INDEX

EXPRIMENT NO - 1

Page No.
Date

AIM - Write an Embedded C program to addition of two Numbers

Software Used - MPLAB X IDE V5.45

Theory -

MPLAB IDE is a software program that runs of a PC to develop applications for microchip Micro-controllers. It is called an Integrated Development Environment, or IDE, because it provides a single integrated "environment" to develop code for embedded microcontrollers.

Differences Between an Embedded Controller and a PC

- The main difference between Embedded controller and PC is that the embedded controller is dedicated to one specific task or set of tasks. A PC is designed to run many different types of programs and to connect to many different external devices.

Data types in C language.

- There are several types of data that can be used in C programming language. A table below shows the range of values which these data can have when used in their basic form.

Variables.

- any number changing its value during program operation is called a variable. Simply put,

Page No.
Date

if the program adds the two numbers (Number 1, and Number 2), it is necessary to have a value to represent what we in everyday life call the sum in this case number 1, number 2 and sum are variables.

Declaring variables.

- Variable name can include any of the alphabetical characters A-Z (a-z), the digit 0-9 and the underscore character '_' The compiler is case sensitive and differentiates between capital and small letters, function and variable names usually contain lower case characters, while constant name contain uppercase characters.
- Variable names must not start with a digit
- some of the names cannot be used as variable names as already being used by the compiler itself, such names are called the key words

Type	size (bits)	Arithmetic Types.
bit	1	Unsigned integer
char	8	signed or unsigned integer
unsigned char	8	Unsigned integer
short	16	signed integer
Unsigned short	16	Unsigned integer
int	16	signed integer
Unsigned int	16	Unsigned integer
short long	24	signed integer
Unsigned short long	24	Unsigned integer
long	32	signed integer
unsigned long	32	Unsigned integer
float	24	signed integer real
double.	24 or 32	real

Page No.
Date

procedure -

step 1 - creating a new object
- Go to the file tab
- click on new project
- step 1 : choose project :
- select : microchip Embedded -> standalone project. click Next.

step 2 - select device :
- select family -> Advanced 8 Bit MCU (PIC18)
- select : Device : PIC18F4550. click Next.

step 3 : select tool : Simulator. click Next.

step 4 : select compiler -> XC8 click Nex

step 5 : select project Name and folder.
- Give project Name
- select project Location using Browse Button
- Uncheck set as main project option.
- click finish.

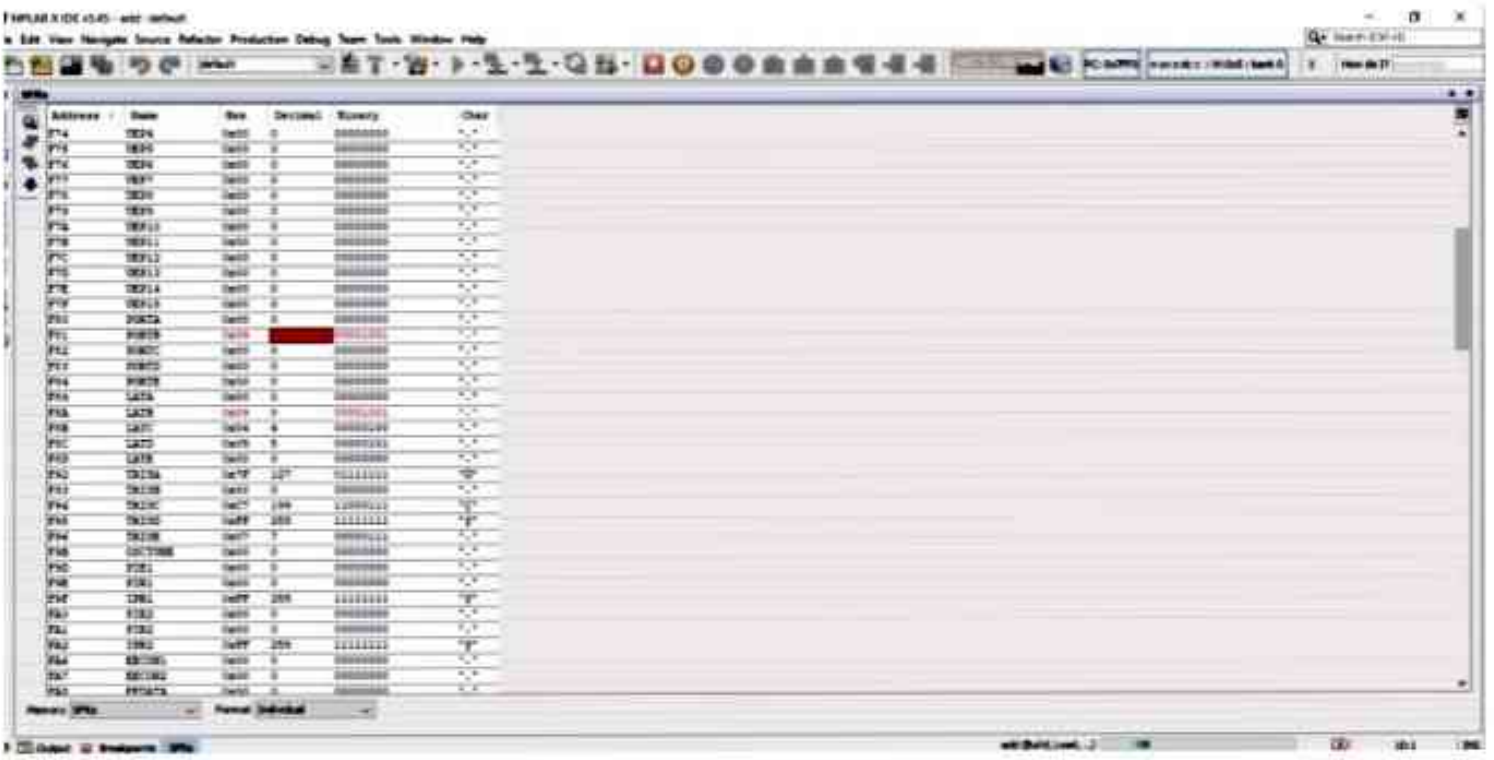
step 6 : creating a new source file
- Go to the project location in the project window.
- click the + sign to open the project space
- Right click source file folder (for a c file)
- New -> c source file.

Step 7 : select Build or clean and Build.
- ~~debug the project location in the project window~~
- Debug the project - check for errors - Build successful will appear on the OUTPUT window
- Go to debug window --- Discreet debug operation - build for debugging - launch for

Addition of two numbers code

```
#include <pic18f4550.h>
void main(void)
{
unsigned int i, j, x;
TRISB=0;                    // port b as O/P
LATB=0;
i=0x04;
j=0x05;
x=i+j;
PORTB=x;
PORTC=i;
PORTD=j;

}
```

Simulation Window



Conclusion -

Thus we have study embedded c program to addition of two numbers.

Page No.
Date

EXPERIMENT NO-2

AIM - Write an Embedded C program to add array of n numbers.

Software Used - MPLAB XIDE V5.45

Theory -

An Array is defined as the collection of similar type of data items stored at contiguous memory locations. Arrays are the derived data type in C programming language which can store the primitive type of data such as int, char, double, float, etc.

- It is necessary to understand basic I/O operations of PIC18f4550 before dealing with its complexities. this presents a way to take simple output from a PIC microcontroler. this learning would also help in interfacing of external devices with the controller.
- PICC18f4550 has a total of 33 I/O (input/output) pins which are distributed among 5 ports. the following table shows the names and number of I/O pins of these 5 ports.

port Name	Number of pins	pins.
PORT A	6	RA0 - RA5
PORT B	8	RB0 - RB7
PORT C	8	RC0 - RC5, RC6 - RC7
PORT D	8	RD0 - DD7
PORT E	3	RE0 - RE2.

The 33 I/O pins of PIC18F4550 are also multiplexed with one or more alternative functions of controller's various peripherals. Each Port of a PIC microcontroller corresponds to three 8-bit Register which should be configured to use the Port for general I/O purpose. the registers are :-

1) TRISx - This is a data direction ~~are also~~ which sets the direction of each port pin as input or output.

2) PORTx - this Register is a stored the input level of pins. (High or low) when a pin configured as input, the input signal from external source is read from PORTx register.

3. LATx - This is output latch register. the data which has to be sent to external hardware as output is stored in LATx register.

PROCEDURE:

Step1: Creating a new project

- Go to the File Tab.
- Click on New Project.
- Step1: Choose Project:
- Select: Microchip Embedded -> Standalone Project. Click Next.

Step2: Select Device:

- Select: Family -> Advanced 8 Bit MCU (PIC18).
- Select: Device: PIC18F4550. Click Next.

Step3: Select Tool: Simulator. Click Next.

Step4: Select Compiler ->XC8. Click Next.

Step5: Select Project Name and Folder.

- Give Project Name.
- Select project Location using Browse Button.
- Uncheck Set as main project option.
- Click Finish.

Step6: Creating a new Source file

- Go to the Project location in the Project window.
- Click the + sign to open the project space.
- Right Click on the Source Files folder (for a C file).
- New - > C Source file.

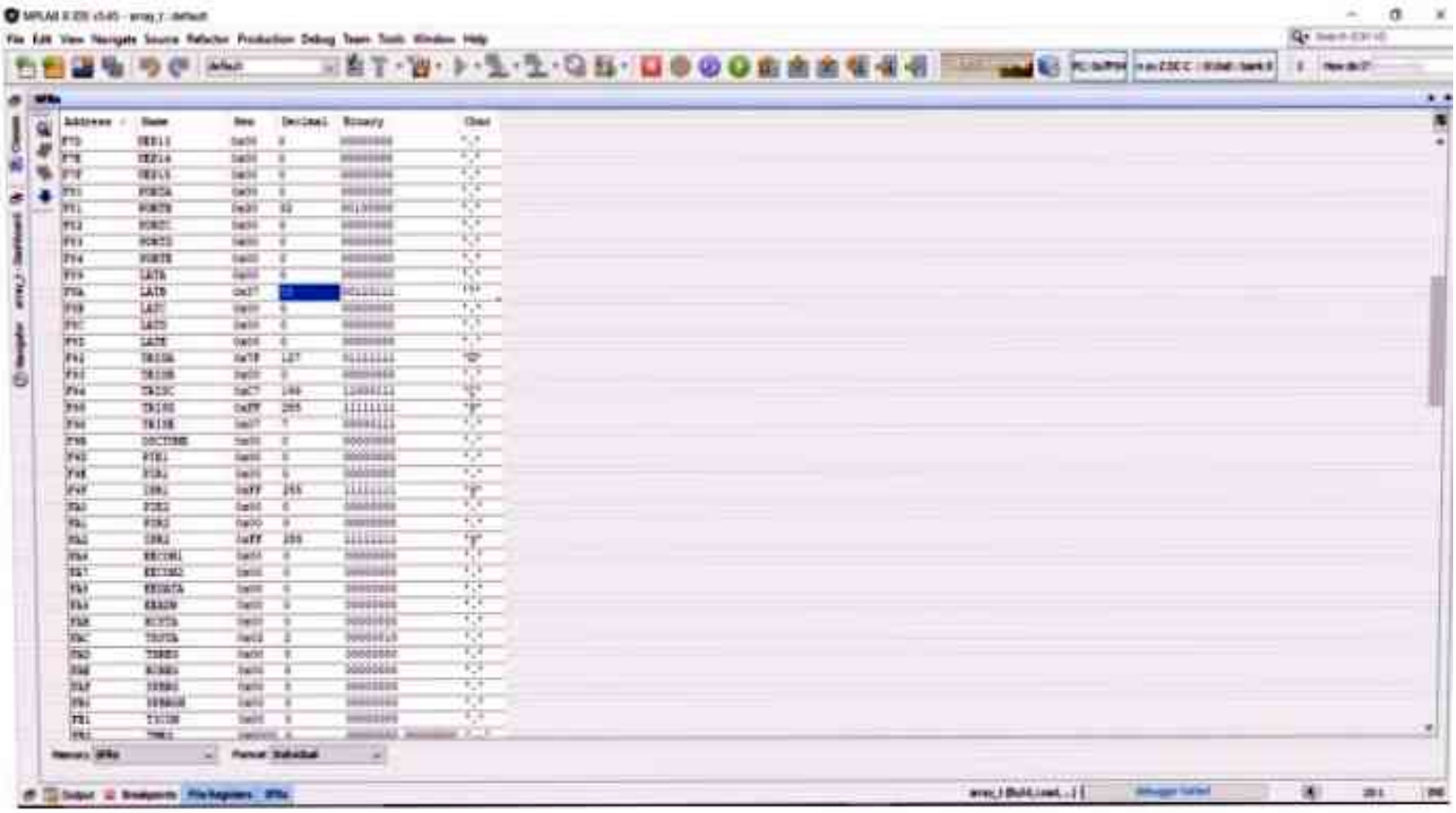
Step7: Select Build or Clean and Build.

- Debug the project – Check for the errors If No errors – Build successful will appear on the OUTPUT window
- Go to Debug window –-Discreet debug operation —build for debugging - Launch for debugger option - For single stepping press F8
- Go to window option -select target memory views-SFRs

Addition of numbers in array code

```
#include <stdio.h>
#include <stdlib.h>
#include <pic18f4550.h>

void main(void) {

  int i,sum,n;

  int number[] = {1,2,3,4,5,6,7,8,9,10};    // array of 10 numbers

  sum = 0;                                  // initialize sum as zero

  for(i=0; i<=9;i++){                       //indexing start from 0 to 9
    sum = sum+number[i];
  }

  TRISB =0;                                 //initialize Port_B as output
  PORTB = sum;                              // from sum to PORT_B

  //n = 0xFF + 0XFF;
}
```

Simulation Window



conclusion

- thus we have study embedded c program to add array of n numbers.

Page No.
Date

EXPERIMENT NO-3.

AIM - Write an Embedded c program to transfer elements from one location to another for internal memory transfer and Internal to external memory transfer.

Software Used - MPLAB X IDE V5.45.

THEORY -

memory orgnization

To used the 4550 judiciously for various applications, it is essential to understand a memory map of the device.

- There are three types of memory in PIC18 enhanced microcontroller devices.
  - Program memory
  - Data memory
  - Data EEPROM.
- As discussed, the 4550 is a Harvard architecture device. Hence concurrent access to program and data memory is enbled. the data EEPROM. for practical porposes, can be regarded as a peripheral device, since it is addressed and accessed through a set of control register.

program memory orgnization.

The 4550 implements a 21-bit program counter which is capable of addressing a 2MB program memory space. It has 32kB of flash memory and can store up to 16,384 single-word instructions.

PIC18fX550

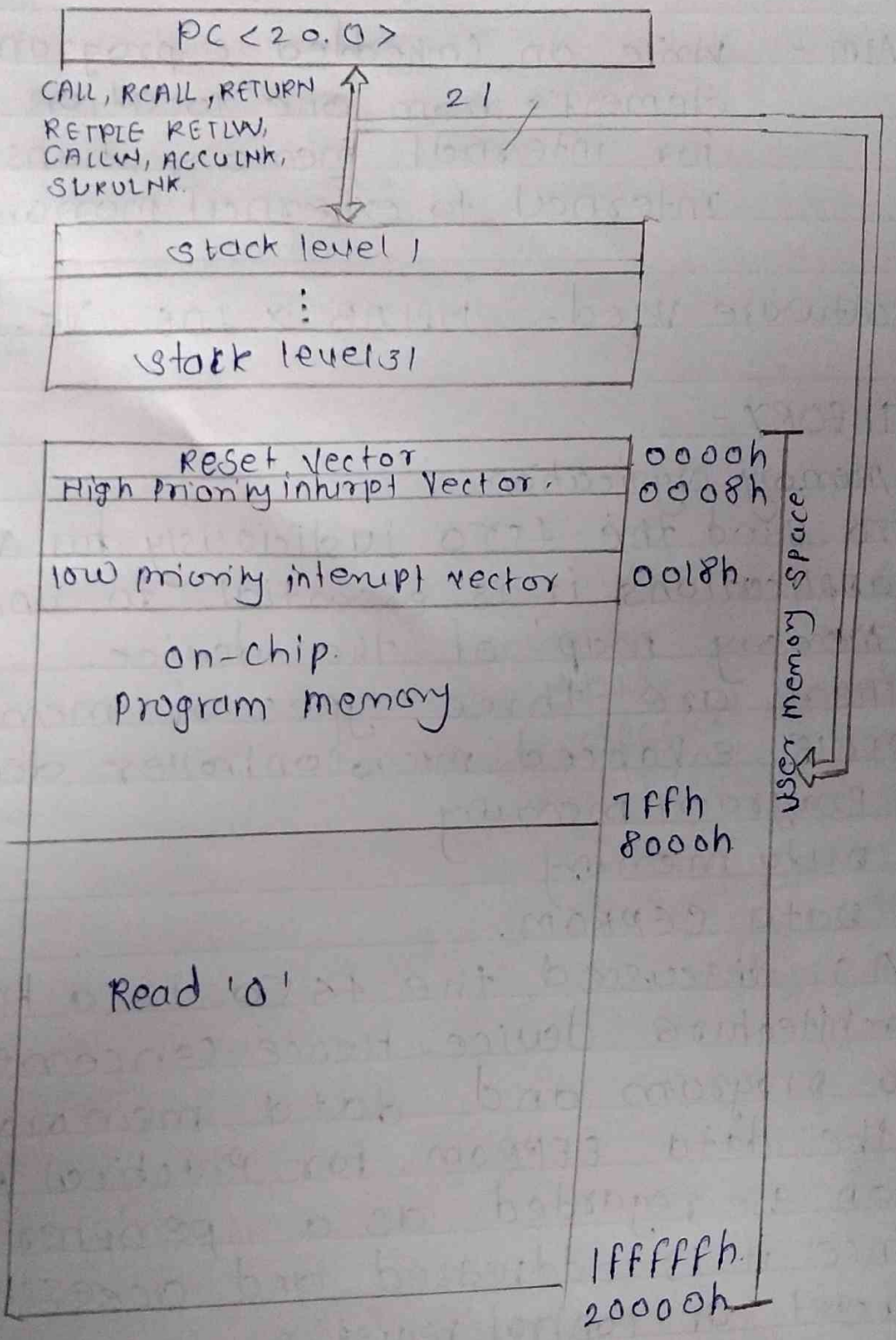
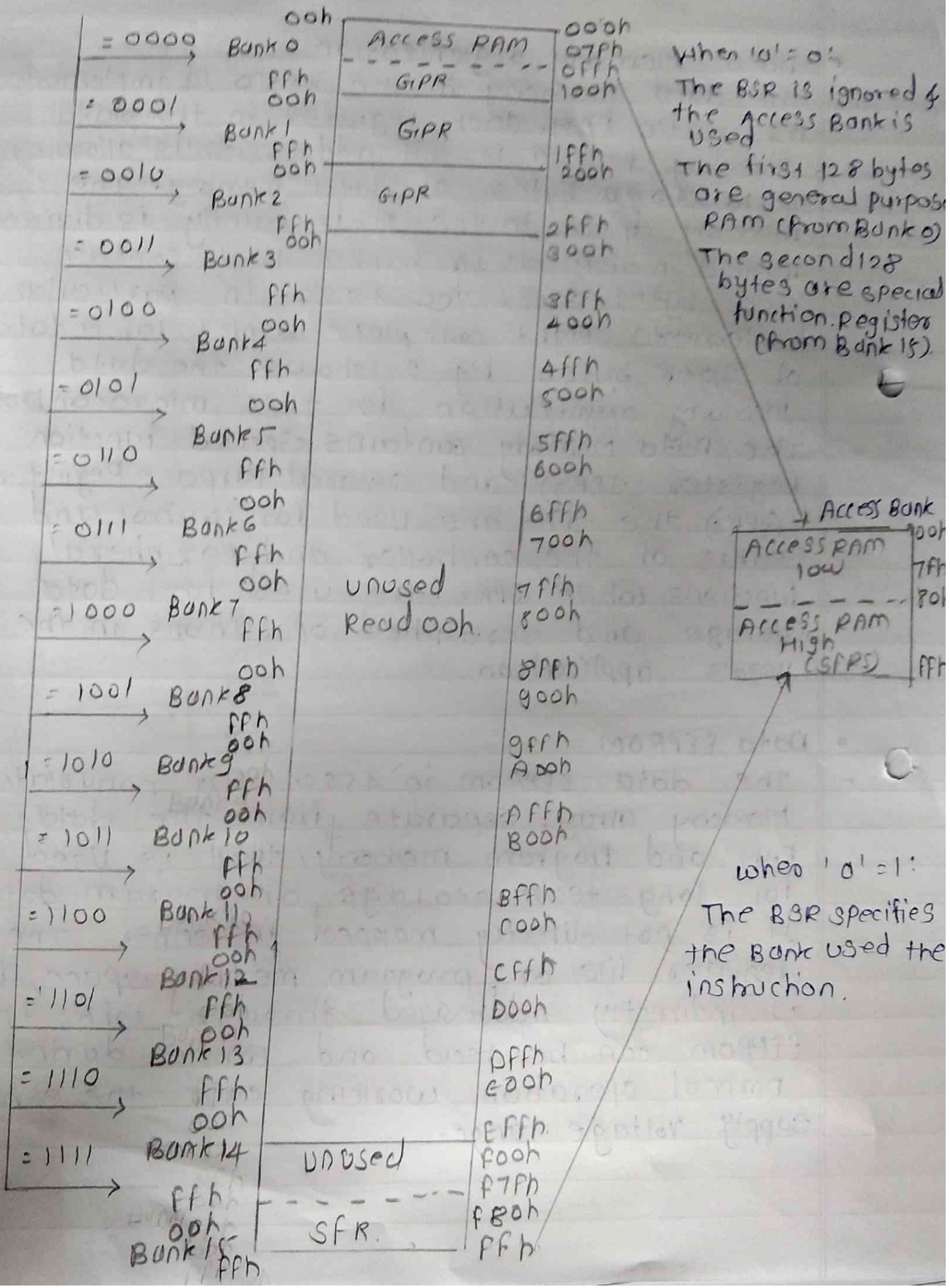


Fig 1 - program memory orgnization in PIC18f4550.

Page No.
Date

- Data Memory organization.
  - The data memory in the 4550 is implemented as static RAM. Each register in the data memory has a 12-bit address, thus allowing up to 4096 bytes of data memory. The memory space in the PIC18 family is divided into as many as 16 banks that contain 256 bytes each. the 4550 in particular implements eight complete banks, for a total of 2048 bytes. fig 2 shows the data memory orgnization for this microcontroller
  - The Data memory contains Special function Register (SFRs) and General Purpose Register (GPRs) the SFRs are used for control and status of the controller and peripheral functions, while GPRs are used for data storage and scratched operations in the user's application.

- Data EEPROM
  - The data EEPROM in 4550 is a nonvolatile memory array, separate from the data RAM and program memory, that is used for long-term storage of program data. It is not directly mapped in either the register file or program memory space, but is indirectly addressed through SFRs the EEPROM can be read and written during normal operation, working over the entire supply voltage range.


= 0000
Bank 0
= 0001
Bank 1
= 0010
Bank 2
= 0011
Bank 3
= 0100
Bank 4
= 0101
Bank 5
= 0110
Bank 6
= 0111
Bank 7
= 1000
Bank 8
= 1001
Bank 9
= 1010
Bank 10
= 1011
Bank 11
= 1100
Bank 12
= 1101
Bank 13
= 1110
Bank 14
= 1111
Bank 15
ffh
00h
Access RAM
GPR
GPR
GPR
unused
Read 00h
unused
SFR
00'0h
07Fh
080h
100h
1FFh
200h
2FFh
300h
3FFh
400h
4FFh
500h
5FFh
600h
6FFh
700h
7FFh
800h
8FFh
900h
9FFh
A00h
AFFh
B00h
BFFh
C00h
CFFh
D00h
DFFh
E00h
EFFh
F00h
F7Fh
F80h
FFh
When 'a' = 0:
The BSR is ignored & the Access Bank is used
The first 128 bytes are general purpose RAM (from Bank 0)
The second 128 bytes are special function register (from Bank 15).
Access Bank
Access RAM Low
Access RAM High (SFRs)
00h
7Fh
80h
FFh
when 'a' = 1:
The BSR specifies the Bank used the instruction.

- memory transfer -
is transfer of information from a memory word to the outside environment is called a read operation. the transfer of new information to be stored into the memory is called a write operation.

- translation. memory exchange. (TMX)
is an XML specification for the exchange of translation memory (TM) data between computer aided translation and localization tools with little or no loss of critical data. It allows the original source and target documents to be recreated from the TMX data.

PROCEDURE:

Step1: Creating a new project

- Go to the File Tab.
- Click on New Project.
- Step1: Choose Project:
- Select: Microchip Embedded -> Standalone Project. Click Next.

Step2: Select Device:

- Select: Family -> Advanced 8 Bit MCU (PIC18).
- Select: Device: PIC18F4550. Click Next.

Step3: Select Tool: Simulator. Click Next.

Step4: Select Compiler ->XC8. Click Next.

Step5: Select Project Name and Folder.

- Give Project Name.
- Select project Location using Browse Button.
- Uncheck Set as main project option.
- Click Finish.

Step6: Creating a new Source file

- Go to the Project location in the Project window.
- Click the + sign to open the project space.
- Right Click on the Source Files folder (for a C file).
- New - > C Source file.

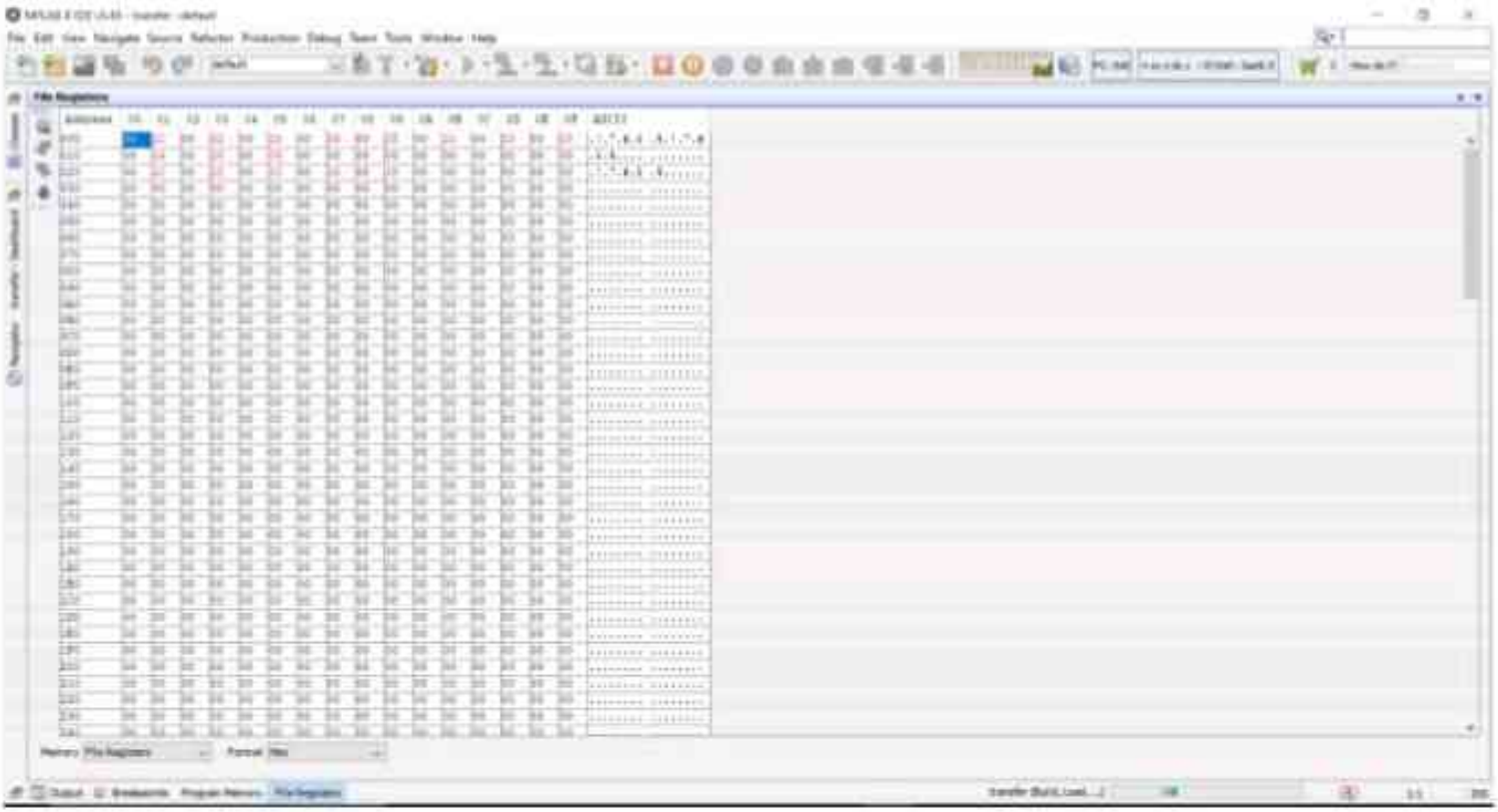
Step7: Select Build or Clean and Build.

- Debug the project – Check for the errors If No errors – Build successful will appear on the OUTPUT window
- Go to Debug window ---Discreet debug operation —build for debugging - Launch for debugger option - For single stepping press F8
- Go to window option -select target memory views-File register

Memory Transfer code

```
#include <stdio.h>
#include <stdlib.h>
#include <pic18f4550.h>
/*
 *
 */
void main(void) {

   int temp,i;
   int source1[] = {0x21,0x22,0x23,0x24,0x25};   // source mem block
   int dest[] = {0x00,0x00,0x00,0x00,0x00};      // destination mem block

   for(i=0; i<=4;i++){                    // counter = 5

    dest[i] = source1[i];                 // source to destination

   }
}
```

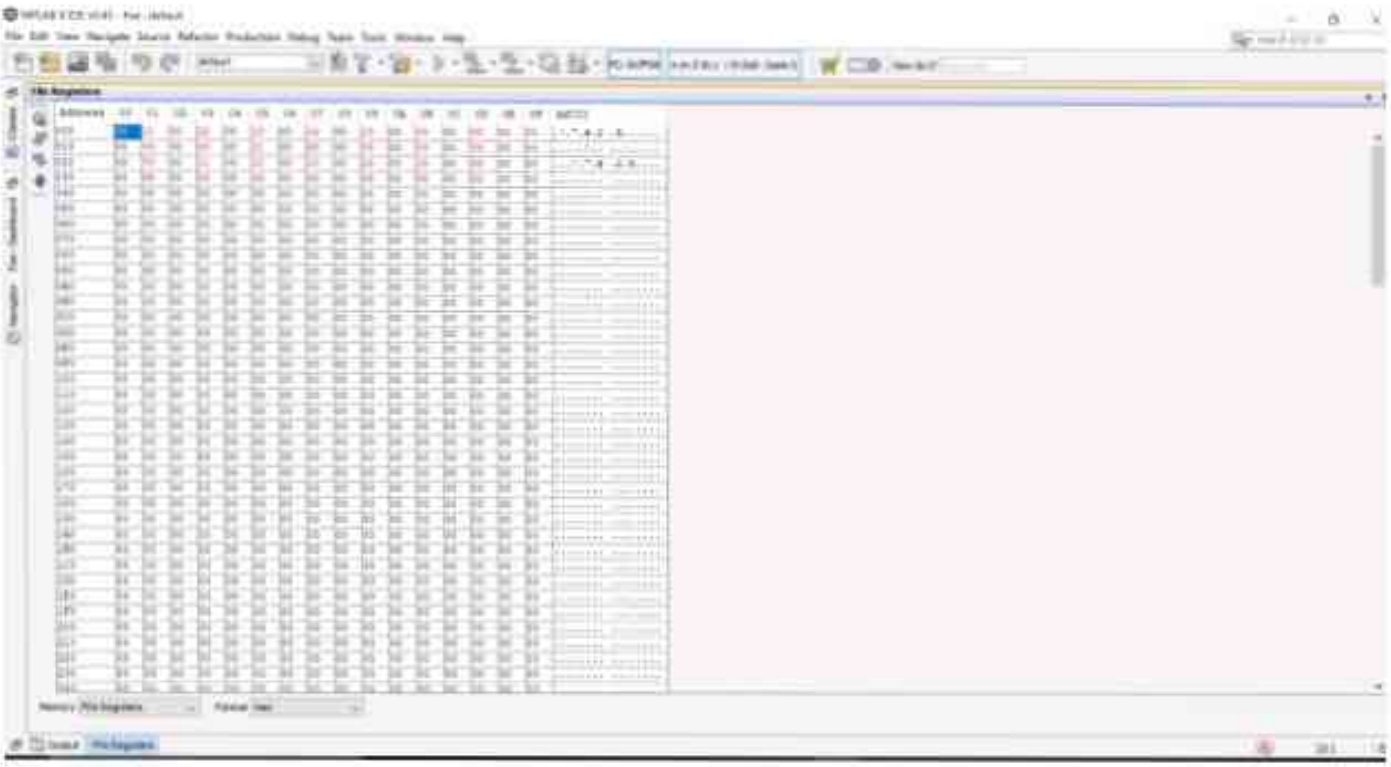
Simulation Window



Memory Exchange code

```
#include <stdio.h>
#include <stdlib.h>
#include <pic18f4550.h>
void main(void) {

   int temp,i;
   int source1[] = {0x21,0x22,0x23,0x24,0x25};
   int dest[] = {0x99,0x99,0x99,0x99,0x99};

   for(i=0; i<=4;i++){
    temp = source1[i];
    source1[i] = dest[i];
    dest[i] = temp;

   }
}
```

Simulation Window



Conclusion-

Thus we have study embedded c program to transfer elements from one location to another for internal to internal memory transfer and internal to external memory transfer.

Page No.
Date

EXPRIMENT NO - 4.

AIM - Write an Embedded C menu driven program to multiply 8 bit number by 8 bit number

Software Used - MPLAB X IDE V5.45

Theory -
Register of PIC 18f.

- The memory of the PIC is devided into a series of register each of the registers has its own adress and memory locations.
- According to the type of working and usage the register in PIC are classified as:
  - Special function Register (SFRS) - used for control and status of the controller and peripheral functions.
  - General purpose Register (GPRS) - Used for data storage and scratch-pad operations in the user's application.
  - WREG - working register (acts as an Accumulator) - Used to perform arithmatic or logical functions.
  - Status register - that the stores flags - Indicates the status of the operation done by ALU.
  - Registers - hold memory address -
    -) Bank Select Register (BSR) : 4 bit register used in direct addressing the data memory

Page No.
Date

- File Select Register (FSRs) - 16 bit register that holds the program memory address while executing programs. this means that the PIC18 family can access program address 000000 to 1FFFFFH a total of 2M bytes of code.

- Stack pointer (SP) - PIC18 has a 5 bit stack pointer. It is used to access the stack

PROCEDURE:

Step1: Creating a new project

- Go to the File Tab.
- Click on New Project.
- Step1: Choose Project:
- Select: Microchip Embedded -> Standalone Project. Click Next.

Step2: Select Device:

- Select: Family -> Advanced 8 Bit MCU (PIC18).
- Select: Device: PIC18F4550. Click Next.

Step3: Select Tool: Simulator. Click Next.

Step4: Select Compiler ->XC8. Click Next.

Step5: Select Project Name and Folder.

- Give Project Name.
- Select project Location using Browse Button.
- Uncheck Set as main project option.
- Click Finish.

Step6: Creating a new Source file

- Go to the Project location in the Project window.
- Click the + sign to open the project space.
- Right Click on the Source Files folder (for a C file).
- New - > C Source file.

Step7: Select Build or Clean and Build.

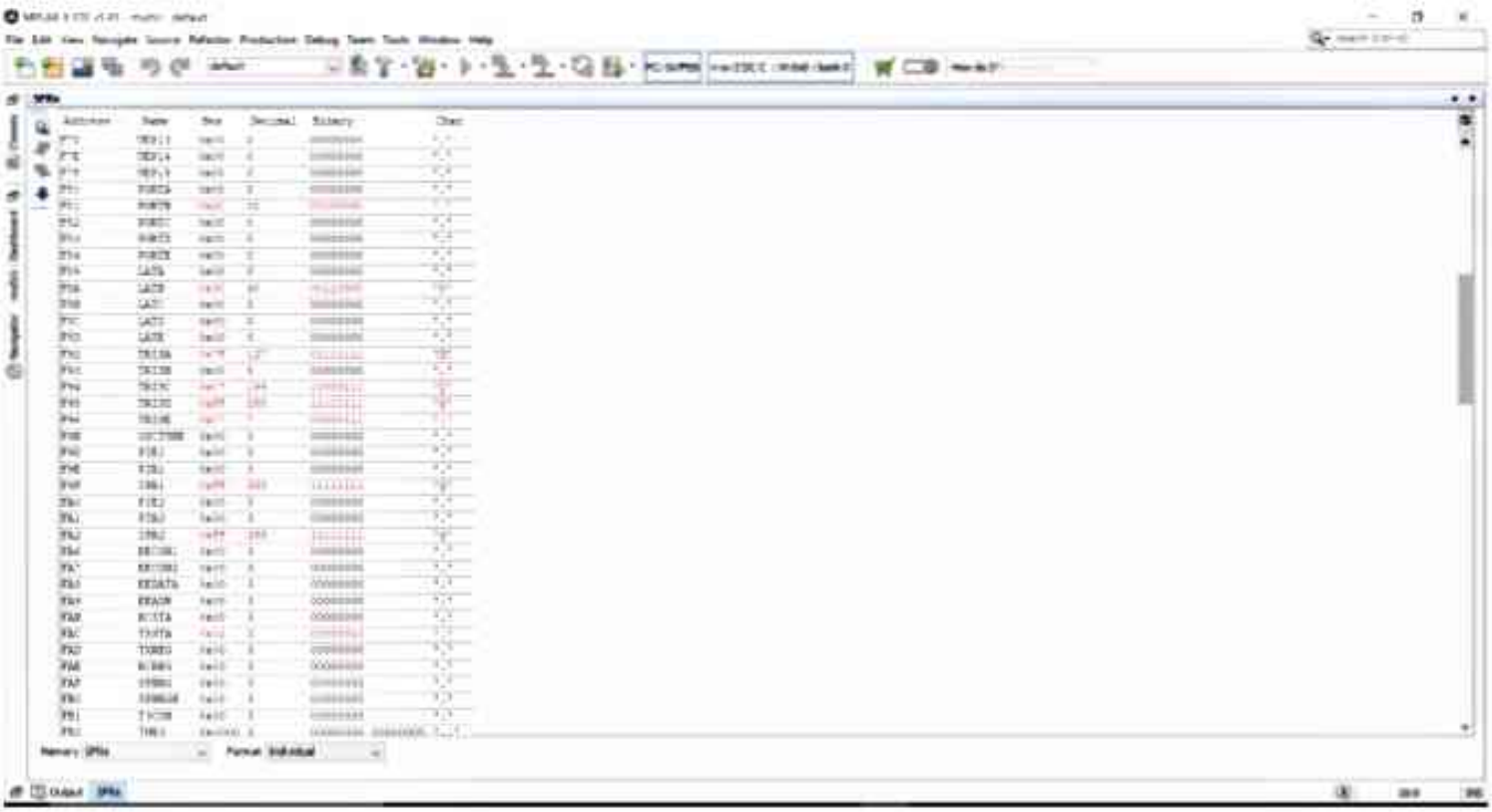
- Debug the project – Check for the errors If No errors – Build successful will appear on the OUTPUT window

- Go to Debug window ---Discreet debug operation —build for debugging - Launch for debugger option - For single stepping press F8
- Go to window option -select target memory views-SFRs

Multiplication code

```
#include <stdio.h>
#include <stdlib.h>
#include <pic18f4550.h>


//multiplication using successive addition
void main(void) {
   //static int v_mem[] = 0x55;      @0x0005
   int num1, num2;
   int result,i;


   result = 0;
   num1 = 0x23;
   num2 = 0x10;

   for(i=1; i<=num2; i++)
   {
      result = result + num1;
   }

   TRISB =0;
   PORTB = result;

}
```

Simulation window



conclusion

Thus we have study embedded c program to menu driven program to multiply 8 bit number.

EXPRIMENT NO. 5

Page No.
Date

AIM: Write an Embedded C program for sorting the numbers in ascending order.

Software Used - MPLAB X IDE V5.45

Theory -
The array can be stored in ascending order by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at the beggining. The algorithm maintains two subarrays in a given array.

1. The subarray which is already sorted
2. Remainning subarray which is unsorted.

PROCEDURE:

Step1: Creating a new project

- Go to the File Tab.
- Click on New Project.
- Step1: Choose Project:
- Select: Microchip Embedded -> Standalone Project. Click Next.

Step2: Select Device:

- Select: Family -> Advanced 8 Bit MCU (PIC18).
- Select: Device: PIC18F4550. Click Next.

Step3: Select Tool: Simulator. Click Next.

Step4: Select Compiler ->XC8. Click Next.

Step5: Select Project Name and Folder.

- Give Project Name.
- Select project Location using Browse Button.
- Uncheck Set as main project option.
- Click Finish.

Step6: Creating a new Source file

- Go to the Project location in the Project window.
- Click the + sign to open the project space.
- Right Click on the Source Files folder (for a C file).
- New - > C Source file.

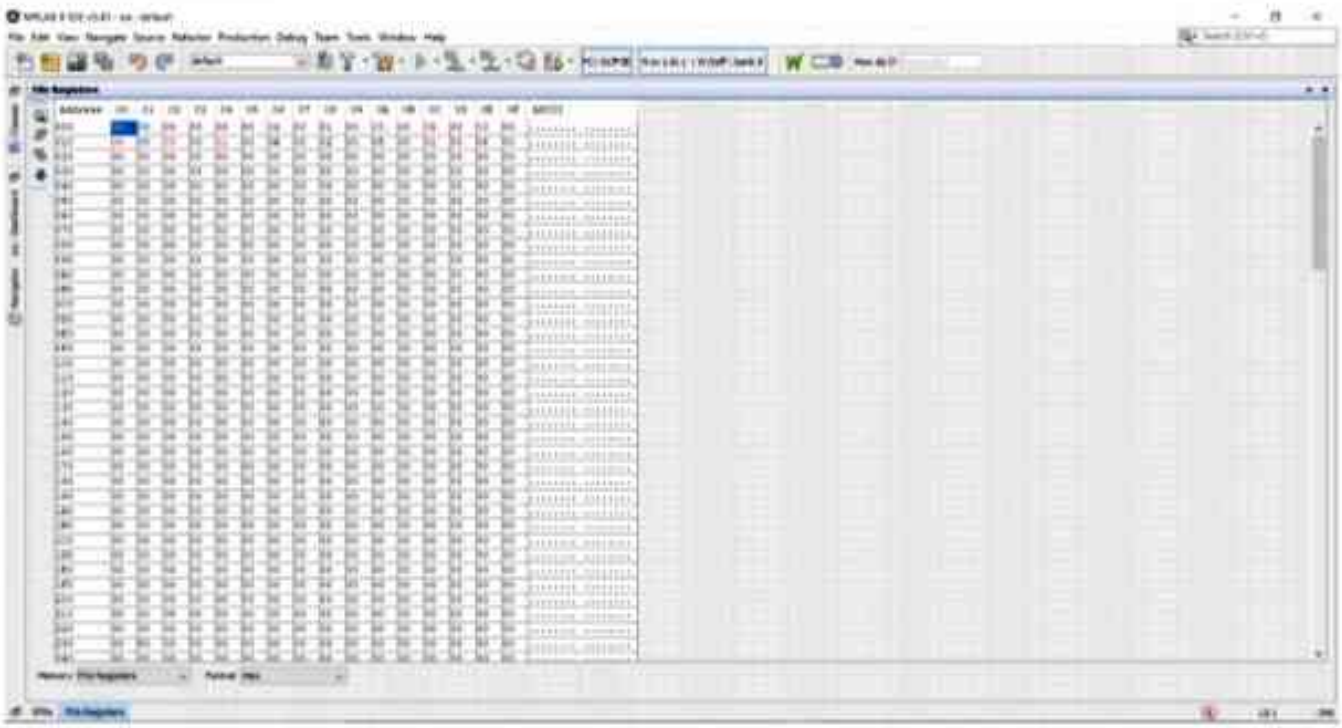
Step7: Select Build or Clean and Build.

- Debug the project – Check for the errors If No errors – Build successful will appear on the OUTPUT window
- Go to Debug window ---Discreet debug operation —build for debugging - Launch for debugger option - For single stepping press F8
- Go to window option -select target memory views-File register

Sorting the array in ascending order code

```
#include <stdio.h>
#include <stdlib.h>
#include <pic18f4550.h>


void main(void) {
   int i,j,temp;
   int num_asc[] = {10,2,5,1,6};

   for(i=0; i<=4; i++){                // point to LHS number
      for(j=i+1;j<=4;j++)              // point to RHS number
      if (num_asc[i] > num_asc[j]){    // if LHS > RHS , change the position
         temp = num_asc[i];
         num_asc[i] =num_asc[j];
         num_asc[j]= temp;
      }
   }
}
```

Simulation Window



Conclusion-

Thus we have study embedded c program to sorting the numbers in ascending order.

EXPRIMENT NO-6

Page No.
Date

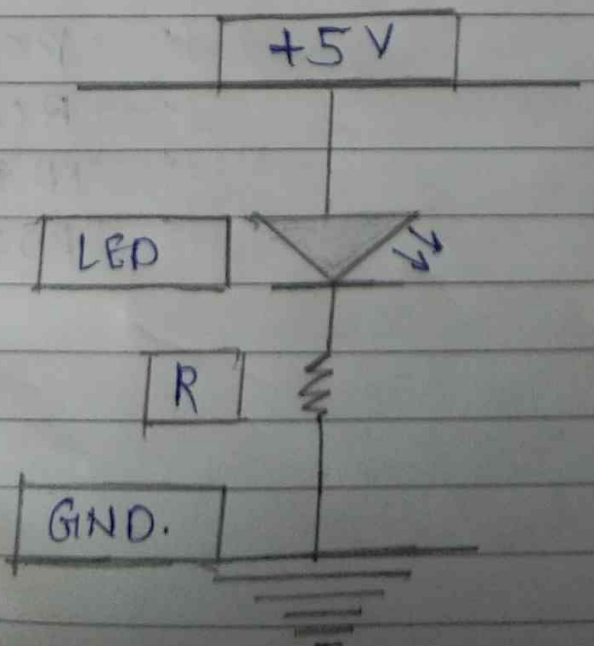
AIM- Write a embedded c program to interface PIC 18fxxx with LED & blinking it using specified delay.

Software used - MPLAB IDE V8.89 and proteus 8 professional

Theory -

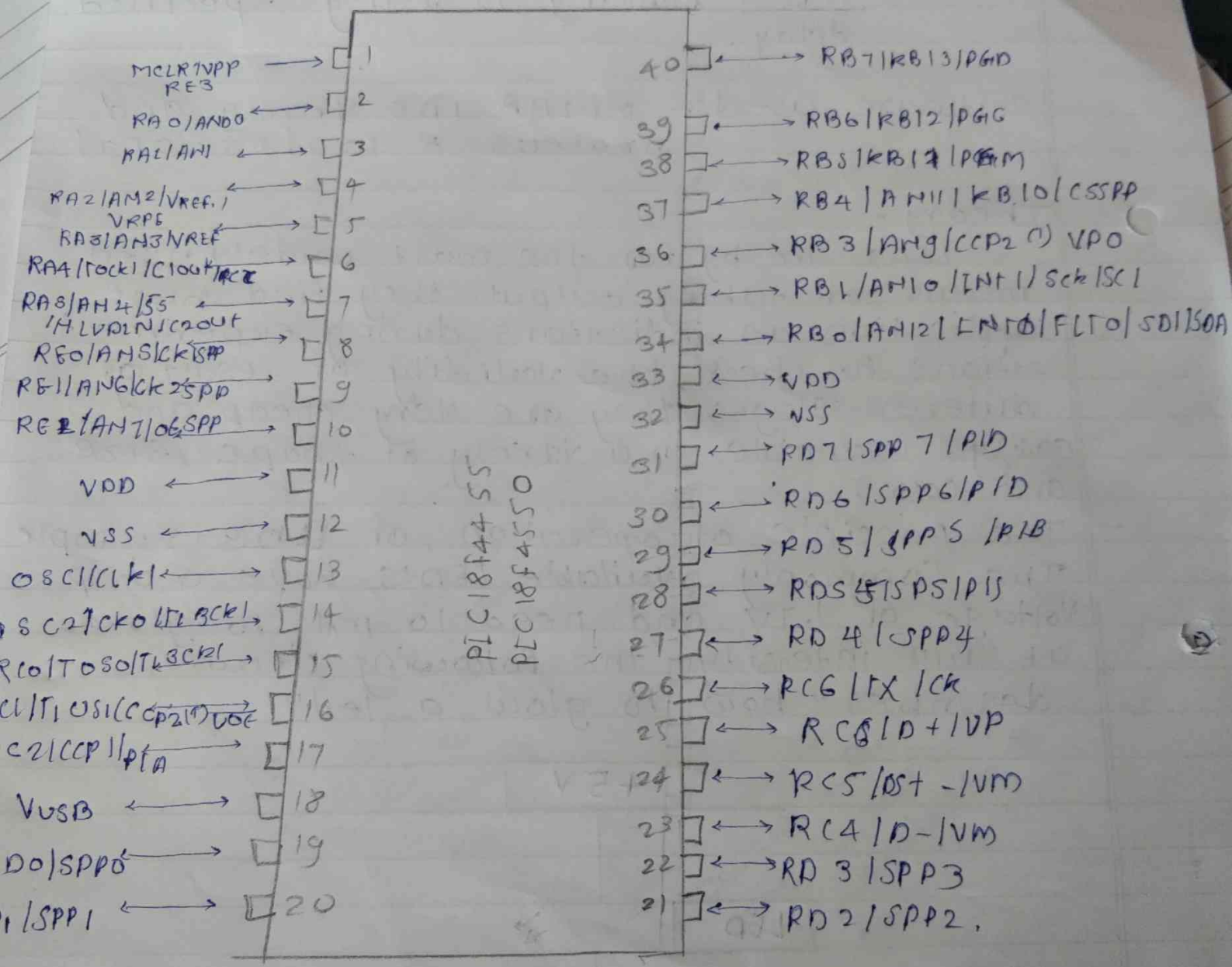
LEDs are by far the most widely used means of taking output. they find huge application as indicators during experiments ations to check the validity of result at different stages. they are very cheap and easily available in a variety of shape, size and colors.

- The principle of operation of LEDs is simple The Commonly available LED's have a drop Voltage of 1.7V and need 10 mA to glow at full intensity. the following circuit describes "how to glow a led".



Pin diagram of PIC18F4550.

40-pin PDIP



Procedure -

Step 1 -

- proteus 8 professional - click on file - new project provide path - next
- Go into schematic Picture - select devices.
- Go to terminal - select VCC and GND.

Step 2 -

- MPLAB IDE V8.89 - Go to project wizard - select device PIC18F4550 - Next - Active toolsuit - microchip C18 Toolsuite - next.
- creat new project file - create new folder click on next - finish
- Go to file - new - write code - save file with c extension - add this file in source file - Right click on source file - add c file.

Step 3 -

- ~~Go to project - Built project HEX file is created Burn the HEX file in proteus 8 Professional~~
- Go to project - Built add project.
- check for the errors If No errors - Built the ~~Hex file in proteus~~ 8 successfull will appear on the output window.

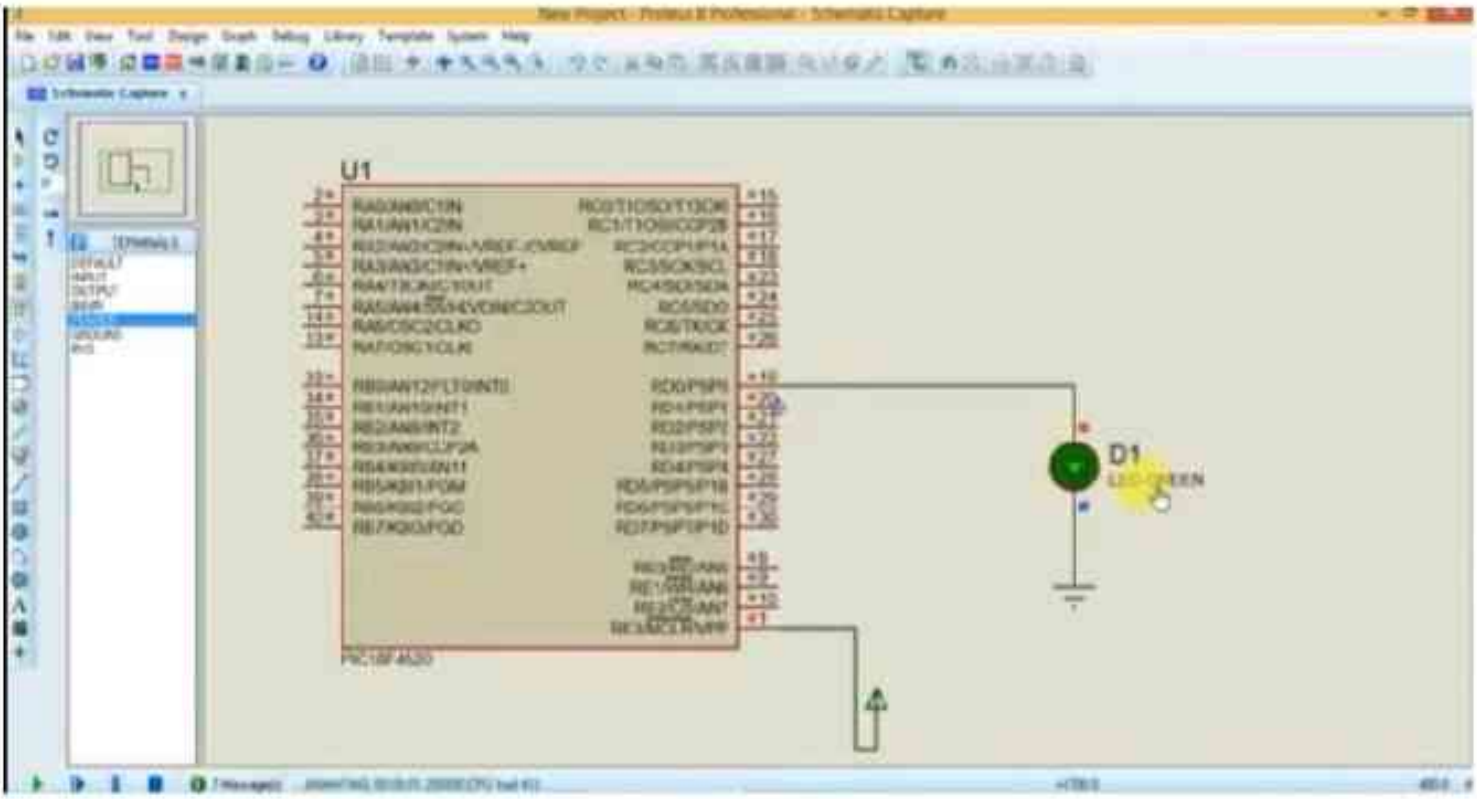
Step 4 -

- After Built project HEX file is created - Burn the HEX file in proteus 8 proffesional.

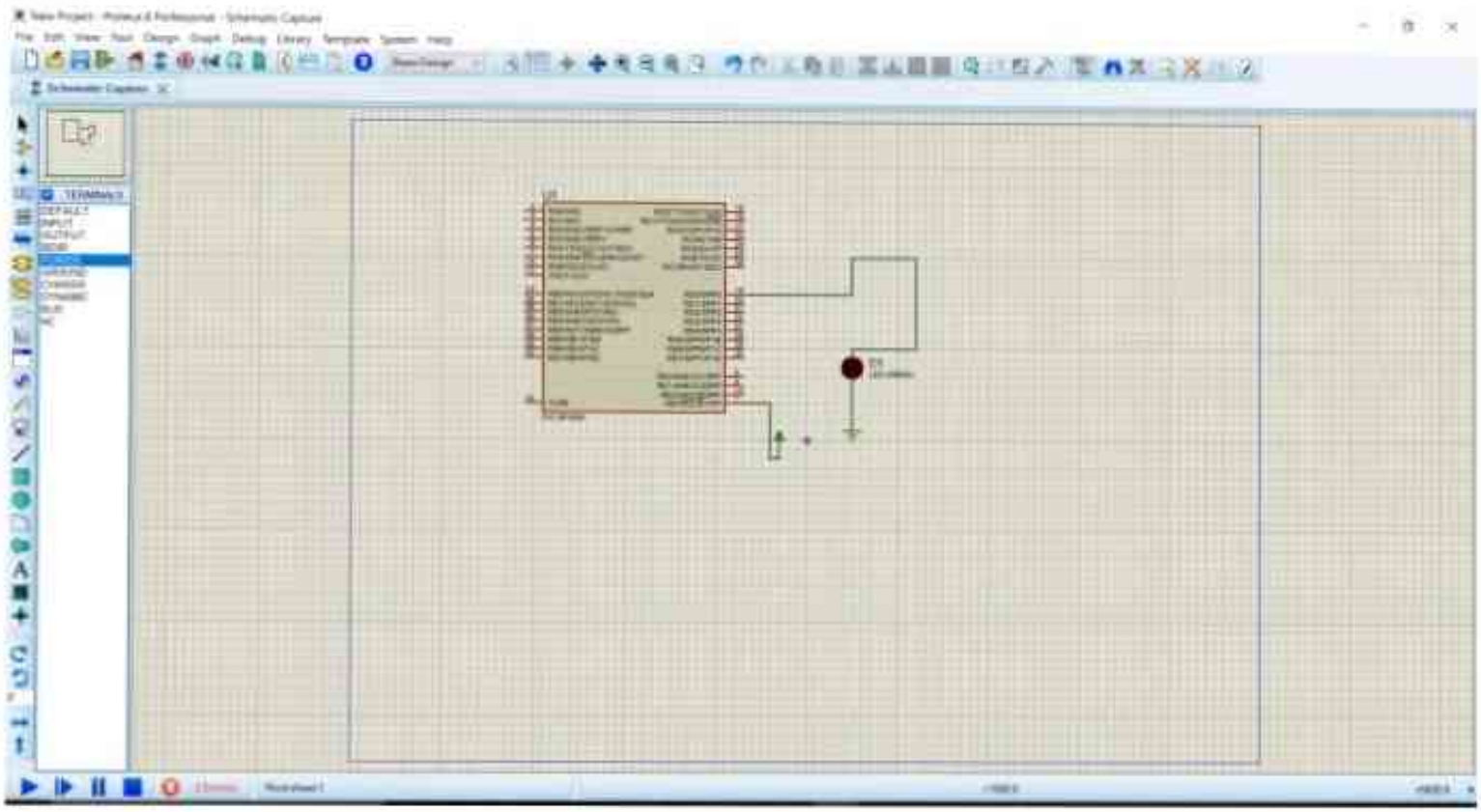
LED interfacing code

```
#include<p18f4550.h>
void DELAY();
#pragma config OSC=HS
#pragma config PWRT=OFF
#pragma config WDT=OFF
#pragma config DEBUG=OFF,LVP=OFF
void main ( )
{
 TRISDbits.RD0=0x00;    //set RD0 pin directin as an output
 while(1)               //forever loop
{
 PORTDbits.RD0=0x01;     //set RD0 pin a HIGH
 DELAY();
 PORTDbits.RD0=0x00;     //clear RD0 pin
 DELAY();                //call delay
}
}
 void DELAY( )
{
 unsigned int i;
 for (i=0; i<10000;i++);
}
```

Simulation window



Proteus Simulation Window



conclusion -

Thus we have study embedded c program to interface PIC 18fxxx with LED & blinking it using specified delay.

Page No.
Date

EXPERIMENT No-7.

AIM- write an embedded c program for timer programming ISR based buzzer on/off and external interrupt input switch press, output at relay.

Software used- MPLAB IDE V8.89 and proteus 8 professional

Theory -

PIC interupt.

PIC microcontroller consists of both hardware and software Interrupts, If the interrupts are generated by external hardware at certain pins of microcontroller, or by inbuit devices like timer, they are called Hardware interrupts. while software interrupt are generated by a piece of code in the program.

IVT table-

- IVT Table stands for Interrupt Vector table It holds the addresses of the interrupt service routines ~~hardware~~. for various interrupt supported by the microcontroller.

Table. 6.3.1 Shows the interrupt vector table fore the PIC 18.

Interrupt	Vector Address (RAM location)
power-on Reset	0000H
High priority Interrupt	0008H
low priority interrupt	0018H.

- programming of timer using Interrupt.
- The timer interrupt is generated when the time register overflows from ffh to 00h in 8-bit mode or ffffh to 0000h in 16-bit mode. this overflow sets the TMRxIF bit. the timer interrupt can be masked by cleaning the TMRxIE bit.
- The TMRxIF bit must be cleared in software by the Timer module Interrupt service Routine before re-enbling this interrupt.
- Note- * may take value 0, 1, 2, 3.
- Table 6.9.1 shows the timers and their corresponing interrupt enable flag bits and interrupt flag bit.

Timer.	Interrupt Enable		Interrupt Flag	
	Bit	Register	Bit	Register
Timer0	TMR0IE	INTCON	TMR0IF	INTCON
Timer 1	TMR1IE	PIE1	TMR1IF	PIR1
Timer 2	TMR2IE	PIE1	TMR2IF	PIR1
Timer 3.	TMR3IE	PIE2	TMR3IF	PIR2

Table 6.9.1 Timers and their corresponding interrupt enable flag bits and interrupt flag bits.

External hardware interrupts-

- PIC18 microcontroller has three external hardware interrupts INT0, INT1, and INT2. they are available on PORTB pins RB0, RB1 and RB2.
- These interrupts are edge triggered interupts i.e. triggered by either a rising edge or by falling edge.

falling edge.

PROCEDURE:

Step 1:

- Proteus 8 Professional- click on file -new project-provoide path-next
- Go into Schematic Picture-Select Devices
- Go to terminal -select VCC and GND

Step 2:

- MPLAB IDE V8.89 -Go to Project-Project Wizard-select device PIC18F4550-next-Active toolsuit-Microchip C18 Toolsuite-next
- Create new Project file-Create new Folder -Click on next -Finish
- Go to file-new-write Code-Save file with .C extension -add this file in source File-Right Click on Source file- add C file

Step 3:

- Go to project-Build all Project
- Check for the errors If No errors – Build successful will appear on the OUTPUT window

Step 4:

- After Build project HEX file is created-Burn the HEX file in Proteus 8 Professional.

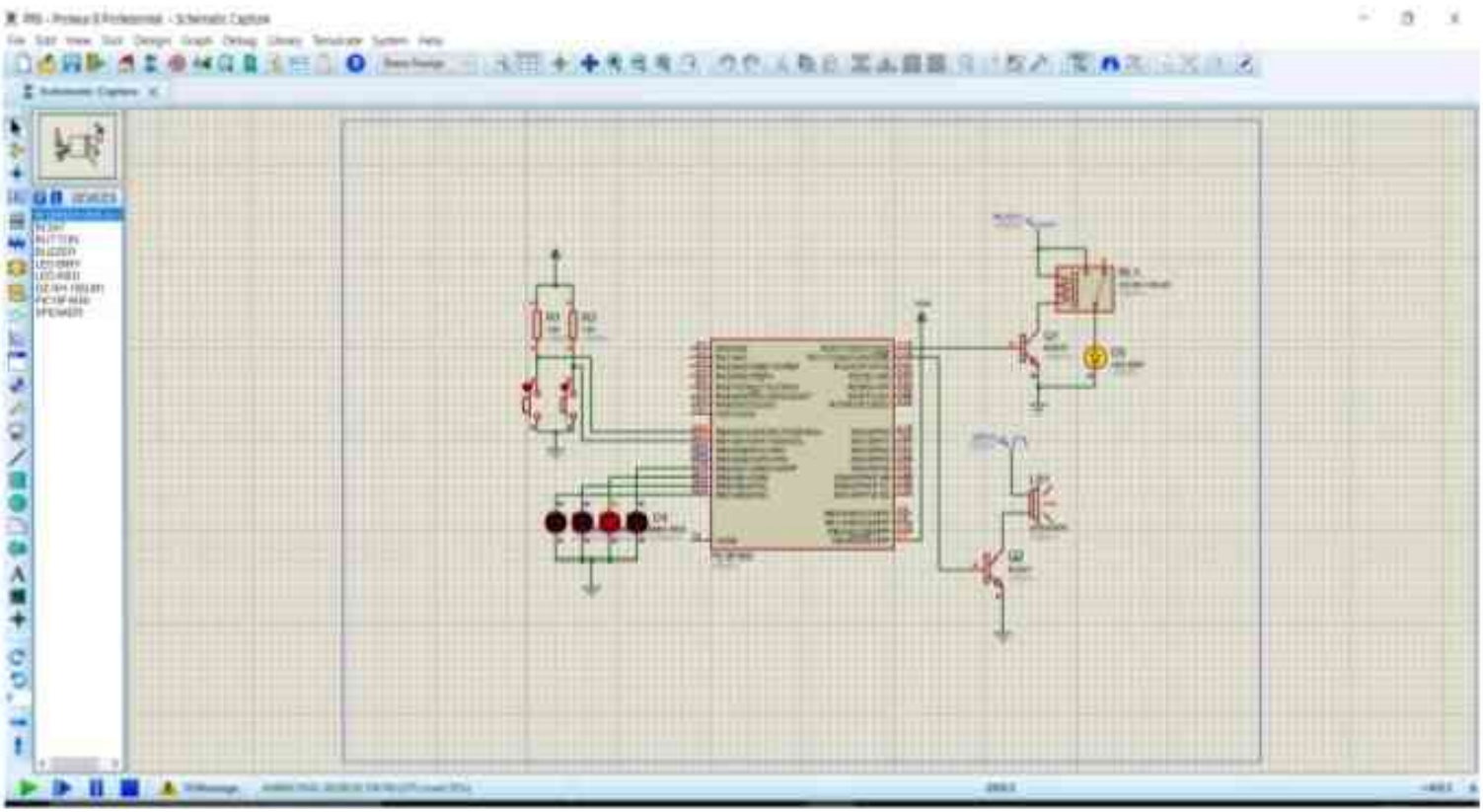
Program

```
#include <p18f4550.h>

/*The following lines of code perform interrupt vector relocation to work with the USB bootloader. These must be used with every application program to run as a USB application.*/

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)

{

}


#pragma code

#pragma code _LOW_INTERRUPT_VECTOR = 0x1018

void low_ISR (void)

{

}

#pragma code

/*End of interrupt vector relocation*/


/*Start of main program*/


void MsDelay (unsigned int time)

{

        unsigned int i, j;
```

```
        for (i = 0; i < time; i++)
                for (j = 0; j < 710; j++);          /*Calibrated for a 1 ms delay in MPLAB*/
}

#define lrbit PORTBbits.RB0
#define rlbit PORTBbits.RB1
#define buzzer PORTCbits.RC0
#define relay PORTCbits.RC1


void main()
{
  unsigned char val=0;
  INTCON2bits.RBPU=0;
        ADCON1 = 0x0F;

        TRISBbits.TRISB0=1;
        TRISBbits.TRISB1=1;
        TRISCbits.TRISC0=0;
        TRISCbits.TRISC1=0;

        TRISB = 0x03;
        PORTB = 0x10;
        buzzer = 0;

        relay = 0;

        while (1)
        {
                if (!(lrbit))
                val = 1;
```

```
        if (!(rlbit))
        val = 0;

        if (val)
        {
                buzzer = 1;
                relay = 1;
                LATB = LATB >>1;
                MsDelay(250);
                if (LATB == 0x10 ||LATB == 0x00)
                {
                        LATB = 0x80;
                        MsDelay(250);
                }
        }
        else
        {
                buzzer = 0;
                relay = 0;
                LATB = LATB <<1;
                MsDelay(250);
                if (LATB == 0x80 ||LATB == 0x00)
                {
                        LATB = 0x10;
                        MsDelay(250);
                }
        }
}
```

Proteus Simulation Window



Conclusion - Thus we have study embedded a program for timer programming ISR based buzzer on/off and external Interrupt input switch press, output at a relay.

Page No.
Date

EXPRIMENT NO - 8

AIM - Write an embedded C program for LCD interfacing with PIC 18FXXX.

Software Used - MPLAB IDE V8.89 and protus 8 professional

Theory -

16x2 LCD module is a very common type of LCD module that is used in embedded system. It consists of 16 rows and 2 columns of 5x7 LCD dot matrices It is available in a 16 pin package with back light, contrast adjustment function and each dot matrix has 5x8 dot resolution the pin numbers, their name and corresponding functions are shown in the below table.

pin NO	Name	function.
1	VSS	This pin must be connected to the ground
2	VCC	Positive supply voltage pin (5V DC)
3	VEE	construct adjustment
4	RS	Register selection
5	RW	Read or write
6	E	Enable
7	DB0	Data
8	DB1	Data
9	DB2	Data
10	DB3	Data
11	DB4	Data
12	DB5	Data
13	DB6	Data
14	DB7	Data
15	LED+	Back light LED+
16	LED-	Back light LED-

VEE pin is ment for adjusting the contrast of the LCD display and the construct can be adjusted by varying the voltage at this pin.

The 16x2 LCD module has a set of commands each ment for doing a particular job with the display. R/W pin is ment for selecting between read and write modes. High level at this pin enables read mode and low level at this pins enables write mode.

E pin is for enabling the module. A high to low transition at this pin will enable the module.

DB0 to DB7 are the data pins. the data to be displayed and the command instruction are placed on these pins.

LED+ in the anode of the back light LED and this pin must be connected to VCC through a suitable series. current limiting register. LED- is the cathode of the back light and this pin must be connected to ground

16 X2 LCD module commands —

16x2 LCD module has a set of preset command instructions. each command will make the module to do a particular task. the commonly used command and their function are given in the table. Hex page-

command	function.
0f	LCD ON, cursor ON, cursor blinking ON.
01	clear screen
2	Return home
4	Decrement cursor
06	~~0e~~ Increment cursor
F	Display ON, cursor ON.
80	force cursor to the beginning of 1st line
C0	force cursor to the beginning of 2nd line
38	Use 2 lines and 5x7 matrix
8C	Activate second line
83	cursor line 1 position.
0C3	Jump to second line position 3
0C1	Jump to second line, position 1.

* LED initialization.
The steps that has to be done for initializing the LCD display is given below and these steps are common for almost all application.
- send 38H to the 8 bit data line for initialization
- send 0fH for making LCD ON, cursor, ON and cursor blinking ON.
- send 06H for incrementing cursor position.
- send 01H for clearing the display and return the cursor.

* Sending data to the LCD-
The steps for sending data to LCD module is given below. I have already said that the LCD module has pins namely RS, R/W and E
It is ~~command and mode~~ 'logic state of these pins that make the module to

determine whether a given data input is a command or data to be displayed.

- Make R/W low.
- make RS = 0 if data byte is a command and make RS = 1 if the data byte is a data to be displayed
- place data byte on the data register
- pulse E from high to low
- Repeat above steps for sending another data.

PROCEDURE:

Step 1:

- Proteus 8 Professional- click on file -new project-provoide path-next
- Go into Schematic Picture-Select Devices
- Go to terminal -select VCC and GND

Step 2:

- MPLAB IDE V8.89 -Go to Project-Project Wizard-select device PIC18F4550-next-Active toolsuit-Microchip C18 Toolsuite-next
- Create new Project file-Create new Folder -Click on next -Finish

- Go to file-new-write Code-Save file with .C extension -add this file in source File-Right Click on Source file- add C file

Step 3:

- Go to project-Build all Project
- Check for the errors If No errors – Build successful will appear on the OUTPUT window

Step 4:

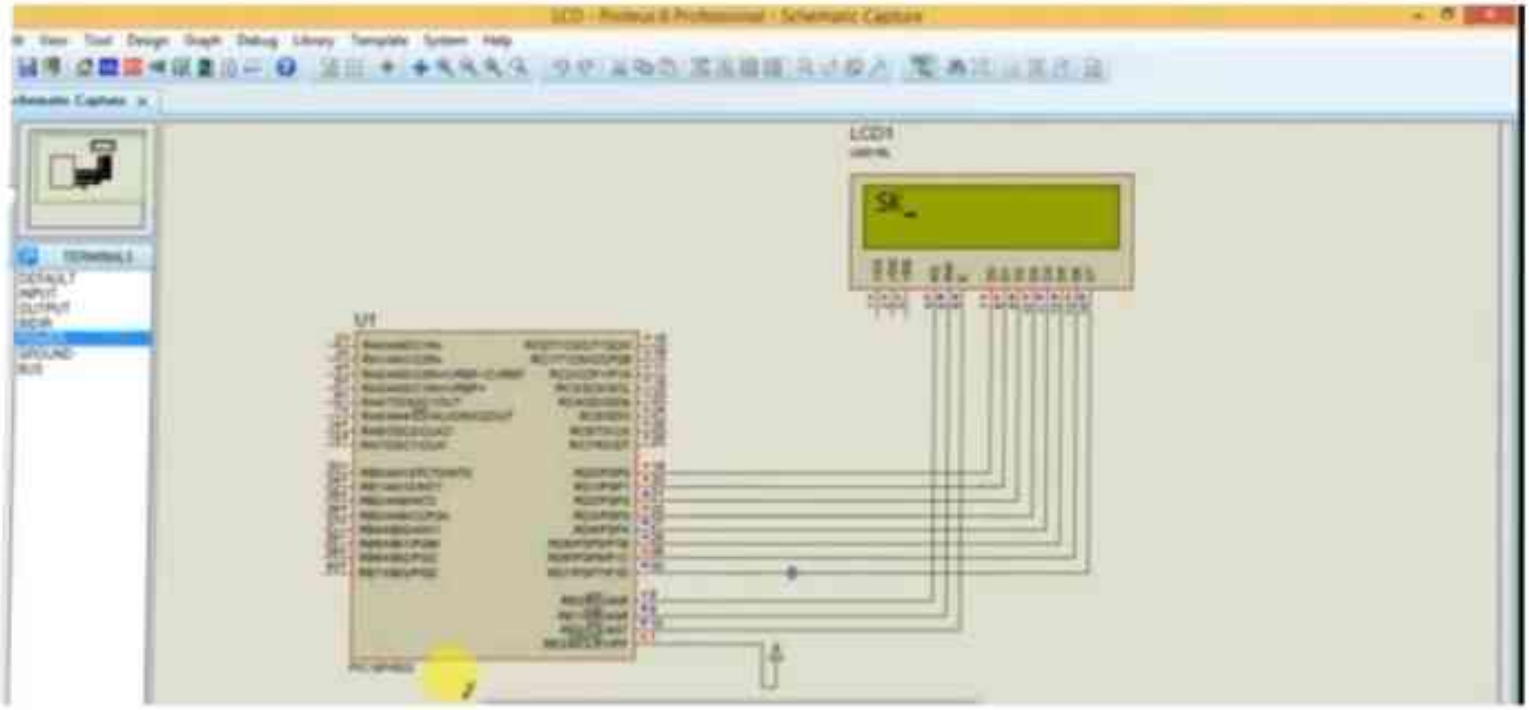
- After Build project HEX file is created-Burn the HEX file in Proteus 8 Professional.

LCD interfacing code

```
#include<p18f4550.h>
#pragma config OSC=HS
#pragma config PWRT=OFF
#pragma config WDT=OFF
#pragma config DEBUG=OFF,LVP=OFF

void lcdmd (unsigned char value);
void lcddata(unsigned char value);
vpid msdelay(unsigned int itime);

#define ldata PORTD              //declare ldata variable for PORTD
#define rs PORTEbits.RE0         //decalre  rs varible for pin RE0
#define rw PORTEbits.RE1         //decalre  rw varible for pin RE1
#define en PORTEbits.RE2         //decalre  en varible for pin RE2

Void main()
{
 TRISD=0x00;                     //set direction of PORTD as output
 ADCON1=0x0F;
 TRISE=0x00;                     //set direction of PORTE as output
 msdelay(50);
 lcdcmd(0x38);                   //16x2 LCD
 msdelay(50);
 lcdcmd(0x0E);
 msdelay(15);
 lcdcmd(0x01);                   //clear display screen
 msdelay(15);
 lcdcmd(0x06);                   //Increment cursor & shift right
 msdelay(15);
 lcdcmd(0x80);                   //Force cursor on first row first position
 lcddata('S')                   //Display character 'S'
 msdelay(50);
 lcddata('K');                   //Display character 'K'
 msdelay(50);
}
 void lcdcmd(unsigned char value)
{
 ldata=value;                    //send the command value to PORTD
 rs=0;                          //selection of command register of LCD
 rw=0;
 en=1;                           //generate high to low pulse Enable pin
 msdelay(1);
 en=0;
}
 void lcddata(unsigned char value)
{
 ldata=value;                    //send the command value to PORTD
 rs=1;                          //selection of data register of LCD
 rw=0;
 en=1;                           //generate high to low pulse Enable pin
 msdelay(1);
 en=0;
}
```

```
 void msdelay(unsigned int itime)
{
  int i,j;
  for (i=0; i<itime; i++);
  for (j=0; j<135; j++);
}
```

Proteus Simulation Window



Conclusion → Thus we have study embedded c program for LCD interfacing with PIC18FXXX.

Page No.
Date

EXPRIMENT NO- 9.

AIM- Write an Embedded c program for Generating PWM signal for DC motor

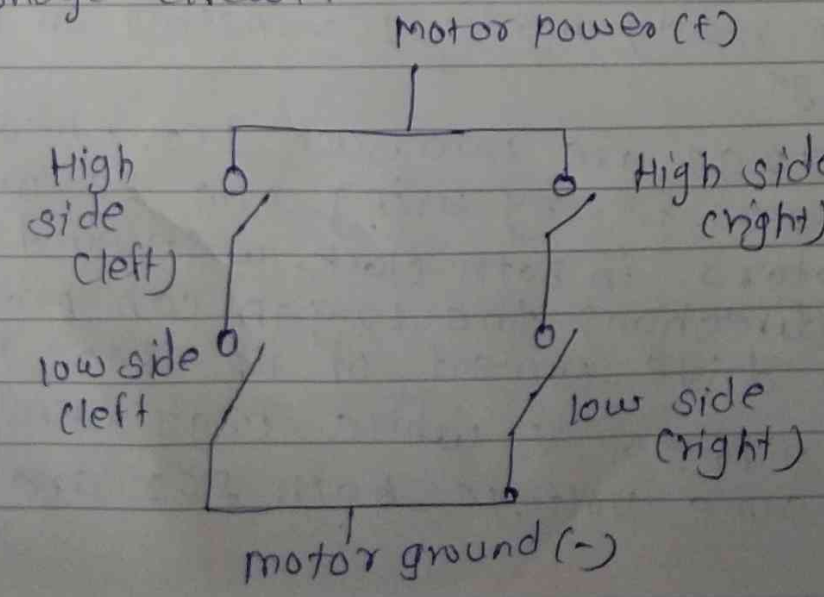
Software Used- MPLAB IDE V8.29 and proteus. 8 professional.

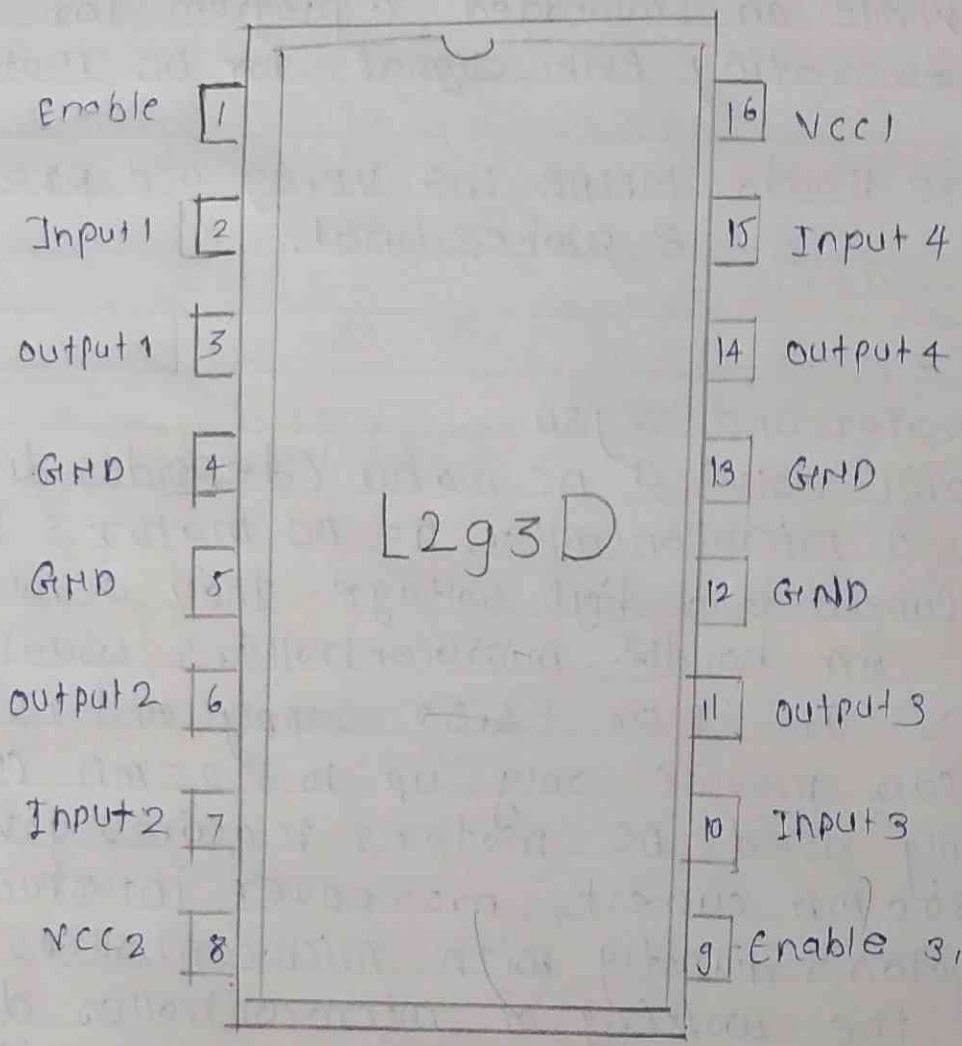
Theory-

DC motor and 12937D.

We can't drive a DC motor (depends) directly with a microcontroller. DS DC motors requires high current and high voltage than a microcontroller can handle. microcontrollers usually operates at +5 or +3.3V supply and it I/O pin can provide only up to 25mA current commonly used DC motors requires 12V supply and 800 mA current, moreover interfacing DC motors directly with microcontrollers may affect the working of microcontroller due to to back emf of the DC motor. thus. it is clear that, it not a good idea to interface DC motor directly with microcontrollers.

Thus solution to above problems is to use H-bridge circuit.





pin Diagram of L293D

It is a special circuit, by using the 4 switches we can control the direction DC motor depending upon our power requirements we can make our own H-bridge using transistors / MOSFETs as. It is better to use ready mode ICs. instead of making our own H-bridge.

L293D and L293 are two such ICs. these are dual H-bridge drivers. i.e. by using one IC we can control two DC motors. in both clock. wise and counter clockwise directions. the L293D can provide bireechional drive current. of up 600 mA at Voltage from 4.5 V to 36 V while L293 can provide up to 1A At some voltage. Both ICs are designed

To drive inductive loads (such as dc motors, bipolar stepping motors, relay and solenoids as well as other high-current or high voltage loads in positive supply applications

Capture/ comp / PWM (CCP) Modules.

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
-(1)	-(1)	DCxB1	DCxB0	CCPxM3	CCPxM2	CCPxM1	CCPxM0
bit7							bit0

Legend:

R = Readable bit W = Writable bit U = Unimplemented bit, read as '0'

-n = Value at POR '1' = Bit is set

'0' = Bit is cleared x = Bit is cleared = Bit is unknown

Bit 7-6 Unimplemented: Read as '0' (1)

bit 5-6 DCxB1: DCxB0: PWM Duty cycle Bit 1 and Bit 0 for CCPx module.

DCxB	DCxB	Decimal point
1	0	
0	0	0 0
0	1	0.25
1	0	0.50
1	1	0.75.

bit 3-0 CCPxM3 : CCPxM0 : CCPx module mode select bits

0000 = Capture / Compare / PWM 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 CCPx match (CCPxIF bit is set)

11xx = PWM mode

PROCEDURE:

Step 1:

- Proteus 8 Professional- click on file -new project-provoide path-next
- Go into Schematic Picture-Select Devices
- Go to terminal -select VCC and GND

Step 2:

- MPLAB IDE V8.89 -Go to Project-Project Wizard-select device PIC18F4550-next-Active toolsuit-Microchip C18 Toolsuite-next
- Create new Project file-Create new Folder -Click on next -Finish
- Go to file-new-write Code-Save file with .C extension -add this file in source File-Right Click on Source file- add C file

Step 3:

- Go to project-Build all Project
- Check for the errors If No errors – Build successful will appear on the OUTPUT window

Step 4:

- After Build project HEX file is created-Burn the HEX file in Proteus 8 Professional

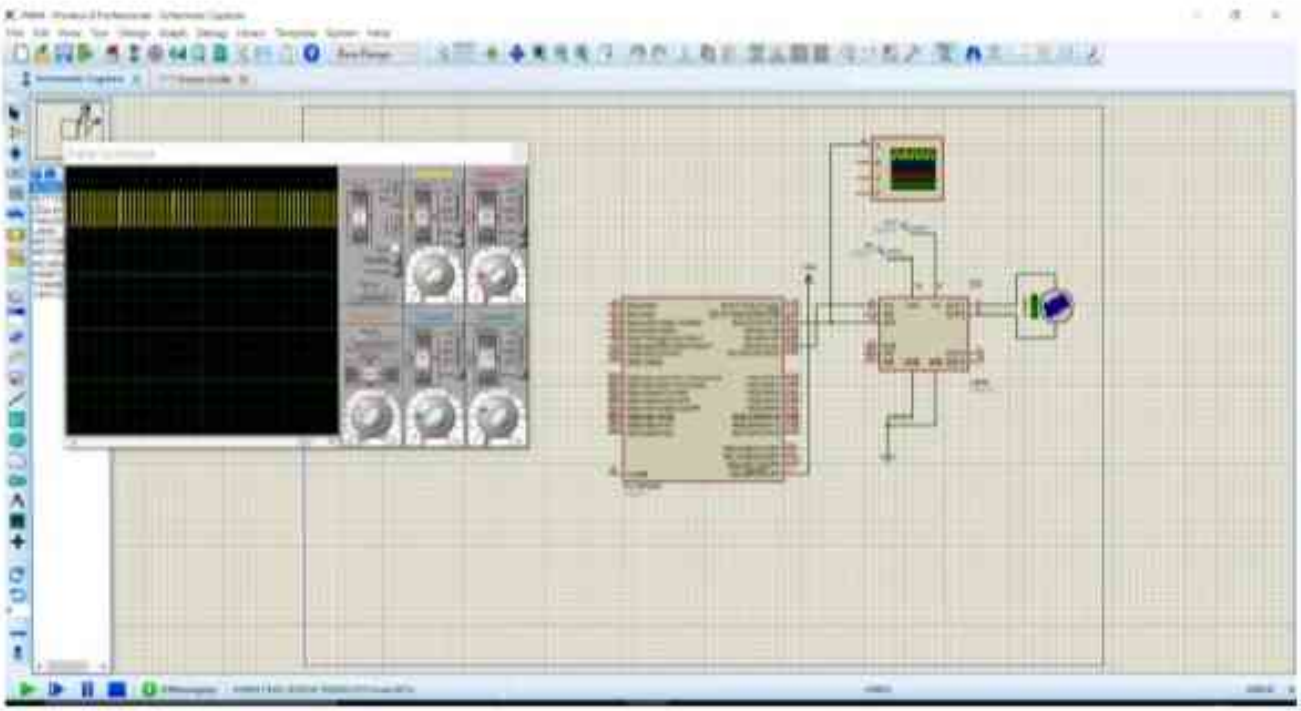
Code

```
#include<P18f4550.h>

/** V E C T O R  R E M A P P I N G **********************************************/

extern void _startup (void);  // See c018i.c
#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 High_ISR _endasm
}

#pragma code _LOW_INTERRUPT_VECTOR = 0x1018
void _low_ISR (void)
{
  // _asm goto Low_ISR _endasm
}

#pragma code

/**********************End Code**********************/

void main(void)
{
   TRISCbits.TRISC0 = 0 ;            // DC Motor IN1
        TRISCbits.TRISC1 = 0 ;                                  // DC Motor IN2
        TRISCbits.TRISC2 = 0 ;                                  // CCP2 PWM output, DC motor
enable


        PORTCbits.RC0 = 0;
        PORTCbits.RC1 = 1;

   CCP1CON  =  0b00001100;     //Select PWM mode and decimal value of Duty cycle
   CCPR1L  = 20;               //Duty cycle (Higher 8 bits out of 10 bits) change it to vary speed
   T2CON  =  0b00000010;           //Select Pre-scalar = 16 in Timer2
   PR2    = 150;               //Period Register (according to frequency of PWM)
   while(1)
   {
        PIR1bits.TMR2IF=0;           // Clear TMR2IF
        TMR2=0;                 //Clear Timer2
        T2CONbits.TMR2ON = 1;          //Turn ON Timer2
```



```
                while(PIR1bits.TMR2IF==0);          // Monitor TMR2IF
   }
}
```

Simulation Window



Conclusion → Thus we have study embedded C program for Generating PWM signal for DC. motor.

EXPRIMENT NO. - 10

AIM: Write an Embedded C program for PC to PC serial communication using UART.

SOFTWARE USED: MPLAB IDE V8.89 and Proteus 8 Professional

THEORY:

Both, Parallel and Serial modes of communication have certain advantages and disadvantages over one another. The serial communication is a preferred option due to its ability of long distance communication with error detection capability. The microcontrollers consist of an inbuilt hardware unit known as USART (Universal Synchronous Asynchronous Reception and Transmission) to facilitate serial transfer of data.

Before starting USART, some general terms related to communication need to be understood. These terms are explained below.

Asynchronous Communication:

In this type of communication, both Transmitter (Tx) and Receiver (Rx) work on different clocks which means that they are not synchronized. Start and Stop bits are also sent with each Data byte to identify the data.

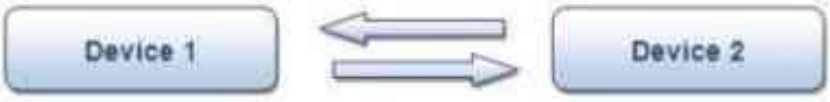
Synchronous Communication:

In this type of communication, both Tx and Rx are synchronized with the same clock and no Start or Stop bits are used.

- **Serial data transmission can be classified on the basis of how transmission occurs .**

Full-duplex Communication:

When either of the devices can send and receive data at the same instant, they are said to have full-duplex communication.



Half-duplex Communication:

In this type of communication, a device can either behave as Transmitter or Receiver at an instant which means that a device can't transmit data when it is receiving and vice versa.

PIC's EUSART

PIC18F452 has an inbuilt EUSART (Enhanced USART). Normally USART can be configured as asynchronous full-duplex communication or synchronous half-duplex communication. EUSART provides additional capabilities as compared to USART, like Automatic Baud-rate Detection. Automatic baud-rate detection means that during reception there is no need to set the baud rate at controller's side, EUSART sets it on its own which is certainly an advantage over USART. Baud rate and its calculation have been explained later in this article.

Registers of EUSART:

To use the EUSART of PIC18f452 microcontroller following registers need to be configured.

1. TXSTA (Transmit Status and Control Register)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
CSRC	TX9	TXEN	SYNC	SENDB	BRGH	TRMT	TX9D

TRMT: This is a read only bit which shows the status of Shift register where data is stored.

1 = Transmit Shift Register Empty

0 = Transmit Shift Register Full

BRGH: This bit is configured to decide the speed of Asynchronous serial communication.

1 = High speed

0 = Low speed

SYNC: The mode of communication is selected by this bit.

1 = Synchronous mode

0 = Asynchronous mode

TXEN: This bit is set to high to enable the transmission.

TX9: This bit is set to high while sending 9-bit long data.

1 = Selects 9-bit transmission

0 = Selects 8-bit transmission

2. RCSTA (Receive Status and Control Register)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D

CREN: This bit is set to high for continuous reception enable. To stop reception the bit is set to zero.

RX9: This bit is used when 9-bit long data is to be received.

1 = Selects 9-bit reception

0 = Selects 8-bit reception

SPEN: This bit is used to enable/disable the serial port. (Tx and Rx pins)

1 = Serial port enabled

0 = Serial port disabled

3. TXREG (EUSART Transmit Register)

The data to be transmitted is stored in TXREG register.

4. RCREG (EUSART Receive Register)

The data to be received is stored in RCREG register.

5. PIR1 (Peripheral Interrupt Request Register 1)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
SPPIF	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF

TXIF: This is transmitter interrupt flag bit. It becomes high when the TXREG register is empty.

RCIF: This is receiver interrupt flag bit. It becomes high when the RCREG register is full.

6. PIE1 (Peripheral Interrupt Enable Register 1)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
SPPIE	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE

TXIE: This bit is set to high to enable the transmission interrupt.

RCIE: This bit is set to high to enable the reception interrupt.

7. BAUDCON (Baud Rate Control Register)

This register controls the baud rate and some special functionalities of serial communication like auto-baud rate detection, inversion of receiving data etc.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ABDOVF	RCIDL	RXDTP	TXCKP	BRG16	—	WUE	ABDEN

BRG16: This bit is used to enable/disable 16-bit baud rate register.

1 = 16-bit Baud Rate Generator (BRG) – SPBRGH and SPBRG

0 = 8-bit Baud Rate Generator (BRG) – SPBRG only

This means if the BRG16 bit is set to high, the baud rate is decided by both SPBRGH and SPBRG registers; and if it is set to low, only SPBRG register will set the baud rate.

7. SPBRG & SPBRGH (EUSART Baud Rate Generator Register Low Byte and High Byte)

These registers are used to set baud rate for the serial communication. The two registers SPBRG & SPBRGH store lower byte and higher byte respectively.

Baud Rate:

The baud rate is the speed of data in serial communication. It is also known as bits per second (bps). PIC's EUSART provides different communication modes like asynchronous, synchronous, high-speed and low speed etc. The baud rate for different EUSART modes can be decided by different formulae. The following table shows the baud-rate formulae for different EUSART communication modes.

Configuration Bits			BRG/EUSART Mode	Baud Rate Formula
SYNC	BRG16	BRGH		
0	0	0	8-bit/Asynchronous	Fosc / [64 (n + 1)]
0	0	1	8-bit/Asynchronous	Fosc / [16 (n + 1)]
0	1	0	16-bit/Asynchronous	
0	1	1	16-bit/Asynchronous	Fosc / [4 (n + 1)]
1	0	x	8-bit/Synchronous	
1	1	x	16-bit/Synchronous	

Here : x = Doesn't matter

Fosc = Crystal frequency

N = Value of *SPBRGH : SPBRG* Register pair

For example, if a Baud rate of 9600 is to be set for a 12MHz crystal and the mode is 8-bit, Asynchronous, then,

SYNC=0, BRG16=0 and BRGH=0 and the baud rate can be calculated, by using the following formula, as given below.

$$\text{Baud Rate} = \frac{\text{Fosc}}{[\,64 \times (\text{SPBRG}+1)\,]}$$

So

$$(\text{SPBRG} + 1) = \frac{\text{Fosc}}{[\,64 \times \text{Baud Rate}\,]}$$

$$\text{SPBRG} = \frac{12000000}{64 \times 9600} - 1$$

$$\text{SPBRG} = 19.531 \sim 20$$

Objective: To first configure PIC18f4550 for asynchronous low-speed, 8-bit serial communication at 9600 baud rate. Next program the microcontroller to receive the serial data from a PC and transmit back the same data serially to the PC.

Window's HyperTerminal has used to send and receive the data. To know more about HyperTerminal and working with it, refer to the last section in the Description of Serial port interfacing with 8051. MAX232 has been used as level convertor between controller and PC. The connections are shown in circuit diagram tab.

Programming steps:

1. Configuration of EUSART-

- Set the baud rate by putting appropriate value in the SPBRG register.
- SPEN bit of the RCSTA register is set to high to activate serial port (Tx and Rx pins) of the controller.
- TXEN and CREN bits (in TXSTA and RCSTA registers) are set to high to activate serial transmission and serial reception respectively.

2. For Transmission-

- Store the data into TXREG register.
- Wait until transmission flag (TXIF) is equal to zero (PIR1 register).

3. For Reception-

- Wait until reception flag (RCIF) is equal to zero (PIR1 register).
- Store the value of RCREG register in some variable. This value is the received data.

PROCEDURE:

Step 1:

- Proteus 8 Professional- click on file -new project-provoide path-next
- Go into Schematic Picture-Select Devices
- Go to terminal -select VCC and GND

Step 2:

- MPLAB IDE V8.89 -Go to Project-Project Wizard-select device PIC18F4550-next-Active toolsuit-Microchip C18 Toolsuite-next
- Create new Project file-Create new Folder -Click on next -Finish
- Go to file-new-write Code-Save file with .C extension -add this file in source File-Right Click on Source file- add C file

Step 3:

- Go to project-Build all Project
- Check for the errors If No errors – Build successful will appear on the OUTPUT window

Step 4:

- After Build project HEX file is created-Burn the HEX file in Proteus 8 Professional

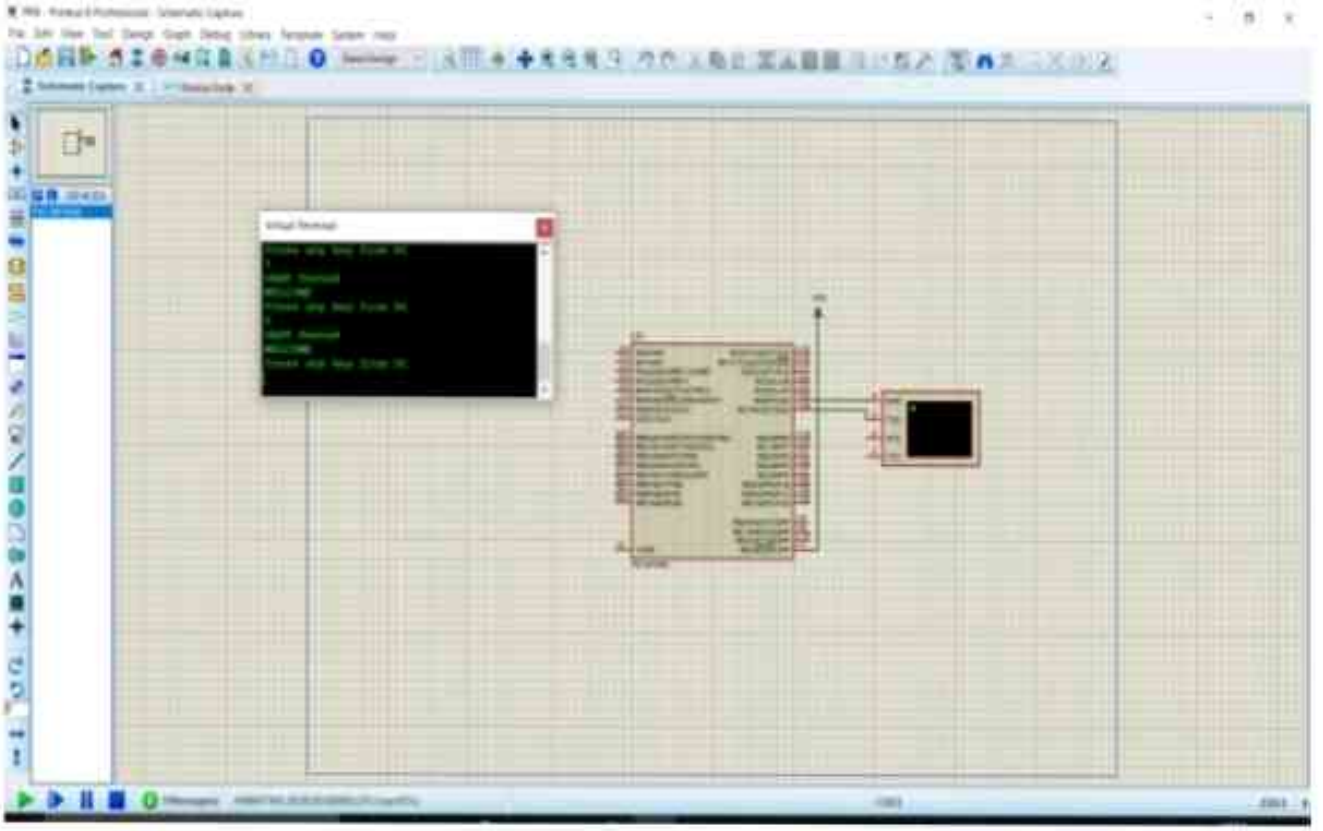
CONCLUSION - Thus we have study embedded C program for PC to PC serial communication using UART.

Code

```
#include <p18f4550.h>
#include <stdio.h>

/*The following lines of code perform interrupt vector relocation to work with the USB
bootloader. These must be used with every application program to run as a USB application.*/
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)
{
}

#pragma code
#pragma code _LOW_INTERRUPT_VECTOR = 0x1018
void low_ISR (void)
{
}
#pragma code
/*End of interrupt vector relocation*/
/*****This application helps in understanding the USART peripheral in PIC microcontrollers.
                The Terminal settings for PC are
                1.> Baudrate --> 19200
                2.> Parity   --> None
                3.> Databits --> 8
                4.> Stop bits --> 1

/*************************************************/

void TXbyte(char data)
{
  while(TXSTAbits.TRMT==0);             //wait tilltransmit buffer in not empty
  TXREG = data;                                 // Transmit Data
}                                                                       //end TXbyte


void main()
{
        unsigned char i, temp, String[]={"WELCOME \n\rPress any key from PC\n\r"},
String1[]={"\n\rUART Tested\n\r"};

        SSPCON1 = 0;          // Make sure SPI is disabled   //Refer Datasheet
        TRISCbits.TRISC7=1;         // RX
        TRISCbits.TRISC6=0;         // TX
```

```
	SPBRG = 0x71;
	SPBRGH = 0x02;	// 0x0271 for 48MHz -> 19200 baud XTAL=20MHz, Fosc=48Mhz
	TXSTA = 0x24;	// TX enable BRGH=1
	RCSTA = 0x90;	// continuous RX
	BAUDCON = 0x08;		// BRG16 = 1
	temp = RCREG;	// Empty Recieve buffer

	for(i=0;String[i]!='\0';i++)
	{
		TXbyte(String[i]);
	}
	while(PIR1bits.RCIF==0);	//Wait util data from PC is received
	temp = RCREG;
	TXbyte(temp);

	for(i=0;String1[i]!='\0';i++)
	{
		TXbyte(String1[i]);
	}

	while(1);		//loop forever
}
```

Simulation Window



EXPRIMENT NO. - 11

AIM: Write an Embedded C program for Temperature sensor interfacing using ADC & display on LCD.

SOFTWARE USED: MPLAB IDE V8.89 and Proteus 8 Professional

THEORY: A microcontroller, a digital device, can read, execute and transmit only digital signals. On the contrary, the outputs of the most of the transducers are analog in nature. Thus it is hard to interface these transducers directly with controllers. Analog-to-digital convertor (ADC) ICs are one way to make the analog input compatible with the microcontroller.

Using an external ADC adds complexity to the circuit. To avoid this complexity, PIC Microcontrollers have in-built ADC module which reduces the cost and connections of the circuit. This article explains the in-built ADC of PIC18F4550 controller.

As mentioned in the summary, a PIC microcontroller has inbuilt ADC for A/D conversion. The ADC module of PIC18F4550 controller has following specifications:

- 10-bit resolution output which means that an analog input gets converted into a corresponding 10-bit digital output.
- 13 channels which means that a total of 13 analog signals can be converted simultaneously into digital.
- Vref+ (RA3) and Vref- (RA2) pins for external reference voltage.
- 8 selectable clock options.
- ADC can be in auto-triggering mode for continuous A/D conversion.

ADC Registers:

To work with the inbuilt ADC of this PIC microcontroller, the certain registers are required to be configured. Each of these ADC registers has been explained below.

1.ADCON0 (A/D CONTROL REGISTER 0)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
—	—	CHS3	CHS2	CHS1	CHS0	GO/DONE	ADON

ADON: This bit is used to enable/disable the ADC peripheral of the PIC.

1 = A/D converter module is enabled

0 = A/D converter module is disabled

GO/DONE: This is A/D conversion status bit. For ADON=1,

1 = A/D conversion in progress

0 = A/D Idle

CHS3: CHS0: These bits are used to select a particular analog channel from 13 available channels (0-12) which are multiplexed with digital I/O pins. The following table shows the bit configuration to select these analog channels:

CHS3:CHS0	Analog Channel	Pin
0000	Channel 0	RA0/AN0
0001	Channel 1	RA1/AN1
0010	Channel 2	RA2/AN2
0011	Channel 3	RA3/AN3
0100	Channel 4	RA5/AN4
0101	Channel 5	RE0/AN5
0110	Channel 6	RE1/AN6
0111	Channel 7	RE2/AN7
1000	Channel 8	RB2/AN8
1001	Channel 9	RB3/AN9
1010	Channel 10	RB1/AN10
1011	Channel 11	RB4/AN11
1100	Channel 12	RB0/AN12

2.ADCON1 (A/D CONTROL REGISTER 1)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
—	—	VCFG1	VCFG0	PCFG3	PCFG2	PCFG1	PCFG0

PCFG0:PCFG3: As mentioned earlier, there are 13 analog channels in PIC18F4550 which are multiplexed with digital I/O pins. This means that such a (multiplexed) pin can act as either a digital I/O pin or an analog input pin. Either of these configurations is selected by these bits. The following table shows the bit configuration to make a pin D (Digital I/O) or A (Analog input):

PCFG3: PCFG0	AN12	AN11	AN10	AN9	AN8	AN7	AN6	AN5	AN4	AN3	AN2	AN1	AN0
0000	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	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

VCGF1-VCGF0: These are voltage configuration bits to select the reference voltage (Vref) for ADC module.

Bit	= 1	= 0
VCGF0	Vref+ (RA3)	Vcc
VCGF1	Vref- (RA2)	Vss (GND)

3.ADCON2 (A/D CONTROL REGISTER 2)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ADFM	—	ACQT2	ACQT1	ACQT0	ADCS2	ADCS1	ADCS0

ADCS2:ADCS0: These bits are used to select the clock option for ADC peripheral. The following table shows the bit configuration to select from different clock options:

ADCS2:ADCS0	Clock Option
000	FRC (clock derived from A/D RC oscillator)
001	FOSC/ 64
010	FOSC/ 16
011	FOSC/ 4
100	FRC (clock derived from A/D RC oscillator)
101	FOSC/ 32
110	FOSC/ 8
111	FOSC/ 2

ACQT2:ACQT0: These bits are used to set the acquisition time of the ADC. The acquisition time is the time required to charge and discharge the holding capacitor of the ADC.

ADFM: This bit is used to select the format of digital output.

1 = Right justified (LSB to MSB)

0 = Left justified (MSB to LSB)

4.ADRESL & ADRESH:

Since the ADC module of PIC provides 10-bit digital output after A/D conversion, this output is stored in two 8-bit registers, namely, ADRESL & ADRESH. The lower byte is stored in ADRESL (A/D Result High register) while the higher byte is stored in ADRESH (A/D Result Low register).

Working with ADC

Objective: To select an analog channel of PIC18F4550's in-built ADC and provide an analog input (0 to 5 volt) to it using a variable resistor or preset (20 k). (See circuit diagram) The main objective is to read the analog signal and display the corresponding digital value on LCD. (Also see interfacing LCD with PIC)

Programming Steps:

1. Set number of analog inputs by setting PCFG3:PCFG0 (ADCON1).
2. Select the analog channel by setting CHS3:CHS0 bits (ADCON0).
3. Select the clock option, acquisition time output format by configuring the ADCON2register.
4. Enable the ADC by making ADON bit (ADCON0) high.
5. Start the conversion by setting GO/DONE bit(ADCON0).
6. Wait until GO/DONE bits becomes low. This indicates that the A/D conversion is over.
7. Store A/D conversion result from ADRESH:ADRESL into a variable.
8. Covert the resultant value to its corresponding ASCII value and display on LCD.

PROCEDURE:

Step 1:

- Proteus 8 Professional- click on file -new project-provoide path-next
- Go into Schematic Picture-Select Devices
- Go to terminal -select VCC and GND

Step 2:

- MPLAB IDE V8.89 -Go to Project-Project Wizard-select device PIC18F4550-next-Active toolsuit-Microchip C18 Toolsuite-next
- Create new Project file-Create new Folder -Click on next -Finish
- Go to file-new-write Code-Save file with .C extension -add this file in source File-Right Click on Source file- add C file

Step 3:

- Go to project-Build all Project
- Check for the errors If No errors – Build successful will appear on the OUTPUT window

Step 4:

- After Build project HEX file is created-Burn the HEX file in Proteus 8 Professional

CONCLUSION - Thus we have study embedded C program for Temperature sensor interfacing using ADC & display on LCD.

Code

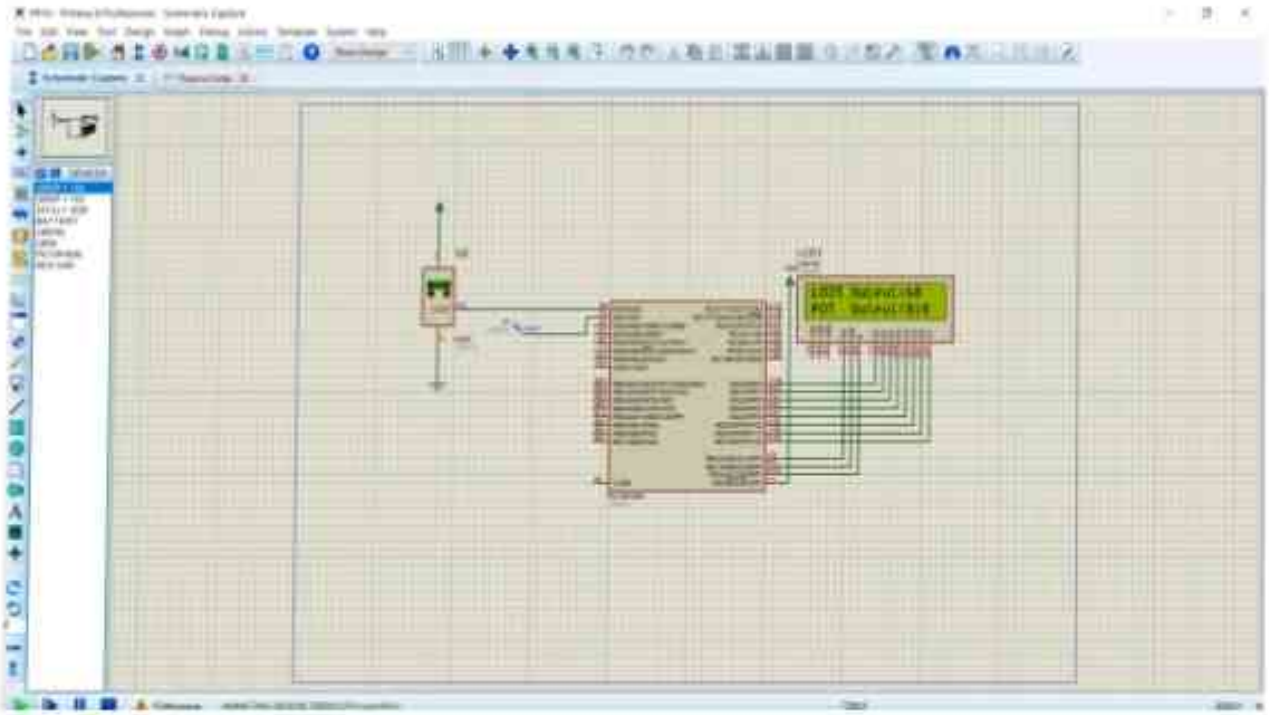
```
#include <p18f4550.h>
#include <stdio.h>
#include <stdlib.h>
/*The following lines of code perform interrupt vector relocation to work with the USB bootloader.
These must be used with every application program to run as a USB application.*/
extern void _startup (void);
#pragma code _RESET_INTERRUPT_VECTOR = 0x1000

void _reset (void)
{
        _asm goto _startup _endasm
}

#pragma code
#pragma code _HIGH_INTERRUPT_VECTOR = 0x1008
void high_ISR (void)
{
}

#pragma code
#pragma code _LOW_INTERRUPT_VECTOR = 0x1018
void low_ISR (void)
{
}
#pragma code
/*End of interrupt vector relocation*/

#define Channel0 0b00000000
#define Channel1 0b00000100
#define LCD_DATA    PORTD           //LCD data port
#define ctrl        PORTE           //LCD signal port
#define en          PORTEbits.RE2   // enable signal
#define rw          PORTEbits.RE1   // read/write signal
#define rs          PORTEbits.RE0   // register select signal


//LCD function definitions
void LCD_cmd(unsigned char cmd);
void init_LCD(void);
void LCD_write(unsigned char data);
void LCD_write_string(static char *str);

void ADC_Init(void);
void Select_Channel(unsigned char Channel);
unsigned int Get_ADC_Result(void);
void Start_Conversion(void);
void myMsDelay (unsigned int time);
```

```
void ADC_Init()
{
        ADCON0=0x00;                //A/D Module is OFF no channel selected
        ADCON1=0b00001110;      // Reference as VDD & VSS, AN0 set as analog pins
        ADCON2=0b10001110;      // Result is right Justified
                                        //Acquisition Time 2TAD
                                        //ADC Clk FOSC/64
        PIR1bits.ADIF=0;    //clear ADC conversion interrupt
        ADCON0bits.ADON=1;      //Turn ON ADC module
}

void Select_Channel(unsigned char Channel)
{
        ADCON0=ADCON0 & 0b11000001;
        ADCON0=ADCON0 | Channel;     //selecting channel no 0
}

void Start_Conversion()
{
        ADCON0bits.GO=1;
}

unsigned int Get_ADC_Result()
{
        unsigned int ADC_Result=0;
        while(ADCON0bits.GO);
        ADC_Result=ADRESL;
        ADC_Result|=((unsigned int)ADRESH) << 8;
        return ADC_Result;
}

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

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

   LCD_cmd(0x0C);      // cursor off
   myMsDelay(15);

   return;
```

```
}
//Function to guess command to the LCD
void LCDr_cmd(unsigned char cmd)
{
   LCD_DATA = cmd;
   rw = 0;
   rw = 0;
   en = 1;
   myMsDelay(15);
   en = 0;
   myMsDelay(15);
   return;
}

//Function to write data in the LCD
void LCD_write(unsigned char data)
{
   LCD_DATA = data;
   rs = 1;
   rw = 0;
   en = 1;
   myMsDelay(15);
   en = 0;
   myMsDelay(15);
   return ;
}
//Function to split the string into individual characters and call the LCD_write function
void LCD_write_string(static char *str)   //store address value of the string in pointer *str
{
   int i = 0;
   while (str[i] != 0)
   {
      LCD_write(str[i]);     //sending data on LCD byte by byte
      myMsDelay(15);
            i++;
   }
   return;
}


void main()
{
        unsigned int i, adc_val;
        static char String1[]={"LM35 Output:   "};
        static char String2[]={"POT Output:   "};
        unsigned char val, pot[4];
        TRISB = 0x00;
        TRISC = 0x0F;

        ADC_Init();             // Init ADC peripheral
        ADCON1 = 0x0E ;
```

```
while(1)
{
        LCD_cmd(0x80);
        myMsDelay(50);
        LCD_write_string(String1);
        Select_Channel(Channel0);          //select channel no
        Start_Conversion();                        //Trigger conversion
        adc_val= Get_ADC_Result();
        itoa(adc_val,pot0);
        LCD_cmd(0x8c);
        for(i=0;pot0[i]!='\0';i++)
        {
                LCD_write (pot0[i]);
        }
        myMsDelay(300);

        LCD_cmd(0xC0);
        myMsDelay(50);
        LCD_write_string(String2);
        Select_Channel(Channel1);          //select channel no
        Start_Conversion();                        //Trigger conversion
        adc_val= Get_ADC_Result();
        itoa(adc_val,pot0);
        LCD_cmd(0xCC);
        for(i=0;pot0[i]!='\0';i++)
        {
                LCD_write (pot0[i]);
        }
        myMsDelay(300);     }

}
```

Simulation Window



EXPRIMENT NO. - 12

Aim-Study of Arduino board and understand the OS installation process on Raspberry-pi.

1)Study of Arduino board

Arduino Uno: The Arduino Uno R3 is a microcontroller board based on the ATmega328. It has 14 digital input/output pins (of which six can be used as PWM outputs), six analog inputs, a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button. Simply connect it to a computer via a USB cable or power it with a AC-to-DC adapter or battery to get started.

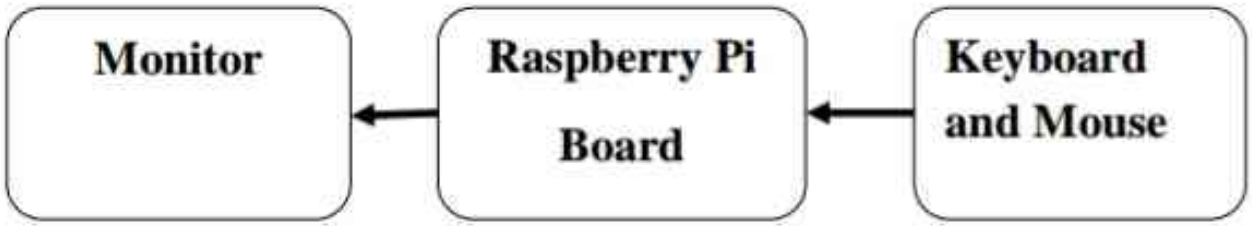
Arduino Nano: The Arduino Nano is a small, breadboard-friendly board based on the ATmega328. The microcontroller has more or less the same functionality of the Arduino Duemilanove, but in a different package. It lacks a DC power jack, and works with a Mini-B USB cable instead of a standard one.

2) Different operating systems for Raspberry –Pi. Understanding the process of installation on Raspberry-pi.

APPARATUS:

1. Raspberry Pi board
2. Monitor
3. VGA to HDMI converter
4. USB keyboard and mouse
5. Class 10 Memory card.

BLOCK DIAGRAM:



THEORY:

Raspberry-Pi:

The Raspberry Pi is the go-to microcomputer for all ages and abilities starting out in the wonderful world of programming and electronics.
There are three key models of Raspberry Pi on the market today - the Raspberry Pi 4 Model B, Raspberry Pi Model A+ and their tiny sibling the Raspberry Pi Zero.
A Raspberry Pi is a computer. A really small, really powerful, 'micro' computer. The latest Raspberry Pi 4 Model B includes a 1.5GHz quad-core processor, 1GB, 2GB or 4GB of RAM, Wi-Fi, Bluetooth and more!
With a highly-active community of forums, blogs, tutorials and learning materials available, learning how to code and make projects has never been easier! It's also more accessible than ever thanks to the ultra-affordable Raspberry Pi Zero W.
With the ability to use code to control real-world 'things' via the Raspberry Pi's GPIO, there are endless projects to make - LED projects, robots, home automation, AI assistants, remote control cars, displays, camera projects, media centers, servers, cluster computers and much, much more!
Our ever-growing range of Raspberry Pi kits, accessories and add-ons are a great way to explore the Raspberry Pi further and make fun projects along the way.

Operating systems runs on Raspberry-Pi:

Raspbian

Raspbian is a Debian-based engineered especially for the Raspberry Pi and it is the perfect general-purpose OS for Raspberry users.
It employs the Openbox stacking window manager and the Pi Improved Xwindows Environment Lightweight coupled with a number of pre-installed software which includes Minecraft Pi, Java, Mathematica, and Chromium.
Raspbian is the Raspberry foundation's official supported OS and is capable of accomplishing any task you throw at it.

OSMC

OSMC (Open Source Media Center) is a free, simple, open-source, and easy-to-use standalone Kodi OS capable of playing virtually any media format. It features a modern beautiful minimalist User Interface and is completely customizable thanks to the several built-in images that it comes with. Choose OSMC if you run the Raspberry Pi for managing media content.

Windows IoT Core

Windows IoT Core is a Windows OS built specially for the Raspberry Pi as a development platform for programmers and coders. Its aim is for programmers to use it to build prototypes of IoT devices using the Raspberry Pi and Windows 10.
It has an emphasis on security, connectivity, creation, and cloud integration. Unlike other titles on this list, you can't use it without running Windows 10 on your PC as you need Visual Studio on a Windows 10 setup to work with it.

RaspBSD

RaspBSD is a free and open-source image of FreeBSD 11 that has been preconfigured in 2 images for Raspberry Pi computers. If you didn't know, FreeBSD isn't Linux, but it works in pretty much the same way as it is a descendant of the research by the Berkeley Software Distribution and it is among the world's most broadly used Operating Systems today with its code existing in game consoles e.g. PlayStation 4, macOS, etc.

Ubuntu Core

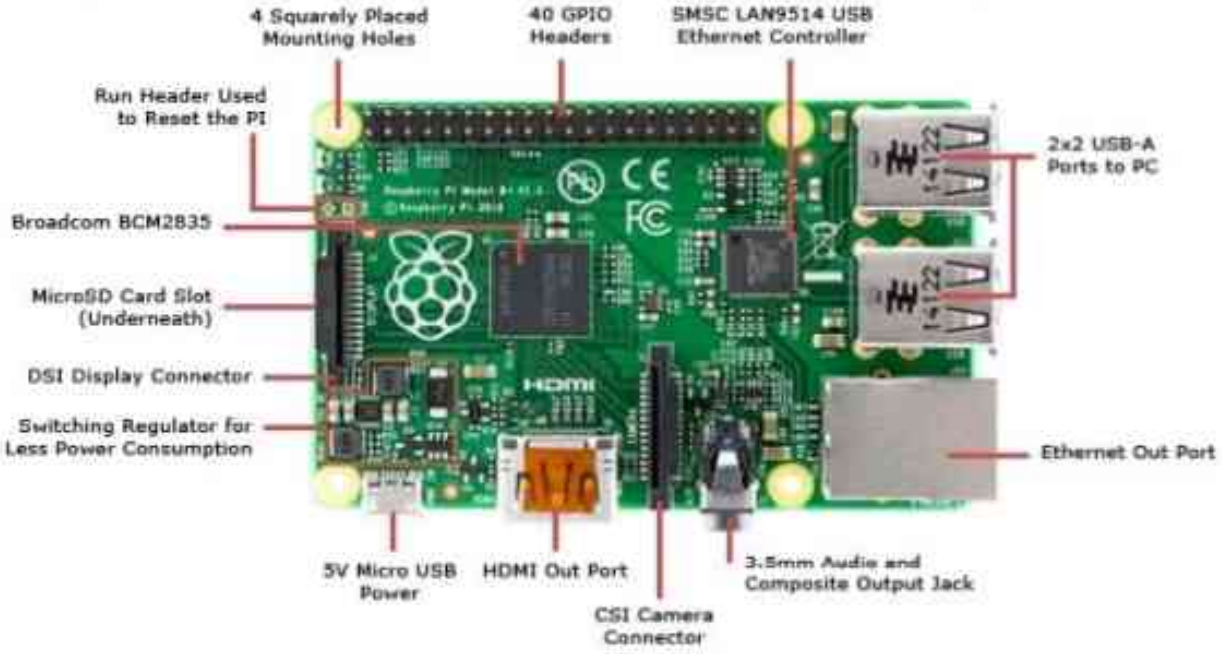
Ubuntu Core is the version of Ubuntu designed for Internet of Things applications. Ubuntu is the most popular Linux-based Operating System in the world with over 20+ derivatives and given that it has an active and welcoming forum, it will be easy to get up and running with Ubuntu Snappy Core on your Raspberry Pi.

Linutop

Linutop OS is a secure Raspbian-based Web Kiosk and digital signage player. It is dedicated to professionals with the need to deploy public Internet stalls and digital signage solutions using Raspberries.
This OS is perfect if you run hotels, restaurants, shops, city halls, offices, museums, etc. and it is compatible with Raspberry Pi B, B+ and 2.

Pin Description:

GPIO Pinout Diagram



PROCEDURE:

Installation of Raspbian:

- Download Raspbian
- Unzip the file
- Write the disk image to your SD card.
- Put the micro SD card in board and boot up.

CONCLUSION: Thus we have Study of Arduino board and understand the OS installation process on Raspberry-pi.

Experiment No. – 14

1) **Aim-** Interfacing of Sensor and actuators with Arduino Board and Raspberry-Pi

Sensors

- IR sensor
- LDR sensor
- PIR sensor
- Ultrasonic sensor
- Temperature and humidity (DHT11) sensor
- Temperature (LM35) (analog) sensor

Actuators

- DC motor
- Servo motor
- Stepper motor

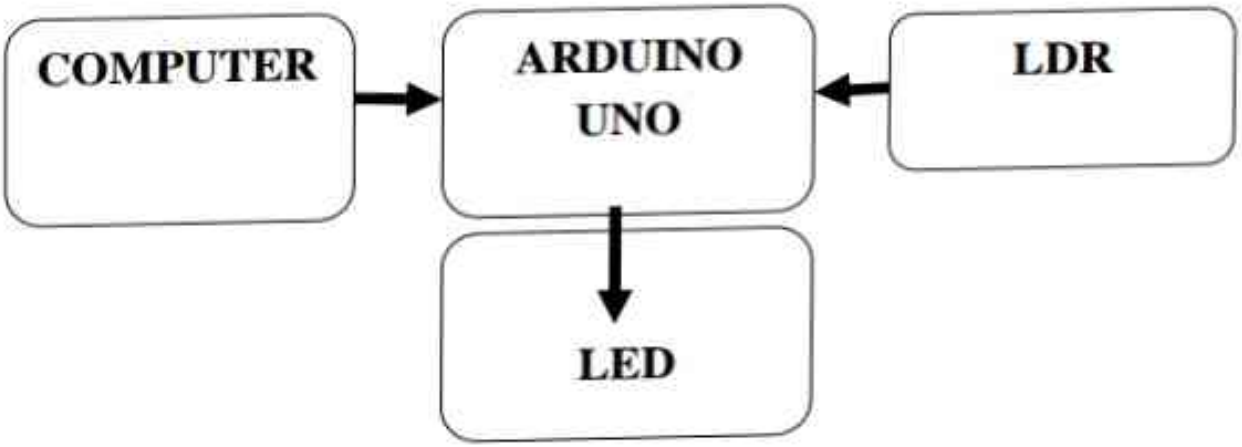
1)Interfacing with Arduion

Inter facing of LDR sensor with Arduino board

APPARATUS:

- Computer set with Arduino IDE
- Arduino Board
- LDR
- LEDs.
- resistor

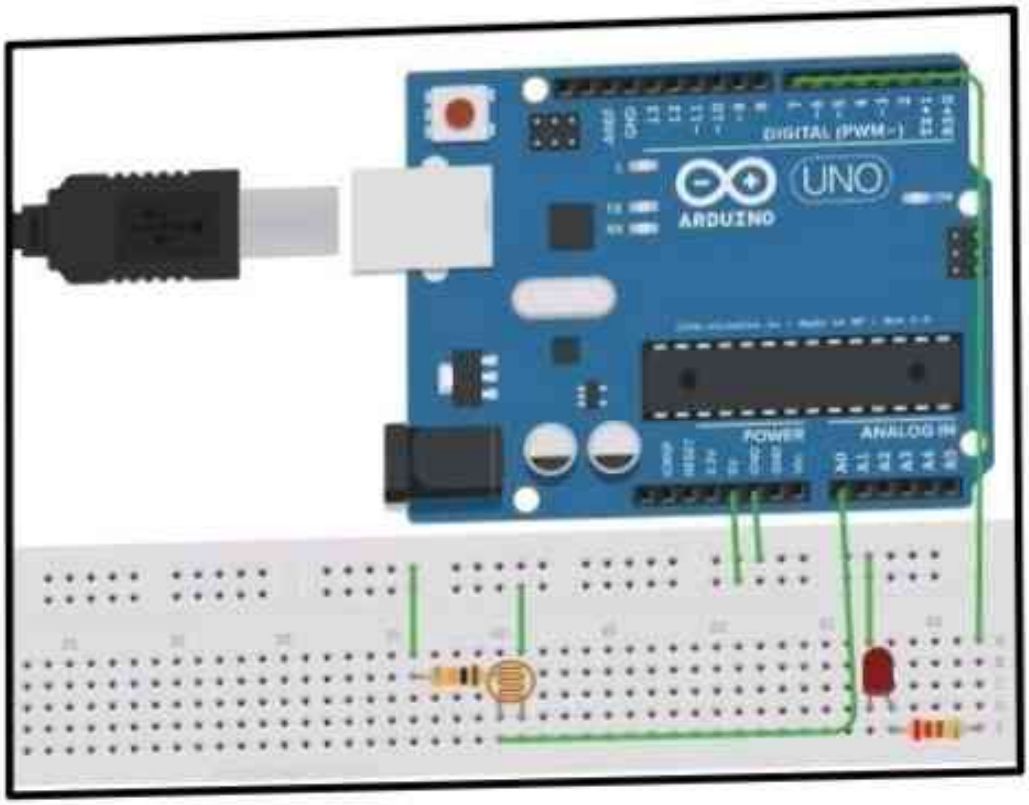
BLOCK DIAGRAM:



PROCEDURE:

1. Get a computer set which has installed Arduino IDE .
2. Connect Arduino to computer via USB and check whether port is detected or not.
3. Make connections as per the shown in connection diagram
4. Write code as per connections
5. Burn code in Arduino.

CONNECTION DIAGRAM:



PROGRAM:

```
constint lamp = 7;
boolean x = true;
.
void setup() {
Serial.begin(9600);
pinMode(lamp , OUTPUT);

}

void loop() {
```

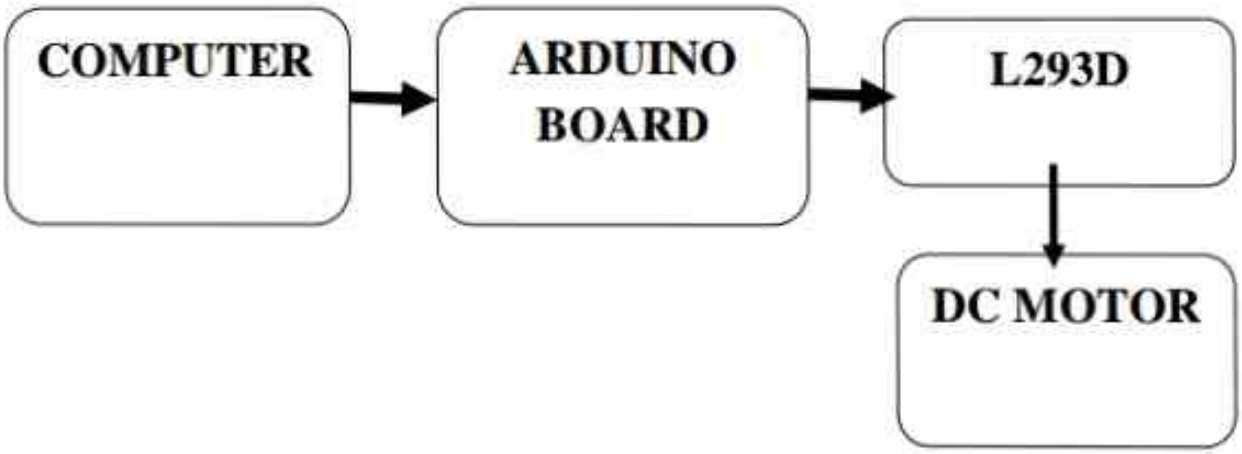
```
int c = analogRead(A0);
delay(500);
if ( c<300 && x == true){
digitalWrite(7,HIGH);
  x = false;
delay(1000);

}
else if ( c <300 && x == false){
x = true;
digitalWrite(7,LOW);
delay(1000);
}
}
```

Interfacing of DC motor with Arduino board

APPARATUS:

- Computer set with Arduino IDE
- Arduino Board
- DC motor
- L293D motor driver

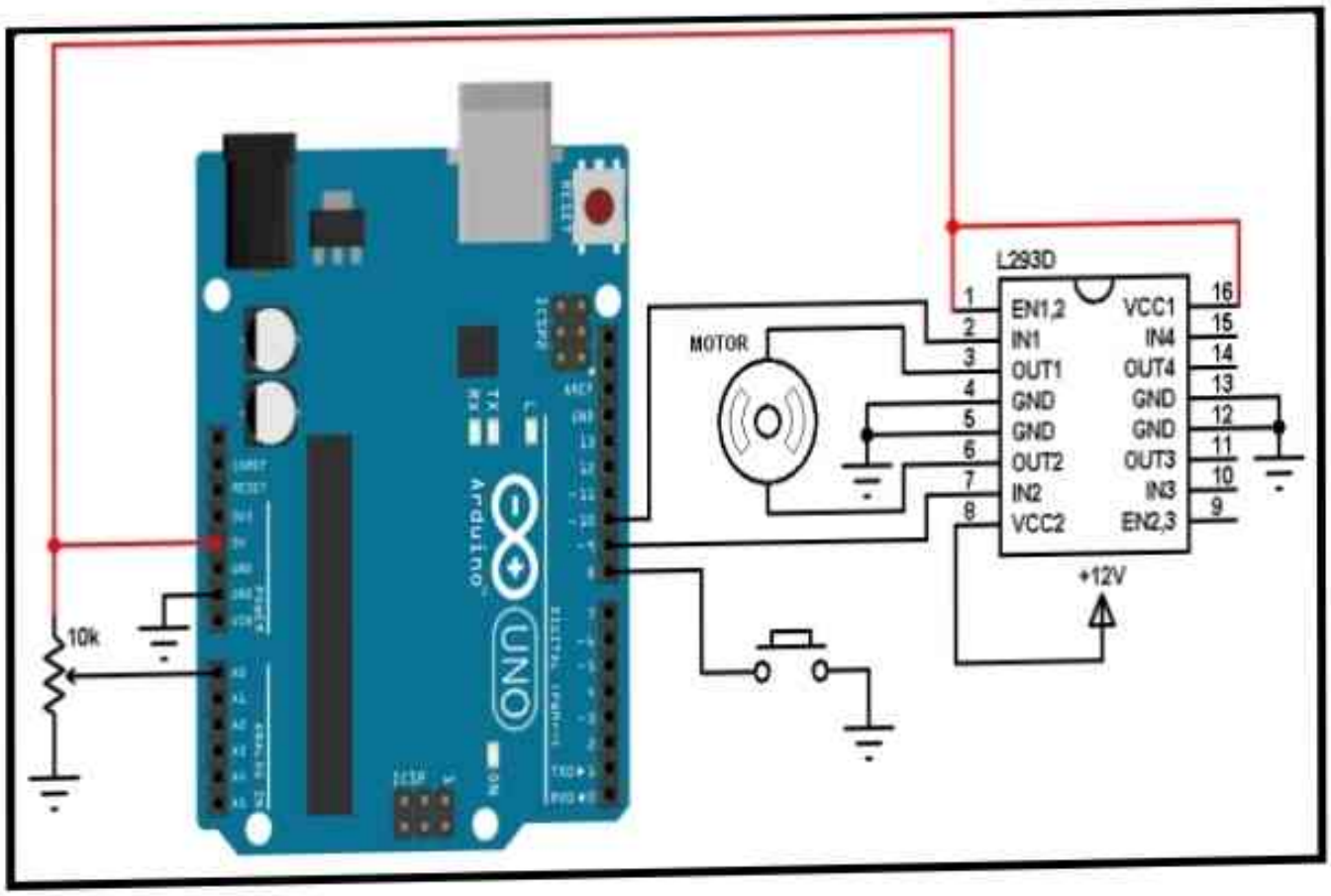
BLOCK DIAGRAM:



PROCEDURE:

- Get a computer set which has installed Arduino IDE .
- Connect Arduino to computer via USB and check whether port is detected or not.
- Make connections as per shown connection diagram.
- Type program of DC motor in Arduino IDE and debug.
- Burn code in Arduino.
- Check for DC motor is ON.

CONNECTION DIAGRAM:



PROGRAM:

```
// Arduino DC motor speed and direction control

#define button   8
#define pot      0
#define pwm1     9
#define pwm2    10

boolean motor_dir = 0;
int motor_speed;

void setup() {
 pinMode(button, INPUT_PULLUP);
 pinMode(pwm1,   OUTPUT);
 pinMode(pwm2,   OUTPUT);
}

void loop() {
 motor_speed = analogRead(pot) / 4;
 if(motor_dir)
  analogWrite(pwm1, motor_speed);
 else
  analogWrite(pwm2, motor_speed);
 if(!digitalRead(button)){              // If direction button is pressed
  while(!digitalRead(button));          // Wait until direction button released
  motor_dir = !motor_dir;               // Toggle direction variable
  if(motor_dir)
   digitalWrite(pwm2, 0);
  else
   digitalWrite(pwm1, 0);
 }
}
```

Interfacing with Raspberry-pi

Interfacing PIR sensor with Raspberry pi

All living beings radiate energy to the surroundings in the form of infrared radiations which are invisible to human eyes. A PIR (Passive infrared) sensor can be used to detect these passive radiations. When an object (human or animal) emitting infrared radiations passes through the field of view of the sensor, it detects the change in temperature and therefore can be used to detect motion.

HC-SR501 uses differential detection with two pyroelectric infrared sensors. By taking a difference of the values, the average temperature from the field of view of a sensor is removed and thereby reducing false positives.

Interfacing HC-SR501 with Raspberry Pi is easy because the output of a sensor is Pi friendly ie. 3.3V and it can be powered from the 5V rail of Pi.

The PIR sensor consist of 3 pins:

1. Vcc – 4.5V to 20V, Input power
2. OUTPUT – TTL output of sensor 0V, 3.3V
3. GND – Ground

The module has a rectangular window with two sub-probes 1 and 2 located at two ends of the rectangle. When a body emitting infrared radiation moves from side to side, the time for each probe for detection varies. Larger the time difference, more sensitive the device. It also uses a

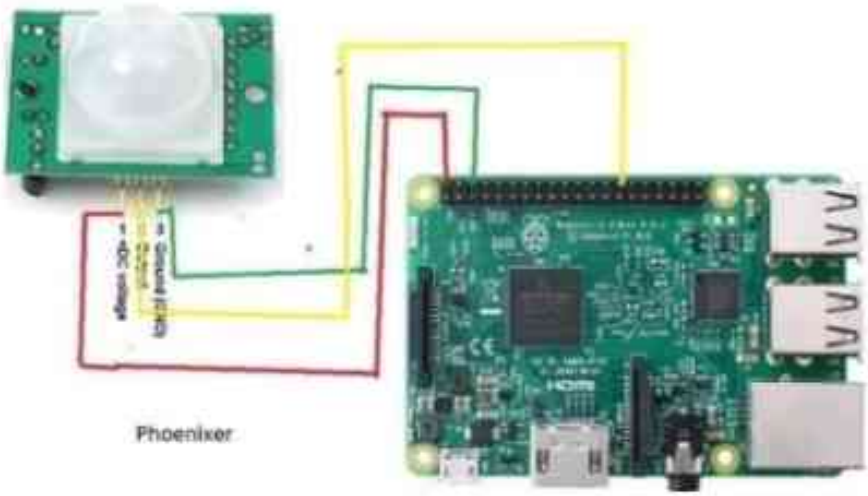
Adjustment:

For adjusting the detection delay (0.3 seconds to 600 seconds): Turn the potentiometer clockwise to increase and anticlockwise to decrease.

For adjusting the sensing distance (3 meters to 7 meters): Turn the potentiometer clockwise to

Hardware Requirement

1. Raspberry Pi setup (screen,mouse,keypad,raspberry pi,AC to 5V DC converter).
2. 3 female to female connector wires
3. A PIR sensor.



Vcc, Output, Ground are connected to 2 (5V), 26 (GPIO) and 6 (GND) pins of Pi respectively as shown in the figure.

Copy and paste given programme into PYTHON Editor of raspberry pi.

Program code

```
import RPi.GPIO as GPIO #Import GPIO library

import time #Import time library

GPIO.setmode(GPIO.BOARD) #Set GPIO pin numbering

pir = 26 #Associate pin 26 to pir

GPIO.setup(pir, GPIO.IN) #Set pin as GPIO in print "Waiting for sensor to settle"

time.sleep(2) #Waiting 2 seconds for the sensor to initiate print "Detecting motion"

while True:

if GPIO.input(pir): #Check whether pir is HIGH print "Motion Detected!"

time.sleep(2) #D1- Delay to avoid multiple detection

time.sleep(0.1) #While loop delay should be less than detection(hardware) delay
```

Run the program.....

Open terminal and goto dir where your program is saved and open that program with using sudo.

Note: The program can be calibrated for smaller detection (hardware) delay by using smaller Program delay(D1).

Output:

you will see "Motion Detected" printing on screen when any motion detected by sensor

a) Interfacing of DC motors with Raspberry Pi 3

Principle of Operation

The main principle in controlling a DC Motor with Raspberry Pi lies with the Motor Driver. A Motor Driver is a special circuit or IC that provides the necessary power (or rather the current) to the motor for smooth and safe operation.

Even a small 5V DC Motor draws a high initial current of around 300 – 400 mA. This current will then fall down 150 – 200 mA as the motor gains speed to around.

This is a huge current for devices like Microcontrollers, Arduino, Raspberry Pi etc. Hence, we should never connect a motor directly to Raspberry Pi (or any other microcontroller).

Motor Driver play an important role in this situation. They take the control signals from Raspberry Pi and provide the necessary drive current to the motor from the power supply.

In this project, the motor driver (L293D) is given with two control signals from Raspberry Pi through GPIO Pins. As per the Python Program, the motor will rotate in either forward or reverse direction.

Components Required

Raspberry Pi 3 Model B

L293D Motor Driver IC or Module

Small DC Motor (5V) Connecting wires

(Jumper Wires)

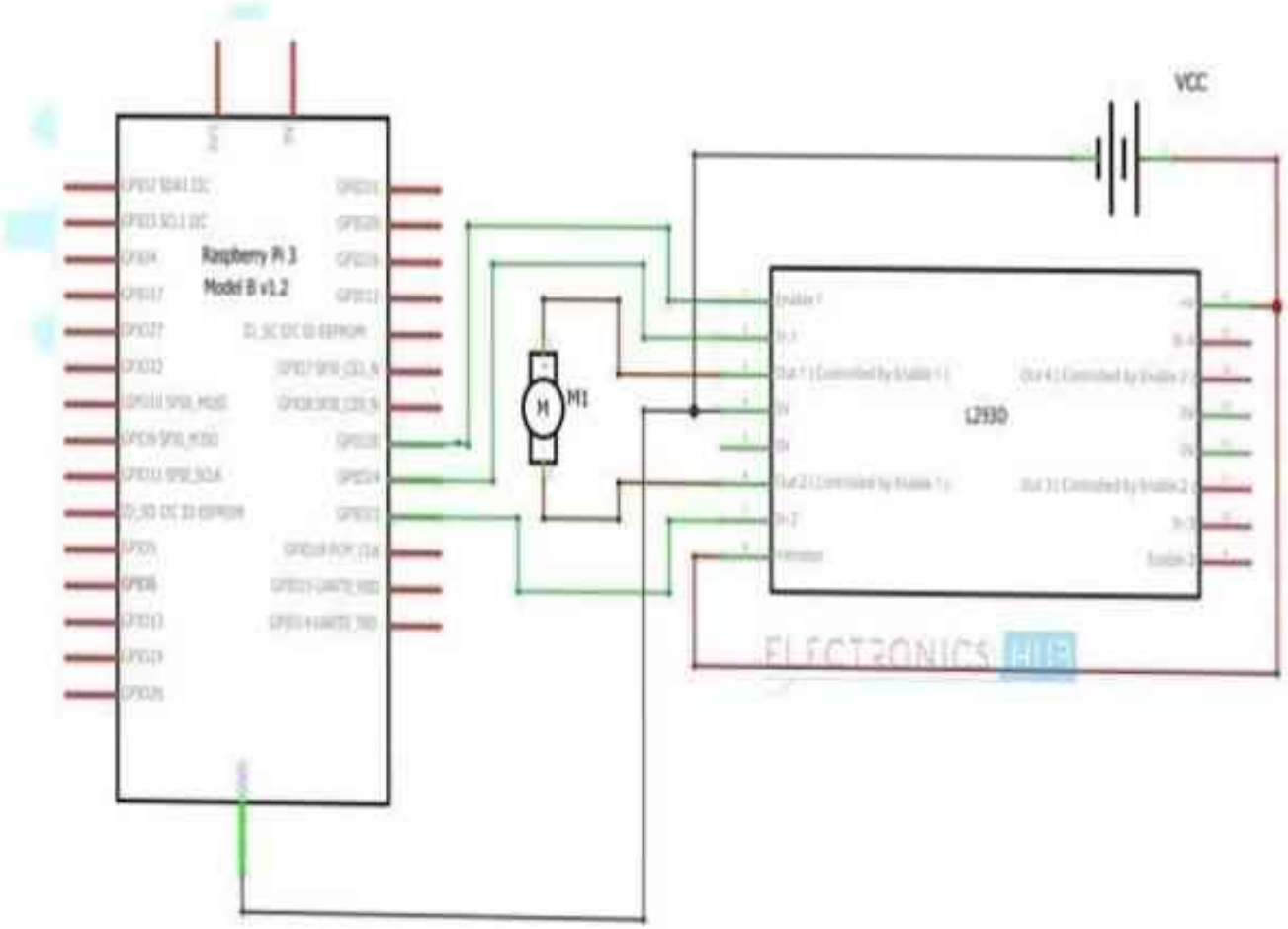
5V – 2A Power Supply for Raspberry Pi

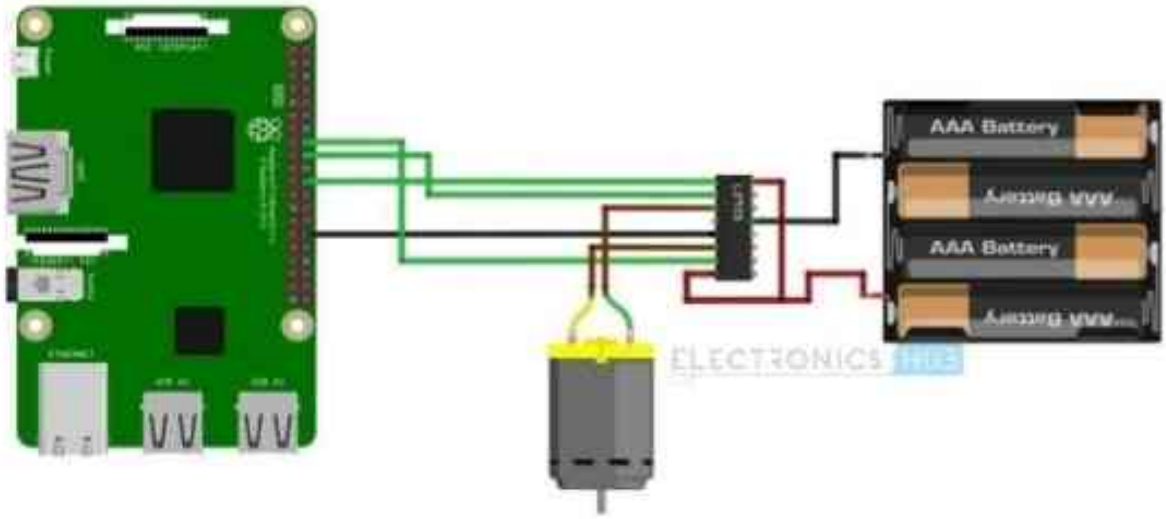
5V Supply for Motor

Miscellaneous (Computer, Ethernet Cable, etc.)

Circuit Diagram

The circuit wiring diagram of the project is shown below. You can easily configure this circuit and the program for controlling two DC Motors with Raspberry Pi and L293D Motor Driver IC.





The design of the circuit for controlling a DC Motor with Raspberry Pi is very simple. First, connect the pins 8 and 16 (VCC2 and VCC1) of L293D to external 5V supply (assuming you are using a 5V Motor).

There are four ground pins on L293D. Connect pin 4 to the GND of supply. Also, connect the ground pin of L293D to GND pin of the Raspberry Pi.

Finally, we have the enable and control input pins. Connect the pin 1 of L293D (1,2EN) to GPIO25 (Physical Pin 22) of Raspberry Pi. Then connect control input pins 2 and 7 (1A and 2A) to GPIO24 (Physical Pin 18) and GPIO23 (Physical Pin 16) respectively.

OPTIONAL: If you want to connect a second motor, all you need to connect are the Enable (3,4EN) and second motor control inputs (3A and 4A) to three different GPIO Pins of the Raspberry Pi.

Program

```
import RPi.GPIO as GPIO

from time import sleep

# Pins for Motor Driver Inputs

Motor1A = 24

Motor1B = 23
```

```
Motor1E = 25

def setup():
    GPIO.setmode(GPIO.BCM)                    # GPIO Numbering
    GPIO.setup(Motor1A,GPIO.OUT) # All pins as Outputs
    GPIO.setup(Motor1B,GPIO.OUT)
    GPIO.setup(Motor1E,GPIO.OUT)
def loop():
    # Going forwards
    GPIO.output(Motor1A,GPIO.HIGH)
    GPIO.output(Motor1B,GPIO.LOW)
    GPIO.output(Motor1E,GPIO.HIGH)
    sleep(5)
    # Going backwards
    GPIO.output(Motor1A,GPIO.LOW)
    GPIO.output(Motor1B,GPIO.HIGH)
    GPIO.output(Motor1E,GPIO.HIGH)
    sleep(5)
    # Stop
    GPIO.output(Motor1E,GPIO.LOW)
def destroy():
    GPIO.cleanup()
if__name__== '__main__':     # Program start from here
    setup()
```

```
try:
        loop()
except KeyboardInterrupt:
        destroy()
```

Applications

DC Motor are found everywhere: robots, drones, RC Cars, etc. By Controlling a DC Motor with Raspberry Pi, we can develop many motor related projects using Raspberry Pi.

Can be used in Raspberry Pi based robotic applications like Line Follower Robot, Obstacle Avoiding Robot, Quadcopter, Web Controlled Robot etc.

Interfacing of Servo motors with Raspberry Pi 3

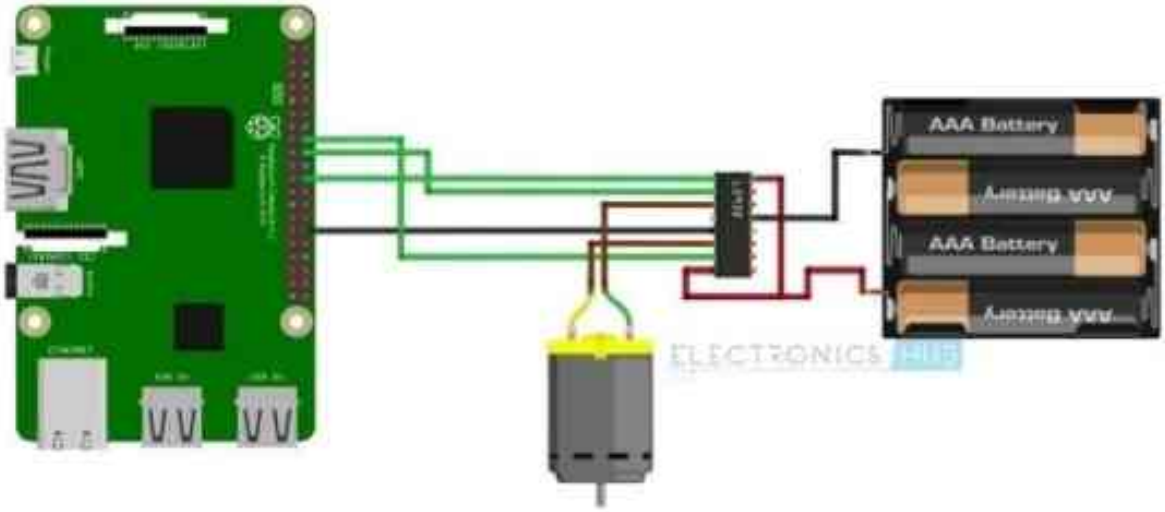
Hardware used:

- Raspberry Pi 3 Model B
- L293D Motor Driver IC or Module Small DC Motor (5V)
- Connecting wires (Jumper Wires)
- 5V – 2A Power Supply for Raspberry Pi
  5V Supply for Motor
- Miscellaneous (Computer, Ethernet Cable, etc.)

Software used:

Python Interpreter

Connection diagram :



Program

```
import RPi.GPIO as GPIO

from time import sleep

# Pins for Motor Driver Inputs

Motor1A = 24

Motor1B = 23
```

```
Motor1E = 25

def setup():
    GPIO.setmode(GPIO.BCM)                    # GPIO Numbering
    GPIO.setup(Motor1A,GPIO.OUT) # All pins as Outputs
    GPIO.setup(Motor1B,GPIO.OUT)
    GPIO.setup(Motor1E,GPIO.OUT)
def loop():
    # Going forwards
    GPIO.output(Motor1A,GPIO.HIGH)
    GPIO.output(Motor1B,GPIO.LOW)
    GPIO.output(Motor1E,GPIO.HIGH)
    sleep(5)
    # Going backwards
    GPIO.output(Motor1A,GPIO.LOW)
    GPIO.output(Motor1B,GPIO.HIGH)
    GPIO.output(Motor1E,GPIO.HIGH)
    sleep(5)
    # Stop
    GPIO.output(Motor1E,GPIO.LOW)
def destroy():
    GPIO.cleanup()
if__name__== '__main__':     # Program start from here
    setup()
```

```
try:
        loop()
except KeyboardInterrupt:
        destroy()
```

CONCLUSION – Thus we have study Interfacing of Sensor and actuators with Arduino Board and Raspberry-Pi