S.E. (I.T) PIC18F LAB Manual

Experiment 1

Aim: Write an Embedded C program to add array of n numbers.

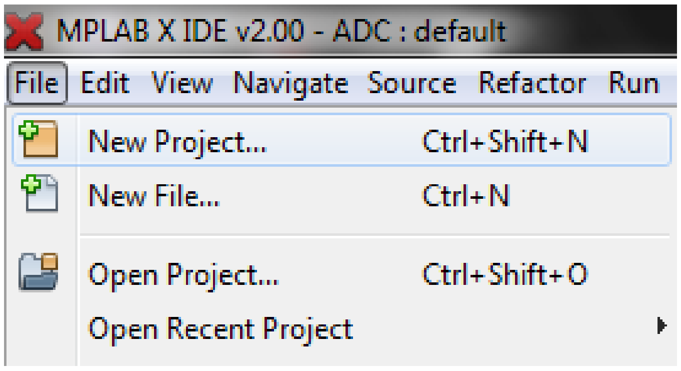
Experimental Setup: MicroPIC18F board, USB cable, Power supply adaptor, MPLABx IDE, PICLoader software.

Theory:

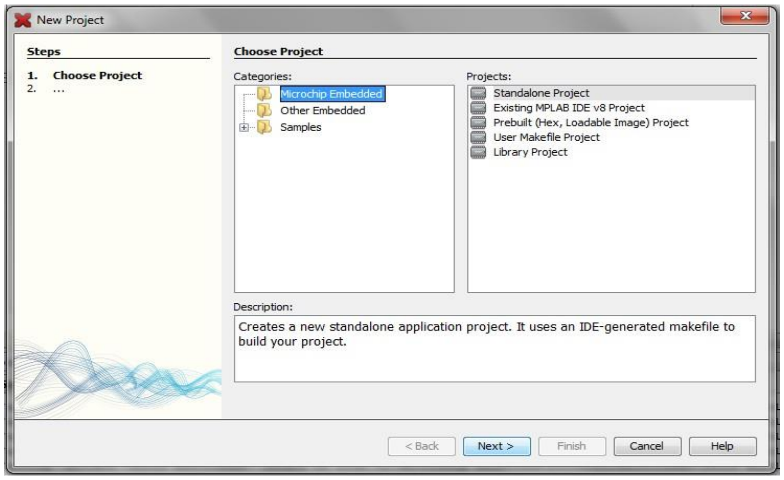
MPLABX IDE for program development.

1. Creating a New Project.

- Create a folder on the PC drive and give it appropriate name.
- Now click on the MPLABX IDE icon and start the IDE.
- Go to the File Tab.
- Click on New Project.

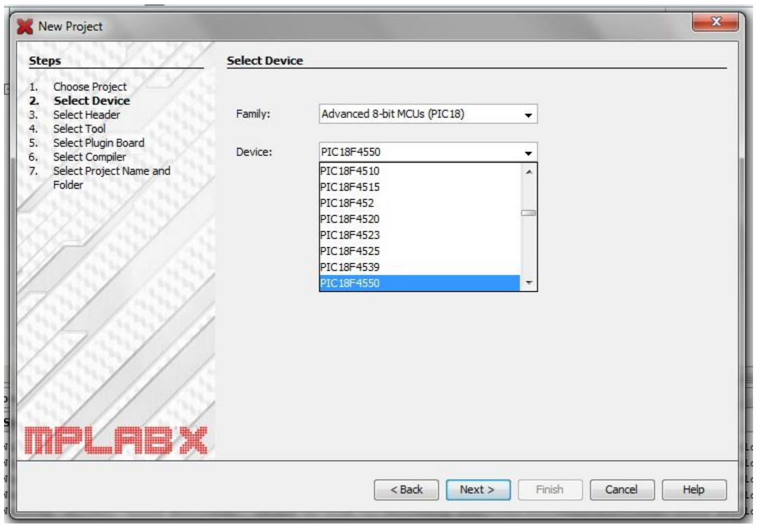

MPLAB X IDE v2.00 - ADC : default
File Edit View Navigate Source Refactor Run
New Project... Ctrl+Shift+N
New File... Ctrl+N
Open Project... Ctrl+Shift+O
Open Recent Project

- Step1: Choose Project:
  - Select : Microchip Embedded -> Standalone Project. Click Next.



Creates a new standalone application project. It uses an IDE-generated makefile to build your project.

- Step2: Select Device:
  - Select: Family ->Advanced 8 Bit MCU (PIC18).
  - Select: Device: **PIC18F4550**. Click Next.



Steps

1. Choose Project
2. **Select Device**
3. Select Header
4. Select Tool
5. Select Plugin Board
6. Select Compiler
7. Select Project Name and Folder

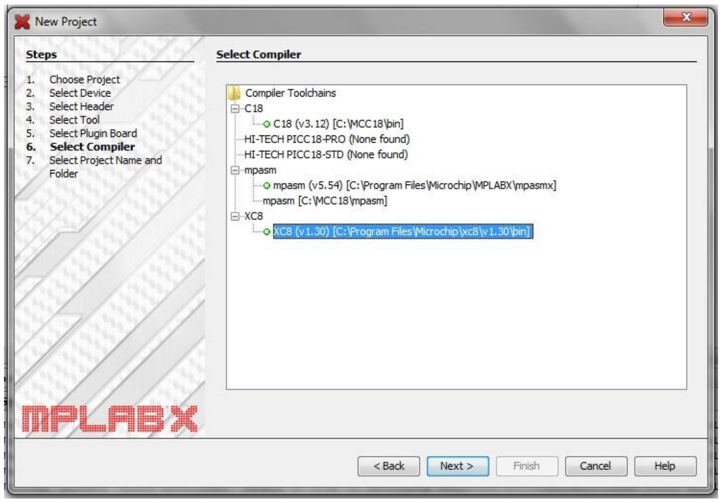
Select Device

Family: Advanced 8-bit MCUs (PIC18)

Device: PIC18F4550

```
PIC18F4510
PIC18F4515
PIC18F452
PIC18F4520
PIC18F4523
PIC18F4525
PIC18F4539
PIC18F4550
```

- Step3: Select Tool: Simulator. Click Next.
- Step4: Select Compiler ->XC8. Click Next
  - **Note: if you are developing a program using assembly language then select the compiler as ->mpasm(v5.68)**



Steps

1. Choose Project
2. Select Device
3. Select Header
4. Select Tool
5. Select Plugin Board
6. **Select Compiler**
7. Select Project Name and Folder

Select Compiler

```
Compiler Toolchains
C18
    C18 (v3.12) [C:\MCC18\bin]
  HI-TECH PICC18-PRO (None found)
  HI-TECH PICC18-STD (None found)
mpasm
    mpasm (v5.54) [C:\Program Files\Microchip\MPLABX\mpasmx]
    mpasm [C:\MCC18\mpasm]
XC8
    XC8 (v1.30) [C:\Program Files\Microchip\xc8\v1.30\bin]
```

- Step5: Select Project Name and Folder.
  - Give Project Name.
  - Select project Location (the project folder we created) using Browse Button.
  - Uncheck **Set as main project option**.
  - Click Finish.

Very Important Note:

Step6: Adjustment for Bootloader. (only for C programs)

- From the Projects window Right click on the project name and Go to properties.
- Select XC8 linker.
- In Option Categories -> select Additional Options.
- In Code Offset: write **800**.

2. Opening an existing project.

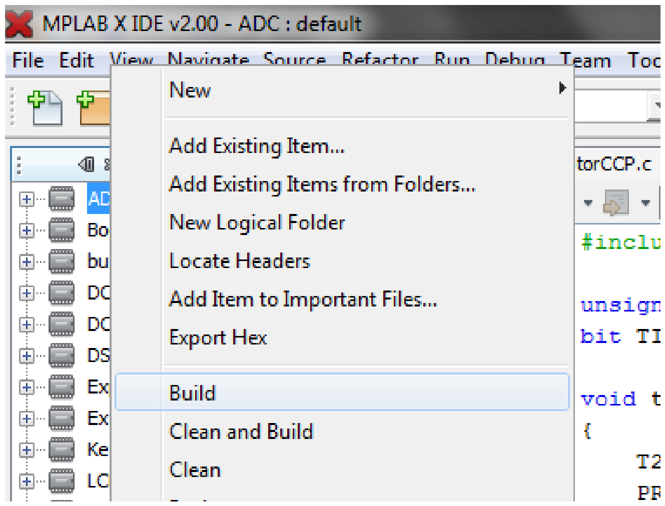
- Go to the File Tab.
- Select Open Project.
- Browse to the location and select the **projectname.X** file (project file). Click on Open Project .

3. Creating a new Source file and Header 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) and **Header files** (for a .h file).
- New -> C Source file / or C Header File.

4. Compiling Project.

- Step1: Go to project window.
    - Right Click on the project folder and select **Build** or **Clean and Build**.


MPLAB X IDE v2.00 - ADC : default
File Edit View Navigate Source Refactor Run Debug Team Too
New
Add Existing Item...
Add Existing Items from Folders...
New Logical Folder
Locate Headers
Add Item to Important Files...
Export Hex
Build
Clean and Build
Clean
torCCP.c

ARRAYS

An array is a collection of elements of the same type placed in contiguous memory locations that can be individually referenced by using an index to a unique identifier.

Five values of type int can be declared as an array without having to declare five different variables (each with its own identifier).

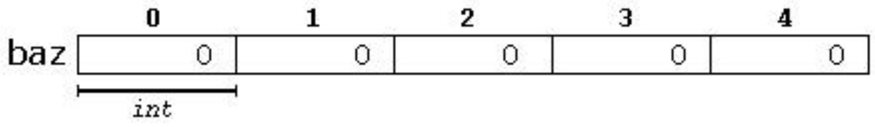
INITIALIZING ARRAYS

By default, are left uninitialized. This means that none of its elements are set to anyparticular value; their contents are undetermined at the point the array is declared.

The initializer can even have no values, just the braces:

```
int baz [5] = { };
```

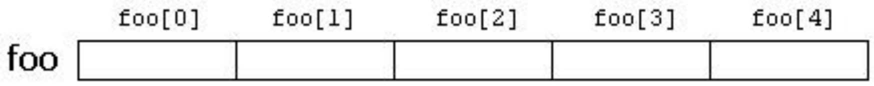
This creates an array of five int values, each initialized with a value of zero:



ARRAY ACCESSING

The values of any of the elements in an array can be accessed just like the value of a regular variable of the same type. The syntax is:

name[index]
Following the previous examples in which foo had 5 elements and each of those elements was of type int, the name which can be used to refer to each element is the following:



Some other valid operations with arrays:

```
foo[0] = a;

foo[i] = 75;

b = foo [i+2];

foo[foo[i]] = foo[i] + 5;
```

Procedure:

Step1: Open MPLABX IDE on the PC for program development and create a new project and save it in a new folder.

Step2: Write the program in C language to insert elements in array. **(in program properties make sure to add the 0x800 offset)**

Step3: Build the program and create hex file. In case of errors correct program and rebuild to create hex file.

Step4: Check File register and respective variables for result.

Program:

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

   unsigned int i, sum, n;
   unsigned int number[] = {1,2,3,4,5,6,7,8,9,10};
   sum = 0;                          // initialize sum as zero
   TRISB =0;              //initialize Port_B as output
   for(i=0; i<=9; i++)
   {
         sum = sum + number[i];
   }
    PORTB = sum;
   return;
}
```

Conclusion: Thus, we have studied embedded C program to add array of n numbers.

Experiment 2

Aim: Write an Embedded C program to transfer elements from one location to another for following.

I) Internal to internal memory transfer

II) Internal to external memory transfer

Theory:

EEPROM Memory (Internal)

What is an EEPROM?

EEPROM, pronounced as Double-E-PROM, stands for Electrically Erasable Programmable Read-Only Memory. This kind of memory devices is re-programmable by the application of electrical voltage and can be addressed to write/read each specific memory location.

The internal EEPROM memories (Built-in Within Microcontrollers) can be accessed for reading/writing operations by code. Writing a few lines of code will enable you of storing and/or retrieving data from the built-in EEPROM memory. And this is going to be our task in this tutorial. To develop the required firmware to drive this memory module.

Applications For EEPROMs

Let me give you a quick recap of the features of EEPROM memories before discussing their applications in real-life. A typical EEPROM device regardless of its type (internal/external) has the following features:

- Electrical erase-ability
- Electrical re-programmability
- Non-volatile memory locations
- Serial/Parallel interfaces for address/data lines (For External EEPROMs)
- Easy programmatically-controlled memory interface (For Internal EEPROMs)

Reading the Data EEPROM

Memory

To read a data memory location, the user must write the address to the EEADR register, clear the EEPGD and CFGS control bits (EECON1<7:6>) and then set control bit RD (EECON1<0>). The data is available in the very next instruction cycle of the EEDATA register; therefore, it can be read by the next instruction. EEDATA will hold this value until another read operation or until it is written to by the user (during a write operation).

Reading from Data EEPROM Memory

1. Write the address of the memory location to be read to EEADR register.

2. To select EEPROM data memory, clear the EEPGD control bit.
3. Set the RD bit to initiate the read cycle.
4. Then we can read data from EEDATA register.

Writing to Data EEPROM Memory

1. Write the address of the memory location to write to EEADR register.
2. Write the 8-bit data to be written in the EEDATA register.
3. To select EEPROM data memory, clear the EEPGD control bit.
4. Set the WREN control bit to enable write operations.
5. Disable Interrupts if enabled in your program. (You may store interrupt register (INTCON) to enable interrupts)
6. Then the special five instruction sequence is executed.
7. Enable Interrupts if using.
8. Disable the program operations by clearing WREN control bit.
9. WR control bit is cleared and EEIF interrupt flag bit is set after completion of the write operation. EEIF bit must be cleared in the program.

Procedure:

Step1: Open MPLABX IDE on the PC for program development and create a new project and save it in a new folder.

Step2: Write the program in C language for memory transfer **(in program properties make sure to add the 0x800 offset)**

Step3: Build the program and create hex file. In case of errors correct program and rebuild to create hex file.

Step4: Check File register, program memory and respective variables for result.

Program:

Memory exchange

```
#include <xc.h>

void main(void) {
    int temp, i;

    int source[]={0x21,0x22,0x23,0x24,0x25};

    int dest[]={0x00,0x00,0x00,0x00,0x00};

    for(i=0;i<=4;i++)

        { temp = source[i];

source[i]=dest[i];

dest[i]=temp;

}

    return;

}
```

Memory Transfer

```
#include <xc.h>

void main(void) {

    int temp, i;

    int source[]={0x21,0x22,0x23,0x24,0x25};

    int dest[]={0x00,0x00,0x00,0x00,0x00};

    for(i=0;i<=4;i++)

        dest[i]=source[i];

    return;

}
```

Result:

Conclusion: Thus, we have studied embedded C program to transfer elements from one location to another.

Experiment 3

Aim: Write an Embedded C program for sorting the numbers in ascending and descending order

Experimental Setup: MicroPIC18F board, USB cable, Power supply adaptor, MPLABx IDE, PICLoader software.

Theory:

<u>Ascending Order:</u>

we need to sort the given array in ascending order such that elements will be arranged from smallest to largest. This can be achieved through two loops. The outer loop will select an element, and inner loop allows us to compare selected element with rest of the elements.

Original array:

5	2	8	7	1

Array after sorting:

1	2	5	7	8

Elements will be sort in such a way that smallest element will appear on extreme left which in this case is 1. The largest element will appear on extreme right which in this case is 8.

Algorithm

1. Declare and initialize an array.
2. Loop through the array and select an element.
3. The inner loop will be used to compare the selected element from the outer loop with the rest of the elements of the array.
4. If any element is less than the selected element then swap the values.
5. Continue this process till entire array is sorted in ascending order.

<u>Descending order:</u>

we need to sort the given array in descending order such that elements will be arranged from largest to smallest. This can be achieved through two loops. Outer loop will select a element and inner loop allow us to compare selected element with rest of the elements.

Original array:

1	2	3	4	5

Array after sorting:

8	7	5	2	1

Elements will be sort in such a way that largest element will appear on extreme left which in this case is 8. Smallest element will appear on extreme right which in this case is 1.

Algorithm

1. Declare and initialize an array.
2. Loop through the array and select an element.
3. Inner loop will be used to compare selected element from outer loop with rest of the elements of array.
4. If any element is greater than the selected element then swap the values.
5. Continue this process till entire list is sorted in descending order.

Procedure:

Step1: Open MPLABX IDE on the PC for program development and create a new project and save it in a new folder.

Step2: Write the program in C language for ascending and descending order. **(in program properties make sure to add the 0x800 offset)**

Step3: Build the program and create hex file. In case of errors correct program and rebuild to create hex file.

Step4: Check File register, program memory and respective variables for result.

Program:

```
#include <xc.h>
#include<pic18f4550.h>

void main(void)
{
   int i, j, temp;
   int num[]= {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[i] < num[j])
              {                 // if LHS < RHS , change the position
                     temp = num[i];
                     num[i] =num[j];
                     num[j]= temp;
              }
         }
     }
}
```

Program:

```
#include <xc.h>
#include<pic18f4550.h>

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

  for (i = 0; i < 5; ++i)
        {

            for (j = i + 1; j < 5; ++j)
               {

               if (num[i] > num[j])
               {

                   temp =  num[i];
                   num[i] = num[j];
                   num[j] = temp;

               }

            }

        }
}
```

Conclusion: Thus, we have studied sorting the numbers in ascending and descending order.

Experiment 4

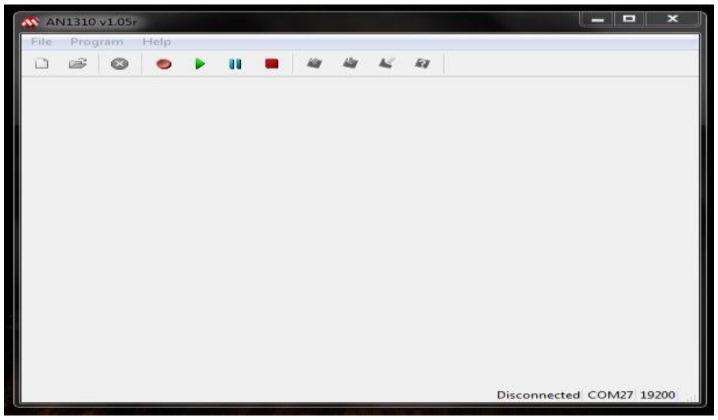
Aim: Write an Embedded C program for interfacing PIC18FXXX to LED and blinking it using specified delay.

Experimental Setup: MicroPIC18F board, USB cable, Power supply adaptor, MPLABx IDE, PICLoader software.

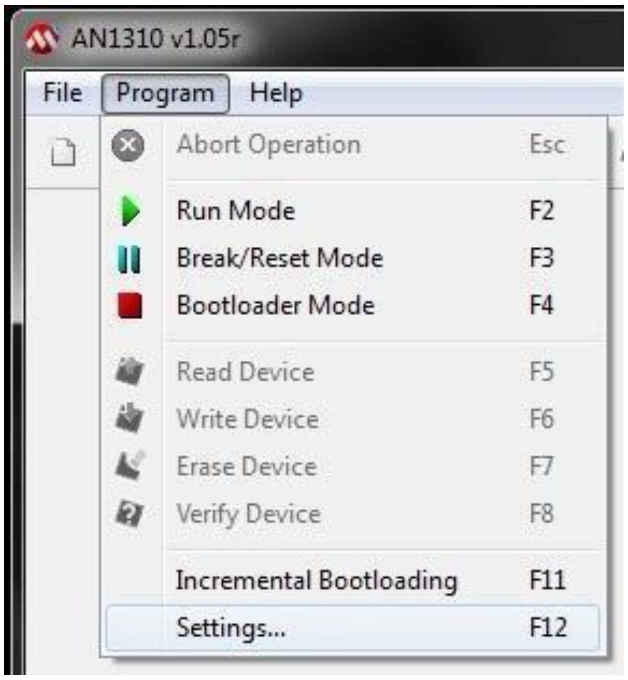
Theory:

Programming the Hex File USING PICLoader.

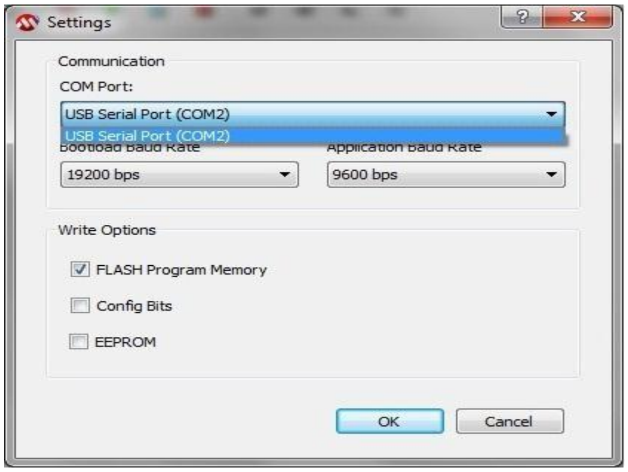
- Connect the USB Cable to the Board.
- Step 1: Double Click the PICloader.exe.



- Step2: Go to Programs -> Settings.



- Select the USB to serial com port. Click OK.



Settings

Communication

COM Port: USB Serial Port (COM2)

Bootload Baud Rate: 19200 bps

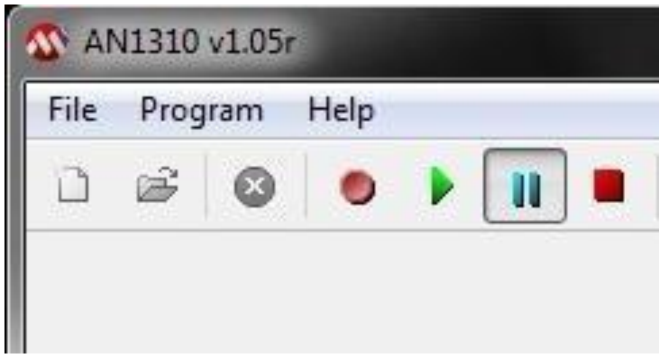
Application Baud Rate: 9600 bps

Write Options

- [✓] FLASH Program Memory
- [ ] Config Bits
- [ ] EEPROM

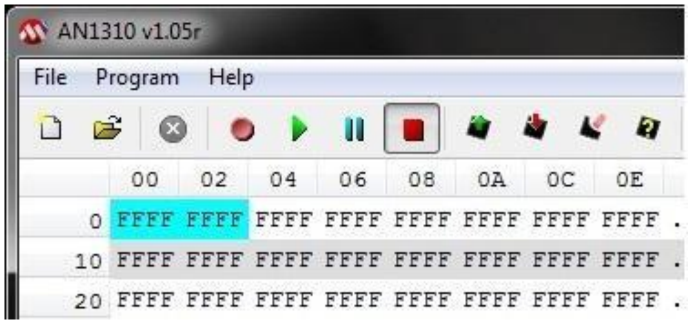
Note: Make sure not to select Config Bits and EEPROM Settings as these can damage the bootloader.

- Step3: Go to Programs -> Break/Reset Mode or Press F3.



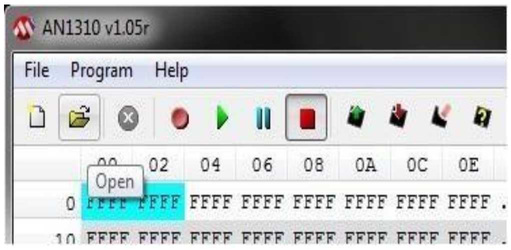
- Step4: Press the Reset Switch on the Micro-PIC18F Board.

- Step5: Go to Programs ->Bootloader Mode or Press F4.



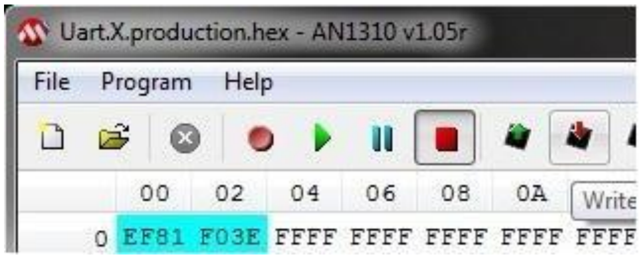
```
     00   02   04   06   08   0A   0C   0E
 0  FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF .
10  FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF .
20  FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF .
```

- Step6 : Select Hex file :
  - File ->Open -> Browse to location.



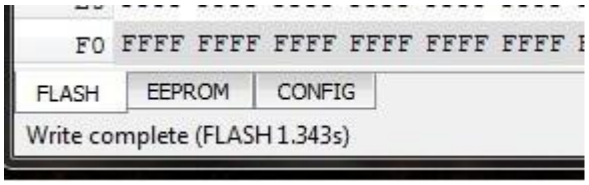
- Project folder -> dist -> default->production.

- Step7: Go to Programs -> Write Device or Press F6



```
0 EF81 F03E FFFF FFFF FFFF FFFF FFFF
```

After successful writing it will display Write Complete at the bottom.


```
F0 FFFF FFFF FFFF FFFF FFFF FFFF
```

Write complete (FLASH 1.343s)

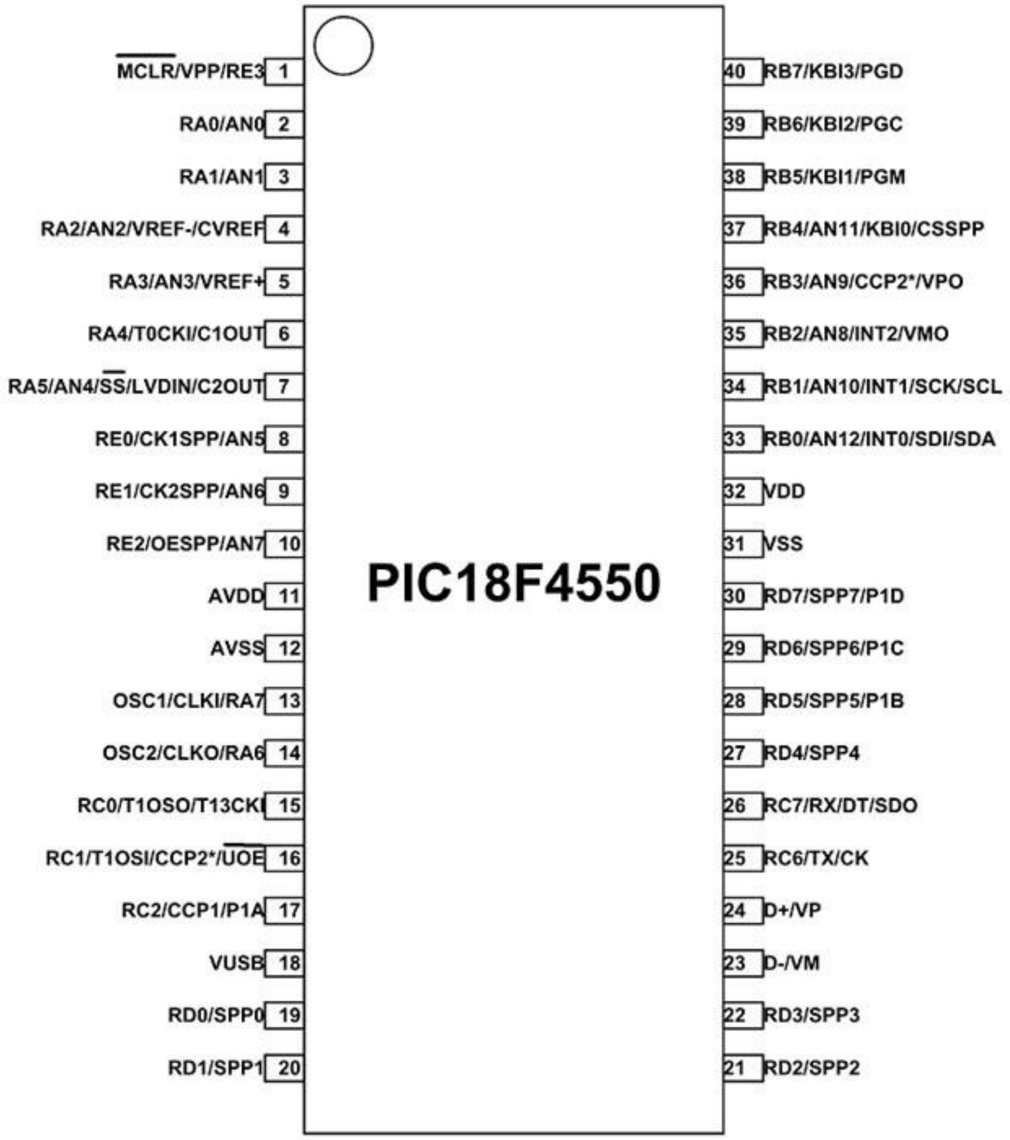
- Step8: Press Reset on the board to Run the program.

Peripheral Device Selection Matrix.

While using the peripherals for checking the output of the program please make the appropriate selection of peripheral switches SW21, SW22 and SW23.

Peripheral Device Selection Matrix			
Device	**SW21**	**SW22**	**SW23**
LED	1-2	2-3	NA
LCD	NA	1-2	NA
Seven Segment	2-3	2-3	NA
COM2	NA	NA	2-3
Keypad	NA	NA	1-2

Pin diagram of PIC18F4550:



Features of PIC18F4550:

- PIC18F4550 belongs to the PIC18F family; PIC18F4550 is an 8bit microcontroller and uses RISC architecture. PIC18F4550 has 40 pins in PDIP (dual in line package) and 44 pin in TQFP (Quad flat package).
- 32KB flash memory, 2048 bytes of SRAM (synchronous Random Access memory), EEPROM (Electrically Erasable Program Read Only Memory) of 256 bytes are embedded in the PIC18F4550.
- It has 35 I/O pins for interfacing and communication with other peripherals, 13channel of 10bit analog to digital converters which are used for interfacing and communicating the analog peripherals (DC motor, LDR, etc.).
- It has 2 CCP and 1 ECCP module that is enhanced capture and compare module which is mainly used for modulation and waveform generation functions. CCP module is of 16bit

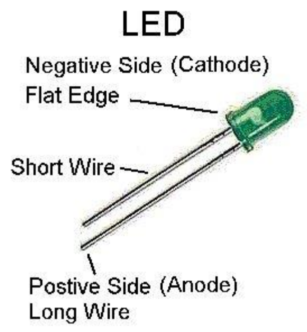
register works as 16 capture bit register, 16 compare bit register, and PWM and duty cycle register.

- PIC18F4550 has SPI (serial peripheral interface) and i2c (inter integrated circuit) for master and slave modes. It has SPP (Streaming Parallel Port) for USB streaming transfer.
- PIC18F4550 is embedded with 4 timer modules (timer0 to timer3), 2 comparator modules and 3 external interrupt. It has Dual Oscillator options allow microcontroller and USB module to run at different clock speeds. It can operate in 2.0V to 5.5V

Introduction of LED

A light-emitting diode (LED) is a two-lead semiconductor light source. It is a p–n junction diode that emits light when activated.[When a suitable voltage is applied to the leads, electrons are able to recombine with electron holes within the device, releasing energy in the form of photons. This effect is called electroluminescence, and the color of the light (corresponding to the energy of the photon) is determined by the energy band gap of the semiconductor.

When operated in a forward biased direction Light Emitting Diodes are semiconductor devices that convert electrical energy into light energy.



Light Emitting Diode color

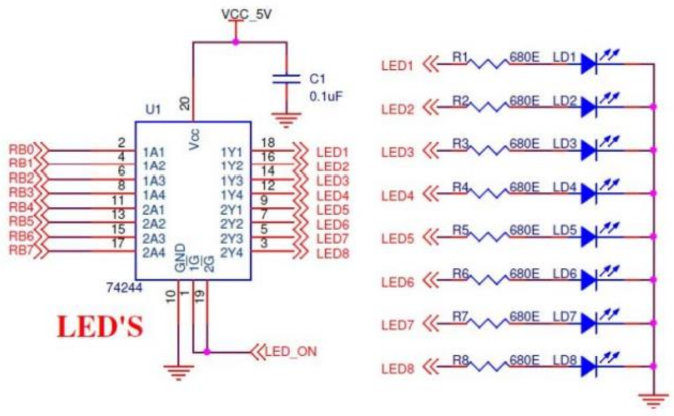
So how does a light emitting diode get its color? Unlike normal signal diodes which are made for detection or power rectification, and which are made from either Germanium or Silicon semiconductor materials, **Light Emitting Diodes** are made from exotic semiconductor compounds

Different LED compounds emit light in specific regions of the visible light spectrum and therefore produce different intensity levels.

The LED is to be connected in a forward bias condition across a power supply it should be *current limited* using a series resistor to protect it from excessive current flow. Never connect an LED directly to a battery or power supply as it will be destroyed almost instantly because too much current will pass through and burn it out.

Interfacing details for LED

Device	Pin Details
LED1	RB0
LED2	RB1
LED3	RB2
LED4	RB3
LED5	RB4
LED6	RB5
LED7	RB6
LED8	RB7



8 general purpose LED’s are provided on the Micro-PIC18F V3 Board in common anode configuration. LD1 thru LD8 are interface to Port Pins RB0 thru RB7. In order to make the LED ON we have to give Logic “1”(HIGH).

For this device to operate correctly connect the switch SW21 between 1-2 and SW22 between 2-3.

Procedure:

Step1: Open MPLABX IDE on the PC for program development and create a new project and save it in a new folder.

Step2: Write the program in C language for interfacing LEDs to PIC18F4550. **(in program properties make sure to add the 0x800 offset)**

Step3: Build the program and create hex file. In case of errors correct program and rebuild to create hex file.

Step4: Prepare the experimental setup by connecting the MicroPIC18F board to the PC using USB cable. Power ON the Board. Check for the USBtoSerial COMx allocated by the PC.

Step5: Using the PICLoader Software flash the hex file in the PIC18F4550.

Step6: Press reset button and execute the program.

Program:

```
#include <p18f4550.h>
void delay(unsigned int time)
{
        unsigned int i,j;
        for(i=0;i<time;i++)
                for(j=0;j<5000;j++);
}
void main(void)
{
      TRISB = 0x00;
      LATB = 0xFF;
      while(1)                                                        //Loop forever;
      {
              LATB = ~LATB;
              delay(200);
      }
}
```

Result: Check if the LEDs are blinking. You can change the delay and vary the blinking rate.

Conclusion: Thus, we have studied interfacing PIC18FXXX to LED and blinking it using specified delay.

Experiment 5

Aim: Write an Embedded C program for ISR based buzzer on/off using Timer.

Experimental Setup: MicroPIC18F board, USB cable, Power supply adaptor, MPLABx IDE, PICLoader software.

Theory:

Buzzer

Buzzer is an electrical device, which is similar to a bell that makes a buzzing noise and is used for signaling. Typical uses of **buzzers** and beepers include alarm devices, timers and confirmation of user input such as a mouse click or keystroke.

What is Active and Passive Buzzer?

Buzzers are mainly divided into two types first one is active buzzers and the second one is passive buzzers. In an active buzzer, additional circuitry is added to make the use it easier, but it only produces a single type of sound or tone. It required **a dc power source** for generating the **sound** or beep. Similarly, passive buzzer can generate different sounds or beep. It depends upon the frequency which is provided to this passive buzzer. For generating any type of sound, the desired frequency signal is provided to its oscillating circuit then it generates the desired frequency sound or beep.

It required an ac power source for generating the sound or beep. For identifying it is active or passive then this could check after supplying ac and **dc voltages** across the buzzer. Now the buzzers are available that could use as an active and passive buzzer just changing the command which is given to this through any type of controller. Such buzzers are called active standard passive buzzers. These are easily available in the market or online shop. A simple active standard passive buzzer is shown is figure1

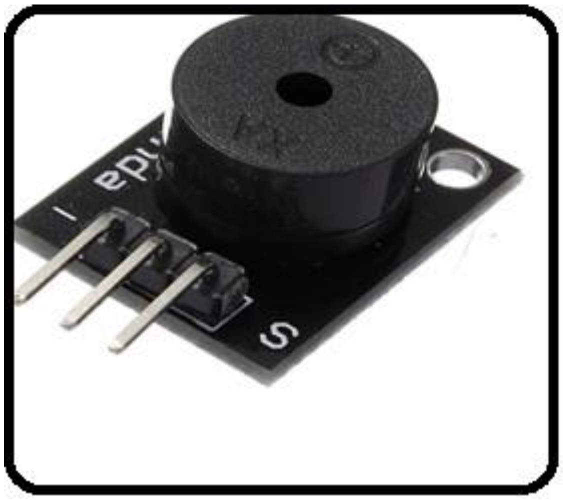


Figure 1 Active Standard Passive Buzzer

Pin Configuration of Active and Passive Buzzer

Every active and passive buzzer consist of three pins such as GND, VCC and signal pin which is shown in figure 2. For power, on this active and passive buzzer, 3.3 volts to 5 volts are given to VCC pin and GND pin. These voltages could be given through any controller or power source and the third pin is because the signal pin, therefore, it is also powered up or given signal through a controller. Through this signal pin, the active and passive buzzer is turned on or off any time. This buzzer has a limited voice if anyone wants to increase the sound or loud the voice then a transistor is added with this buzzer. Sometimes it takes more current that could not possibly form controller. In this condition, an NPN or PNP transistor is added in the circuit for safely turning on or off the buzzer form controller. Normally the resistance of active and passive buzzer is 16 ohm. It takes 100 m amps or more current therefore this could be easily operated from transistor because any NPN or PNP transistor can easily drive .5Amps current.

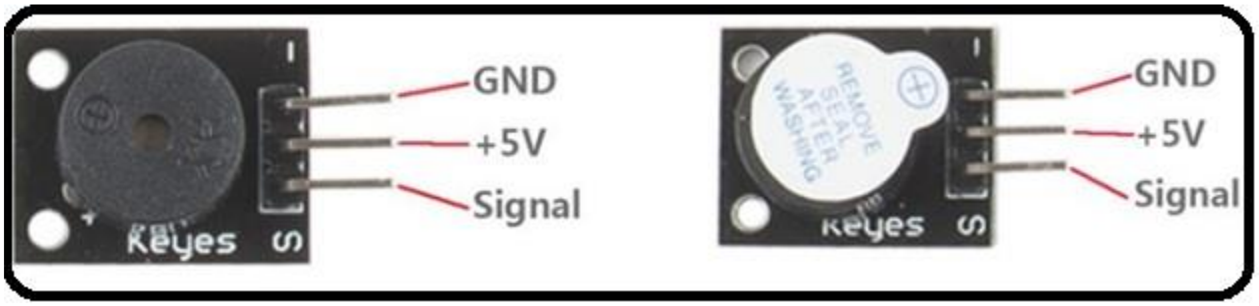


Figure 2 Passive and Active Configuration of Active Standard Passive Buzzer

Working Principle of Active and Passive Buzzer

The working principle of the active and passive buzzer is very simple. For operating it with any controller then the wiring of this buzzer is done according to its pin configuration which is mentioned in detail in the above paragraph. Actually, this buzzer consists of an electromagnetic speaker. When it is turned on through any controller then electromotive waves are generated in the form of sound. For operating it with active buzzer configuration, dc voltages are supplied this buzzer through dc power source. Similarly, for operating it in passive buzzer configuration ac voltages are supplied to this buzzer. Both types of voltages could be taken to this buzzer with any controller but the programming commands of both are different.

Procedure:

Step1: Open MPLABX IDE on the PC for program development and create a new project and save it in a new folder.

Step2: Write the program in C language for interfacing Buzzer to PIC18F4550, using Timer ISR. **(in program properties make sure to add the 0x800 offset)**

Step3: Build the program and create hex file. In case of errors correct program and rebuild to create hex file.

Step4: Prepare the experimental setup by connecting the MicroPIC18F board to the PC using USB cable. Power ON the Board. Check for the USBtoSerial COMx allocated by the PC.

Step5: Using the PICLoader Software flash the hex file in the PIC18F4550.

Step6: Press reset button and execute the program.

Program:

```
#include <pic18f4550.h>       /* Contains PIC18F4550 specifications */
#define Buzzer LATAbits.LATA5             /* Define buzzer pin */
unsigned int count = 0;

void interrupt Timer1_ISR()
{
    if(TMR1IF==1)
    {
    //1 ms delay time in timer
    TMR1L = 0x20;
    TMR1H = 0xD1;
    count ++;

    if (count >= 1000) //measure upto 1000 ms i.e. 1 seconds
    {
        Buzzer = ~Buzzer;     /* Toggle buzzer pin  */
        count = 0;  //reset count
    }
    TMR1IF = 0; //timer1 overflow flag to 0
    }
}


void main()
{
    TRISB=0;                  /* Set as output port */
    TRISAbits.TRISA5 = 0;       //set buzzer pin RA5 as output
    GIE=1;                      /* Enable Global Interrupt */
    PEIE=1;                     /* Enable Peripheral Interrupt */
    TMR1IE=1;                   /* Enable Timer1 Overflow Interrupt */
    TMR1IF=0;

    /* Enable 16-bit TMR1 register,no pre-scale,internal clock, timer OFF */
    T1CON=0x80;          /*1:8 prescale*/
    TMR1L = 0x20;
    TMR1H = 0xD1;
    TMR1ON=1;            /* Turn ON Timer1 */

    while(1);
}
```

Result: Check if the buzzer is sounding ON/OFF and the ISR is getting executed with the specified timer delay. You can change the delay and vary the sounding rate.

Conclusion: Thus, we have studied ISR based buzzer on/off using Timer.

Experiment 6

Aim: Write an Embedded C program for External Interrupt input switch press, output at Relay.

Experimental Setup: MicroPIC18F board, USB cable, Power supply adaptor, MPLABx IDE, PICLoader software.

Theory:

Introduction

Interrupts are the signals which alter the flow of an executing program by causing the microcontroller jumps to the Interrupt Service Routine to serve to interrupt. These interrupts are given to the microcontroller unit through external pins of the microcontroller.

- Once the controller completes the routine, it returns to the location from where it had made a jump.
- In cases where interrupts are not used, the program would need to constantly poll the input signals to monitor external events for catching the pulses when they occurred. But in some cases, the program can miss an event.
- E.g. An infrared slot sensor trying to catch a coin drop. In this situation, using an interrupt can free the microcontroller to get some other work done while not missing the input. In such cases, external interrupts are used.

External Interrupt Pins

PIC18F4550 has three external hardware interrupts - INT0, INT1, and INT2. They are on PORTB pins RB0, RB1, and RB2

These interrupts are edge-triggered interrupts i.e. triggered by either a rising edge or by a falling edge.

The edge trigger bit is present in an INTCON2 register.

So let’s see the INTCON2 Register.

Steps for Programming

Initialization

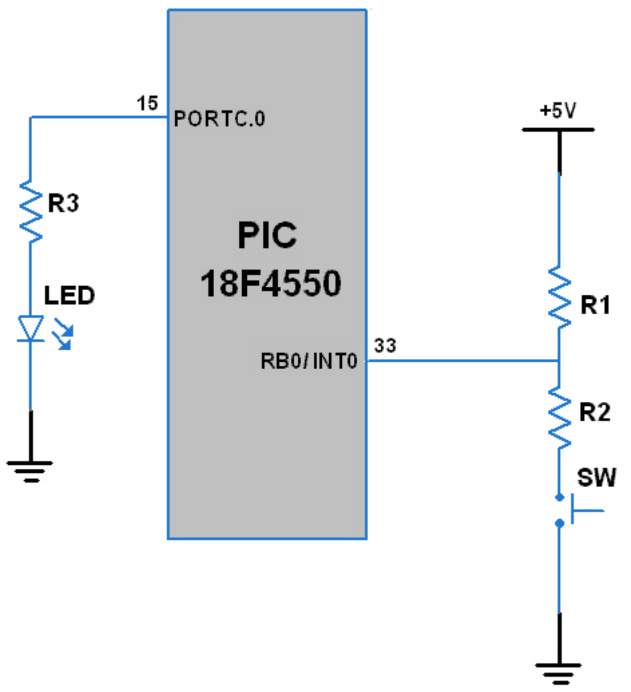
1. Set PORTB External interrupt pin as an input.

2. Also, make PORTB pins as digital input as it is multiplexed with ADC channels. This is done by disabling the ADON bit in the ADCON1 register or by making the PBADEN configuration a bit low.
3. Configure INTCON2 register for edge trigger i.e. positive or negative edge.
4. Enable external interrupt (INT0, INT1, INT2) by setting a respective interrupt to enable bit in the INTCON register.
5. Enable Global Interrupt (GIE).

Application

We are going to develop a small application on PIC18F4550 using an external interrupt. In this, we will toggle LED which is connected to the PORTC.0 pin when an external interrupt occurs.

Interfacing Diagram



- Here, the external interrupt event is coming from the switch connected to pin PORTB.0 (INT0). This event is a negative edge triggered. When the switch is pressed, a low pulse is generated on the INT0 pin. This will generate an interrupt as the INTCON2 register is configured to detect a negative edge trigger (high to low pulse).
- Then the control will move to the ISR in which the LED on PORTC.0 will toggle and the control will move to the main program.

- So there is no need to poll the PORTB.0 pin continuously.

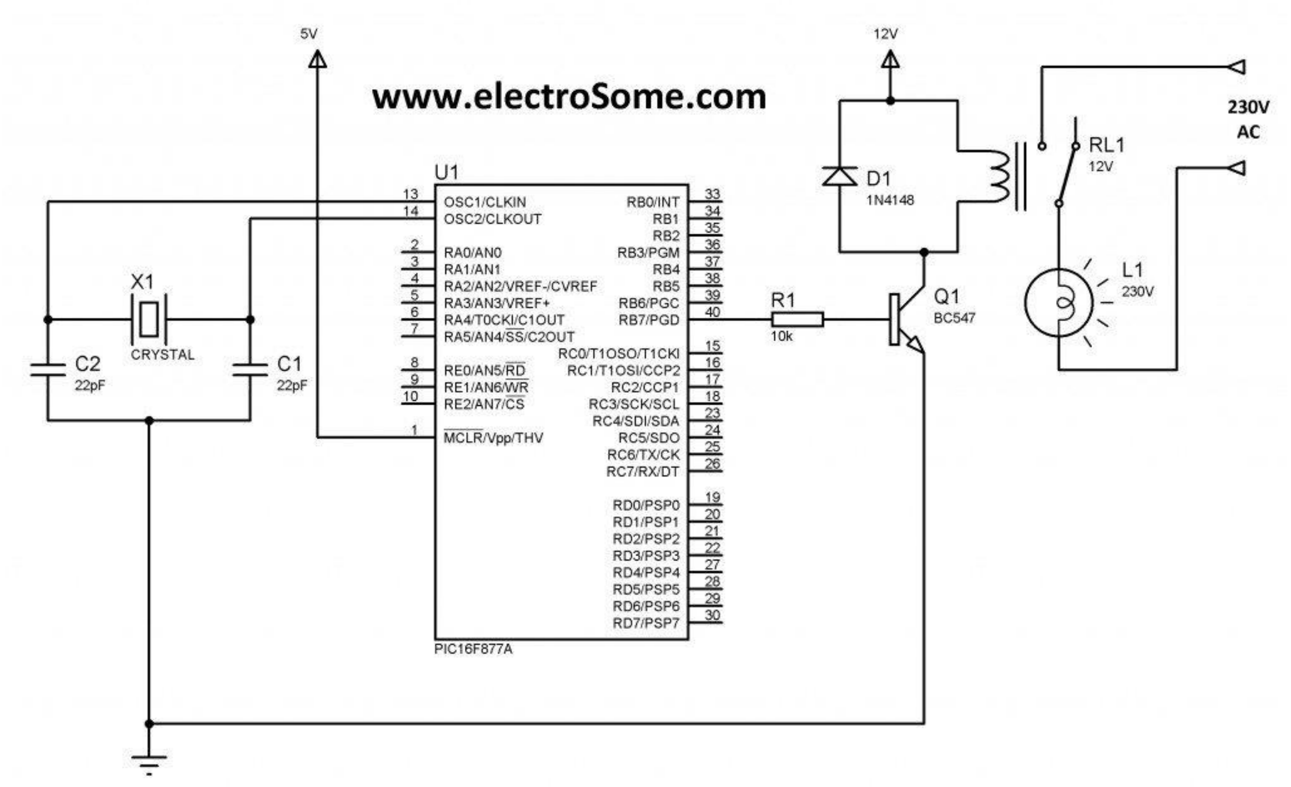
Interfacing Relay with PIC Microcontroller

A relay is an electromagnetic switch which is used to switch High Voltage/Current using Low power circuits. Relay isolates low power circuits from high power circuits. It is activated by energizing a coil wounded on a soft iron core. For detailed working of relay please visit this page. A relay should not be directly connected to a microcontroller, it needs a driving circuit.
A relay should not be connected directly to a microcontroller due to following reasons..

- A microcontroller is not able to supply current required for the working of a relay. The maximum current that a PIC Microcontroller can source or sink is 25mA while a relay needs about 50 – 100mA current.
- A relay is activated by energizing its coil. Microcontroller may stop working by the negative voltages produced in the relay due to its back emf.

Interfacing Relay with PIC Microcontroller using Transistor

A relay can be easily interfaced with microcontroller using a transistor as shown below. Transistor is wired as a switch which carries the current required for operation of the relay. When the pin RB7 of the PIC microcontroller goes high, the transistor BC547 turns On and current flows through the relay. The diode D1 is used to protect transistor and the microcontroller from Back EMF generated in the relays coil. Normally 1N4148 is preferred as it is a fast switching diode having a peak forward current of 450mA. This diode is also known as freewheeling diode.



Note: VDD and VSS of the pic microcontroller is not shown in the circuit diagram. VDD should be connected to +5V and VSS to GND.

Procedure:

Step1: Open MPLABX IDE on the PC for program development and create a new project and save it in a new folder.

Step2: Write the program in C language for interfacing Relay to PIC18F4550, using External Interrupt ISR. **(in program properties make sure to add the 0x800 offset)**

Step3: Build the program and create hex file. In case of errors correct program and rebuild to create hex file.

Step4: Prepare the experimental setup by connecting the MicroPIC18F board to the PC using USB cable. Power ON the Board. Check for the USBtoSerial COMx allocated by the PC.

Step5: Using the PICLoader Software flash the hex file in the PIC18F4550.

Step6: Press reset button and execute the program.

Program:

```
#include <pic18f4550.h>

#define RELAY_PIN LATAbits.LATA4

void interrupt extint_isr(void)
{
    unsigned int i;
    if(INT1F)
    {
        INT1F = 0;
        INT1IE = 0;
        RELAY_PIN = ~RELAY_PIN;
        for(i=0; i<10000; i++);      //small delay for debouncing
        INT1IE = 1;
    }
}

int main()
{
    ADCON1 = 0x0F;          //set pins as Digital
    TRISAbits.TRISA4 = 0;   //set relay pin RA4 as output
    TRISBbits.TRISB1 = 1;   //Interrupt pin as input
    RELAY_PIN = 1;

    INT1IE  =   1;                      //Enable external interrupt INT1
    INTEDG1 =   0;                      //Interrupt on falling edge
    GIE     =   1;                      // Enable global interrupt

    while(1);
}
```

Result: Check if the Relay is switching ON/OFF when external interrupt switch is pressed and the ISR is getting executed .

Conclusion: Thus, we have studied External Interrupt input switch press, output at Relay.

Experiment 7

Aim: Write an Embedded C program for generating PWM signal for DC/Servo motor on PIC18Fxxx.

Experimental Setup: MicroPIC18F board, USB cable, Power supply adaptor, MPLABx IDE, PICLoader software.

Theory:

What Are The Applications Of PWM?

There are numerous situations in which it's ideal to have a PWM output signal. That's why there are countless applications which are mainly dependent on the pulse width modulation technique. I'm going to introduce some of them down below:

- Light intensity control.
- Motor speed control.
- Audio signal generation.
- Servo control (valves, motors, etc).
- Voltage regulation. By switching voltage to the load with the appropriate duty cycle, the output will approximate a voltage at the desired level. The switching noise is usually filtered with an LC network.
- The solar tracking and charging systems.

And much more, you can google "PWM Applications" to find out hundreds of applications related to PWM.

Notes For PWM Mode

I- **Clearing the CCP1CON register**:

Clearing the CCP1CON register will force the CCP1 PWM output latch to the default low level. This is not the PORTC I/O data latch.

II- **Timer2 postscaler**:

The Timer2 postscaler is not used in the determination of the PWM frequency. The postscaler could be used to have a servo update rate at a different frequency than the PWM output.

III- **Duty Cycle value**:

If the PWM duty cycle value is longer than the PWM period, the CCP1 pin will not be cleared.

IV- **Double-Buffering & Glitches**:

CCPR1L and CCP1CON<5:4> can be written to at any time, but the duty cycle value is not latched into CCPR1H until after a match between PR2 and TMR2 occurs (i.e., the period is complete). In PWM mode, CCPR1H is a read-only register. The CCPR1H register and a 2-bit internal latch are used to double-buffer the PWM duty cycle. This double-buffering is essential for glitch-free PWM operation.

V- **PWM Resolution**:

The PWM duty cycle is specified by writing to the CCPR1L register and to the CCP1CON<5:4> bits. Up to 10-bit resolution is available. The maximum PWM resolution (bits) for a given PWM frequency is given by the following formula.

$$Resolution = \frac{log(\frac{F_{osc}}{F_{PWM}})}{log(2)} bits$$

The typical resolution (in bits) for a specific PWM frequency, with a specific Prescaler ratio PS, for a system running at Fosc clock rate is given by the equation down below

$$Resolution = \frac{log(\frac{F_{osc}}{PS \times F_{PWM}})}{log(2)} bits$$

Configuring The CCP1 Module For PWM Operation

Step1 – Configure the CCP module for PWM operation

- As mentioned in the datasheet in the CCP1CON register's description. Writing 11xx to the CCP1M0:CCP1M3 4-Bits selects the PWM mode of operation. Note that xx means that Bit0,1 are don't cares! that's why we'll neglect them while writing the code.

Step2 – Configure the CCP1 pin (RC2) to be an output pin by clearing the corresponding TRISC2 bit

- The CCP1/RC2 pin should be configured as an output pin.

Step3 – Determine the PWM frequency and get the PWM period value

- Let's say we need to have a PWM signal with a frequency of 5kHz. This means that the time period should be = 1/F = 1/5000 = 200µs

Step4 – Using The PWM period, Calculate the value which we'll load to the PR2 Register

- This step involves choosing a value for Timer2 prescaler prior to determining the PR2 register's value. For this step, we'll be using the following equation

$$PWM_{Period} = [(PR2) + 1] \times 4 \times T_{osc} \times Timer2PreScalerValue$$

- We now have the PWMperiod, the Tosc = 1/Fosc, and we'll pick an initial value for Timer2Prescaler (say 1 for 1:1 ratio). It should be easy to solve for PR2 to get its value. The rule of thumb is that PR2 register is an 8-Bit register which means its value ranges from (0 to 255). Your result should never exceed this 255 cap. Otherwise, you should try another value for the Timer2Prescaler. Finally, after getting the PR2 value, we should move this number to the PR2 register.

Step5 – Set the prescaler of Timer2

- Adjust the Timer2 prescaler value to match the value which you've already used in the previous step. This is done by writing to the T2CON register T2CKPSx bits.

Step6 – Step The PWM Duty Cycle, by writing to the CCPR1L register & CCP1CON<5:4> Bits

- To set or change the Duty Cycle of the PWM output signal, you should calculate and write the DC value to the 10-Bit buffer register which consists of CCPR1L:CCP1CON<5:4>. The CCPR1L contains the eight MSBs and the CCP1CON<5:4> contains the two LSbs. This 10-bit value is represented by CCPR1L:CCP1CON<5:4>.
- The following equation is used to calculate the PWM duty cycle in time:

$$PWM_{DutyCycle} = (CCPR1L : CCP1CON < 5 : 4 >) \times T_{osc} \times TMR2_{presclaer}$$

- Let's say you're willing to get 40% DC for your PWM output. Then you should multiply 0.40 by PWMperiod To get the PWMDutyCycle in time. Now, substitute for PWMDutyCycle , Tosc , TMR2prescaler in the previous equation and solve for the 10-Bit value between the ( )
- After getting the 10-Bit duty cycle from your calculations, just write this value to the buffer CCPR1L:CCP1CON<5:4>

Step7 – Turn ON Timer2

Don't forget to turn ON the Timer2 module! it should be obvious that it's the working-horse of our system. However, it happens to find out that you've missed doing so and that's why nothing is working at all.

Procedure:

Step1: Open MPLABX IDE on the PC for program development and create a new project and save it in a new folder.

Step2: Write the program in C language for interfacing DC motor to PIC18F4550 and varying speed using PWM . **(in program properties make sure to add the 0x800 offset)**

Step3: Build the program and create hex file. In case of errors correct program and rebuild to create hex file.

Step4: Prepare the experimental setup by connecting the MicroPIC18F board to the PC using USB cable. Power ON the Board. Check for the USBtoSerial COMx allocated by the PC.

Step5: Using the PICLoader Software flash the hex file in the PIC18F4550.

Step6: Press reset button and execute the program.

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
    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);
        }
    }
}
```

Result: Check if the DC motor speed varies .

Conclusion: Thus, we have studied generating PWM signal for DC/Servo motor on PIC18Fxxx.

Experiment 8

Aim: Write an Embedded C program for LCD interfacing with PIC18Fxxx.

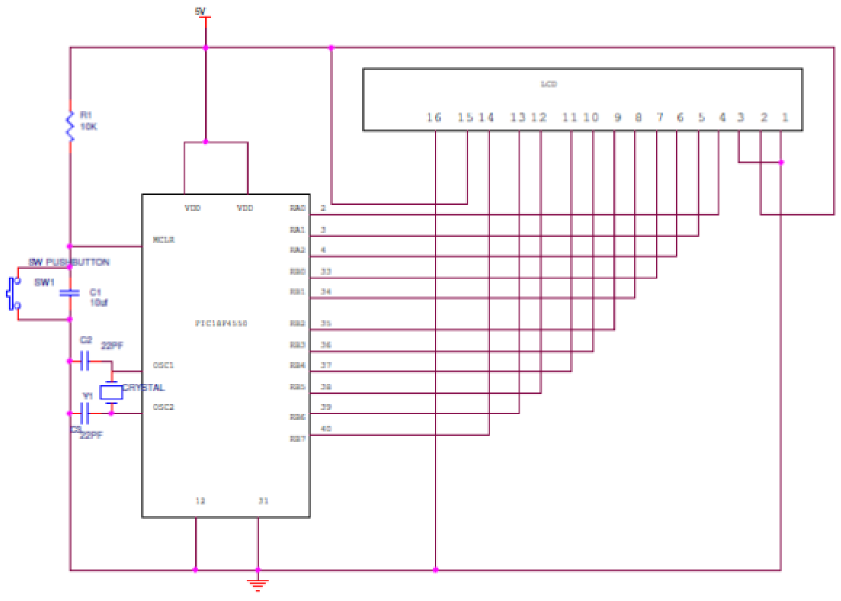
Experimental Setup: MicroPIC18F board, USB cable, Power supply adaptor, MPLABx IDE, PICLoader software.

Theory:

Features of PIC18F4550:

- PIC18F4550 belongs to the PIC18F family; PIC18F4550 is an 8bit microcontroller and uses RISC architecture. PIC18F4550 has 40 pins in PDIP (dual in line package) and 44 pin in TQFP (Quad flat package).
- 32KB flash memory, 2048 bytes of SRAM (synchronous Random Access memory), EEPROM (Electrically Erasable Program Read Only Memory) of 256 bytes are embedded in the PIC18F4550.
- It has 35 I/O pins for interfacing and communication with other peripherals, 13channel of 10bit analog to digital converters which are used for interfacing and communicating the analog peripherals (DC motor, LDR, etc.).
- It has 2 CCP and 1 ECCP module that is enhanced capture and compare module which is mainly used for modulation and waveform generation functions. CCP module is of 16bit register works as 16 capture bit register, 16 compare bit register, and PWM and duty cycle register.
- PIC18F4550 has SPI (serial peripheral interface) and i2c (inter integrated circuit) for master and slave modes. It has SPP (Streaming Parallel Port) for USB streaming transfer.
- PIC18F4550 is embedded with 4 timer modules (timer0 to timer3), 2 comparator modules and 3 external interrupt. It has Dual Oscillator options allow microcontroller and USB module to run at different clock speeds. It can operate in 2.0V to 5.5V

16X2 LCD Interfacing PIC Microcontroller Circuit Diagram:



16X2 LCD Interfacing PIC Microcontroller – Circuit Explanation:
The resistor R1 is used for giving the contrast to the LCD. The crystal oscillator of 12 MHz is connected to the OSC1 and OSC2 pins of Pic microcontroller PIC18F4550 for system clock. The capacitor C2 and C3 will act filters to the crystal oscillator. You can use different ports or pins for interfacing the LCD before going to different ports please check the data sheet whether the pins for general purpose or they are special function pins.

Programming PIC for Interfacing 16X2 LCD:

In the pic programming also for initializing the LCD the R/W pin should be low for writing the data, Enable pins should be high and register select pin (RS) should be high for writing the data. For sending a command the RS should be low, R/W pin should be low and enable pin should be high.

Initializing the LCD function:
lcdcmd(0x38); //Configure the LCD in 8-bit mode,2 line and 5×7 font
lcdcmd(0x0C); // Display On and Cursor Off
lcdcmd(0x01); // Clear display screen
lcdcmd(0x06); // Increment cursor
lcdcmd(0x80); // Set cursor position to 1st line,1st column

Sending command to the LC:

- rs=0; Register select pin is low.

.

- rw=0; Read/write Pin is also for writing the command to the LCD.
- en=1;enable pin is high.

Sending data to the LCD:

- rs=1; Register select pin is high.
- rw=0; Read/write Pin is also for writing the command to the LCD.
- en=1; enable pin is high.
-

Peripheral Device Selection Matrix.

While using the peripherals for checking the output of the program please make the appropriate selection of peripheral switches SW21, SW22 and SW23.

Peripheral Device Selection Matrix			
Device	**SW21**	**SW22**	**SW23**
LED	1-2	2-3	NA
LCD	NA	1-2	NA
Seven Segment	2-3	2-3	NA
COM2	NA	NA	2-3
Keypad	NA	NA	1-2

Procedure:

Step1: Open MPLABX IDE on the PC for program development and create a new project and save it in a new folder.

Step2: Write the program in C language for interfacing 16x2 LCD to PIC18F4550.
(in program properties make sure to add the 0x800 offset)

Step3: Build the program and create hex file. In case of errors correct program and rebuild to create hex file.

Step4: Prepare the experimental setup by connecting the MicroPIC18F board to the PC using USB cable. Power ON the Board. Check for the USBtoSerial COMx allocated by the PC.

Step5: Using the PICLoader Software flash the hex file in the PIC18F4550.

Step6: Press reset button and execute the program.

Result: Check if the characters are getting printed on the LCD screen .

Program:

```
#include <p18f4550.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<100;j++);
    }
}


void SendInstruction(unsigned char command)
{
     LCD_RS = 0;        // select command
     register 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 1st line

}
```

```
unsigned char *String1 = " Microembedded";
unsigned char *String2 = " PIC-18F Board";

void main(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 1st line

 while(*String1)
 {
  SendData(*String1);
  String1++;
 }

 SendInstruction(0xC0);      //set address to 2nd line
 while(*String2)
 {
  SendData(*String2);
  String2++;
 }

 while(1);

}
```

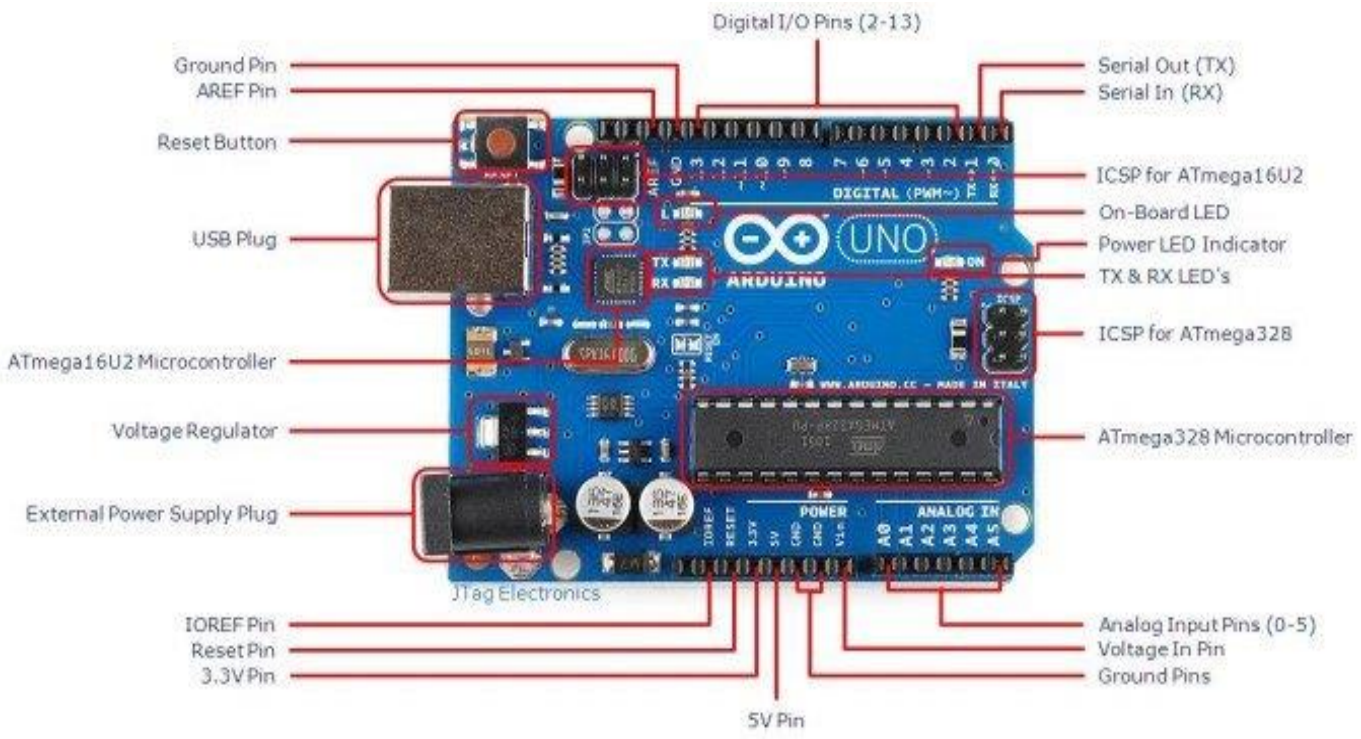
Conclusion: Thus, we have studied Embedded C program for LCD interfacing with PIC18Fxxx.

Experiment 9

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

Theory:

Arduino:- Arduino is an open-source hardware and software company, project and user community that designs and manufactures single-board microcontrollers and microcontroller kits for building digital devices and interactive objects that can sense and control objects in the physical and digital world. Arduino board designs use a variety of microprocessors and controllers. The boards are equipped with sets of digital and analog input/output (I/O) pins that may be interfaced to various expansion boards or breadboards (shields) and other circuits. The boards feature serial communications interfaces, including Universal Serial Bus (USB) on some models, which are also used for loading programs from personal computers. The microcontrollers are typically programmed using a dialect of features from the programming languages C and C++. In addition to using traditional compiler tool chains, the Arduino project provides an integrated development environment (IDE) based on the Processing language project. Arduino is open-source hardware. The hardware reference designs are distributed under a Creative Commons Attribution Share-Alike 2.5 license and are available on the Arduino website. Layout and production files for some versions of the hardware are also available.



Here are the various components on the Arduino board: Microcontrollers ATmega328P (used on most recent boards) ATmega168 (used on most Arduino Diecimila and early Duemilanove) ATmega8 (used on some older board) Digital Pins In addition to the specific functions listed below, the digital pins on an Arduino board can be used for general purpose input and output via the pinMode(), digitalRead(), and digitalWrite() commands. Each pin has an internal pull-up resistor which can be turned on and off using digitalWrite() (w/ a value of HIGH or LOW, respectively) when the pin is configured as an input. The maximum current per pin is 40 mA. Analog Pins In addition to the specific functions listed below, the analog input pins support 10-bit analog-to-digital conversion (ADC) using the analogRead()

function. Most of the analog inputs can also be used as digital pins: analog input 0 as digital pin 14 through analog input 5 as digital pin 19. Analog inputs 6 and 7 (present on the Mini and BT) cannot be used as digital pins. Power Pins • VIN (sometimes labelled "9V"). The input voltage to the Arduino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin. Note that different boards accept different input voltages ranges, please see the documentation for your board. Also note that the LilyPad has no VIN pin and accepts only a regulated input.

Other Pins

• AREF. Reference voltage for the analog inputs. Used with analogReference().

• Reset. (Diecimila-only) Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

• Analog Reference pin (orange)

• Digital Ground (light green)

• Digital Pins 2-13 (green)

• Digital Pins 0-1/Serial In/Out - TX/RX (dark green) - These pins cannot be used for digital i/o (digitalRead and digitalWrite) if you are also using serial communication (e.g. Serial.begin).

• Reset Button - S1 (dark blue)

• In-circuit Serial Programmer (blue-green) • Analog In Pins 0-5 (light blue)

• Power and Ground Pins (power: orange, grounds: light orange) • External Power Supply In (9-12VDC) - X1 (pink)

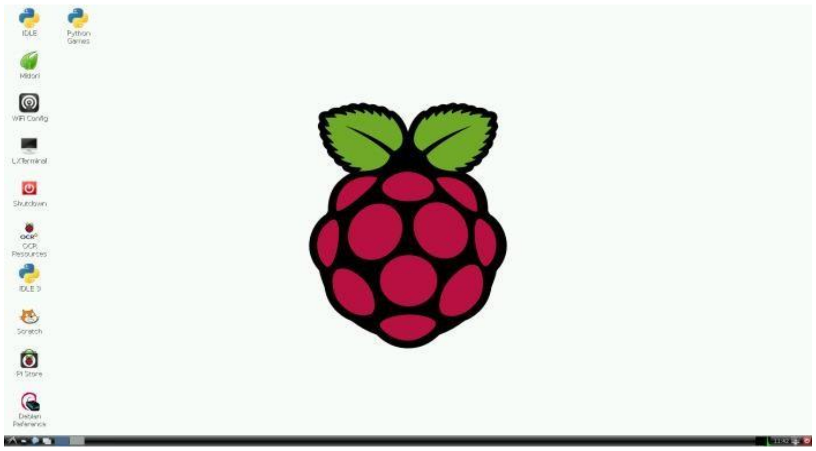
• Toggles External Power and USB Power (place jumper on two pins closest to desired supply) - SV1 (purple)

• USB (used for uploading sketches to the board and for serial communication between the board and the computer; can be used to power the board) (yellow)

OS installation process on Raspberry-pi.

1. Raspbian

A free Debian-based OS optimized for Raspberry Pi's hardware, Raspbian comes with all the basic programs and utilities you expect from a general-purpose operating system. Supported officially by the Raspberry foundation, this OS is popular for its fast performance and its more than 35,000 packages.



The easiest way to install Raspbian on your Pi is by deploying its image file onto an SD card. It uses the lightweight LXDE desktop for its user-friendly graphical session. Its helpful community supports and develops free software for Raspbian, and you'll need a 8 GB SD card to load its latest version, Jessie.

How to install Raspbian on Raspberry-Pi:

Step 1: Download Raspbian

Step 2: Unzip the file. The Raspbian disc image is compressed, so you'll need to unzip it. The file uses the ZIP64 format, so depending on how current your built-in utilities are, you need to use certain programs to unzip it.

Step 3: Write the disc image to your microSD card. Next, pop your microSD card into your computer and write the disc image to it. The process of actually writing the image will be slightly different across these programs, but it's pretty self-explanatory no matter what you're using. Each of these programs will have you select the destination (make sure you've picked your microSD card!) and the disc image (the unzipped Raspbian file). Choose, double-check, and then hit the button to write.

Step 4: Put the microSD card in your Pi and boot up. Once the disc image has been written to the microSD card, you're ready to go! Put that sucker into your Raspberry Pi, plug in the peripherals and power source, and enjoy. The current edition to Raspbian will boot directly to the desktop. Your default credentials are username pi and password raspberry.

Installation using NOOBS Operating Systems Raspbian

The Raspberry Pi is an incredible device, but it won't do much of anything without an operating system. Luckily, choosing and installing an appropriate operating system on your Raspberry Pi has never been easier. One simple method is to use NOOBS, or "New Out of Box Software." As the name suggests, NOOBS is perfect for Pi newbies. It lets you choose your preferred operating system and install it right then and there. But how do you load NOOBS itself? Here's our complete guide on how to install NOOBS on the Raspberry Pi.

Luckily for us, the process is extremely simple. All you'll need is a Raspberry Pi, a computer, and an SD or microSD card. Check out the complete instruction below.

How to install NOOBS on the Raspberry Pi

We've called our article "How to install NOOBS on the Raspberry Pi," but what we're technically doing is installing it on a flash drive, booting to the drive on the Raspberry Pi, and then using NOOBS to choose and install an operating system.

NOOBS has plenty of operating systems for us to choose from when we reach that step – the most notable of which is Raspbian. For now, though let's concentrate on how to install NOOBS on the Raspberry Pi. We will briefly discuss the operating system installations later, in our final step.

The optional easy route: buy a NOOBS SD card.

Installing NOOBS on an SD card isn't hard, but it also isn't necessary. If you'd like, you can choose to buy an SD card that comes pre-loaded with NOOBS. If you go that route, you can skip all the way to the final step!

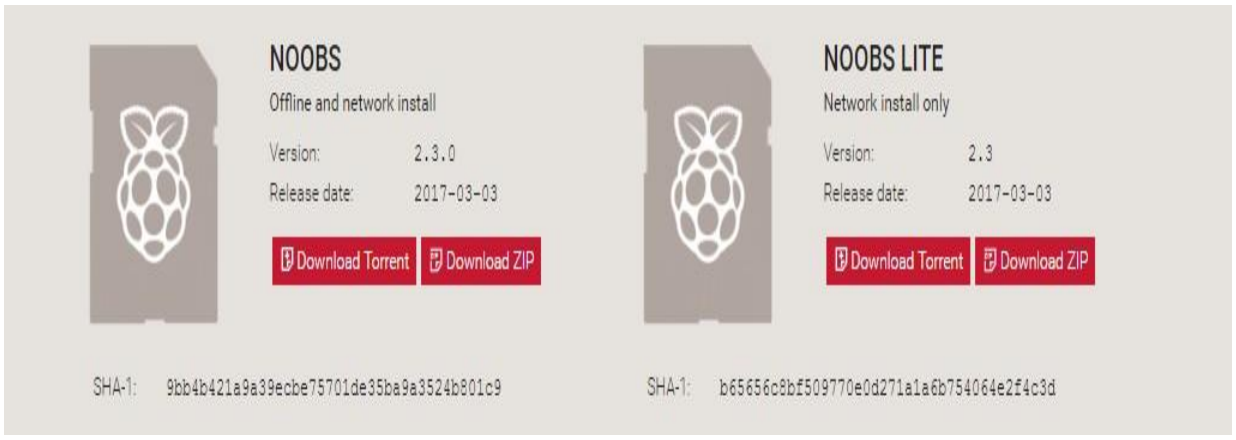
If you want to do things yourself, though, just read on.

What you'll need to install NOOBS on the Raspberry Pi

This project is pretty simple. Besides your Raspberry Pi and essential peripherals, here's all you'll need:

- A computer with an SD card slot
- An SD or microSD card of at least 8 GB

Step 1: Download NOOBS and extract it



NOOBS
Offline and network install
Version: 2.3.0
Release date: 2017-03-03
SHA-1: 9bb4b421a9a39ecbe75701de35ba9a3524b801c9

NOOBS LITE
Network install only
Version: 2.3
Release date: 2017-03-03
SHA-1: b65656c8bf509770e0d271a1a6b754064e2f4c3d

You're going to use your computer to put NOOBS on an SD card – so step one is to get NOOBS onto your computer!

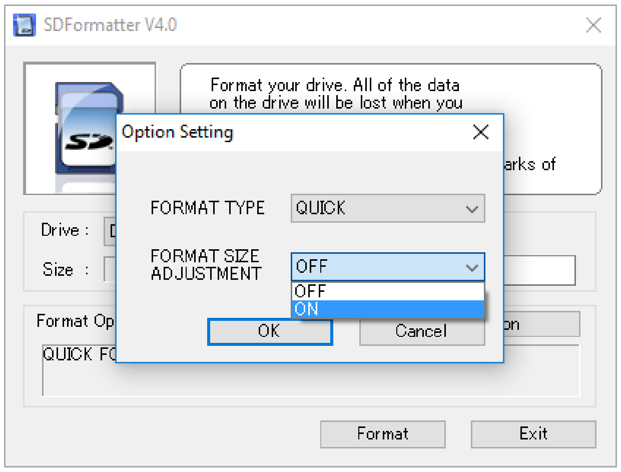
The NOOBS download page will let you choose between NOOBS and "NOOBS Lite." NOOBS includes a full version of Raspbian, so you can install that particular operating system without using the internet at all. With NOOBS Lite, on the other hand, you'll need a network connection to install any of the operating systems NOOBS makes available – even Raspbian.

Go ahead and choose whichever version you would like. NOOBS will download as a .zip file, so before you do anything else, go ahead and extract it.

Step 2: Format an SD card

Now you're going to want to go ahead and stick your SD card into the corresponding slot on your computer. You're going to want to format it as FAT. There are a few ways to do this:

On Mac or Windows, use the SD Association's Formatting Tool (Mac users can also just use the disk utility). Make sure the "Format size adjustment" option is set to "on." Then erase it in FAT (or MS-DOS) format.

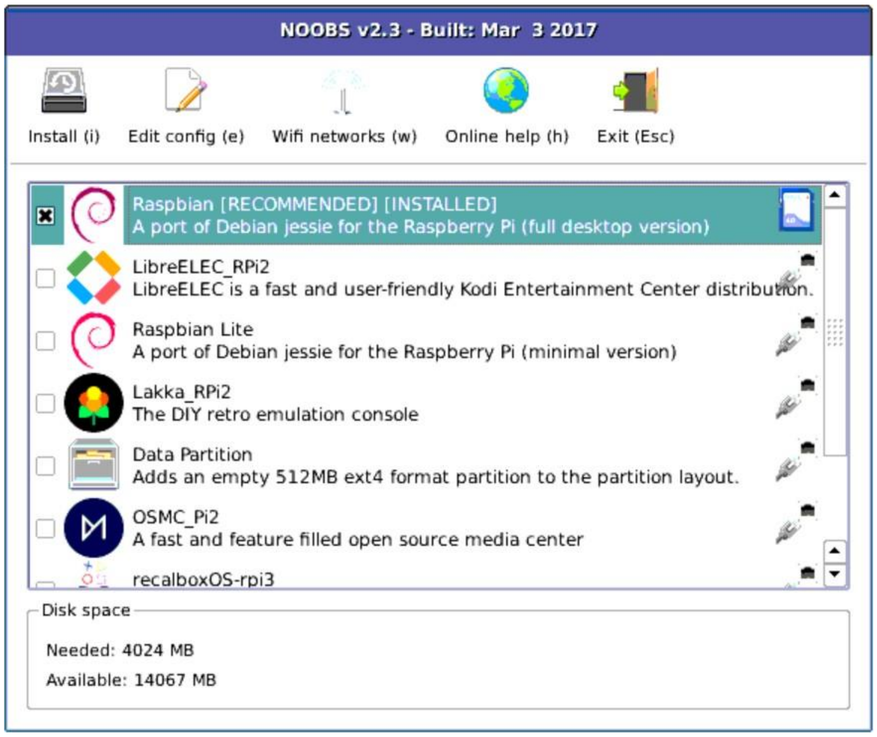


FORMAT TYPE QUICK
FORMAT SIZE ADJUSTMENT OFF

Step 3: Put the NOOBS files on the SD card

Now, just drag and drop the NOOBS files into your newly formatted SD card. You want the files only, so if your .zip extracted to a folder, open that folder up and select only the stuff inside of it.

Step 4: Put your SD card into your Raspberry Pi and boot it up



Raspbian [RECOMMENDED] [INSTALLED]
A port of Debian jessie for the Raspberry Pi (full desktop version)
LibreELEC_RPi2
LibreELEC is a fast and user-friendly Kodi Entertainment Center distribution.
Raspbian Lite
A port of Debian jessie for the Raspberry Pi (minimal version)
Lakka_RPi2
The DIY retro emulation console
Data Partition
Adds an empty 512MB ext4 format partition to the partition layout.
OSMC_Pi2
A fast and feature filled open source media center
recalboxOS-rpi3

Needed: 4024 MB
Available: 14067 MB

Once you have NOOBS on your SD card, using it is incredibly easy. Just put the SD card into your Raspberry Pi and start that sucker up. As we said before, while this guide is called "How to install NOOBS on the Raspberry Pi," the endgame here is actually to install an operating system like Raspbian, LibreELEC, OSMC, or any of the others NOOBS gives you access to.

This is the step in which that happens. After booting to NOOBS, you'll be greeted with a menu that will let you choose which operating system you'd like to install on your Pi. Your menu may look a little bit different than the one in the screenshot above, because NOOBS ingeniously adapts to your generation and model of Raspberry Pi.

Which OS should you choose? Well, that's up to you. Raspbian is probably the most frequently used, and you'll find plenty of projects here on our site that utilize it. OSMC acts as a media center, and LibreELEC boots directly to the popular media center app Kodi. Ultimately, it's all a matter of personal preference!

Once you've decided, just hit "Install" and sit back. From now on, your Pi will boot directly to that operating system. Easy, right?

And if you're not happy with the operating system you pick, you're not stuck. Just hold down the SHIFT key while booting up, and you'll be back in the NOOBS menu ready to try out a different option

Booting Your Raspberry Pi OS for the First Time

With Raspberry Pi OS installed, you'll need to login with the following credentials:

Username: pi

Password: raspberry

For other operating systems, check the documentation to find the default login credentials.

Remember that the password will not be displayed as you type it; there are no Windows-style * symbols representing the letters. Instead, it will appear that you haven't entered a password. This is a security feature in Linux to prevent people guessing the length of your passphrase. Just type the password regardless.

Once Raspberry Pi OS has booted, change your password. You should be prompted to do this in the Change Password window.

Alternatively, open **Menu > Preferences > Raspberry Pi Configuration** and in the **System Tab** click **Change Password**.

You can also change the password in the command line using the passwd command.
Installing an OS on a Raspberry Pi Is Easy

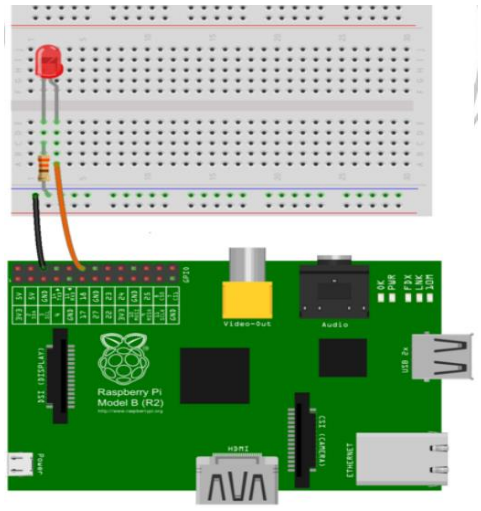
Options for installing one or more operating systems on Raspberry Pi are plentiful. You can either install single operating systems with Raspberry Pi Imager, Etcher, or a simple command line instruction, or use tools like NOOBS, Berryboot, and PINN for dual booting.

Conclusion: Thus, we have studied Arduino board and understood the OS installation process on Raspberry-pi.

Experiment 10

Aim: Write simple program using Open source prototype platform like Raspberry-Pi/Beagle board/Arduino for digital read/write using LED and switch Analog read/write using sensor & actuators.

Theory:
An application of blinking LEDs using Raspberry Pi In this assignment, we will learn how to interface an LED with Raspberry Pi. To implement this, we will gather the requirements and follow the instructions given below.
Requirements:
1. Raspberry Pi3 Kit
2. Breadboards
3. LEDs
4. Jumper wires

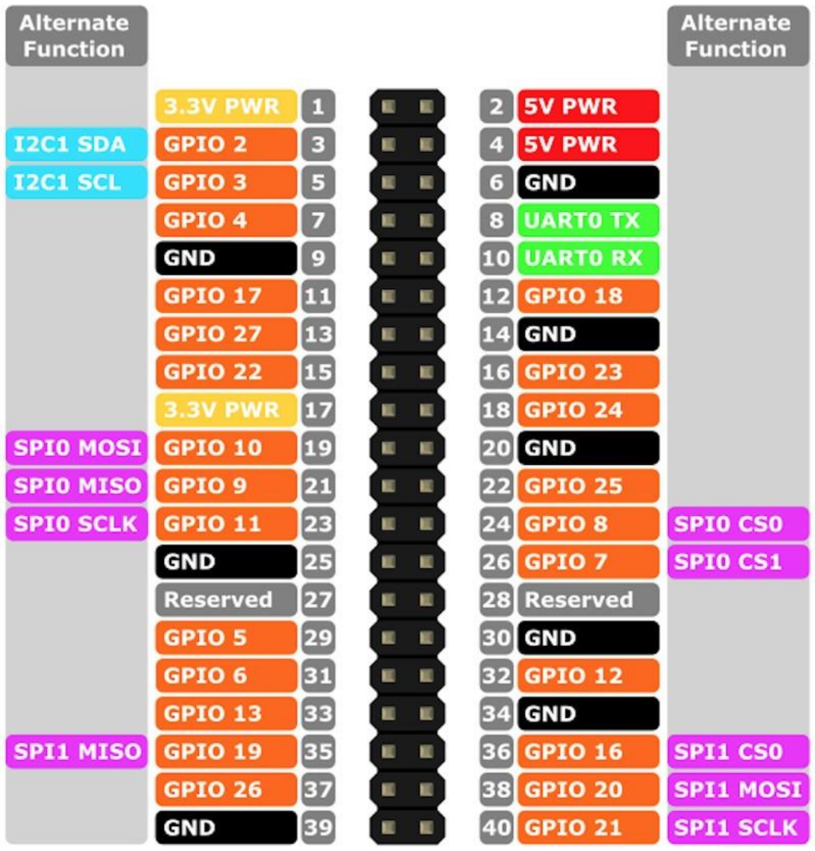


Instructions to Glow LED:
Use GPIO diagram of Raspberry Pi, which will give you understanding of GPIO structure.
Algorithm:
1. Connect GPIO 18 (Pin No. 12) of Raspberry Pi to the Anode of the LED through connecting jumper wires and Breadboard.

2. Connect Ground of Raspberry Pi to Cathode of the LED through connecting wires and breadboard.
3. Now power up your Raspberry Pi and boot.
4. Open terminal and type nano blinkled.py
5. It will open the nano editor. Use following pseudo code in the python to blink LED and save

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
GPIO.setup(18, GPIO.OUT)
while True:
        GPIO.output(18, GPIO.HIGH)
        time.sleep(1)
        GPIO.output(18, GPIO.LOW)
        time.sleep(1)
```

6. Now run the code. Type python blinkled.py
7. Now LED will be blinking at an interval of 1s. You can also change the interval by modifying time.sleep in the file.
8. Press Ctrl+C to stop LED from blinking.



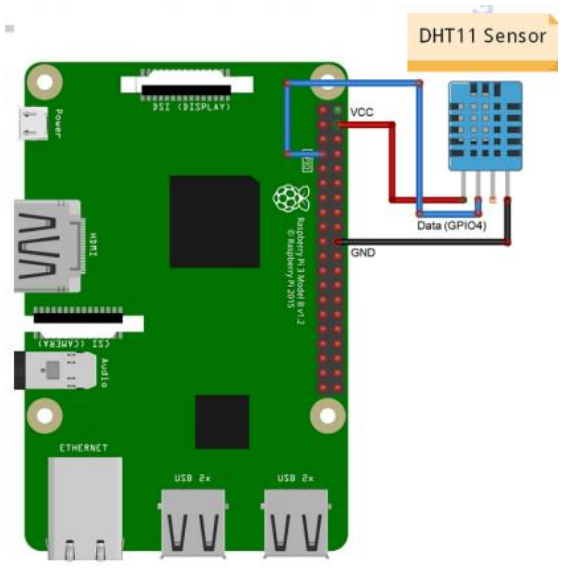
Alternate Function					Alternate Function
	3.3V PWR	1	2	5V PWR	
I2C1 SDA	GPIO 2	3	4	5V PWR	
I2C1 SCL	GPIO 3	5	6	GND	
	GPIO 4	7	8	UART0 TX	
	GND	9	10	UART0 RX	
	GPIO 17	11	12	GPIO 18	
	GPIO 27	13	14	GND	
	GPIO 22	15	16	GPIO 23	
	3.3V PWR	17	18	GPIO 24	
SPI0 MOSI	GPIO 10	19	20	GND	
SPI0 MISO	GPIO 9	21	22	GPIO 25	
SPI0 SCLK	GPIO 11	23	24	GPIO 8	SPI0 CS0
	GND	25	26	GPIO 7	SPI0 CS1
	Reserved	27	28	Reserved	
	GPIO 5	29	30	GND	
	GPIO 6	31	32	GPIO 12	
	GPIO 13	33	34	GND	
SPI1 MISO	GPIO 19	35	36	GPIO 16	SPI1 CS0
	GPIO 26	37	38	GPIO 20	SPI1 MOSI
	GND	39	40	GPIO 21	SPI1 SCLK

An application to read the environment temperature and humidity & display it.
In this assignment, we will learn how to find environment temperature and humidity using DHT11 sensor and Raspberry Pi.

Temperature and Humidity Sensor- DHT11
• It’s a very basic and slow sensor, easy to use for small scale.
• The DHT sensors are made of two parts, a capacitive humidity sensor and thermistor.
• There is also a very basic chip inside that does some analog to digital conversion and spits out a digital signal with the temperature and humidity.
• The digital signal is fairly easy to read using any microcontroller. • These sensors are frequently used in remote weather stations, soil monitors, and home environment control systems.

Features of DHT11 Sensor: -
• Ultra-low cost
• 3 to 5V power and I/O
• 2.5mA max current use during conversion (while requesting data)
• Good for 20–80% humidity readings with 5% accuracy
• Good for 0–50°C temperature readings ±2°C accuracy
• No more than 1 Hz sampling rate (once every second)
• Body size 15.5mm x 12mm x 5.5mm
• 4 pins with 0.1" spacing

Requirements:
1. Raspberry Pi3 Kit
2. Breadboards
3. DHT11 Sensor
4. Jumper wires



Algorithm:

1. Connect GPIO 18 (Pin No. 12) of Raspberry Pi to the Data pin of DHT11 sensor through connecting jumper wires and Breadboard.
2. Connect Power (5V) and Ground of Raspberry Pi to DHT11 sensor through connecting wires and breadboard.
3. Now power up your Raspberry Pi and boot.
4. Download Adafruit_Python_DHT.tar.gz package
5. Open terminal and type tar -xvzf Adafruit_Python_DHT.tar.gz command to extract Adafruit package.
6. In the same terminal, type nano temp.py
7. It will open the nano editor. Use following pseudo code in the python to blink LED and save

```
import Adafruit_DHT as dht
import time
while True:
        hum, temp = dht.read_retry(11,18)
        print "Temp: ", temp
        print "Hum: ",
        hum time.sleep(1)
```

8. Now run the code. Type python temp.py
9. Now temperature and humidity will be displayed at an interval of 1s. You can also change the interval by modifying time.sleep in the file.
10. Press Ctrl+C to stop displaying temperature and humidity.

Conclusion: - Thus, we have studied how to monitor LEDs and measure temperature and humidity in the environment using DHT11 sensor and Raspberry Pi 3.