:
    a. LATB = 0b00000000; or 0x00
    b. PORTB=0b00000000; or 0x00
    c. LATBbits.LATB3 = 0; or
    d. PORTBbits.RB3 = 0;

Interfacing Relay and Buzzer with microcontroller:



Relay Interfacing

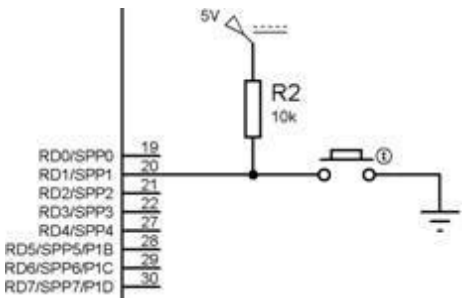


Buzzer interfacing

To turn the relay and buzzer ON and OFF, follow the same steps as that of LED. However, from the figure, we can see that the logic of the the buzzer is inverted i.e. when we write '0' to the PORT pin connected to the buzzer, it is turned ON; whereas on writing a '1' turns the buzzer OFF.

Interfacing a BUTTON with microcontroller:

To interface a push-button with a pin of the microcontroller, one end of the button is grounded, while the other end is connected to the pin of the microcontroller.
The microcontroller pin is pulled HIGH with the help of a Pull-Up resistor or by setting the LAT register bit HIGH.
When the button is not pressed i.e. when it is open, the PORT pin receives logic '1' through the Pull-Up resistor or through the LAT register.
When the button is pressed, the PORT pin receives logic '0' since it is connected to the ground directly via the closed push-button.
The status or the logic received by the pin is read with the help of the PORT register.



Button Interfacing

Procedure:

1. Open MPLABX IDE and create a new project.

   File-> New Project->Microchip Embedded -> Standalone
   Project. Select device PIC18F4550.
   Select compiler XC8.
   Project name ->
   Finish
2. Create a C source file: Right click on the Source
   Files New -> C source file
   Give a name to the file.
3. Copy or type the source program in the C file created.
4. Save the file.
5. Right-click on the project name, and select “Clean and Build”.
6. Open PIC Loader.
7. Press F3
8. Press Reset
9. Press F4

CONCLUSION :

Course – Microcontrollers
Academic Year 2025-26

Semester 1

EXPERIMENT NO. - 7

TITLE : Generate square wave using timer with interrupt

AIM/ OBJECTIVE : Write embedded C language program to generate a square wave of 50Hz any port pin.

HARDWARE/ SOFTWARE USED:

1. Keil uVision2 Compiler / Assembler
2. SST Flash Software for downloading
3. 8051 Trainer kit.

Theory:

Time Delay Generation:

Most used subroutine is one that generates a programmable time delay. Time delays may be done by using software loops that essentially do nothing for some period or by using hardware timers that count internal clock pulses. The key to writing this program is to calculate the exact time each instruction will take at the clock frequency in use. Following terms are very helpful to write a subroutine which generates desired time delay.

T-state: T-state is defined as one subdivision of the operation performed in one clock period. The terms T-state and clock period are often used synonymously.

Machine Cycle: Machine cycle in 8051 is defined as 12 oscillator periods. The 8051, take one to four machine cycles to execute an instruction. To calculate the machine cycle for the 8051, we take 1/12 of crystal frequency, and then take its inverse.

Assume crystal frequency of 11.0592 MHz

M/C frequency = 11.0592 MHz/12 = 921.6 KHz Machine Cycle =1/921.6 KHz = 1.085 us (microseconds)

Instruction Cycle: Instruction cycle is defined as the time required for completing the execution of an instruction. One instruction cycle consists of one to four machine cycles.

e.g. 2 Machine cycles are required for instruction DJNZ R2, target to be executed. Then instruction cycle is calculated as follows.

Instruction cycle = No. Machine cycles x Machine cycle period

= 2 x 1.085 us

= 2.17 us

Timer /Counter in 8051:

8051 has 2 timer/ counter. They can be used a timers to generate a time delay or as counter to count events happening outside the microcontroller. Both Timer 0 and Timer 1 are 16 bits wide. Since the 8051 has an 8-bit architecture, each 16-bit timer register is accessed as two separate registers of low byte and high byte.

- Timer 0 can be accessed as –
- TL0 – Timer 0 lower byte
- TH0 – Timer 0 higher byte
- Timer 1 can be accessed as –
- TL1 - Timer 1 lower byte
- TH1 – Timer 1 higher byte

Both timer shares the Timer control (TCON) register, which controls the timer/ counter operation and Timer mode (TMOD) register, which is used to configure the timer for different operating modes.

TMOD (Timer Mode Register):

GATE	T/C	M1	M0	GATE	T/C	M1	M0
Timer/Counter1				Timer/Counter0			

Both the timers used the same 8 bit register to set various timer operation mode. TMOD is 8- bit register where lower 4 byte are set aside for timer 0 and higher 4 bytes for timer 1. Since, it is not bit addressable; the corresponding bit value is directly loaded into TMOD.

Gate:

8051 has both hardware and software controls to start and stop the timers. By the means of software controlling instruction timers are used to control to start timer or stop.

C/T:

This bit in TMOD is used to determine whether timer is to be used as delay generator or event counter.
If C/T = 0 – used as timer
If C/T = 1 – used as
counter

M1 M0:

M1, M0 selects the timer mode.

M1	M0	Mode	Operation
0	0	0	13 bit counter, 8 bit C/T with THX and TLX as 5 bit Prescalar.
0	1	1	16 bit counter, 8 bit C/T with THX and TLX cascaded with no Prescalar.
1	0	2	8 bit auto reload, THX hold the value which is to be loaded into TLX after each overflow.
1	1	3	Split timer mode.

TCON (Timer Control Register):

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Timer run control bits TR0 and TR1 and timer overflow flags TF0 and TF1 are the part of 8 bit register called TCON. The upper 4 bits are used to store TR and TF flags of both timer- 0 and timer 1 while the lower 4 bits are set aside for interrupt. Timer run control bit TR0/TR1 is used to start the corresponding timer / counter. Timer overflow flag bit TF0/TF1 is set when corresponding timer/ counter is overflowed i.e. count value FFFF h to 0000 h.

Different Modes of timer/ counter –

1. Mode 0 (13 bit timer/counter):

- Mode 0 is exactly like mode 1 except it is 13 bit timer.

C/T:

This bit in TMOD is used to determine whether timer is to be used as delay generator or event counter.
If C/T = 0 – used as timer
If C/T = 1 – used as
counter

M1 M0:

M1, M0 selects the timer mode.

M1	M0	Mode	Operation
0	0	0	13 bit counter, 8 bit C/T with THX and TLX as 5 bit Prescalar.
0	1	1	16 bit counter, 8 bit C/T with THX and TLX cascaded with no Prescalar.
1	0	2	8 bit auto reload, THX hold the value which is to be loaded into TLX after each overflow.
1	1	3	Split timer mode.

TCON (Timer Control Register):

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Timer run control bits TR0 and TR1 and timer overflow flags TF0 and TF1 are the part of 8 bit register called TCON. The upper 4 bits are used to store TR and TF flags of both timer- 0 and timer 1 while the lower 4 bits are set aside for interrupt. Timer run control bit TR0/TR1 is used to start the corresponding timer / counter. Timer overflow flag bit TF0/TF1 is set when corresponding timer/ counter is overflowed i.e. count value FFFF h to 0000 h.

Different Modes of timer/ counter –

2. Mode 0 (13 bit timer/counter):

- Mode 0 is exactly like mode 1 except it is 13 bit timer.

The size of the time delay depends on two factors, (a) the crystal frequency and (b) the timer's 16-bit register in mode 1. The largest delay is achieved by the making both TH and TL zero.
Formula for delay calculations using mode 1 of the timer for crystal frequency of XTAL = 11.0592 MHz, (TH, TL) = $(NNNNN)_{10}$ is as follows.
Time Delay (Td) = (65536 - NNNNN) x 1.085 us.

Therefore Maximum delay = (65536 – 0000) x 1.085 us = 71 ms. *Finding the Values to be loaded into timer for desired delay* Assume the 11.0592 MHz as crystal frequency for 8051.

i. Divide the desired time delay by 1.085us. (n = Td/1.085us)
ii. Perform 65536 – n. where n is the decimal value from step 1.
iii. Convert the result of step 2 to hexadecimal, where yyxx is the initial hex value to be loaded into the timer's registers.
iv. Set TL = xx and TH = yy.

To generate the 2 KHz square wave on port pin

A. Main Program

1. Load the value "10h" in TMOD register indicating Timer-1 is to be used and timer mode 1 is selected.
2. Load the registers TL and TH with initial count values i.e. DC00h. (TH = DCh, TL = 00h)
3. Start the timer by setting TR1 bit in TCON register.
4. Halt the program.

CALCULATIONS:

Frequency = 50 Hz

Machine Cycle = 1.085 us

(microseconds) Time period = T_P

=1/50Hz = 20 ms

Required duty cycle is 50%

Therefore, $T_{on} = T_{off} = 1/20 = 10$ ms

Desired time delay is $T_D = 10$ ms

Divide T_D by $1.085 \times 10^{-6} = n = 10 \times 10^{-3} / 1.085 \times 10^{-6} = 9216$

Subtract n from 65536 = 65536 – 9216 = 56320

Convert above decimal value in to Hex value =

DC00Ah Load this value into Timer Register.

(TH = DCh , TL = 00h)

CONCLUSION:

Course – Microcontrollers
Academic Year 2025-26

Semester 1

EXPERIMENT NO. - 8

TITLE : Interface analog voltage 0-5V to internal ADC and display value on LCD

AIM/ OBJECTIVE : Interface analog voltage 0-5 to internal ADC and display value on LCD

HARDWARE/ SOFTWARE USED: MPLAB IDE, PIC Loader, PIC18F4550 Trainer Kit

THEORY:

PIC 18F4520 has on chip 10 bit ADC.

It has 4 registers for controlling and storage of result.

1. ADCON0 – ADC Control Register 0
2. ADCON1 – ADC Control Register 1
3. ADRESH – ADC Result High Register
4. ADRESL – ADC Result Low

Register A/D Control Register 0 (ADCON0)

The ADCON0 register, shown in the below image, controls the operation of the A/D module i.e. Used to Turn ON the ADC, Select the Sampling Freq, and also Start the conversion.

ADCON0 REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0
ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/$\overline{\text{DONE}}$	—	ADON
bit 7							bit 0

ADCS1-ADCS0: A/D Conversion Clock Select bits. These bits are based on ADCON1 Register's ADCS2 bit.

CHS2-CHS0: Analog Channel Select bits

000 = Channel 0 (AN0)

001 = Channel 1 (AN1)

010 = Channel 2 (AN2)

011 = Channel 3 (AN3)

100 = Channel 4 (AN4)

101 = Channel 5 (AN5)

110 = Channel 6 (AN6)

111 = Channel 7 (AN7)

GO/DONE: A/D Conversion Status
bit When ADON = 1:
1 = A/D conversion in progress (setting this bit starts the A/D conversion which is automatically cleared by hardware when the A/D conversion is complete)

0 = A/D conversion not in progress
ADON: A/D On bit
1 = A/D converter module is powered up

0 = A/D converter module is shut-off and consumes no operating
current A/D Control Register 1 (ADCON1)
The ADCON1 register, shown below, configures the functions of the port pins i.e Used to configure the GPIO pins for ADC. The port pins can be configured as analog inputs (RA3 can also be the voltage reference) or as digital I/O.

ADCON1 REGISTER

R/W-0	R/W-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	ADCS2	—	—	PCFG3	PCFG2	PCFG1	PCFG0
bit 7							bit 0

Steps :

1. Initialize the ADC Module: We have already learnt how to initialize an ADC so we just call this below function to initialize the ADC

The void ADC_Initialize() function is be as follows.

```
void ADC_Initialize()
{
  ADCON0 = 0b01000001; //ADC ON and Fosc/16 is selected
  ADCON1 = 0b11000000; // Internal reference voltage is selected
}
```

2. Select the analog channel: Now we have to select which channel we are going to use to read the ADC value. Lets make a function for this, so that it will be easy for us to shift between each channel inside the *while* loop.

```
unsigned int ADC_Read(unsigned char channel)
{
    //****Selecting the channel**///
  ADCON0 &= 0x11000101; //Clearing the Channel Selection Bits
  ADCON0 |= channel<<3; //Setting the required Bits
  //**Channel selection complete***///
}
```

Then channel to be selected is received inside the variable channel. In the line

```
ADCON0 &= 0x1100101;
```

The previous channel selection (if any) is cleared. This is done by using the bitwise and operator “&”. The bits 3, 4 and 5 are forced to be 0 while the others are left to be in their previous values.

Then the desired channel is selected by left shifting the channel number thrice and setting the bits using the bitwise or operator “|”.

```
ADCON0 |= channel<<3; //Setting the required Bits
```

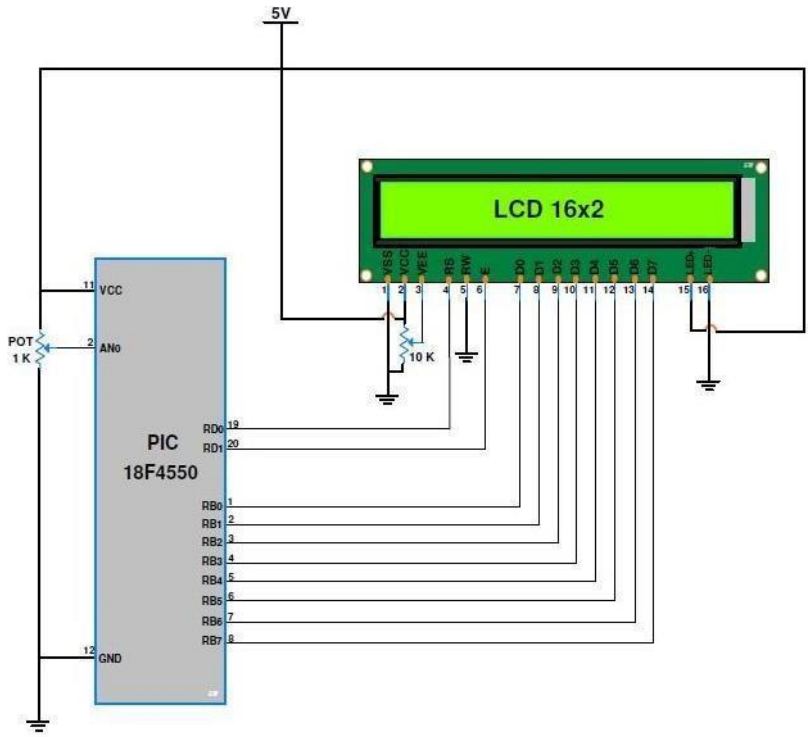
3. Start ADC by making Go/Done bit high: Once the channel is selected we have to start the ADC conversion simply by making the GO_DONE bit high:

```
GO_nDONE = 1; //Initializes A/D Conversion
```

4. Wait for the Go/DONE bit to get low: The GO/DONE bit will stay high until the ADC conversion has been completed, hence we have to wait till this bit goes low again. This can be done by using a *while* loop.

```
while(GO_nDONE); //Wait for A/D Conversion to complete
```

Circuit Diagram:



PROGRAM FOR ANALOG VOLATGE TO INTERNAL ADC & DISPLAY ON LCD:-

```
#include <p18f4550.h>
#include<stdio.h>
#define LCD_EN LATAbits.LA1
#define LCD_RS LATAbits.LA0
#define LCDPORT LATB

void lcd_delay(unsigned int time)
{
 unsigned int i , j ;

   for(i = 0; i < time; i++)
   {
        for(j=0;j<50;j++);
   }
}
void SendInstruction(unsigned char command)
{
    LCD_RS = 0;                 // RS low :
    Instruction LCDPORT = command;
    LCD_EN = 1;                 // EN
    High lcd_delay(10);
    LCD_EN = 0;                 // EN Low; command sampled at EN falling
    edge lcd_delay(10);
}
void SendData(unsigned char lcddata)
{
    LCD_RS = 1;                 // RS HIGH : DATA
    LCDPORT = lcddata;
    LCD_EN = 1;                 // EN
    High lcd_delay(10);
    LCD_EN = 0;                 // EN Low; data sampled at EN falling edge
```

```
    lcd_delay(10);
}

void InitLCD(void)
{
   ADCON1 = 0x0F;
   TRISB = 0x00; //set data port as output
   TRISAbits.RA0 = 0; //RS pin
   TRISAbits.RA1 = 0; // EN pin

   SendInstruction(0x38);    //8 bit mode, 2 line,5x7
   dots SendInstruction(0x06);            // entry mode
   SendInstruction(0x0C);    //Display ON cursor OFF
   SendInstruction(0x01);    //Clear display
   SendInstruction(0x80);    //set address to 0
}

void ADCInit(void)
{
   TRISEbits.RE1 = 1;                //ADC channel 6 input
   TRISEbits.RE2 = 1;                //ADC channel 7 input

   ADCON1 = 0b00000111;                  //Ref voltages Vdd & Vss; AN0 -  AN7
channels Analog
   ADCON2 = 0b10101110;                 //Right justified; Acquisition time 12T;
Conversion clock Fosc/64
}

unsigned short Read_ADC(unsigned char Ch)
{
   ADCON0 = 0b00000001 | (Ch<<2);  //ADC on; Select channel;
   GODONE = 1;             //Start Conversion

   while(GO_DONE == 1 );     //Wait till A/D conversion is
   complete return ADRES;     //Return ADC result
```

```
}

void DisplayResult(unsigned short ADCVal)
{
 unsigned char i,text[16];
 unsigned short tempv;
 tempv = ADCVal;

 SendInstruction(0x80);            //set to 1st line
 for(i=0;i<10;i++)                 //Display the 10 bit ADC result on LCD
 {
  if(tempv & 0x200)
  {
   SendData('1');
  }
  else
  {
    SendData('0');
  }
  tempv = tempv<<1;
 }

 ADCVal = (5500/1024)*ADCVal;                  //Convert binary data to mV;  1 bit
<=> (5500/1024)mV
 sprintf(text,"ADC value=%4dmv",ADCVal);       //Convert integer data to string

 SendInstruction(0xC0);              //set to 2nd line
 for(i=0;i<16;i++)                   //Display string on
 LCD
 {
  SendData(text[i]);
 }
}
```

```
void main()
{
    unsigned short Ch_result;


TRISB = 0x00;                    //PORTB connected to LCD is
output ADCInit();
    InitLCD();
 while(1)
    {
        Ch_result = Read_ADC(7);
        DisplayResult(Ch_result);
        lcd_delay(1000);
    }
}
```

CONCLUSION:

Course – Microcontrollers
Academic Year 2025-26

Semester 1

EXPERIMENT NO. - 9

Aim: Generation of PWM signal for DC motor control.

Software and Hardware used: MPLAB IDE, PIC Loader, PIC18F4550 Trainer Kit

APPLICATION:

In Industries DC motors are using. In some application speed control is required. Here we have assembly of small DC motor with speed control.

THEORY:

The direct current (DC) motor is one of the first machines devised to convert electrical power into mechanical power. Permanent magnet (PM) direct current converts electrical energy into mechanical energy through the interaction of two magnetic fields. One field is produced by a permanent magnet assembly; the other field is produced by an electrical current flowing in the motor windings. These two fields result in a torque which tends to rotate the rotor. As the rotor turns, the current in the windings is commutated to produce a continuous torque output. The stationary electromagnetic field of the motor can also be wire- wound like the armature (called a wound-field motor) or can be made up of permanent magnets (called a permanent magnet motor). The purpose of a motor speed controller is to take a signal representing the demanded speed, and to drive a motor at that speed

PWM Signal

Pulse Width Modulation (PWM) is a digital signal which is most commonly used in control circuitry. This signal is set high (5v) and low (0v) in a predefined time and speed. The time during which the signal stays high is called the "on time" and the time during which the signal stays low is called the "off time". There are two important parameters for a PWM as discussed below:

Duty cycle of the PWM:

The percentage of time in which the PWM signal remains HIGH (on time) is called as duty cycle. If the signal is always ON it is in 100% duty cycle and if it is always off it is 0% duty cycle.

Duty Cycle =Turn ON time/ (Turn ON time + Turn OFF time)

PWM using PIC18F4550:

PWM signals can be generated in our PIC Microcontroller by using the **CCP (Compare Capture PWM)** module. The resolution of our PWM signal is 10-bit, that is for a value of 0 there will be a duty cycle of 0% and for a value of 1024 (2^10) there be a duty cycle of 100%. There are two CCP modules in our PIC MCU (CCP1 And CCP2), this means we can generate two PWM signals on two different pins (pin 17 and 16) simultaneously, in our tutorial we are using CCP1 to generate PWM signals on pin 17.

The following registers are used to generate PWM signals using our PIC MCU:

1. CCP1CON (CCP1 control Register)
2. T2CON (Timer 2 Control Register)
3. PR2 (Timer 2 modules Period Register)
4. CCPR1L (CCP Register 1 Low)

Programming PIC to generate PWM signals:

In our program we will read an Analog voltage of 0-5v from a potentiometer and map it to 0- 1024 using our ADC module. Then we generate a PWM signal with frequency 5000Hz and vary its duty cycle based on the input Analog voltage. That is 0-1024 will be converted to 0%-100% Duty cycle. This tutorial assumes that you have already learnt to use ADC in PIC if not, read it from here, because we will skip details about it in this tutorial.

So, once the configuration bits are set and program is written to read an Analog value, we can proceed with PWM.
The following steps should be taken when configuring the CCP module for PWM operation:

1. Set the PWM period by writing to the PR2 register.

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 TRISC<2> bit.
4. Set the TMR2 prescale value and enable Timer2 by writing to T2CON.
5. Configure the CCP1 module for PWM operation.

CCP1CON REGISTER/CCP2CON REGISTER (ADDRESS 17h/1Dh)

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	CCPxX	CCPxY	CCPxM3	CCPxM2	CCPxM1	CCPxM0
bit 7							bit 0

bit 3-0 **CCPxM3:CCPxM0**: CCPx Mode Select bits

0000 = Capture/Compare/PWM disabled (resets CCPx module)
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, set output on match (CCPxIF bit is set)
1001 = Compare mode, clear output on match (CCPxIF bit is set)
1010 = Compare mode, generate software interrupt on match (CCPxIF bit is set, CCPx pin is unaffected)
1011 = Compare mode, trigger special event (CCPxIF bit is set, CCPx pin is unaffected); CCP1 resets TMR1; CCP2 resets TMR1 and starts an A/D conversion (if A/D module is enabled)
11xx = PWM mode

The timer module's prescaler is set by making the bit T2CKPS0 as high and T2CKPS1 as low the bit TMR2ON is set to start the timer.

T2CON: TIMER2 CONTROL REGISTER (ADDRESS 12h)

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0
bit 7							bit 0

bit 1-0 **T2CKPS1:T2CKPS0**: Timer2 Clock Prescale Select bits

00 = Prescaler is 1
01 = Prescaler is 4
1x = Prescaler is 16

To set the Frequency of the PWM signal. The value of the frequency has to be written to the PR2 register. The desired frequency can be set by using the below formulae:

```
PWM Period = [(PR2) + 1] * 4 * TOSC * (TMR2 Prescale Value)
```

Rearranging these formulae to get PR2 will give

```
PR2 = (Period / (4 * Tosc * TMR2 Prescale )) - 1
```

Our PWM signal has 10-bit resolution hence this value cannot be stored in a single register since our PIC has only 8-bit data lines. So we have use to other two bits of CCP1CON<5:4> (CCP1X and CCP1Y) to store the last two LSB and then store the remaining 8 bits in the CCPR1L Register.

The PWM duty cycle time can be calculated by using the below formulae:

```
PWM Duty Cycle = (CCPRIL:CCP1CON<5:4>) * Tosc * (TMR2 Prescale Value)
```

Rearranging these formulae to get value of CCPR1L and CCP1CON will give:

```
CCPRIL:CCP1Con<5:4> = PWM Duty Cycle / (Tosc * TMR2 Prescale Value)
```

The value of our ADC will be 0-1024 we need that to be in 0%-100% hence, PWM Duty Cycle = duty/1023. Further to convert this duty cycle into a period of time we have to multiply it with the period (1/ PWM_freq)

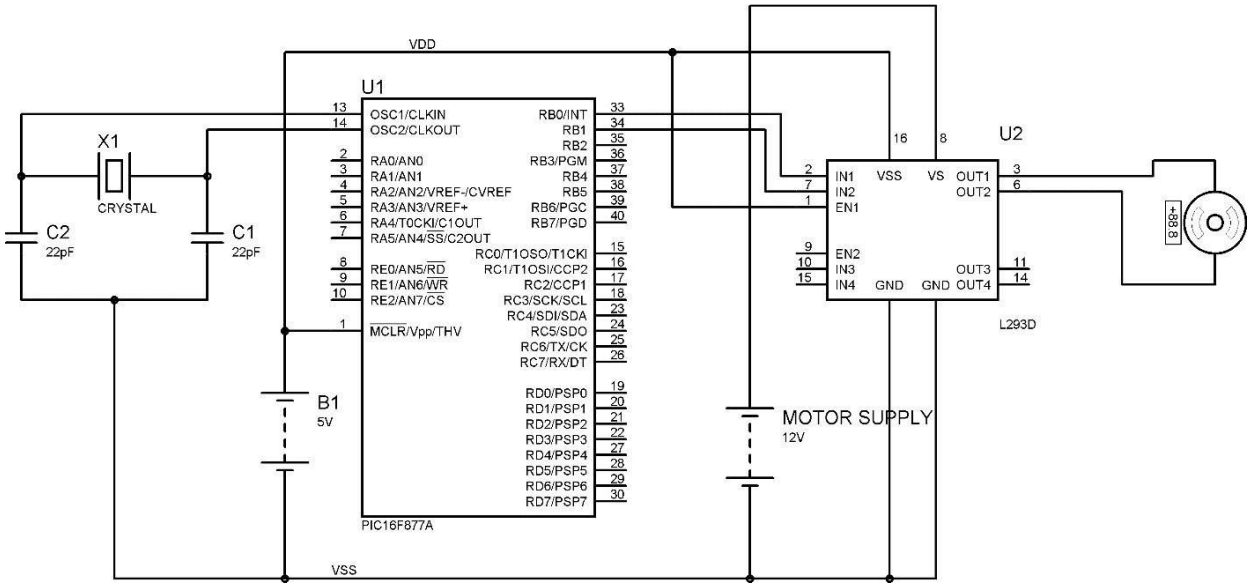
We also know that Tosc = (1/PWM_freq), hence..

```
Duty = ( ( (float)duty/1023) * (1/PWM_freq) ) / ( (1/_XTAL_FREQ) * TMR2PRESCALE) ;
```

Resolving the above equation will give us:

```
Duty = ( (float)duty/1023) * (_XTAL_FREQ / (PWM_freq*TMR2PRESCALE));
```

Circuit Diagram:



/* Calculations

* Fosc = 48MHz

* PWM Period = [(PR2) + 1] * 4 * TMR2 Prescale Value / Fosc

* PWM Period = 200us

* TMR2 Prescale = 16

* Hence, PR2 = 149 or 0x95

* Duty Cycle = 10% of 200us

* Duty Cycle = 20us

* Duty Cycle = (CCPR1L:CCP1CON<5:4>) * TMR2 Prescale Value / Fosc

* CCP1CON<5:4> = <1:1>

* Hence, CCPR1L = 15 or 0x0F

Program:

```
#include<p18f4550.h>
unsigned  char
count=0; bit
TIMER,SPEED_UP;


void timer2Init(void)

{

   T2CON  =  0b00000010;         //Prescalar = 16; Timer2
   OFF PR2                        =  0x95;  //Period Register
}


void delay(unsigned int time)

{

   unsigned int i,j;
   for(i=0;i<time;i++)
      for(j=0;j<1000;j++);

}


void main(void)

{


   unsigned int i;
   TRISCbits.TRISC1      = 0;          //RC1 pin as output
   TRISCbits.TRISC2      = 0;          //CCP1 pin as
                                       output
   LATCbits.LATC1        = 0;

   CCP1CON  =  0b00111100;            //Select PWM mode; Duty cycle
LSB CCP1CON<4:5> = <1:1>
   CCPR1L  =  0x0F;                  //Duty cycle
   10% timer2Init();               //Initialise Timer2
   //Interrupt_Init();                //Initialise interrupts
```

```
    //SPEED_UP = 1;
    TMR2ON = 1;                    //Timer2
    ON while(1)                 //Loop forever
    {
       for(i=15;i<150;i++)
       {
          CCPR1L = i;
          delay(100);
       }
       for(i=150;i>15;i--)
       {
          CCPR1L = i;
          delay(100);
       }
    }
}
```

Conclusion:

Course – Microcontrollers
Academic Year 2025-26
Semester 1

EXPERIMENT NO. - 10

TITLE : Case study on LED Interfacing with Proteus

AIM/ OBJECTIVE : To simulate LED interfacing using Proteus

HARDWARE/SOFTWARE USED

- Keil uVision3 (for Embedded C coding and simulation)
- Proteus Design Suite (for circuit simulation)

Theory:

8051 Microcontroller (AT89C51 or similar)

LED interfacing with a microcontroller helps visualize logic levels and memory operations in embedded systems. LEDs act as binary indicators—ON for logic 1 and OFF for logic 0. In this experiment, data is written from internal memory and transferred to a port where LEDs are connected. Using embedded C, we perform memory transfers and reflect the data pattern on the LED array.

Proteus Design :-

Proteus Design Suite is a powerful electronic design automation (EDA) software developed by Lab center Electronics. It is widely used for simulating electronic circuits and embedded systems before actual hardware implementation. Proteus integrates several modules, with the most commonly used being ISIS (for schematic capture and circuit simulation), ARES (for PCB layout design), and VSM (Virtual System Modelling) which allows real-time simulation of microcontrollers such as Arduino, PIC, AVR, ARM, and 8051. With Proteus, users can draw circuit diagrams, place components from a vast built-in library, and simulate the behavior of digital and analog circuits as well as complete microcontroller-based systems. One of its major strengths lies in the ability to load compiled code (like HEX files) into virtual microcontrollers, enabling users to verify and debug their firmware without needing the physical hardware.

This makes Proteus especially valuable in educational environments, prototype development, and project visualization. Additionally, ARES allows users to translate these circuits into PCB layouts, complete with 3D previews and export options for manufacturing. In essence, Proteus offers an end-to-end solution for electronics design, making it an essential tool for hobbyists, students, and professionals alike.

Proteus software

- Test your circuit **without physical components**
- Debug embedded code **without burning microcontrollers**
- Save time and cost in prototyping
- Ideal for **academic projects**, **IoT systems**, and **industrial automation**

CONCLUSION: