Nirma University
Institute of Technology
B. Tech. Sem. VII (EC)

2ECOE53 – Arduino for Engineers

Laboratory Manual

Electronics & Communication Engineering Department
Institute of Technology
Ahmedabad

General Instructions to Students:

1. The lab consists of 10 experiments and one project. The lab is not a separate component from the theory class.

2. The evaluation of the lab component will be done as follows.

Lab Experiments (0.75)	**Final Viva (0.25)**
10 experiments (75%)	25%

3. Students should come with thorough preparation of the Pre-Lab questionnaire for the experiment to be conducted.
4. Students will not be permitted to attend the laboratory unless they submit the practical record/report of the last covered experiment fully completed in all aspects.
5. Resources (Kits & Tools) to be used in the Laboratory will be provided by the department. In case students are preferring online mode of experiments, they should keep the following components handy for hands-on experimentation.
    - Arduino UNO or Mega Board and IDE in the Computer
    - USB Programming Cable
    - Bread Board and Hook Up Wires
    - Variable Resistor, LED, Push Button Switches
    - 16 x 2 LCD with and without I2C interface
    - DC and Servo Motor

Index

Exp. No. : **1** **Date: / /**
Title : **Introduction to Arduino boards and Arduino IDE**

Objective:

To acquaint the students with Arduino boards, Arduino IDE, and programming.

Learning Outcomes:
After completion of this experiment, students will be able to

- Identify various Arduino boards
- Configure the IDE to program various Arduino boards
- Differentiate between analog and digital pins of Arduino

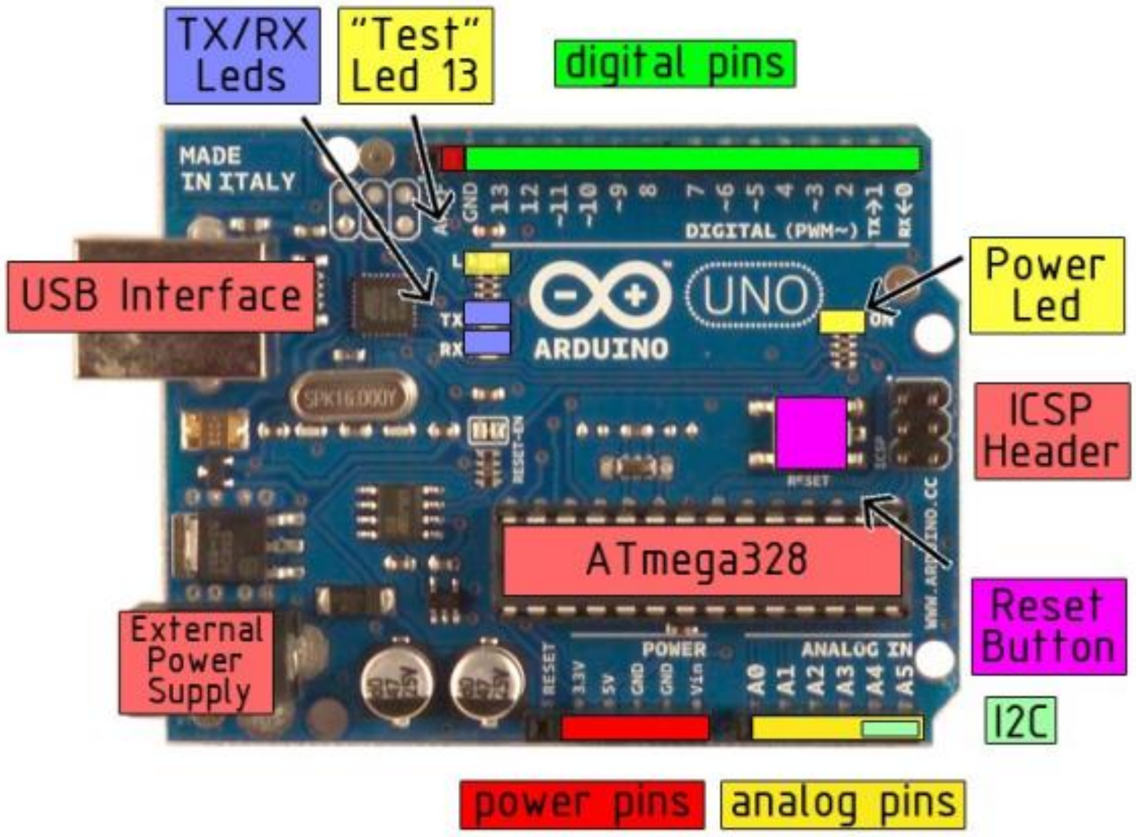
Pre Lab Exercise:

- Download Arduino IDE from the official Arduino website

Theory:
In this section types of Arduino boards, Arduino IDE setup, basics of Arduino programming, and interfacing of Arduino board with a compute are given.

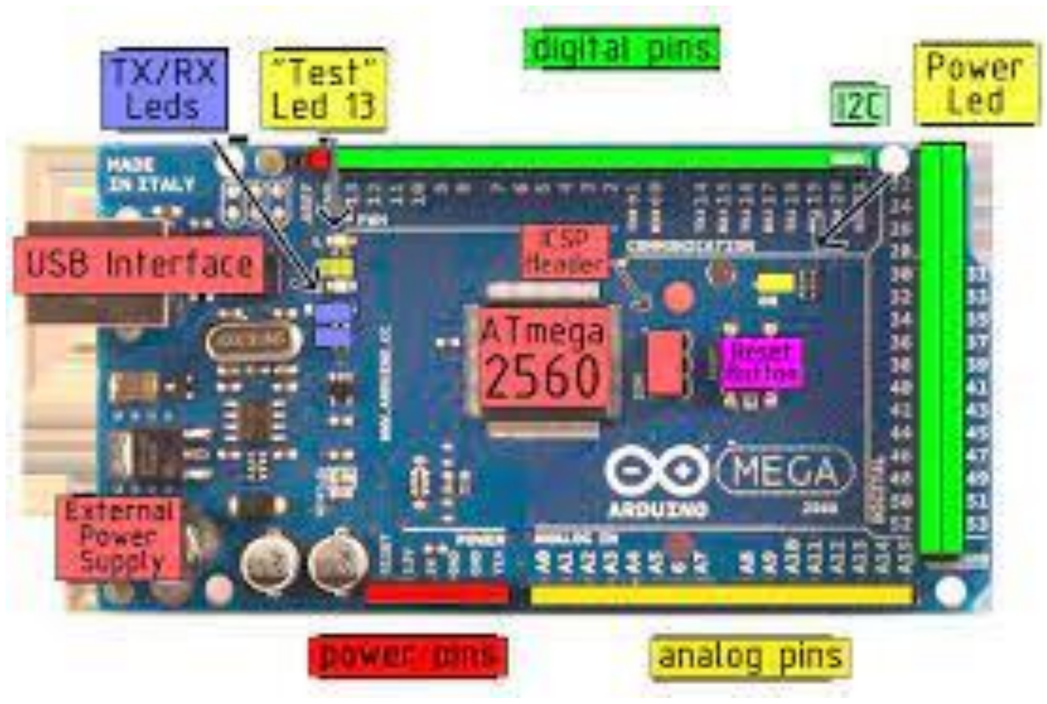
- **Arduino Boards**

Arduino uno



Courtesy: A000066-Arduino-datasheet-38879526.pdf (octopart.com)

Arduino Mega

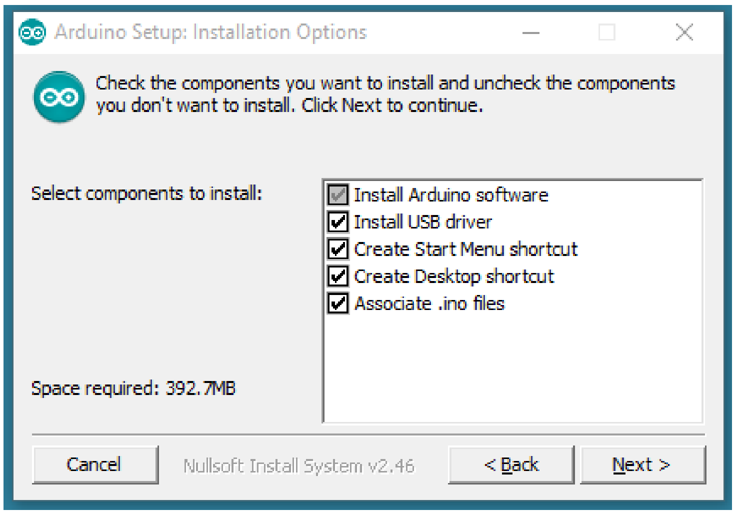


Courtesy: www.mantech.co.za/datasheets/products/A000047.pdf

- **Arduino IDE Installation guidelines:**

  Get the latest version of Arduino IDE from https://www.arduino.cc/en/Main/Software. You can choose between the Installer (.exe) and the Zip packages. Use the first one that installs directly, everything you need to use the Arduino Software (IDE), including the drivers. With the Zip package, you need to install the drivers manually. When the download finishes, proceed with the installation and allow the driver installation process when you get a warning from the operating system. Once the download is done, follow these steps for installation:

  1. Choose the components to install


Arduino Setup: Installation Options

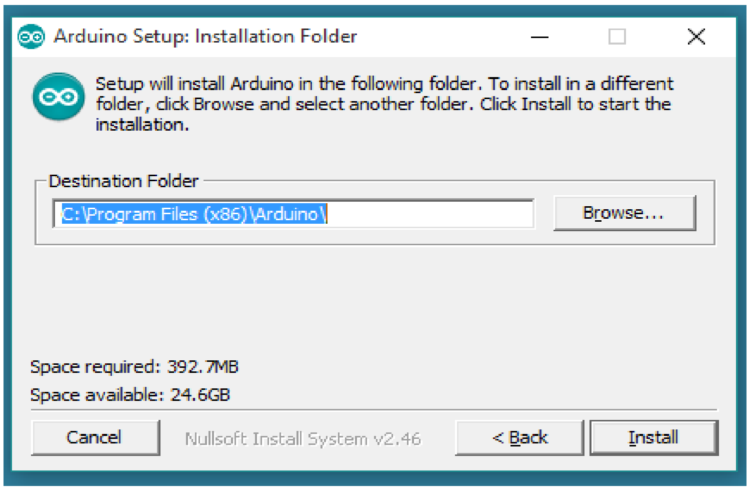
Check the components you want to install and uncheck the components you don't want to install. Click Next to continue.

Select components to install:

- [✓] Install Arduino software
- [✓] Install USB driver
- [✓] Create Start Menu shortcut
- [✓] Create Desktop shortcut
- [✓] Associate .ino files

Space required: 392.7MB

2. Choose the installation directory (keep the default one)


Arduino Setup: Installation Folder

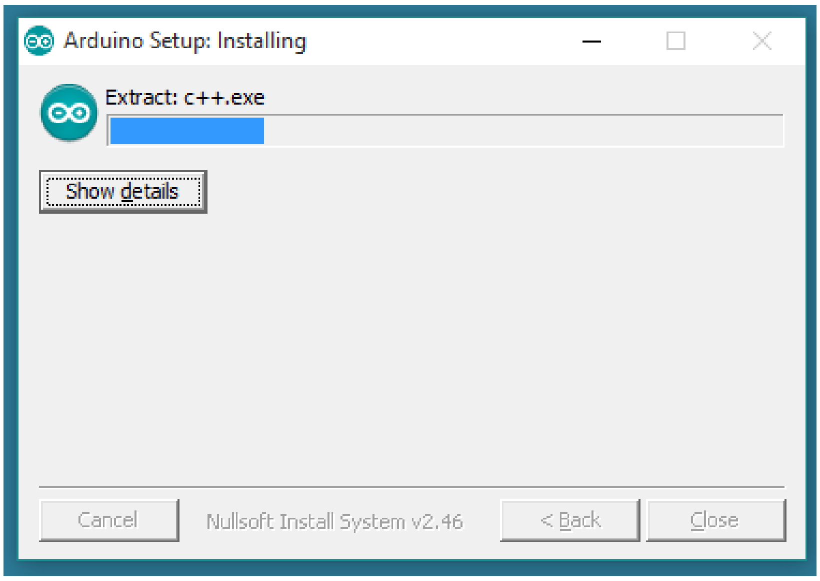
Setup will install Arduino in the following folder. To install in a different folder, click Browse and select another folder. Click Install to start the installation.

Destination Folder

C:\Program Files (x86)\Arduino\

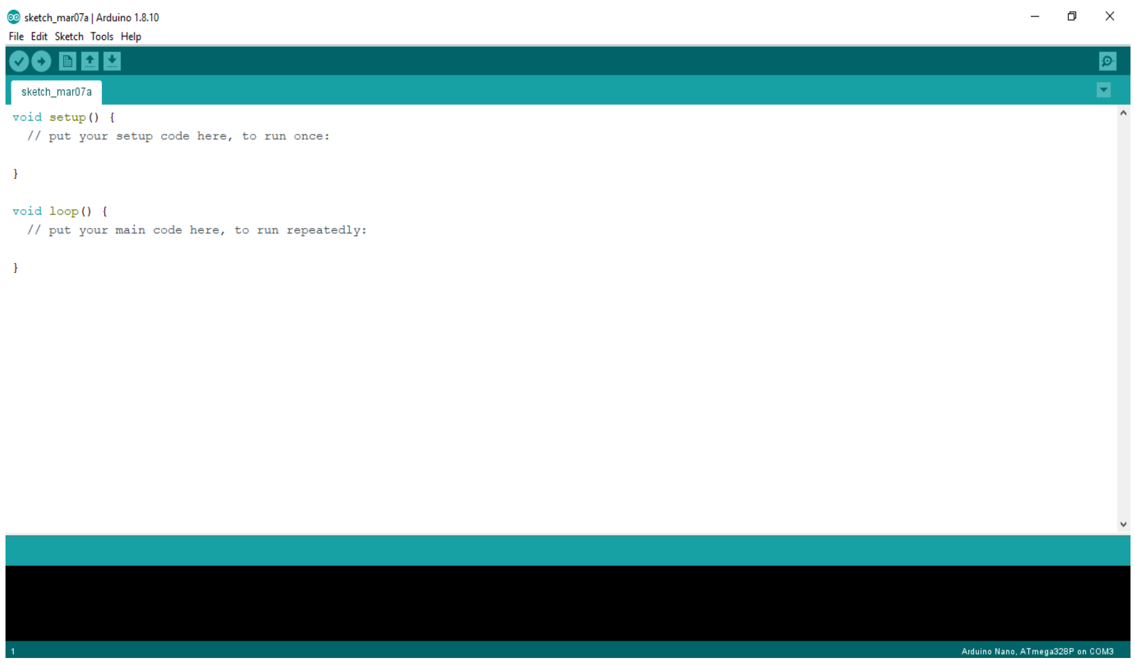
Space required: 392.7MB
Space available: 24.6GB

3. The process will extract and install all the required files to execute properly the Arduino Software (IDE)


Arduino Setup: Installing

Extract: c++.exe

4. After installation, an icon would be created on the desktop & when you click on that icon then following screen will appear on window

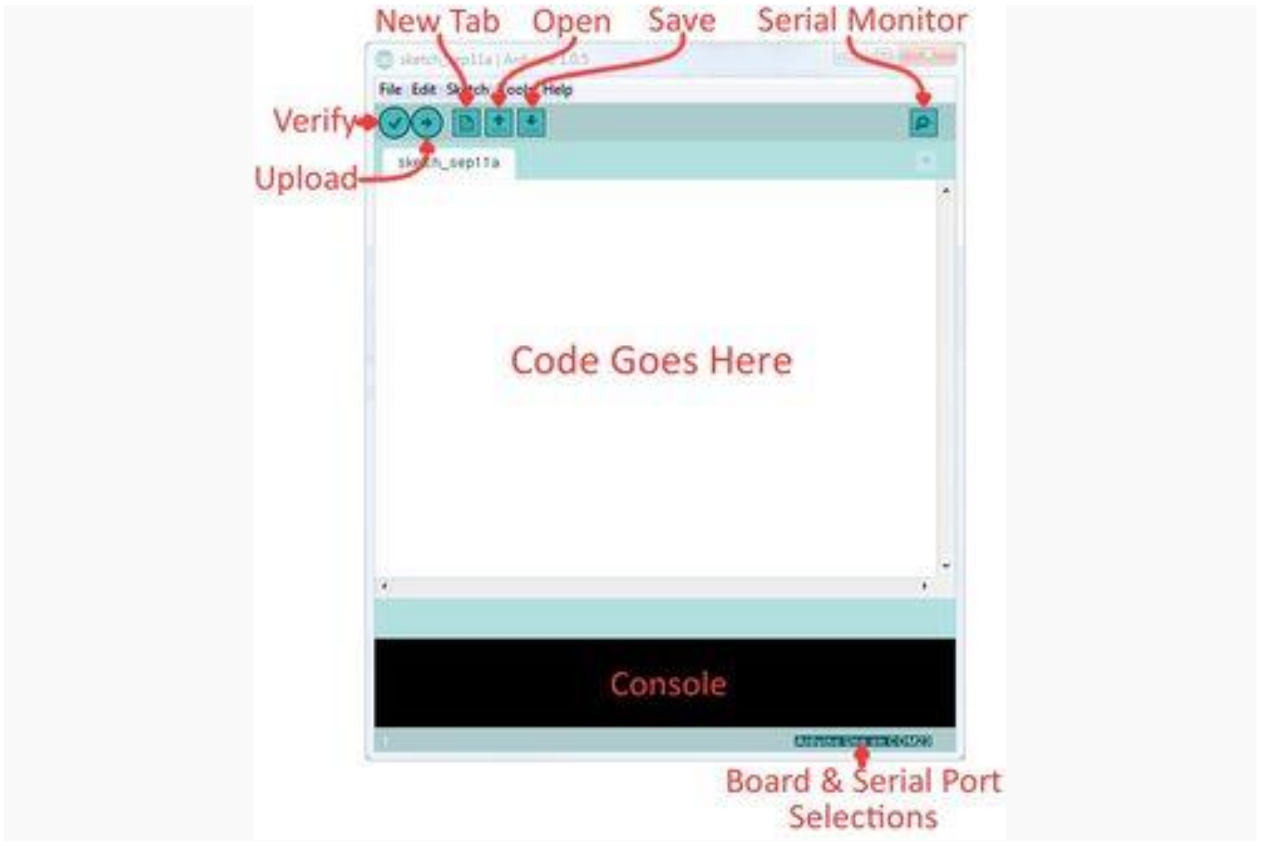


```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

```
Arduino Nano, ATmega328P on COM3
```

- **The Arduino IDE**

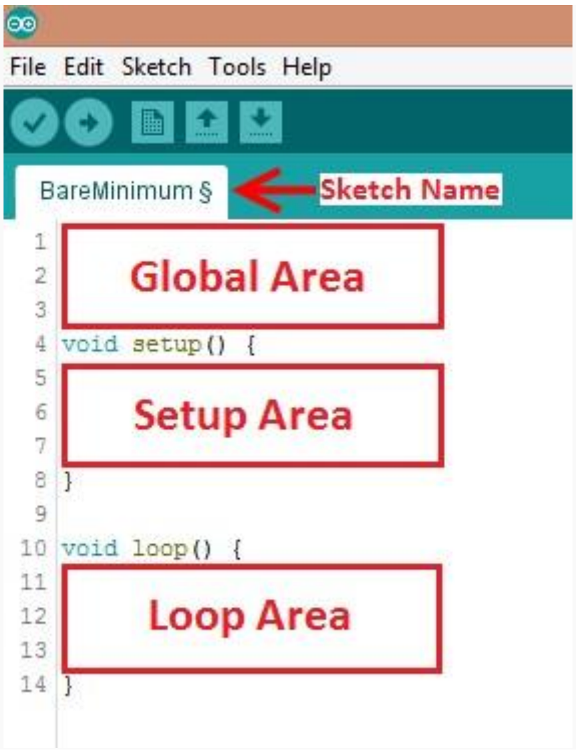
  The Arduino IDE is a program that can be used to edit, compile, and upload code to a supported microcontroller. The figure below shows a screenshot of the editor window:



Verify: checks code for errors and points to where the errors occurred after it finishes. Upload: verifies code and then uploads it to the Arduino board if there are no errors.

Console: shows any errors the software found in hardware.

Serial Monitor: a tool used to see how a program is running. It's like a multimeter for the program. Programs written in Arduino are called sketches. A basic sketch can be broken up into 3 different areas: global, setup, and loop; these areas are pictured below.



```
void setup() {

}

void loop() {

}
```

Global: constants and imported libraries go here.

Setup: activate the pins and sensors used. This code only runs once.

Loop: the code that runs continuously such as reading sensors and turning pins HIGH or LOW.

- **Arduino Programming**

The Arduino programming language is based on C/C++, but it is designed to be simpler and easier to learn. The most intuitive way to think about programming is like building with LEGO blocks: certain rules must be followed and different building blocks can be used to build bigger parts.

- Every line must either end with a semicolon ';' unless it's a conditional, loop, or function
- Comments start with a //
- Comments are text that the program ignores
- Used to label and explain code

- ***Datatypes***

  Datatypes are the different kinds of data values that can be used, manipulated, and stored using C++. The table below includes the most basic and widely used data types.

Datatype	**What it stores (examples)**	**Default value**	**Notes**
Boolean	A true value (1, TRUE, HIGH) or a false value (0, FALSE, LOW)	0, FALSE, LOW	-
Int	An integer number (-5, 15, 1047, etc.)	0	Can be positive or negative
Double	A decimal number (-0.5, 123.77, etc.)	0	Can be positive or negative
Char	A single character ('c', 'A', '5', '?', etc.)	Indeterminate	Must be enclosed in single quotes
String	A sequence of characters ("Hello World!", "10", "157+5", etc.)	Empty ("")	Must be enclosed in double quotes

- ***Operators***

  Operators perform operations on variables and constants. The results of these operations are usually stored in a variable. The table below includes common operators.

Operator	**What it does**	**Notes**
=	Assigns a value to a variable	
+	Adds two or more values	
-	Subtracts two or more values	
*	Multiplies two or more values	
/	Divides two or more values	
++	Increment by 1	Usually used in loops
--	Decrement by 1	Usually used in loops
==	Checks if two values are equal	Usually used in conditionals
!=	Checks if two values are not equal	Usually used in conditionals
> or <	Less than/Greater than comparison	Usually used in conditionals
<= or >=	Less than/greater than or equal to comparison	Usually used in conditionals

&& or ||	Boolean AND or Boolean OR is Used to cascade multiple Boolean operations	Usually used in conditionals

- ***Constants and Variables***

  Constants and variables hold data according to their data types. They need to be given a name so they can be referred to later. Constants hold data that will NOT change while a program is running. Constants usually contain PINs or sensor threshold values. Variables contain data that will change while a program is running. Variables usually contain sensor values and other values that need to have mathematical operations done on them. Below is an example of how to create different constants and variables.

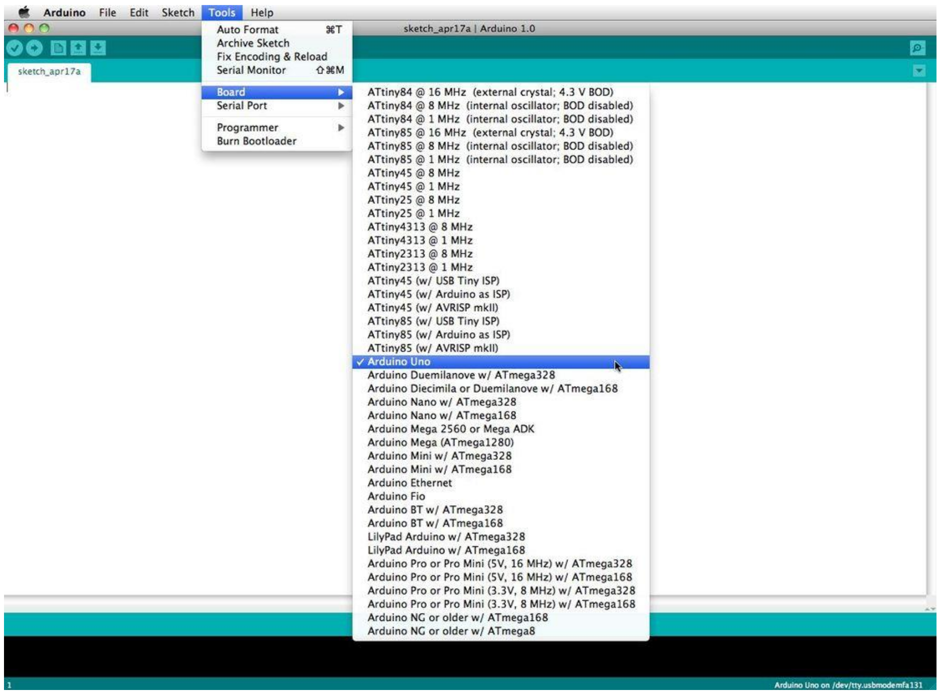
- **Arduino Interfacing with Computer**

1. Plug the USB cable into the ARDUINO & connect it to the PC

2. Before you proceed with programming in the Arduino, you must set the board type, and serial port.

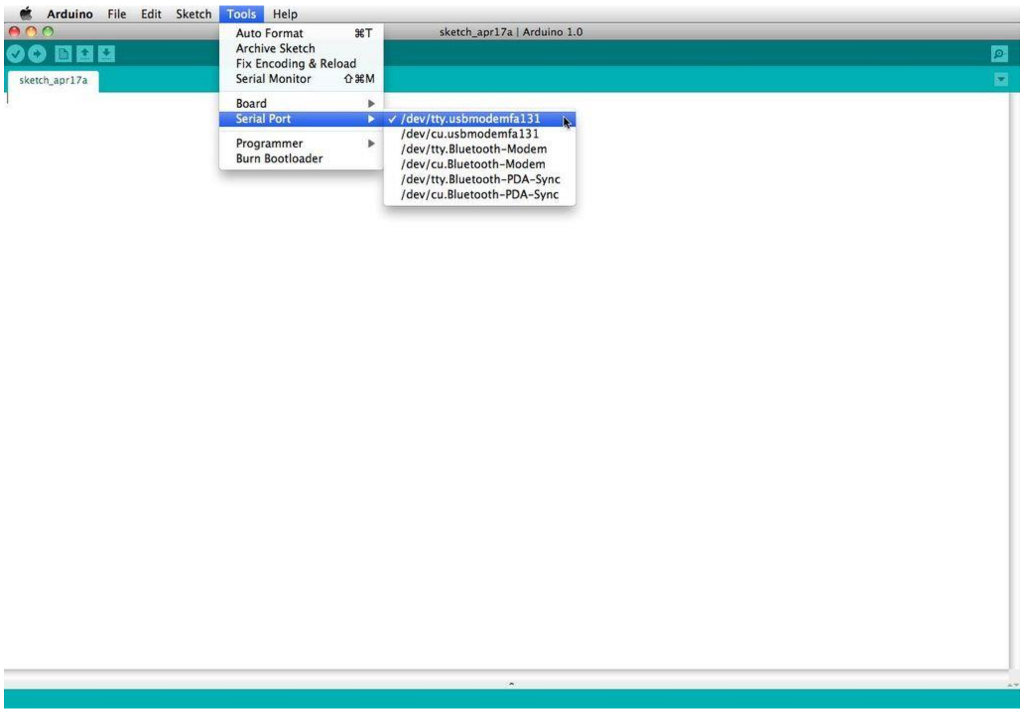
- To set up the board, follow the given path.

  Tools --> Boards. Select the version of the board that you are using as shown in the figure below.


Arduino File Edit Sketch Tools Help
sketch_apr17a | Arduino 1.0
Auto Format ⌘T
Archive Sketch
Fix Encoding & Reload
Serial Monitor ⇧⌘M
Board
Serial Port
Programmer
Burn Bootloader
sketch_apr17a
ATtiny84 @ 16 MHz (external crystal; 4.3 V BOD)
ATtiny84 @ 8 MHz (internal oscillator; BOD disabled)
ATtiny84 @ 1 MHz (internal oscillator; BOD disabled)
ATtiny85 @ 16 MHz (external crystal; 4.3 V BOD)
ATtiny85 @ 8 MHz (internal oscillator; BOD disabled)
ATtiny85 @ 1 MHz (internal oscillator; BOD disabled)
ATtiny45 @ 8 MHz
ATtiny45 @ 1 MHz
ATtiny25 @ 8 MHz
ATtiny25 @ 1 MHz
ATtiny4313 @ 8 MHz
ATtiny4313 @ 1 MHz
ATtiny2313 @ 8 MHz
ATtiny2313 @ 1 MHz
ATtiny45 (w/ USB Tiny ISP)
ATtiny45 (w/ Arduino as ISP)
ATtiny45 (w/ AVRISP mkII)
ATtiny85 (w/ USB Tiny ISP)
ATtiny85 (w/ Arduino as ISP)
ATtiny85 (w/ AVRISP mkII)
✓ Arduino Uno
Arduino Duemilanove w/ ATmega328
Arduino Diecimila or Duemilanove w/ ATmega168
Arduino Nano w/ ATmega328
Arduino Nano w/ ATmega168
Arduino Mega 2560 or Mega ADK
Arduino Mega (ATmega1280)
Arduino Mini w/ ATmega328
Arduino Mini w/ ATmega168
Arduino Ethernet
Arduino Fio
Arduino BT w/ ATmega328
Arduino BT w/ ATmega168
LilyPad Arduino w/ ATmega328
LilyPad Arduino w/ ATmega168
Arduino Pro or Pro Mini (5V, 16 MHz) w/ ATmega328
Arduino Pro or Pro Mini (5V, 16 MHz) w/ ATmega168
Arduino Pro or Pro Mini (3.3V, 8 MHz) w/ ATmega328
Arduino Pro or Pro Mini (3.3V, 8 MHz) w/ ATmega168
Arduino NG or older w/ ATmega168
Arduino NG or older w/ ATmega8
Arduino Uno on /dev/tty.usbmodemfa131

- To set the serial port, go to the following:

  Tools --> Serial Port



3. Write the code in the Blank Space, and press VERIFY ICON ✓ , which compiles the code & checks it for syntactical error.
4. Then press the UPLOAD ICON ➔ , which compiles your code and uploads it to the configured board. After Uploading, the user can arrange the components according to the circuit and then power the Arduino board and see the results.

Program to blink Arduino in-built LED

Code:

```
/*
Blink
Turns on an LED for one second, then off for one second, repeatedly.


*/

// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
       int led = 13;

       // the setup routine runs once when you press reset:
       void setup() {
       // initialize the digital pin as an output.
       pinMode(led, OUTPUT);
       }

       // the loop routine runs over and over again forever:
       void loop() {
       digitalWrite(led, HIGH); // turn the LED on (HIGH is the voltage level)
       delay(1000); // wait for a second
       digitalWrite(led, LOW); // turn the LED off by making the voltage LOW
       delay(1000); // wait for a second
       }
```

Explanation of Code:

The first actual line of code is:

```
int led = 13;
```

As the comment above explains, this is giving a name to the pin that the LED is attached to. This is 13 on most of the Arduinos, including the Uno. Next, we have the 'setup' function. Again, as the comment says, this is run when the reset button is pressed. It is also run whenever the board resets for any reason, such as power first being applied to it, or after a sketch has been uploaded.

```
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}
```

Every Arduino sketch must have a 'setup' function, and the part of it where you might want to add instructions of your own is between the { and the}.

In this case, there is just one command there, which, as the comment states tells the Arduino board that we are going to use the LED pin as an output pin.

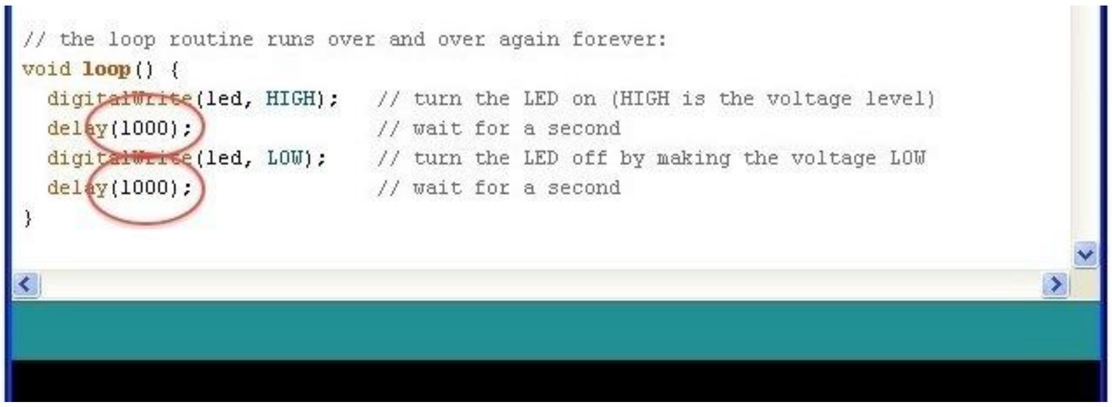
It is also mandatory for a sketch to have a 'loop' function. Unlike the 'setup' function that only runs once, after a reset, the 'loop' function will, after it has finished running its commands, immediately start again.

```
void loop() {
  digitalWrite(led, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);               // wait for a second
  digitalWrite(led, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);               // wait for a second
}
```

Inside the loop function, the commands first of all turn the LED pin on (HIGH), then 'delay' for 1000 milliseconds (1 second), then turn the LED pin off and pause for another second.

Blinking Faster

You are now going to make your LED blink faster. As you might have guessed, the key to this lies in changing the parameter in () for the 'delay' command.



```
// the loop routine runs over and over again forever:
void loop() {
  digitalWrite(led, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);               // wait for a second
  digitalWrite(led, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);               // wait for a second
}
```

These delay periods are in milliseconds, so if you want the LED to blink twice as fast, change the value from 1000 to 500. This would then pause for half a second each delay rather than a whole second. Upload the sketch again and you should see the LED start to flash more quickly.

Self-Evaluation:

1. How many analog and digital pins are available in the Arduino UNO?
2. What is the maximum clock frequency for Arduino UNO?
3. Which microcontroller is available in the Arduino UNO? List the feature of the same.
4. List the differences between Arduino Uno and Arduino Mega.

Conclusion

Exp. No. : 2 **Date: / /**
Title : To Utilize Digital Input-Output pins of Arduino board.

Objective:

To experiment with digital I/O pins of Arduino.

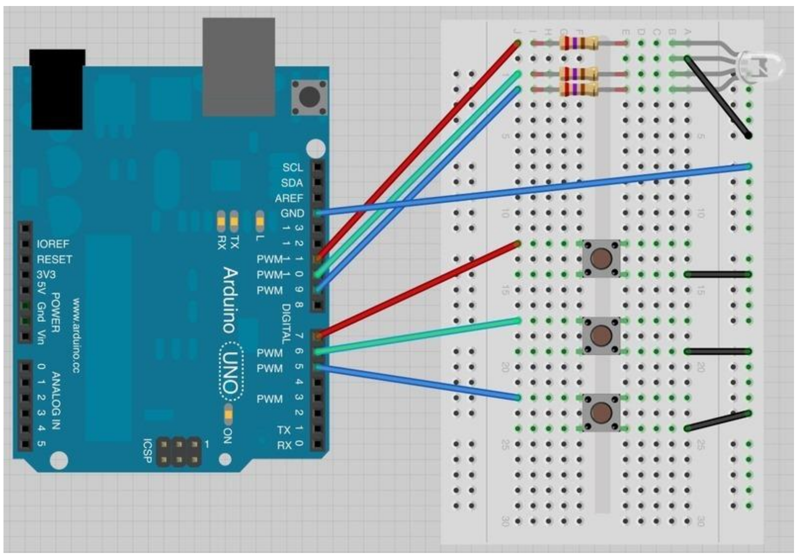
Learning Outcomes:
After completion of this experiment, students will be able to
- Interface push button or on/off switches
- Interface digital devices at Arduino board
- Develop a program for controllinsingle-bitit digital I/O devices

Pre Lab Exercise:
- Explain the following Arduino commands
    - pinMode
    - digitalWrite
    - digitalRead
    - delay

Components

- 10mm Common Cathode RGB LED -1
- 270 Ω Resistor (red, purple, brown stripes)- 3
- Tactile push switch -3
- Half-size Breadboard- 1
- Arduino Uno R3 -1
- Jumper wire pack-1

Circuit Diagram:



Sample code for single push button interfacing

```
// constants won't change. They're used here to set PINs:
const int buttonPin = 2;     // the number of the pushbutton pin
const int ledPin =  13;      // the number of the LED pin

// variables will change:
int buttonState = 0;         // variable for reading the pushbutton status

void setup() {
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // initialize the pushbutton pin as an input:
  pinMode(buttonPin, INPUT);
}

void loop() {
  // read the state of the pushbutton value:
  buttonState = digitalRead(buttonPin);

  // check if the pushbutton is pressed. If it is, the buttonState is HIGH:
  if (buttonState == HIGH)
 {
    // turn LED on:
    digitalWrite(ledPin, HIGH);
  }
else
{
    // turn LED off:
    digitalWrite(ledPin, LOW);
  }
}
```

Lab Exercise:

1. Develop a code for matrix keyboard that takes input from push button switches and controls the output LEDs connected in matrix form, accordingly.

Self-Evaluation:

1. Explore the sample codes given in Arduino examples under digital heads.

Conclusion:

Exp. No. : 3 **Date: / /**
Title : To Utilize Analog Input-Output pins of Arduino board.

Objective:

To experiment with analog I/O pins of Arduino.

Learning Outcomes:

After completion of this experiment, students will be able to

- Interface analog sensors with Arduino Board

Pre Lab Exercise:

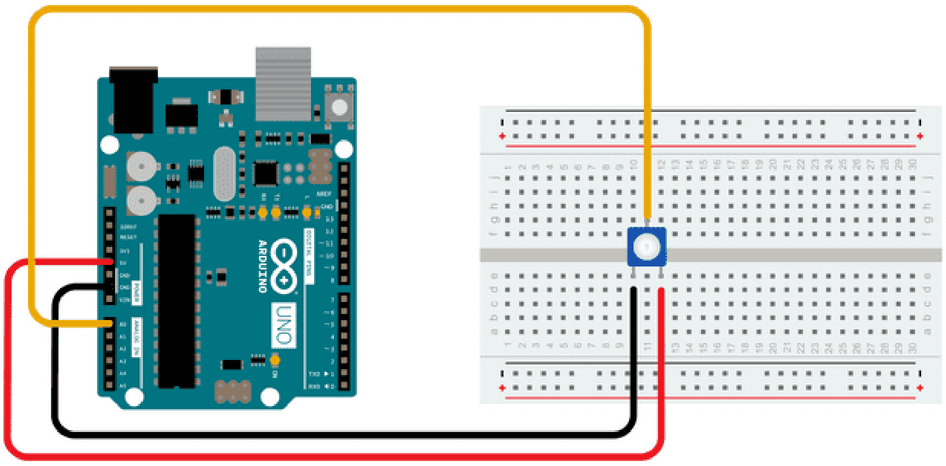
- Explain the following Arduino commands
  - analogWrite
  - analogRead
  - map
  - constraint

Components

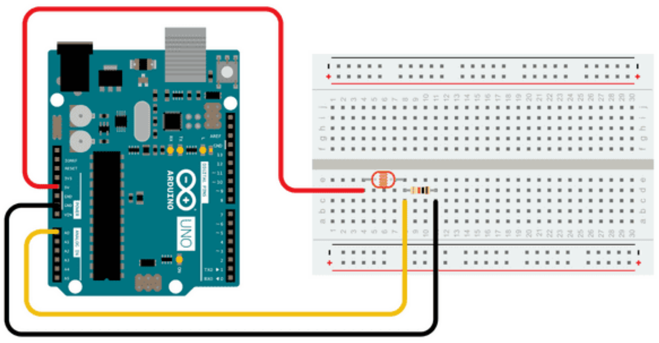
- Arduino Board
- Potentiometer or
- 10K ohm photoresistor and 10K ohm resistor
- built-in LED on pin 13 or
- 220-ohm resistor and red LED

Circuit Diagram:

- With a potentiometer



- With a photoresistor



Sample code

```
*/
int sensorPin = A0;    // select the input pin for the
potentiometer
int ledPin = 13;       // select the pin for the LED
int sensorValue = 0;  // variable to store the value coming from
the sensor
void setup() {
  // declare the ledPin as an OUTPUT:
  pinMode(ledPin, OUTPUT);
}
void loop() {
  // read the value from the sensor:
  sensorValue = analogRead(sensorPin);
  int x;
  x = 1500 * sensorValue;
  // turn the ledPin on
  digitalWrite(ledPin, HIGH);
  // stop the program for <sensorValue> milliseconds:
  delay(sensorValue);
  // turn the ledPin off:
  digitalWrite(ledPin, LOW);
  // stop the program for for <sensorValue> milliseconds:
```

```
  delay(sensorValue);
}
```

Lab Exercise:

1. Prepare a code that accepts analog input from the sensor and controls three output LEDs depending on the value received from the analog sensor.

Conclusion:

Exp. No. : 4 **Date: / /**
Title : To perform serial communication using Arduino

Objective:

To perform serial communication with a computer

Learning Outcomes:
After completion of this experiment, students will be able to

- Comprehend the concept of the baud rate for serial communication
- Interface serial port of the computer with Arduino
- Interface the devices, which work with the RS-232 protocol

Pre Lab Exercise:

- Explain the following Arduino commands
  - Serial.print
  - Serial.begin (baudrate)
  - Serial.println
  - Serial.read
  - Serial.write
  - Serial.available

Components

- Arduino Board
- Computer/Laptop

Theory:

The Arduino environment's built-in serial monitor to communicate with an Arduino board. Click the serial monitor button in the toolbar and select the same baud rate used in the call to begin(). Serial communication on pins TX/RX uses TTL logic levels (5V or 3.3V depending on the board). Don't connect these pins directly to an RS232 serial port; they operate at +/- 12V and can damage your Arduino board. To use these extra serial ports to communicate with your personal computer, you will need an additional USB-to-serial adaptor, as they are not connected to the Mega or UNO's USB-to-serial adaptor. To use them to communicate with an external TTL serial device, connect the TX pin to your device's RX pin, the RX to your device's TX pin, and the ground of your Mega or UNO to your device's ground.

Sample Code

```
/*
  Uses a for loop to print numbers in various formats.
*/
void setup() {
  Serial.begin(9600); // open the serial port at 9600 bps:
}
```

```
void loop() {
  // print labels
  Serial.print("NO FORMAT");  // prints a label
  Serial.print("\t");         // prints a tab

  Serial.print("DEC");
  Serial.print("\t");

  Serial.print("HEX");
  Serial.print("\t");

  Serial.print("OCT");
  Serial.print("\t");

  Serial.print("BIN");
  Serial.println();        // carriage return after the last
label

  for (int x = 0; x < 64; x++) { // only part of the ASCII
chart, change to suit
    // print it out in many formats:
    Serial.print(x);       // print as an ASCII-encoded decimal
- same as "DEC"
    Serial.print("\t\t");  // prints two tabs to accomodate the
label lenght

    Serial.print(x, DEC);  // print as an ASCII-encoded decimal
    Serial.print("\t");    // prints a tab

    Serial.print(x, HEX);  // print as an ASCII-encoded
hexadecimal
    Serial.print("\t");    // prints a tab

    Serial.print(x, OCT);  // print as an ASCII-encoded octal
    Serial.print("\t");    // prints a tab

    Serial.println(x, BIN);  // print as an ASCII-encoded binary
    // then adds the carriage return with "println"
    delay(200);            // delay 200 milliseconds
  }
  Serial.println();        // prints another carriage return
}
```

Lab Exercise:

1. Write a code that accepts analog input from the sensor, and displays the sensor value on the computer.

Conclusion:

Exp. No. : 5 **Date: / /**
Title : To interface LCD display

Objective:

To interface 16X2 LCD display (Parallel Communication) with Arduino

Learning Outcomes:
After completion of this experiment, students will be able to

- Interface with 16X2 LCD display with Arduino board
- Control various display for Arduino IDE

Pre Lab Exercise:

- Explain the following Arduino commands
    - lcd.begin()
    - lcd.print()
    - lcd.setCursor()
    - Blink
    - Cursor
    - Display
    - TextDirection
    - Scroll
    - Serial display
    - SetCursor
    - Autoscroll

Components:

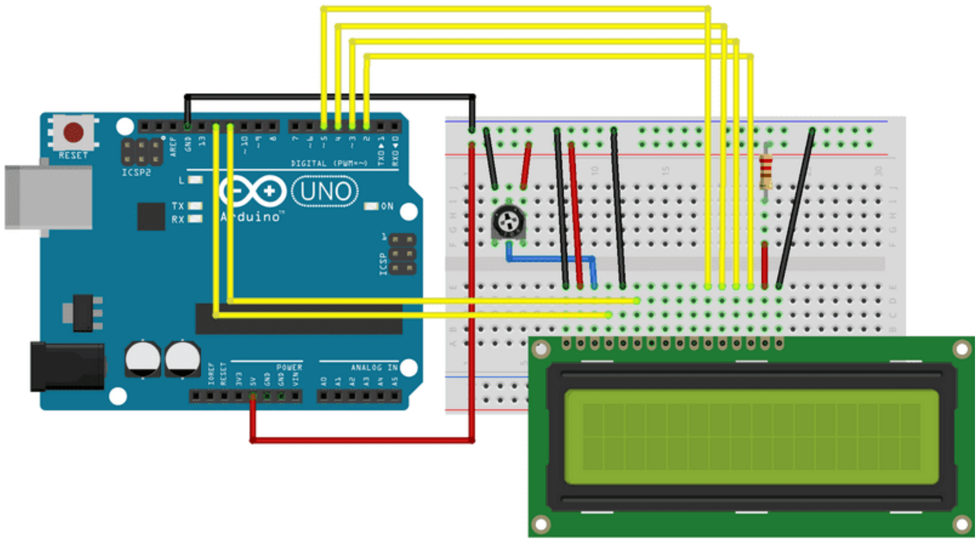
- Arduino Board
- LCD Screen (compatible with Hitachi HD44780 driver)
- Pin headers to solder to the LCD display pins
- 10k ohm potentiometer
- 220-ohm resistor
- Hook-up wires
- Breadboard

Circuit Diagram:

Before wiring the LCD screen to your Arduino board we suggest to solder a pin header strip to the 14 (or 16) pin count connector of the LCD screen, as you can see in the image above. To wire your LCD screen to your board, connect the following pins:

- LCD RS pin to digital pin 12
- LCD Enable pin to digital pin 11
- LCD D4 pin to digital pin 5
- LCD D5 pin to digital pin 4
- LCD D6 pin to digital pin 3
- LCD D7 pin to digital pin 2
- LCD R/W pin to GND
- LCD VSS pin to GND

- LCD VCC pin to 5V
- LCD LED+ to 5V through a 220-ohm resistor
- LCD LED- to GND
- Additionally, wire a 10k pot to +5V and GND, with its wiper (output) to LCD screens VO pin (pin3).



Sample code to display “hello world”

```
// include the library code:
#include <LiquidCrystal.h>

// initialize the library by associating any needed LCD
interface pin
// with the arduino pin number it is connected to

const int rs = 12, en = 11, d4 = 5, d5 = 4, d6 = 3, d7 = 2;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

void setup() {
  // set up the LCD's number of columns and rows:
  lcd.begin(16, 2);
  // Print a message to the LCD.
  lcd.print("hello, world!");
}

void loop() {
  // set the cursor to column 0, line 1
  // (note: line 1 is the second row, since counting begins with
0):
  lcd.setCursor(0, 1);
  // print the number of seconds since reset:
  lcd.print(millis() / 1000);
}
```

Lab Exercise:

1. Develop a code that displays scrolling message on LCD.

Conclusion:

Exp. No. : 6 **Date: / /**
Title : To establish communication using I2C protocol

Objective:

To interface 16X2 LCD display using I2C protocol

Learning Outcomes:

After completion of this experiment, students will be able to

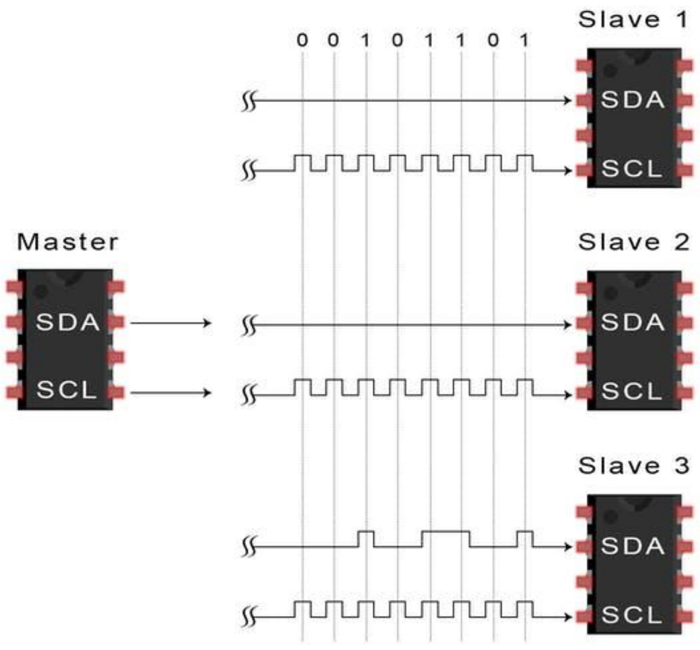
- Comprehend the I2C protocol
- Interface the devices working with I2C protocol to the Arduino Board

Pre Lab Exercise:

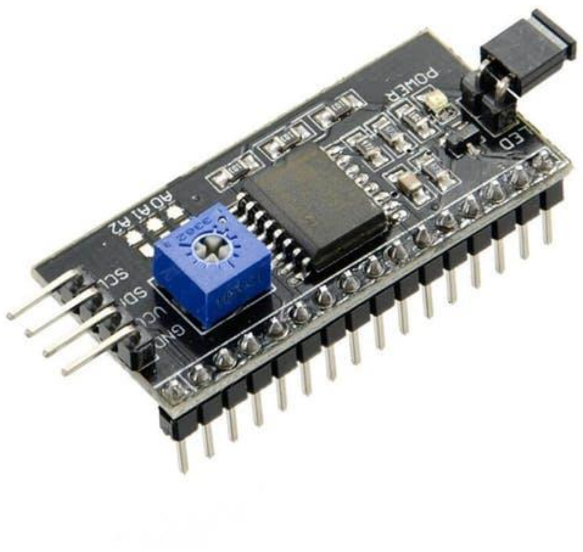
- Explain the following Arduino commands
  - Wire.begin()
  - Wire.requestFrom()
  - Wire.available()
  - Wire.read()
  - Wire.write()
  - Wire.beginTransmission()
  - Wire.endTransmission()
  - Wire.onReceive()

Theory:

The use of the I2C protocol requires less no of wires to connect the LCD with the Arduino board. It works as an intermediator between the LCD and Arduino. I2C stands for Inter-IC BUS. It is designed by Philips semiconductors. I2C is a synchronous, multi slave, multi-master packet-switched, single-ended serial bus. i.e. multiple chips can be connected to the same bus. I2C uses only two bidirectional open collector or open drain lines, Serial Data Line (SDA) and Serial Clock Line (SCL), pulled up with resistors. Typical voltages used are +5 V or +3.3 V, although systems with other voltages are permitted. Typical I2C communication is illustrated in the below figure.



I2C Communication



I2C Serial interface Adapter

It is also known as I2C Module. It has a total of 20 male pins. 16 pins are faced to the rear side and 4 pins are faced towards the front side. These 16 pins are connected to a 16x2 LCD and the 2 pins out of 4 pins are SDA and SCL. SDA is the serial data pin and SCL is the clock pin. The rest 2 pins for the power supply (Vcc and ground). There is a POT on the I2C Module. We can control the contrast of the LCD display by rotating this POT. And there is a jumper fixed on the module. When we remove the jumper, the backlight of the LCD display will go OFF.

Address of I2C LCD:
Every device which can be attached to MCU has an address. This address is used to communicate with that particular device which is labeled as A0, A1

and A2. Three solder pads on the I2C module. This is Address selectors. i.e., each solder pads have one upper potion and a one lower potion. If there is a connection between upper potion with lower connection it is called "Connected" otherwise it is called "Not connected". When A0, A1, A2 are in "Not Connected" condition (A0 = 0, A1 = 0, A2 = 0) the address would be 0x27. In default the A0, A1, A2 are in "Not connected" condition. And some time default address is 0x3F. There is no need to change the address of the I2C module when we use only one LCD. But when we use more than one LCD, need to change the address. Because two or more different device can't communicate with the same address. For more address see the table given below.

I2C Module Address Select

A0	**A1**	**A2**	**Hex Address**
0	0	0	0x27
1	0	0	0x26
0	1	0	0x25
1	1	0	0x24
0	0	1	0x23
1	0	1	0x22
0	1	1	0x21
1	1	1	0x20

0 = Not Connected

1= Connected

Steps to interface the LCD is as follows:

Step - 1 & Step -2 are related with the address of the LCD

In some cases, A0, A1, and A2 are in "Not connected" states, but the address is not 0x27. We can't communicate with this address. So we need to find the original address of that device. For that, we need to run the Arduino with the "I2C Scanner" code.

I2C Scanner Code

It is used to find the number of I2C devices and the address of I2C devices. First, add the header file by including "Wire.h" library. Then in the setup part, begin the "Wire" library by "Wire.begin()". Then begin the serial monitor as the baud rate of 9600 by "Serial.begin()". Next in the loop part, define two variables with the datatype "byte" named "error" and "address". Then define another variable with the "Integer (int)" datatype named "Devices". And set the initial value as 0. Next, start a for loop with a minimum value of 1 and a maximum of 127. "address" is used as theloop variable. Next input the

address to the wire with the function "Wire.beginTransmission()". The i2c_scanner uses the return value of the "Write.endTransmisstion()" to see if a device did acknowledge to the address. This return value store the value to the variable "error". The return value becomes 0, if a device acknowledges to the address. Otherwise, the return value becomes 4. Next, use a if. And the condition is "error==0". Then print the particular address to the serial monitor only if the address<16. Here we print the address in Hexadecimal. The printing instruction is "Serial.print(address, HEX)". And count the Device. The complete I2C Scanner code is given below
Sample code of 16X2 LCD interfacing using I2C protocol

```
#include <Wire.h>
void setup()
{
Wire.begin();
Serial.begin(9600);
Serial.println("\nI2C Scanner");
}
void loop()
{
byte error, address;
int Devices;
Serial.println("Scanning...");
Devices = 0;
for(address = 1; address < 127; address++ )
{
Wire.beginTransmission(address);
error = Wire.endTransmission();
if (error == 0)
{
Serial.print("I2C device found at address 0x");
if (address<16)
Serial.print("0");
Serial.print(address,HEX);
Serial.println("  !");
Devices++;
}
else if (error==4)
{
Serial.print("Unknown error at address 0x");
if (address<16)
Serial.print("0");
Serial.println(address,HEX);
}
}
if (Devices == 0)
Serial.println("No I2C devices found\n");
else
Serial.println("done\n");
delay(5000);
}
```

Lab Exercise:

1. Write a code to display your roll number on the first line and name on the second line of the LCD.

Conclusion:

Exp. No. : 7 **Date: / /**

Title : To control the speed of DC and servo motors

Objective:

To interface DC and the servo motor with Arduino

Learning Outcomes:

After completion of this experiment, students will be able to

- Interface DC motor with Arduino board
- Control speed of DC motor
- Interface servo motor with the Arduino board
- Control the angle of servo motor through Arduino board

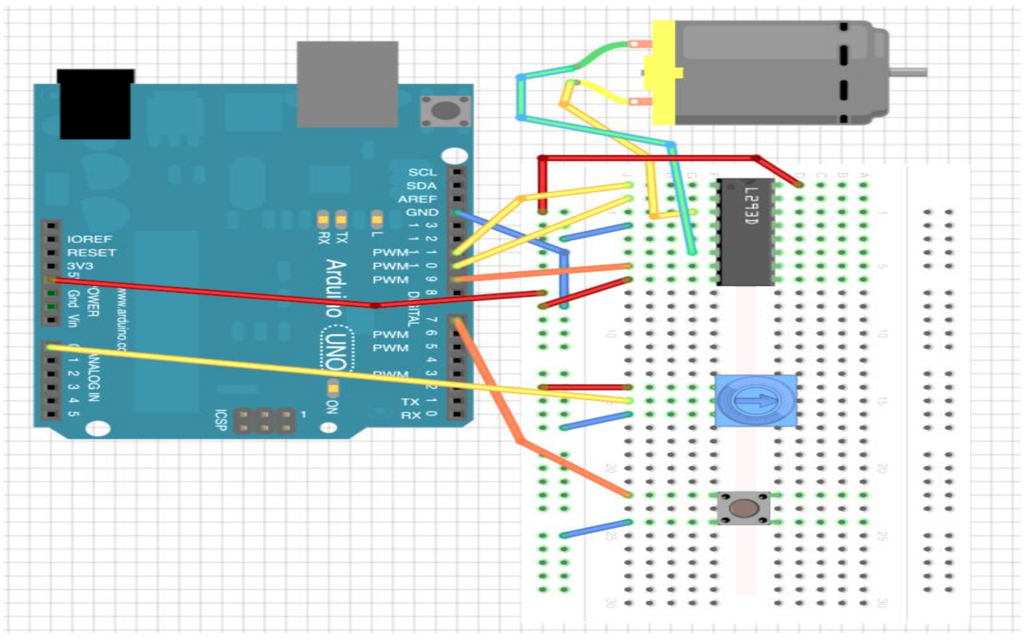
Pre Lab Exercise:

- Explain the following.
  - PWM
  - setMotor
  - Servo
  - .attach(pin)

Components:

- Small 6V DC Motor -1
- Servo motor
- L293D IC -1
- 10 kΩ variable resistor (pot)- 1
- Tactile push switch -1
- Half-size Breadboard- 1
- Arduino Uno R3 -1
- Jumper wire pack -1

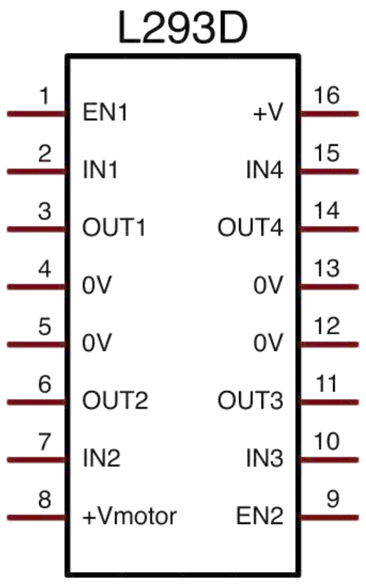
Circuit Diagram for DC Motor Interfacing:



Arduino manages the Enable, In1, and In2 pins to control the motor directly. When the project builds on the breadboard, IC should be inserted the right way around. The notch should be towards the top of the breadboard.

L293D

This is a very useful chip. It can actually control two motors independently. We are just using half the chip in this lesson; most of the pins on the right-hand side of the chip are for controlling a second motor.



A second motor would be attached between OUT3 and OUT4. You will also need three more control pins.

EN2 is connected to a PWM enabled output pin on the Arduino.

IN3 and IN4 are connected to digital outputs on the Arduino.

The L293D has two +V pins (8 and 16). The pin '+Vmotor' (8) provides the power for the motors, and +V (16) for the chip's logic. We have connected both of these to the Arduino 5V pin. However, if you were using a more powerful motor, or a higher voltage motor, you would provide the motor with a separate power supply using pin 8 connected to the positive power supply, and the ground of the second power supply is connected to the ground of the Arduino.

The three pins of L293D that we are interested in are Pin 1 (Enable), Pin 2 (In1), and Pin 7 (In2). These are attached to either 5V or GND using the purple, yellow, and orange jumper wires.

The motor should be turning in one direction, let's call that direction A. If you move Pin 1 (Enable) to GND the motor will stop, no matter what you do with the control pins In1 and In2. Enable turns everything on and off. This makes it useful for using a PWM output to control the motor speed. Reconnect Pin 1 to 5V so that the motor starts again.

Now try moving In1 (pin 2, yellow) from 5V to GND. In1 and In2 are both now connected to GND, so again the motor will stop.

Moving In2 from GND to 5V will cause the motor to turn in the opposite direction (direction B).

Finally, moving In1 back to 5V so that both In1 and In2 are at 5V will again cause the motor to stop.

The effect of the pins In1 and In2 on the motor are summarized in the table below:

In1	**In2**	**Motor**
GND	GND	Stopped
5V	GND	Turns in Direction A
GND	5V	Turns in Direction B
5V	5V	Stopped

Sample Code

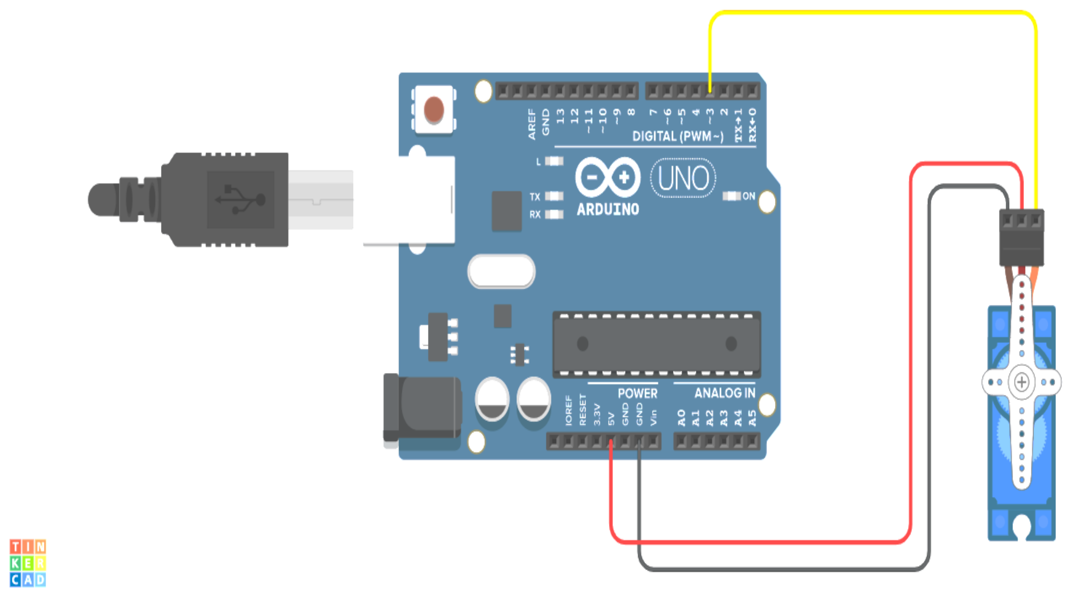
```
#include <LiquidCrystal.h>
LiquidCrystal lcd(13, 12, 11, 10, 9, 8);

//pin 1 in  LCD 1 >>>>>>>>>>>>> ground
//pin 2 in  LCD 2 >>>>>>>>>>>>> vcc
//pin 3 in  LCD 3 >>>>>>>>>>>>> ground
//pin 4 in  LCD 4 >>>>>>>>>>>>> 13
//pin 5 in  LCD 5>>>>>>>>>>>>> ground
// LCD connection
//pin 6 in  LCD 6 >>>>>>>>>>>>> 12
//pin 7 in  LCD 7 >>>>>>>>>>>>> NO
//pin 8 in  LCD 8 >>>>>>>>>>>>> NO
//pin 9 in  LCD 9 >>>>>>>>>>>>> NO
//pin 10 in LCD 10 >>>>>>>>>>>>> NO
//pin 11 in LCD 11  >>>>>>>>>>>>> 11
//pin 12 in LCD 12 >>>>>>>>>>>>> 10
//pin 13 in LCD 13 >>>>>>>>>>>>> 9
//pin 14 in LCD 14 >>>>>>>>>>>>> 8
//pin 15 in LCD 15 >>>>>>>>>>>>> vcc
//pin 16 in LCD 16 >>>>>>>>>>>  ground
//*************************************************************
******************************************
const int In1 = A1;     //pin  5 in lm298 >>>>>>>> A1 in
arduino
```

```
const int In2 = 7;      //pin  7 in lm298 >>>>>>>> 7 in
arduino            // L298 motor driver connection
const int En = 6;       //pin  6 in lm298 >>>>>>>> 6(PWM) in
arduino
                        // pin 2 in l298  >>>>>>>> Motor pin
                        // pin 3 in l298  >>>>>>>> Motor pin
                        // pin 9  in l298  >>>>>>>>> 5 volt
                        // pin 4  in l298  >>>>>>>>> Motor
voltage
                        //pin  8  in l298  >>>>>>>>> ground
//**************************************************************
**********************************************
const int Pot = A0;        //pin at middle in variable
resistance >>>>>>>> A0 in arduino          // variable
resistance  connection
                           // others pin to VCC & Ground
//**************************************************************
**************************************************
const int SW = 5;          //vcc (5 Volt )  switch for
direction with (resistance 10 k to ground )  >>>>>>>>>>>> 5 in
arduino     // switch connection

//**************************************************************
**********************************************************
const int red_led = 2;      // led with resistance 330 ohm to
>>>>>>>>>> 2 in arduino
const int yellow_led = 3;   // led with resistance 330 ohm to
>>>>>>>>>> 3 in arduino                 //LED connection
const int green_led = 4;   // led with resistance 330 ohm to
>>>>>>>>>> 4 in arduino
//**************************************************************
***********************************************************
volatile float pot_read = 0.0;
volatile int i = 0;
int flag = 1;
//**************************************************************
**********************************
void red() {
  digitalWrite(red_led, HIGH);                  // open RED LED
  digitalWrite(yellow_led, LOW);
  digitalWrite(green_led, LOW);
}
//**********************************************
void yellow() {
  digitalWrite(red_led, LOW);                          //
open yellow LED
  digitalWrite(yellow_led, HIGH);
  digitalWrite(green_led, LOW);
}
```

```
//*************************************************************
**********************
void green() {
  digitalWrite(red_led, LOW);          // open Green  LED
  digitalWrite(yellow_led, LOW);
  digitalWrite(green_led, HIGH);
}
//*************************************************************
*****************
void CW() {
  digitalWrite(In1, HIGH);
  digitalWrite(In2, LOW); // Make motor run in Clock wise
direction
  Serial.println(" Clock Wise ");
}
//*************************************************************
*********************
void CCW() {
  digitalWrite(In2, HIGH);
  digitalWrite(In1, LOW);
// Make motor run in counter  Clock wise direction
  Serial.println("counter Clock Wise ");
}
//*************************************************************
****************
void STP() {
  digitalWrite(In1, LOW);
  digitalWrite(In2, LOW);
// Make motor STOP
  Serial.println(" STOP ");
}
//*************************************************************
************************
//Make input and output pins
void setup() {
  // put your setup code here, to run once:
  lcd.begin(16, 2);
   lcd.print("Speed Control");
  delay(1000);
  pinMode(In1, OUTPUT);
  pinMode(In2, OUTPUT);
  pinMode(En, OUTPUT);
  pinMode(red_led, OUTPUT);
  pinMode(yellow_led, OUTPUT);
  pinMode(green_led , OUTPUT);
  pinMode(SW, INPUT);
  pinMode(Pot, INPUT);
  Serial.begin(9600);
}
```

```
//***************************************************************
*******************
void loop() {
  pot_read = analogRead(Pot);    // read variable resistance
  pot_read = pot_read / 4.0;
  pot_read = pot_read / 254.0;
  pot_read = pot_read * 100.0;

  delay(20);

  analogWrite(En, pot_read);
  lcd.setCursor(5, 1);

  lcd.print("PWM =");
  lcd.print(pot_read);
  lcd.print("%");


  Serial.print("PWM =");
  Serial.print(pot_read);


  Serial.println("%");

  //  Serial.print(" pot_read= ");
  //  Serial.println( pot_read );

  if (digitalRead(SW) == 1)
  {
    i++;
    switch (i)
    {
      case 1:
        CW();
        lcd.setCursor(0, 1);

        lcd.print(" CW");

        green();
        Serial.println(" Clock Wise ");
        break;
      case 2:
        CCW();
        lcd.setCursor(0, 1);

        lcd.print("CCW");
        yellow();
        Serial.println("counter Clock Wise ");
        break;
      case 3:
        STP();
        red();
        lcd.setCursor(0, 1);

        lcd.print("STP");
        Serial.println(" STOP ");
        i = 0;
        break;
    }
    while (digitalRead(SW) == 1);
  }

}
```

Circuit Diagram Servo Motor Interfacing:



Sample Code:

In this project, a library named "Servo.h" is required, which is an inbuilt library with Arduino IDE.

```
#include <Servo.h>
Servo Myservo;
int pos;
void setup()
{
Myservo.attach(3);
}

void loop()
{
 for(pos=0;pos<=180;pos++){
Myservo.write(pos);
delay(15);
}
  delay(1000);
  for(pos=180;pos>=0;pos--){
Myservo.write(pos);
delay(15);
}
  delay(1000);
```

Conclusion

Exp. No. : 8 **Date: / /**
Title : To handle external interrupt with Arduino

Objective:

To perform external interrupt service routine

Learning Outcomes:
After completion of this experiment, students will be able to

- Perform time and counter operation with Arduino board
- Utilize interrupt service routing in Arduino environment

Pre Lab Exercise:

- Comprehend the concept of interrupt.
- Explain the command "attachInterrupt"

Components:

- Arduino Board (In this tutorial Arduino NANO is used)
- Push button - 2
- LED - 1
- Resistor (10K) - 2
- LCD (16x2) - 1
- Bread Board
- Connecting Wires

Theory

There are two types of interrupt: External Interrupt and Pin Change Interrupt.

External Interrupt:
These interrupt are interpreted by hardware and are very fast. These interrupts can be set to trigger on the event of RISING or FALLING or LOW levels.

Arduino Board	**External Interrupt pins:**
UNO , NANO	2,3
Mega	2,3,18,19,20,21

Pin Change Interrupts:
Arduinos can have more interrupt pins enabled by using **pin change** interrupts. In ATmega168/328 based Arduino boards any pins or all the 20 signal pins can be used as interrupt pins. They can also be triggered using RISING or FALLING edges.

Using Interrupts in Arduino
In order to use interrupts in Arduino the following concepts are need to be understood.

Interrupt Service Routine (ISR)
Interrupt Service Routine or an Interrupt handler is an event that has small set of instructions in it. When an external interrupt occurs, the processor first executes these code that is present in ISR and returns back to state where it left the normal execution.
ISR has following **syntax in Arduino:**

```
attachInterrupt(digitalPinToInterrupt(pin), ISR, mode);
```

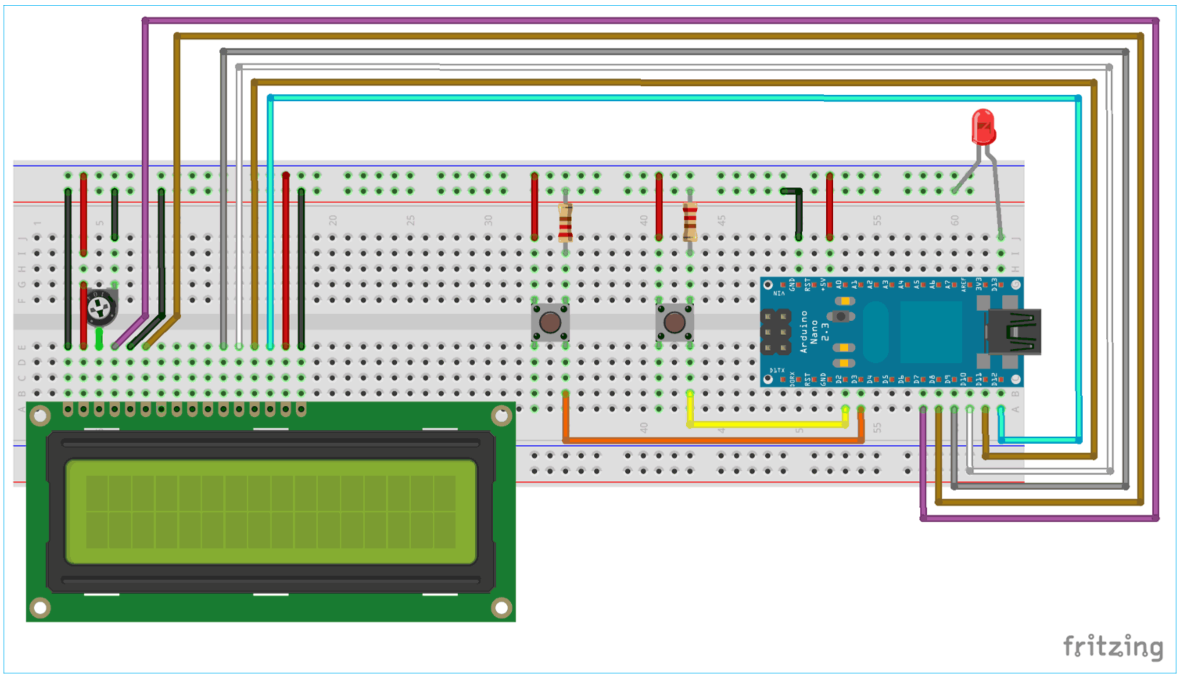
digitalPinToInterrupt(pin): In Arduino Uno, NANO the pins used for interrupt are 2,3 & in mega 2,3,18,19,20,21. Specify the input pin that is used for external interrupt here.

ISR: It is a function that is called when an external interrupt is done.

Mode: Type of transition to trigger on, e.g. falling, rising, etc.
RISING: To trigger an interrupt when the pin transits from LOW to HIGH.
FALLING: To trigger an interrupt when the pin transits from HIGH to LOW.
CHANGE: To trigger an interrupt when the pin transits from LOW to HIGH or HIGH to LOW (i.e., when the pin state changes).

Some Conditions while using Interrupt
Interrupt Service Routine function (ISR) must be as short as possible.
Delay () function doesn't work inside ISR and should be avoided.
In this **Arduino Interrupt tutorial**, a number is incremented from 0 and two push buttons are used to trigger Interrupt, each one is connected to D2 & D3. A LED is used to indicate the Interrupt. **If one push button is pressed** the led goes ON and display shows interrupt2 and goes off, and **when another push button is pressed** the led goes OFF and the display shows interrupt1 and goes off.

Circuit Diagram:



Circuit Connection between Arduino Nano and 16x2 LCD display:

LCD	Arduino Nano
VSS	GND
VDD	+5V
V0	To Potentiometer Centre PIN For Controlling Contrast of the LCD
RS	D7
RW	GND
E	D8
D4	D9
D5	D10
D6	D11
D7	D12
A	+5V
K	GND

Two push buttons are connected to Arduino Nano at pin D2 & D3. They are used for using two external interrupts, one for turning LED ON and another for turning OFF a LED. Each push button has a **pull down resistor** of 10k connected to ground. So when push button is pressed it is logic HIGH (1) and when not pressed it is logic LOW (0). A Pull down resistor is compulsory otherwise there will be floating values at the input pin D2 & D3.

A **LED** is also used to indicate that a Interrupt has been triggered or a button has been pressed.

Sample Code

```
//Interrupts using Arduino
//Circuit Digest

#include<LiquidCrystal.h>                               // Including
lcd display library
LiquidCrystal lcd (7,8,9,10,11,12);                     // Define LCD
display pins RS,E,D4,D5,D6,D7
```

```
volatile int output = LOW;
int i = 0;

void
setup()

{
  lcd.begin(16,2);                              //  setting LCD
as 16x2 type
  lcd.setCursor(0,0);
  lcd.print("CIRCUIT
DIGEST");
  lcd.setCursor(0,1);
  lcd.print("ArduinoInterrupt");
  delay(3000);

  lcd.clear();

  pinMode(13,OUTPUT);

  attachInterrupt(digitalPinToInterrupt(2),buttonPressed1,RISING
);  //  function for creating external interrupts at pin2 on
Rising (LOW to HIGH)
  attachInterrupt(digitalPinToInterrupt(3),buttonPressed2,RISING
);  //  function for creating external interrupts at pin3 on
Rising (LOW to HIGH)

}

void
loop()
{
   lcd.clear();

   lcd.print("COUNTER:");

   lcd.print(i);

   ++i;

   delay(1000);
   digitalWrite(13,output);     //Turns LED ON or OFF depending
upon output value
}

void buttonPressed1()           //ISR function excutes when push
button at pinD2 is pressed
{
   output = LOW;                //Change Output value to
LOW
   lcd.setCursor(0,1);
   lcd.print("Interrupt 1");
}

void buttonPressed2()           //ISR function excutes when push
button at pinD3 is pressed
```

```
{
   output = HIGH;   //Change Output value to
HIGH
   lcd.setCursor(0,1);
   lcd.print("Interrupt2");
}
```

Self-Evaluation:

1. Prepare a code for a visitor counter

Conclusion

Exp. No. : **9** **Date: / /**

Title : **To control peripherals using ESP8266 and BLYNK application**

Objective:

To control various peripherals connected with ESP8266 through wifi using BLYNK application.

Learning Outcomes:

After completion of this experiment, students will be able to

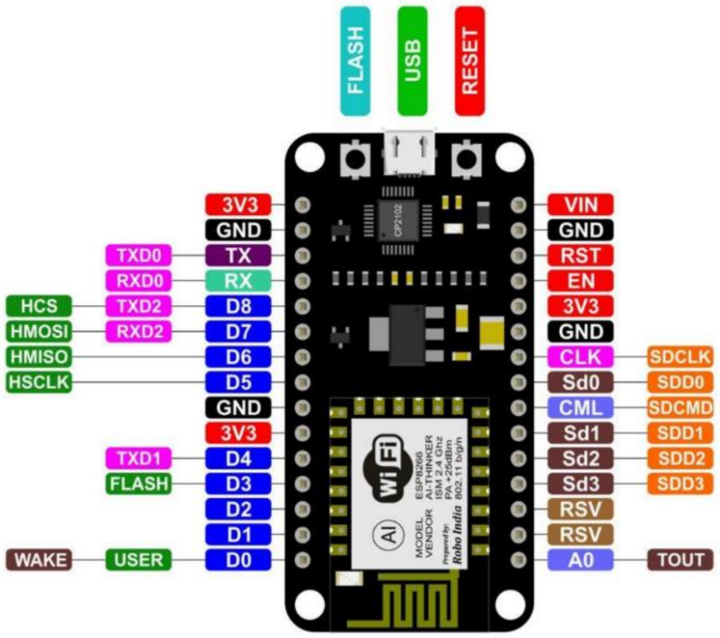
- Find out the MAC address of ESP8266 board
- Utilize ESP8266 for various applications
- Communicate with BLYNK and ESP8266 over WiFi

Components:

- ESP8266 Board
- LED - 1
- Mobile

Theory

The Node MCU is an open-source firmware and development kit that helps to prototype IoT products with Arduino IDE or in a few Lau script lines. It includes firmware that runs on the ESP8266 Wi-Fi SoC. And the hardware that is based on the ESP-12 module.



Connecting NodeMCU with Arduino IDE

1. Open up the Arduino IDE.
2. Go to File -> Preferences -> Additional Boards Manager

URLs: http://arduino.esp8266.com/stable/package_esp8266com_index.json ->click OK

3. Close the IDE and open it up again.
4. Go to Tools -> Board (where you'd select your version of Arduino) -> Boards Manager, find the ESP8266 and click Install. You now should be able to use the ESP8266 as an Arduino. Simply select the NODEMCU 1.0 as your board with Port and you should be ready to code. Now, with ESP8266 board installed to Arduino IDE, we can program NodeMCU using Arduino IDE directly.

Getting MAC address:
The MAC (or Machine Access Control) address is a unique value associated with a network adapter. So, MAC addresses are hardware addresses that uniquely identify a network adapter. It may also be known as an Ethernet Hardware Address (EHA), Hardware Address or Physical Address. It is must to know the MAC address of NodeMCU before starting any communication. The following code needs to be uploaded to get the MAC address.

```
#include <ESP8266WiFi.h>

void setup()
{
Serial.begin(115200);
delay(500);
Serial.println();
Serial.print("MAC: ");
Serial.println(WiFi.macAddress());
}
void loop(){
}
```

After uploading this code open serial monitor and one can find MAC address of the respective NodeMCU.

Blynk Application:

Blynk is an Internet of Things platform, which makes controlling hardware remotely and visualizing its data very easy. One can create own interfaces using the free Blynk App. Every WiFi, Bluetooth/BLE, Ethernet and Serial device is able to connect to the Blynk cloud or a locally running server.

Install Blynk libraries:

1) Install the latest release of the Blynk libraries on GitHub
2) Unpack it
3) Move the libraries to *C:/User//Documents/Arduino/libraries*

Install Blynk App:
1) Download the App for iOS or Android
2) Create an account or sign in
3) Click 'Create New Project'
4) Choose the device and connection type (NodeMCU, WiFi)

5) Receive and note down your 'Auth Token'
6) Open the 'Widget Box' ('+')
7) Add a button
8) Define the output pin the LED is connected to (anode Dx, cathode GND)

The screenshot of the process is shown in the fig.1

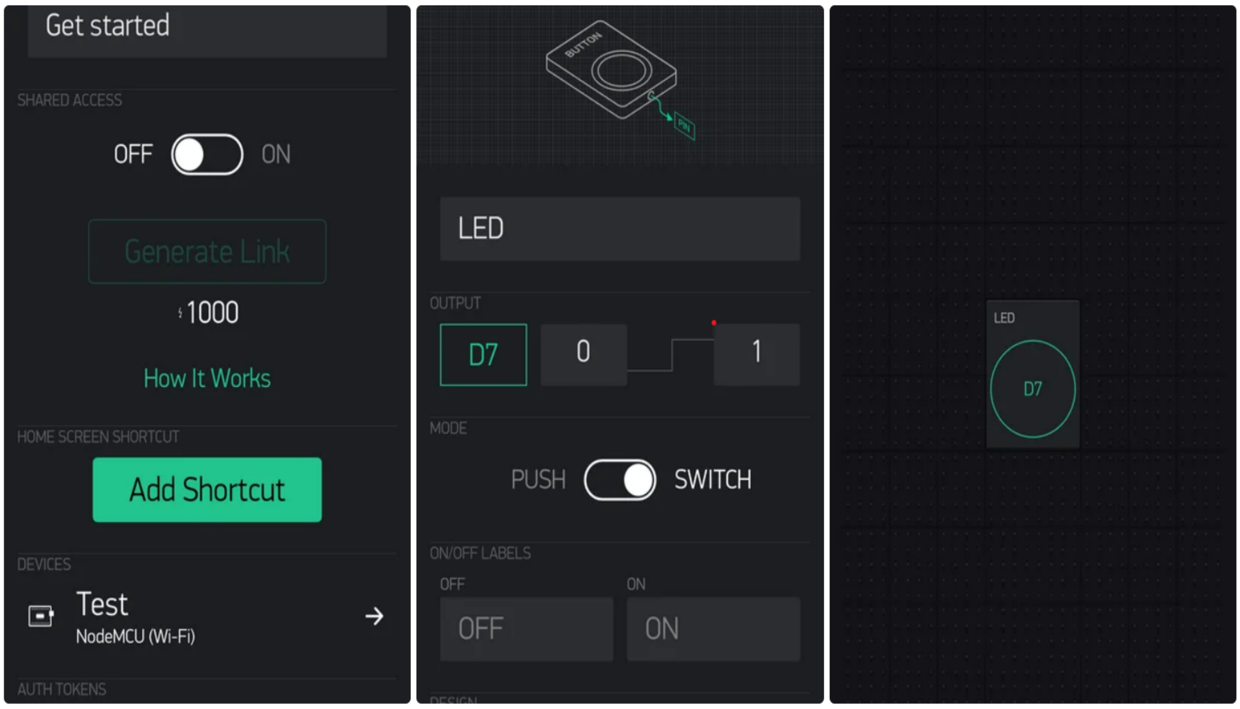
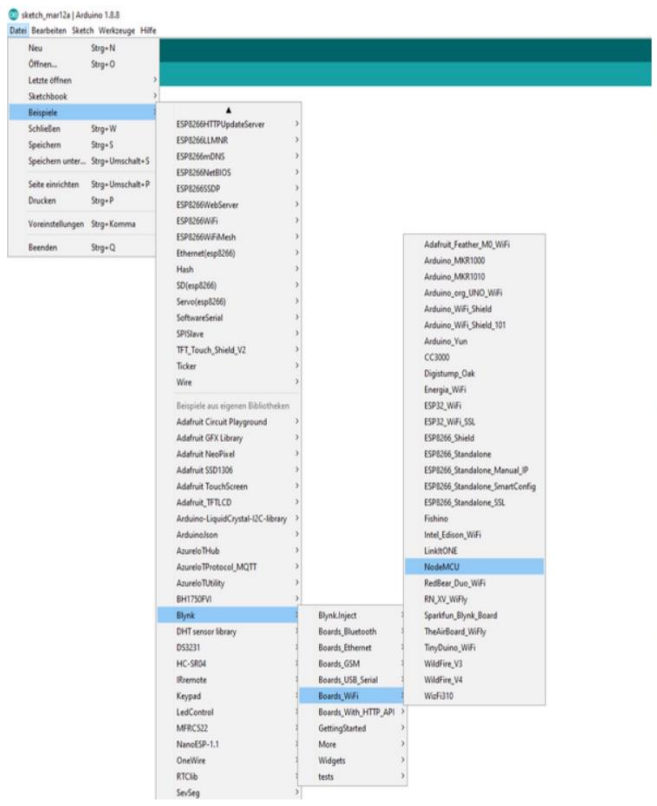


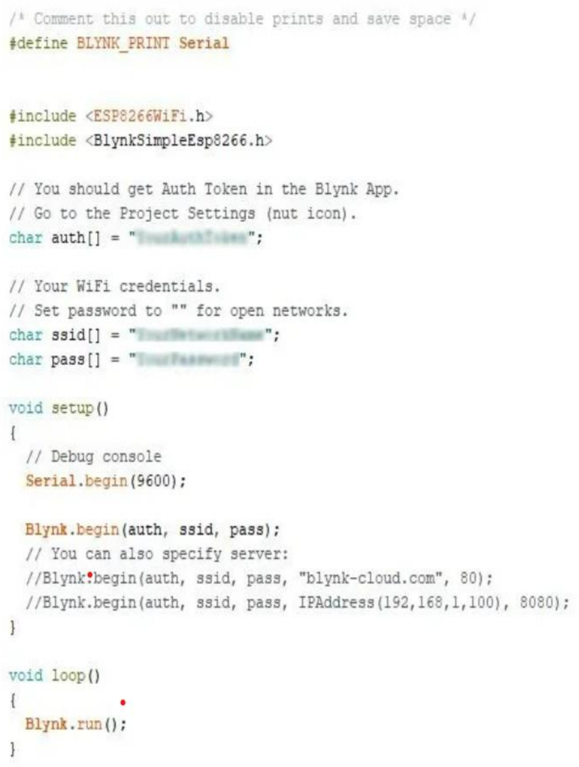
Fig.1 : Blynk application project screenshot

The step for the client-side code for remote controlling an LED is as follows:

1) Open the Arduino IDE
2) Goto Examples > Blynk > Boards_WiFi and select your dev board
3) Enter your 'Auth Token' (char auth[])
4) Enter your WiFi credentials (char ssid[], char pass[])
5) Compile and Upload
6) Open the Serial Monitor and check whether connecting was successful.

Fig. 2 illustrate the given steps with the code to blink LED.





```
/* Comment this out to disable prints and save space */
#define BLYNK_PRINT Serial


#include <ESP8266WiFi.h>
#include <BlynkSimpleEsp8266.h>

// You should get Auth Token in the Blynk App.
// Go to the Project Settings (nut icon).
char auth[] = "[redacted]";

// Your WiFi credentials.
// Set password to "" for open networks.
char ssid[] = "[redacted]";
char pass[] = "[redacted]";

void setup()
{
  // Debug console
  Serial.begin(9600);

  Blynk.begin(auth, ssid, pass);
  // You can also specify server:
  //Blynk.begin(auth, ssid, pass, "blynk-cloud.com", 80);
  //Blynk.begin(auth, ssid, pass, IPAddress(192,168,1,100), 8080);
}

void loop()
{
  Blynk.run();
}
```

Fig.2 Procedure to write a code for blinking a LED

Conclusion

Exp. No. : 10 **Date: / /**
Title : To monitor sensor data on Thingspeak using ESP8266

Objective:

To monitor sensor data on Thingspeak using ESP8266

Learning Outcomes:
After completion of this experiment, students will be able to

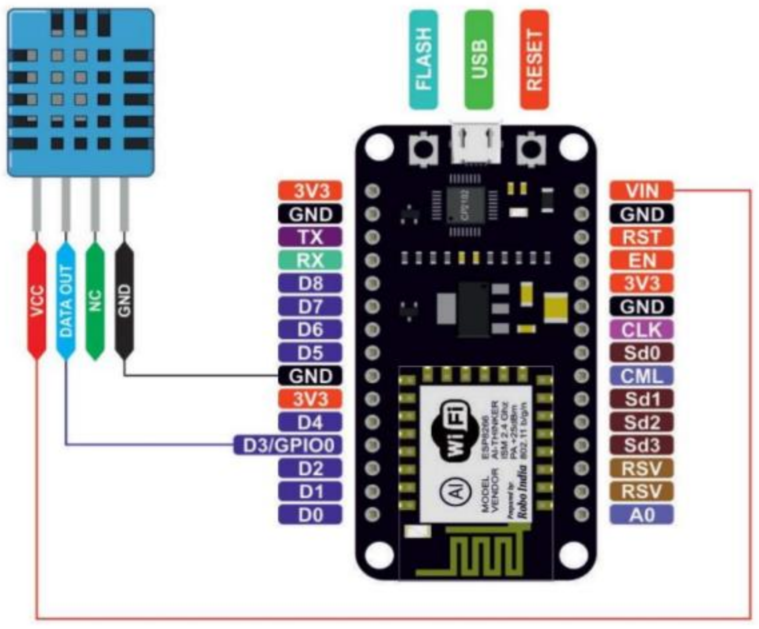
- Utilize Thingspeak for sensor data monitoring

Components:

- Node MCU
- Temperature and humidity sensor
- Jumper cables

Theory

Circuit Diagram



After making the above connections, Follow the given steps to configure the channel on Thinkspeak.com and get API keys
Getting API Key
1) Go to https://thingspeak.com/ and create an account if you do not have one.
2) Login to your account.
3) Create a new channel by clicking on the button.
4) Enter basic details of the channel then Scroll down and save the channel.

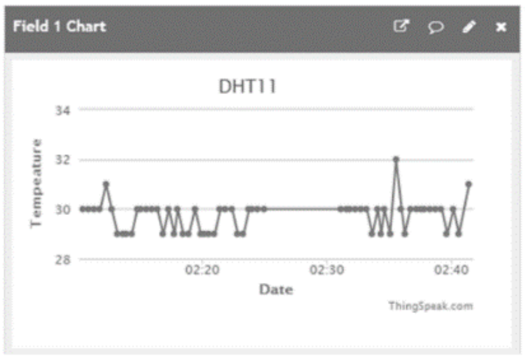
5) Channel Id is the identity of your channel. Note down this. Then go to API keys. copy and paste this key to a separate notepad file will need it later.

Sample Code:

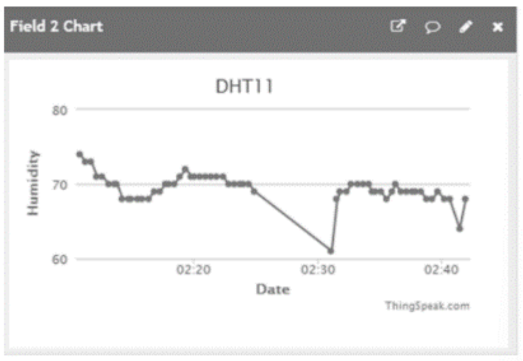
```
// Hardware: NodeMCU,DHT11

#include <DHT.h> // Including library for dht

#include <ESP8266WiFi.h>

String apiKey = "Your API of thingsspeak"; // Enter your Write
API key from ThingSpeak

const char *ssid = "Your wifi Network name"; // replace with
your wifi ssid and wpa2 key

const char *pass = "Network password";

const char* server = "api.thingspeak.com";

#define DHTPIN 0 //pin where the dht11 is connected

DHT dht(DHTPIN, DHT11);

WiFiClient client;

void setup()

{

Serial.begin(115200);

delay(10);

dht.begin();

Serial.println("Connecting to ");

Serial.println(ssid);

WiFi.begin(ssid, pass);

while (WiFi.status() != WL_CONNECTED)

{

delay(500);

Serial.print(".");

}

Serial.println("");

Serial.println("WiFi connected");

}
```

```
void loop()

{

float h = dht.readHumidity();

float t = dht.readTemperature();

if (isnan(h) || isnan(t))

{

Serial.println("Failed to read from DHT sensor!");

return;

}

if

(client.connect(server,80)) // "184.106.153.149" or

api.thingspeak.com

{

String postStr = apiKey;

postStr +="&field1=";

postStr += String(t);

postStr +="&field2=";

postStr += String(h);

postStr += "\r\n\r\n";

client.print("POST /update HTTP/1.1\n");

client.print("Host: api.thingspeak.com\n");

client.print("Connection: close\n");

client.print("X-THINGSPEAKAPIKEY: "+apiKey+"\n");

client.print("Content-Type: application/x- www-form-
urlencoded\n");

client.print("Content-Length: ");

client.print(postStr.length());

client.print("\n\n");

client.print(postStr);

Serial.print("Temperature: ");

Serial.print(t);
Serial.print(" degrees Celcius, Humidity:");

Serial.print(h);

Serial.println("%. Send to Thingspeak.");

}

client.stop();

Serial.println("Waiting...");

// thingspeak needs minimum 15 sec delay between updates,

delay(10000);

}
```

After successful upgradation on Thingspeak, one can easily monitor the sensor data as shown in the below figure. On every successful data upload, success message is displayed on serial monitor.



Temperature



Humidity

Conclusion

Appendix : Adding an external library to Arduino environment

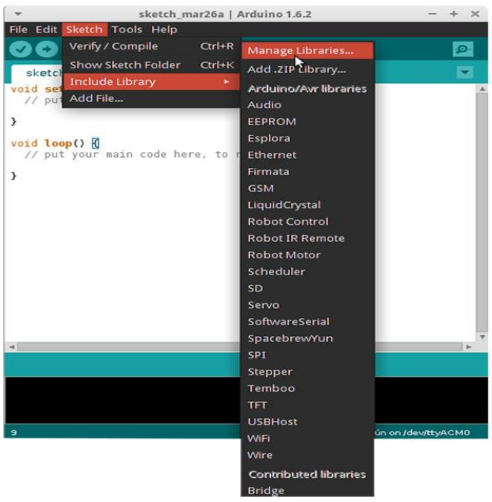
Lot many devices can be connected with the Arduino board. It is not possible that one has knowledge of interfacing of all such devices. The library is a collection of code that makes the programming easy for interfacing the external device with the Arduino. For example, the built-in LiquidCrystal library makes it easy to interface LCD displays and run various commands related to LCD interfacing. There are hundreds of additional libraries available for Arduino.

Libraries can be installed or added in the Arduino IDE in following ways:

- Using the Library Manager
- Importing a .zip Library
- Manual installation

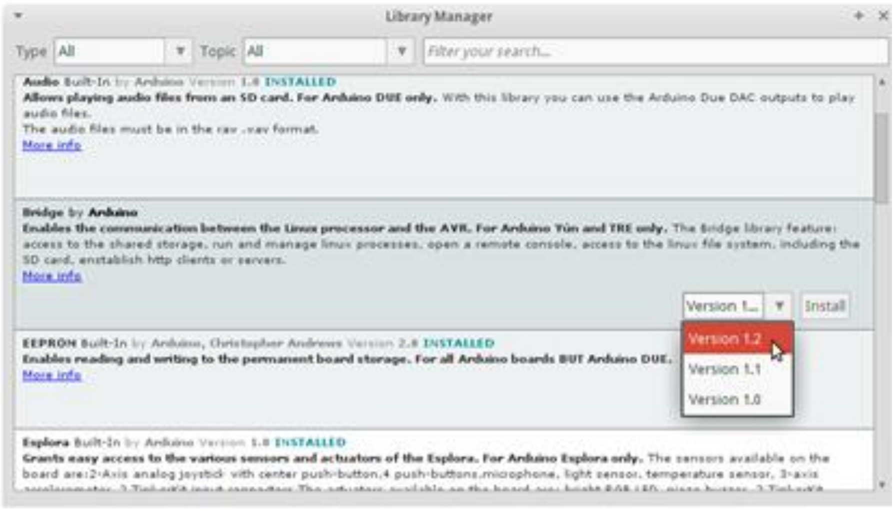
Using the library manager

To install a new library into Arduino IDE one can use the Library Manager. Open the IDE and click to the "Sketch" menu and then *Include Library > Manage Libraries.*



```
void set
  // pu
}

void loop() {
  // put your main code here, to r
}
```

Then the Library Manager will open and you will find a list of libraries that are already installed or ready for installation. In this example we will install the Bridge library. Scroll the list to find it, click on it, then select the version of the library you want to install. Sometimes only one version of the library is available. If the version selection menu does not appear, don't worry: it is normal.


Library Manager

Type All ▼ Topic All ▼ Filter your search...

Audio Built-In by **Arduino** Version **1.0** INSTALLED
Allows playing audio files from an SD card. For Arduino DUE only. With this library you can use the Arduino Due DAC outputs to play audio files.
The audio files must be in the raw .wav format.
More info

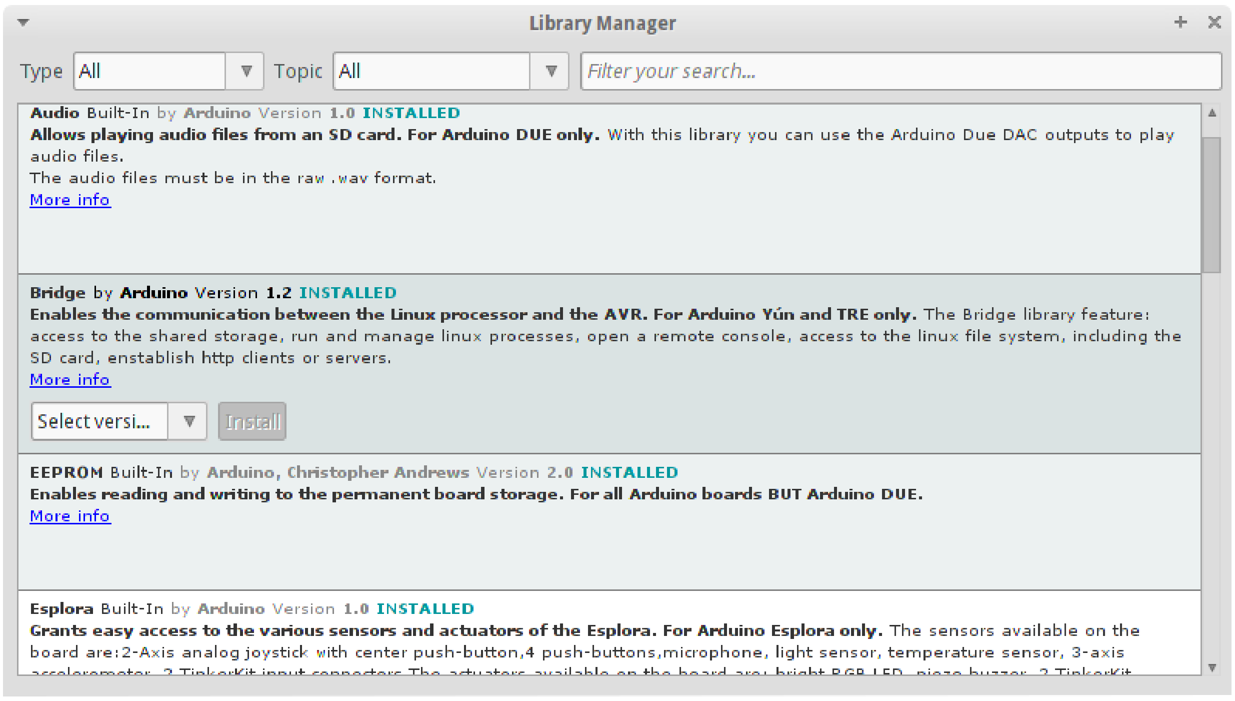
Bridge by **Arduino**
Enables the communication between the Linux processor and the AVR. For Arduino Yún and TRE only. The Bridge library feature: access to the shared storage, run and manage linux processes, open a remote console, access to the linux file system, including the SD card, enstablish http clients or servers.
More info

Version 1... ▼ Install
Version 1.2
Version 1.1
Version 1.0

EEPROM Built-In by **Arduino, Christopher Andrews** Version **2.0** INSTALLED
Enables reading and writing to the permanent board storage. For all Arduino boards BUT Arduino DUE.
More info

Esplora Built-In by **Arduino** Version **1.0** INSTALLED
Grants easy access to the various sensors and actuators of the Esplora. For Arduino Esplora only. The sensors available on the board are:2-Axis analog joystick with center push-button,4 push-buttons,microphone, light sensor, temperature sensor, 3-axis

Finally click on install and wait for the IDE to install the new library. Downloading may take time depending on your connection speed. Once it has finished, an *Installed* tag should appear next to the Bridge library. You can close the library manager.


Library Manager

Type All ▼ Topic All ▼ Filter your search...

Audio Built-In by **Arduino** Version **1.0** INSTALLED
Allows playing audio files from an SD card. For Arduino DUE only. With this library you can use the Arduino Due DAC outputs to play audio files.
The audio files must be in the raw .wav format.
More info

Bridge by **Arduino** Version **1.2** INSTALLED
Enables the communication between the Linux processor and the AVR. For Arduino Yún and TRE only. The Bridge library feature: access to the shared storage, run and manage linux processes, open a remote console, access to the linux file system, including the SD card, enstablish http clients or servers.
More info

Select versi... ▼ Install

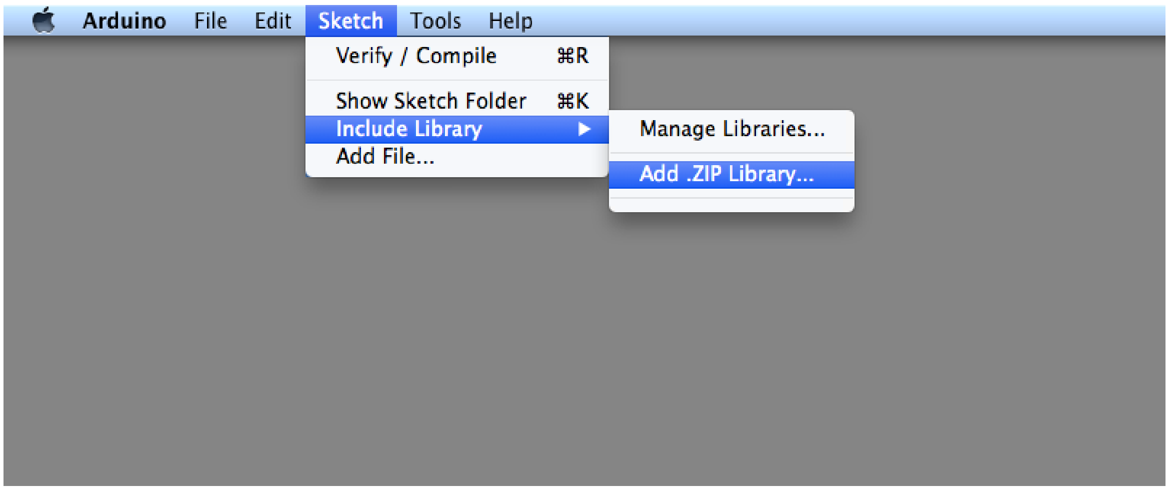
EEPROM Built-In by **Arduino, Christopher Andrews** Version **2.0** INSTALLED
Enables reading and writing to the permanent board storage. For all Arduino boards BUT Arduino DUE.
More info

Esplora Built-In by **Arduino** Version **1.0** INSTALLED
Grants easy access to the various sensors and actuators of the Esplora. For Arduino Esplora only. The sensors available on the board are:2-Axis analog joystick with center push-button,4 push-buttons,microphone, light sensor, temperature sensor, 3-axis

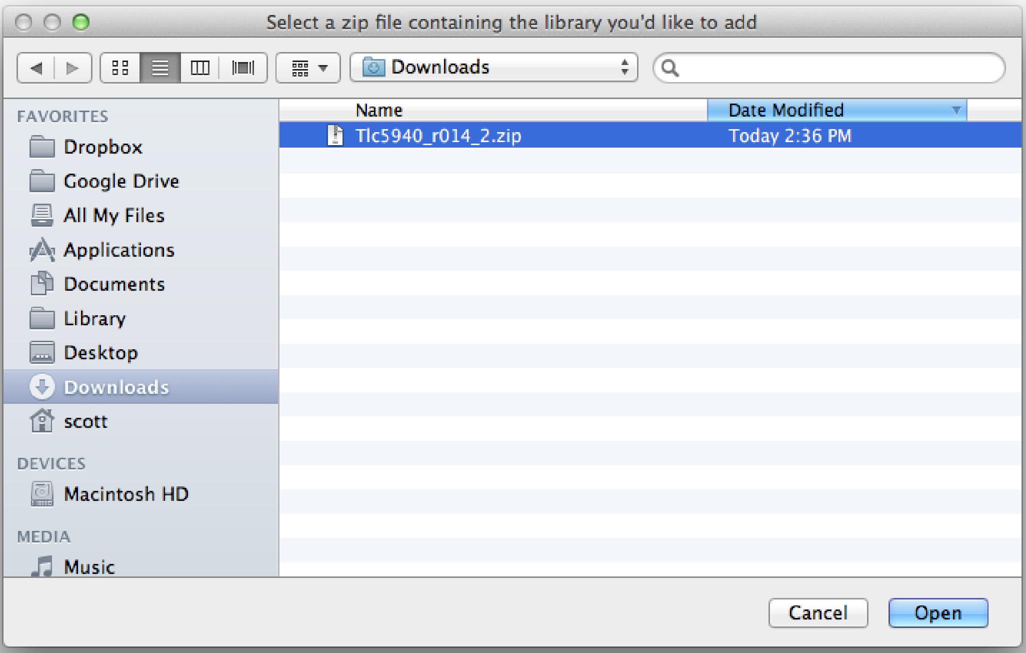
You can now find the new library available in the *Sketch* > *Include Library* menu.

Importing a .zip Library

The third pary libraries are sometimes available as a ZIP file or folder. The name of the folder is the name of the library. Inside the folder will be a .cpp file, a .h file and often a keywords.txt file, examples folder, and other files required by the library. In the Arduino IDE, navigate to *Sketch* > *Include Library* > *Add .ZIP Library*. At the top of the drop down list, select the option to "Add .ZIP Library''.



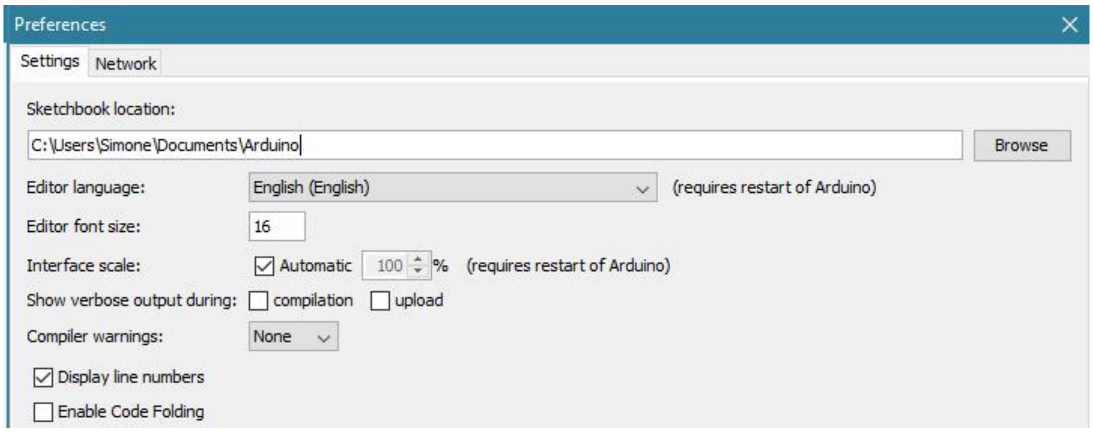
You will be prompted to select the library you would like to add. Navigate to the .zip file's location and open it.



Return to the *Sketch > Include Library menu.* menu. You should now see the library at the bottom of the drop-down menu. It is ready to be used in your sketch. The zip file will have been expanded in the *libraries* folder in your Arduino sketches directory.

Manual installation

When you want to add a library manually, you need to download it as a ZIP file, expand it and put in the proper directory. The ZIP file contains all you need, including usage examples if the author has provided them. The library manager is designed to install this ZIP file automatically as explained in the former chapter, but there are cases where you may want to perform the installation process manually and put the library in the *libraries* folder of your sketchbook by yourself. You can find or change the location of your sketchbook folder at *File > Preferences > Sketchbook* location.


Preferences

Settings | Network

Sketchbook location: C:\Users\Simone\Documents\Arduino

Editor language: English (English) (requires restart of Arduino)

Editor font size: 16

Interface scale: [✓ Automatic] 100 % (requires restart of Arduino)

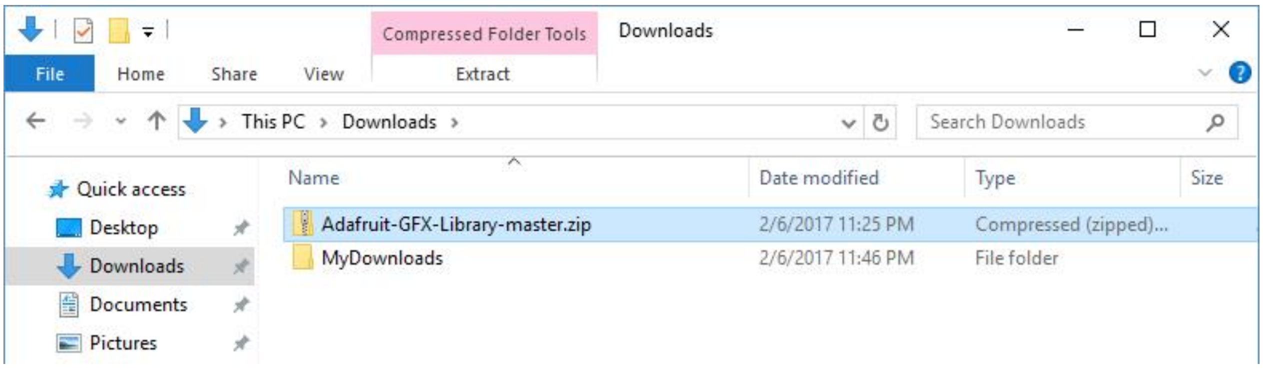
Show verbose output during: ☐ compilation ☐ upload

Compiler warnings: None

[✓ Display line numbers]

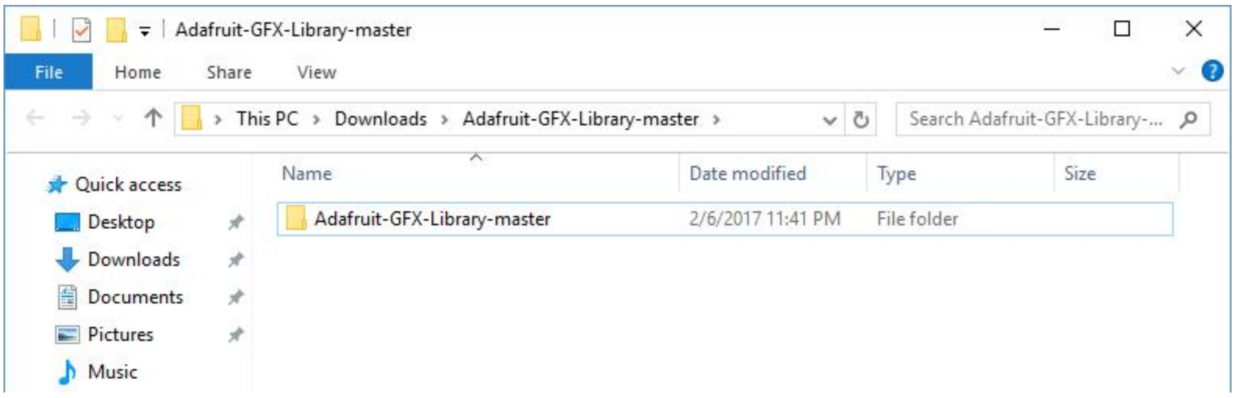
☐ Enable Code Folding

Go to the directory where you have downloaded the ZIP file of the library



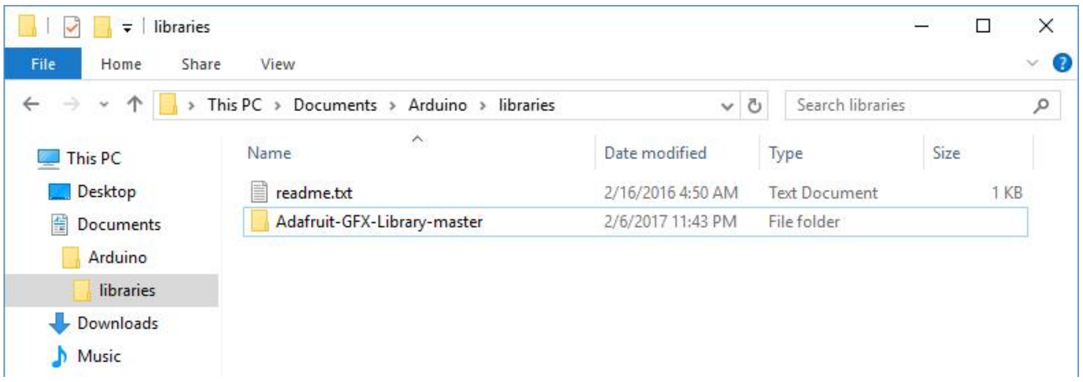
Name	Date modified	Type	Size
Adafruit-GFX-Library-master.zip	2/6/2017 11:25 PM	Compressed (zipped)...	
MyDownloads	2/6/2017 11:46 PM	File folder	

Extract the ZIP file with all its folder structure in a temporary folder, then select the main folder, that should have the library name


Adafruit-GFX-Library-master
File
Home
Share
View
This PC > Downloads > Adafruit-GFX-Library-master >
Search Adafruit-GFX-Library-...
Quick access
Desktop
Downloads
Documents
Pictures
Music

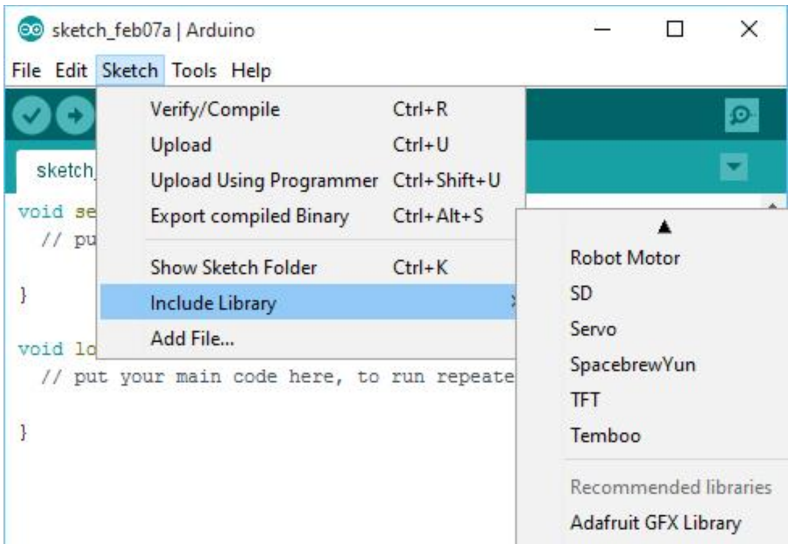
Name	Date modified	Type	Size
Adafruit-GFX-Library-master	2/6/2017 11:41 PM	File folder	

Copy it in the “libraries” folder inside your sketchbook.



Name	Date modified	Type	Size
readme.txt	2/16/2016 4:50 AM	Text Document	1 KB
Adafruit-GFX-Library-master	2/6/2017 11:43 PM	File folder	

Start the Arduino Software (IDE), go to *Sketch* > *Include Library*. Verify that the library you just added is available in the list.



```
void se
  // pu

}

void lo
  // put your main code here, to run repeate

}
```

Arduino libraries are managed in three different places: inside the IDE installation folder, inside the core folder and in the libraries folder inside your sketchbook. The way libraries are chosen during compilation is designed to allow the update of libraries present in the distribution. This means that placing a library in the "libraries" folder in your sketchbook overrides the other libraries versions.

The same happens for the libraries present in additional cores installations. It is also important to note that the version of the library you put in your sketchbook may be lower than the one in the distribution or core folders, nevertheless it will be the one used during compilation. When you select a specific core for your board, the libraries present in the core's folder are used instead of the same libraries present in the IDE distribution folder.