جامعة
الأقصر

Computer science Department

Faculty of Computer and information

Luxor University

Embedded systems Lab manual

Contents

Lab 1

Objectives

- Understand the layout of Arduino UNO board.
- Understand Arduino IDE, and how to write, compile, and upload a code to Arduino.
- Deal with LED using Arduino digital output pins.
- Understand the difference between Pull-up and Pull-down resistors configurations.
- Use push buttons
- Understand switch bouncing problem and how to overcome it by simple coding solution.
- Make some simple application in section

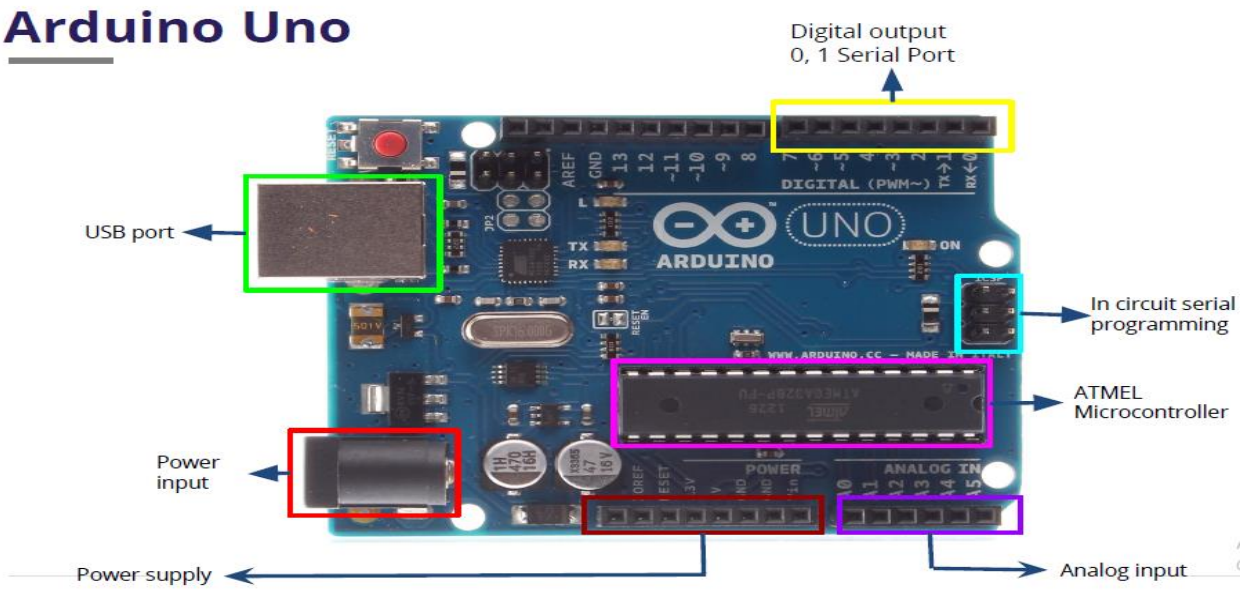
Arduino

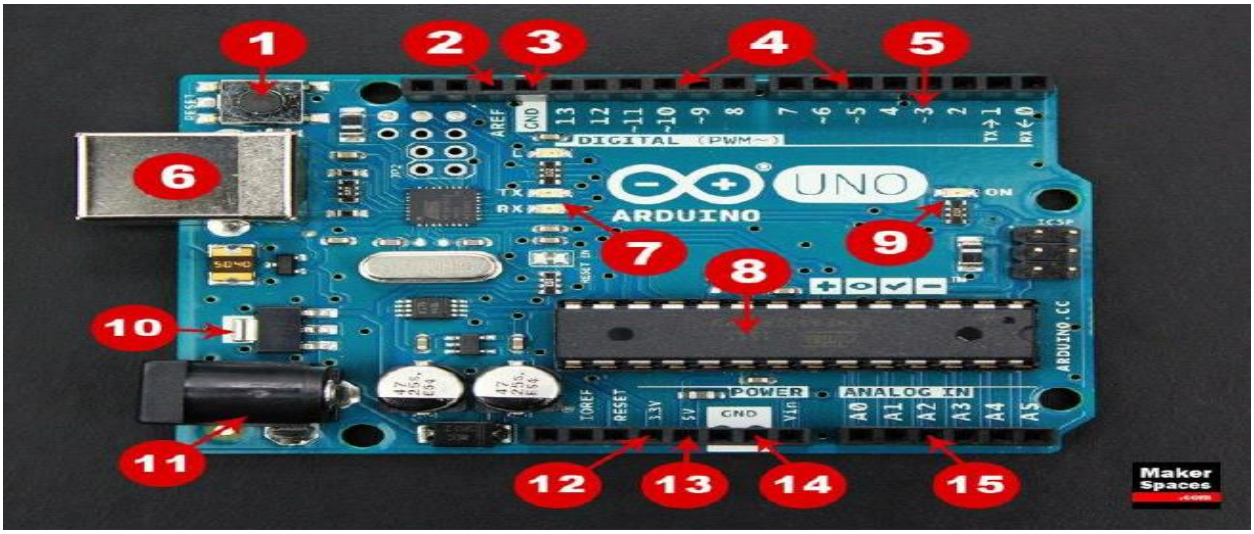
An Arduino board consists of anAtmel8-bit AVR microcontroller with complementary components that facilitate programming and incorporation into other circuits. It's an open-source physical computing platform based on a simple microcontroller board, and a development environment for writing software for the board.

Types of Arduinos

Arduino Board	Processor	Memory	Digital I/O	Analogue I/O
Arduino Uno	16Mhz ATmega328	2KB SRAM, 32KB flash	14	6 input, 0 output
Arduino Due	84MHz AT91SAM3X8E	96KB SRAM, 512KB flash	54	12 input, 2 output
Arduino Mega	16MHz ATmega2560	8KB SRAM, 256KB flash	54	16 input, 0 output
Arduino Leonardo	16MHz ATmega32u4	2.5KB SRAM, 32KB flash	20	12 input, 0 output

But in this course, we will focus on Arduino UNO and ESP2866.





Board Breakdown

Here are the components that make up an Arduino board and what each of their functions are.

1. **Reset Button** – This will restart any code that is loaded to the Arduino board
2. **AREF** – Stands for "Analog Reference" and is used to set an external reference voltage
3. **Ground Pin** – There are a few ground pins on the Arduino and they all work the same
4. **Digital Input/Output** – Pins 0-13 can be used for digital input or output
5. **PWM** – The pins marked with the (~) symbol can simulate analog output
6. **USB Connection** – Used for powering up your Arduino and uploading sketches
7. **TX/RX** – Transmit and receive data indication LEDs
8. **ATmega Microcontroller** – This is the brains and is where the programs are stored
9. **Power LED Indicator** – This LED lights up anytime the board is plugged in a power source
10. **Voltage Regulator** – This controls the amount of voltage going into the Arduino board
11. **DC Power Barrel Jack** – This is used for powering your Arduino with a power supply
12. **3.3V Pin** – This pin supplies 3.3 volts of power to your projects
13. **5V Pin** – This pin supplies 5 volts of power to your projects
14. **Ground Pins** – There are a few ground pins on the Arduino and they all work the same
15. **Analog Pins** – These pins can read the signal from an analog sensor and convert it to digital

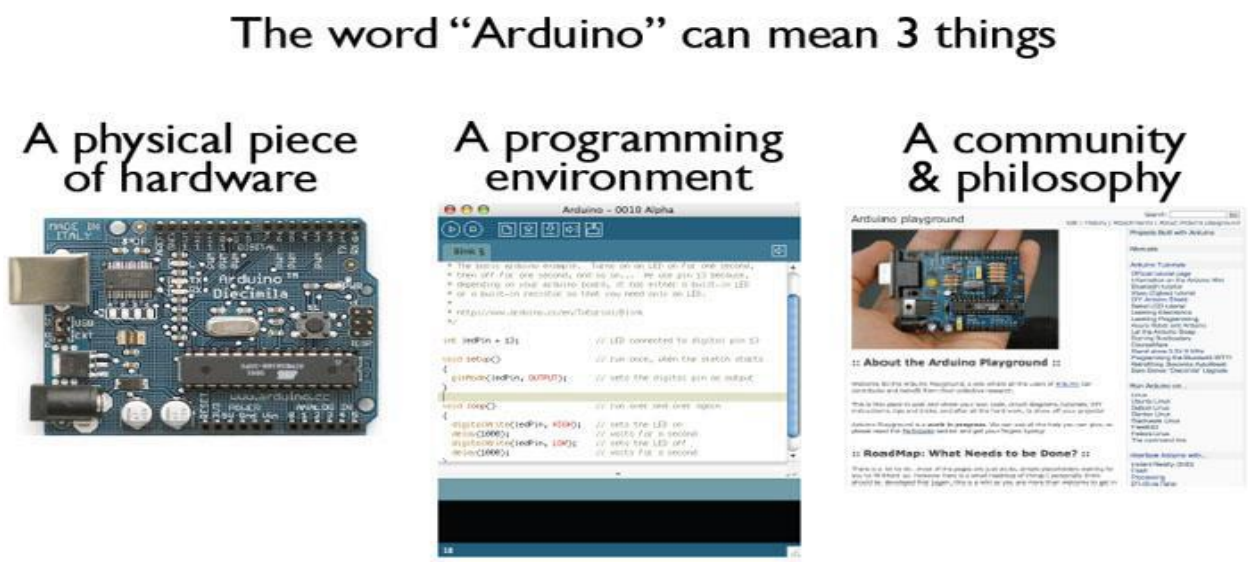
Analog and digital pins

- The Arduino can input and output analog signals as well as digital signals.
- An analog signal is one that can take on any number of values, unlike a digital signal which has only two values: HIGH and LOW.

Arduino Power Supply

The Arduino Uno needs a power source in order for it to operate and can be powered in a variety of ways. You can do what most people do and connect the board directly to your computer via a USB cable. If you want your project to be mobile, consider using a 9V battery pack to give it juice. The last method would be to use a 9V AC power supply

.



Why Arduino?

- Open source and extensible software
    - The Arduino software is published as open-source tools, available for extension by experienced programmers. The language can be expanded through C++ libraries
- Open source and extensible hardware
    - The Arduino is based on Atmel's ATMEGA 8 and ATMEGA 168 microcontrollers. The plans for the modules are published under a Creative Commons license, so experienced circuit designers can make their own version of the module, extending it and improving it.
- Inexpensive
- Cross-platform
- Simple, clear programming environment

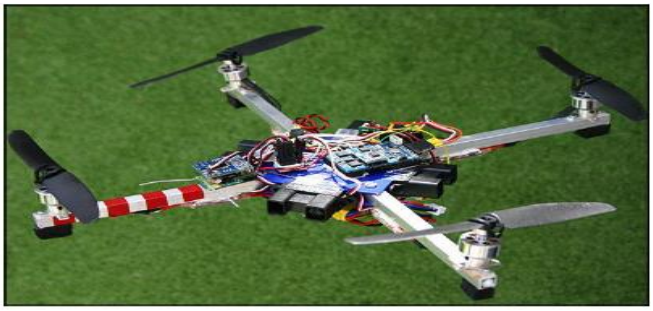
Arduino Language

- Simplified C/C++
- Peripheral libraries
    - LCD, sensors, 12C, ect.

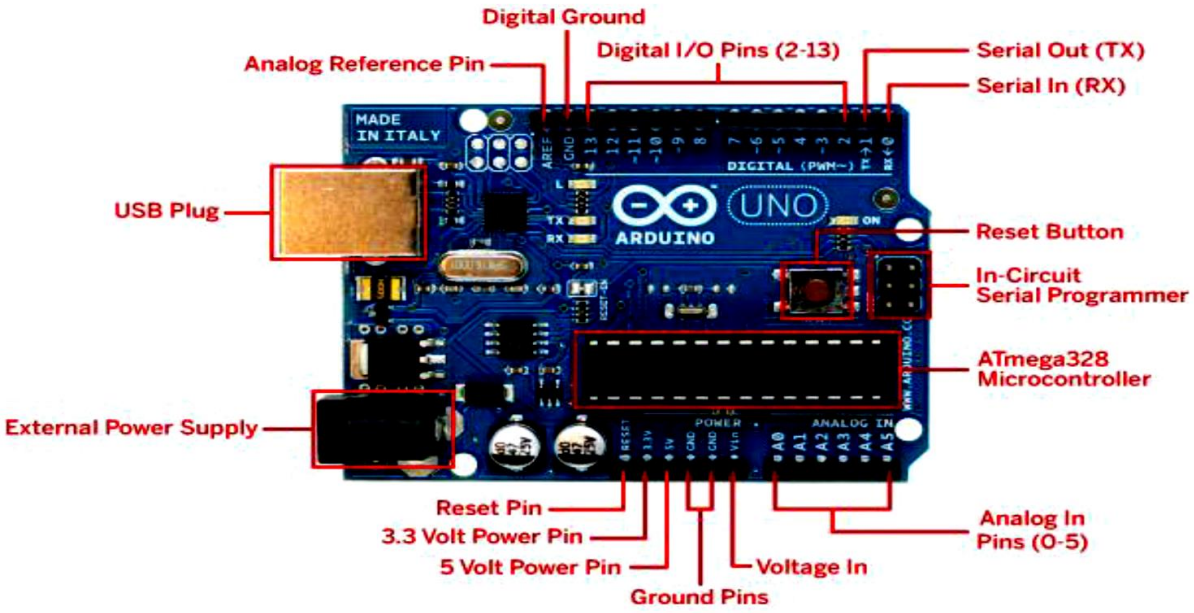
What is a Library?

A library is a big collection of procedures, where all the procedures are related! If you, say, want to control a motor, you may want to find a Motor Control Library: a collection of procedures that have already been written for you that you can use without having to do the dirty work of learning the nuances of motors.

Examples of Arduino Projects



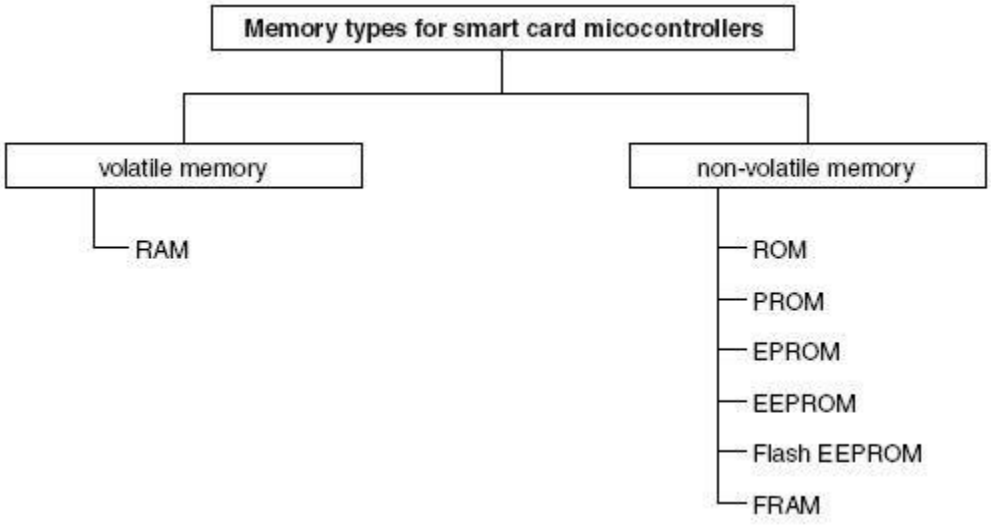
Arduino UNO board



UNO specs

Microcontroller	ATmega328
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	6
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Clock Speed	16 MHz
Length	68.6 mm
Width	53.4 mm
Weight	25 g

Memory Types



Arduino has:

- Flash memory: it's a rewritable non-volatile memory. This means that its content will still be there if you turn off the power. It's a bit like the hard disk on the Arduino board. Your program is stored here. code for writing and retrieving any data structure to EEPROM easily.

- RAM: it's like the RAM in your computer. Its content disappears when you turn off the power, but it can be read and written really fast. Every normal variable in your sketch is held in RAM while your sketch runs.
- EEPROM: it's an older technology to implement rewritable non-volatile memory. It's normally used to store settings and other parameters between resets.

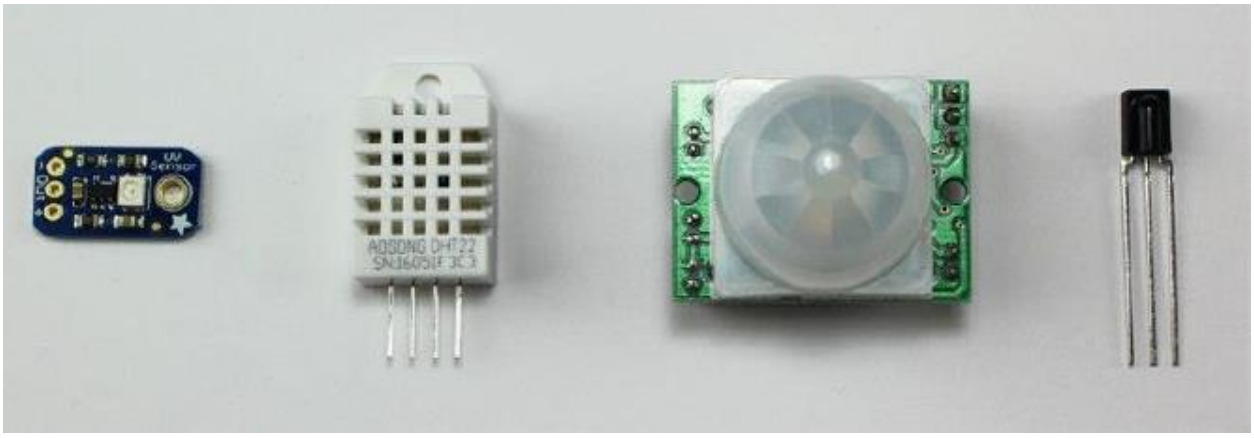
Power

- The Arduino Uno can be powered via the USB connection or with an external power supply. The power source is selected automatically.
- External (non-USB) power can come either from an AC-to-DC adapter (wall-wart) or battery. The adapter can be connected by plugging a 2.1mm center-positive plug into the board's power jack. Leads from a battery can be inserted in the Gnd and Vin pin headers of the POWER connector.
- The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may be unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts.

Arduino Sensors

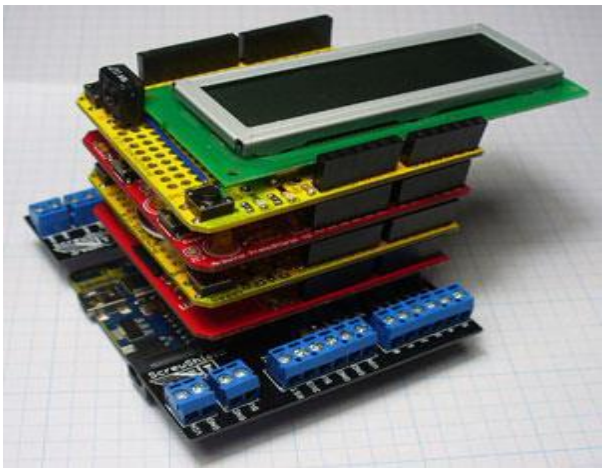
If you want your Arduino to sense the world around it, you will need to add a sensor. There are a wide range of sensors to choose from and they each have a specific purpose. Below you will find some of the commonly used sensors in projects.

- Distance Ranging Sensor
- PIR Motion Sensor
- Light Sensor
- Degree of Flex Sensor
- Pressure Sensor
- Proximity Sensor
- Acceleration Sensor
- Sound Detecting Sensor
- RGB and Gesture Sensor
- Humidity and Temperature Sensor



Arduino Shields

Shields are boards that can be plugged on top of the Arduino PCB extending its capabilities. The different shields follow the same philosophy as the original toolkit: they are easy to mount, and cheap to produce.

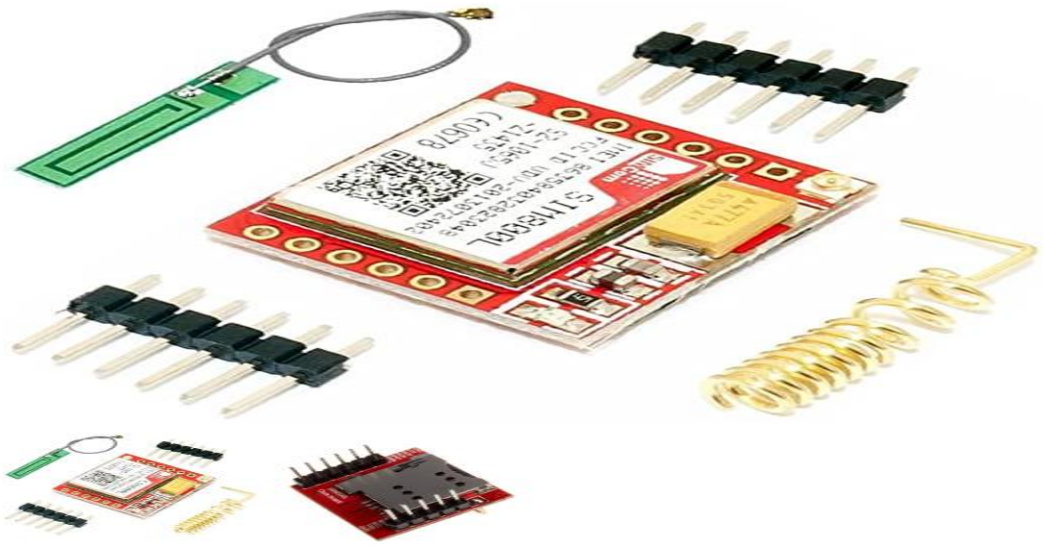


Examples Arduino Shields

- If you want to add a very specific functionality to your Arduino, you will need to use a shield. Arduino shields plug into the top of the Arduino board and can add capabilities such as WiFi, Bluetooth, GPS and much more.
- Arduino Wi-Fi Shield-This is the Arduino Ethernet Shield sans wires. This shield can get your Arduino connected to a WiFi router, so it can host webpages and scour the Internet.
- Cellular Shield w/ SM5100B-Turn your Arduino into a cellular phone! Send SMS text messages, or hook up a microphone and speaker and use it to replace your iPhone.
- GPS Shield-GPS isn't as complicated as you might think. With a GPS Shield, your Arduino will always know where it is.

- GSM/ GPRS: The GPRS Shield provides you a way to use the GSM cell phone network to receive data from a remote location. The shield allows you to achieve this via any of the three methods:
    - Short Message Service
    - Audio
    - GPRS Service

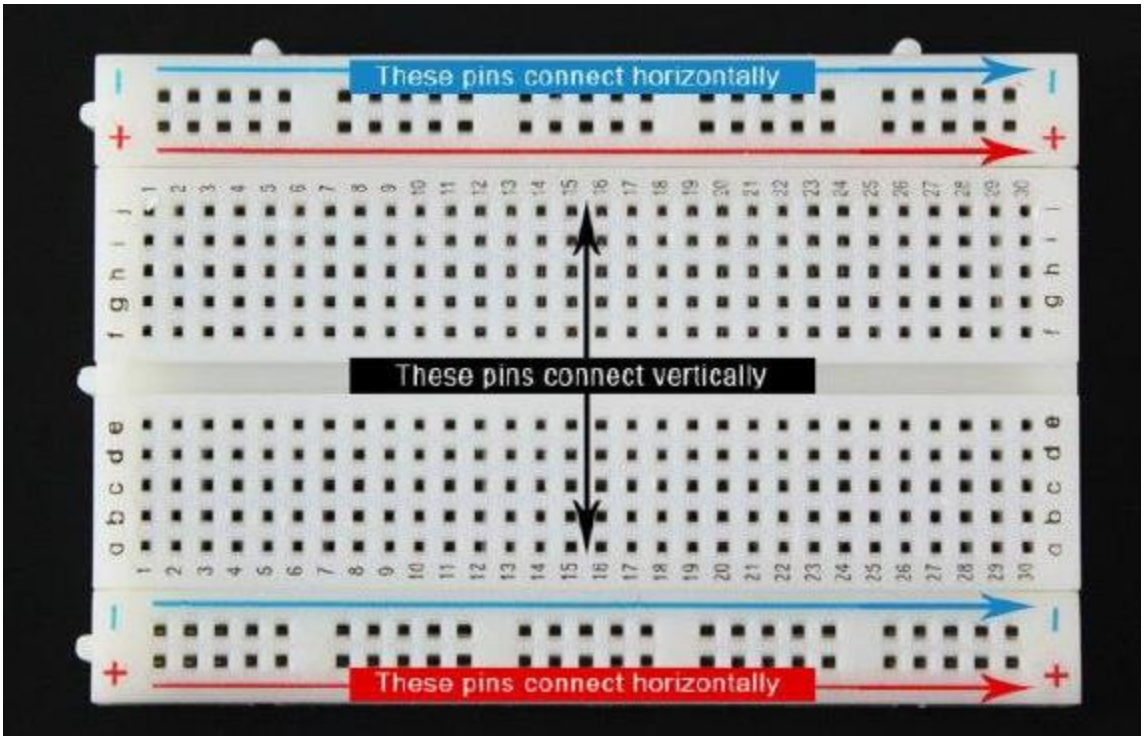
  The GPRS Shield is compatible with all boards which have the same form factor (and pinout) as a standard Arduino Board. The GPRS Shield can be configured and controlled through UART by simple AT commands. Based on the SIM900 module from SIMCOM, the GPRS Shield is like a cell phone. Besides the communications features, the GPRS Shield has 12 GPIOs, 2 PWMs and an ADC.



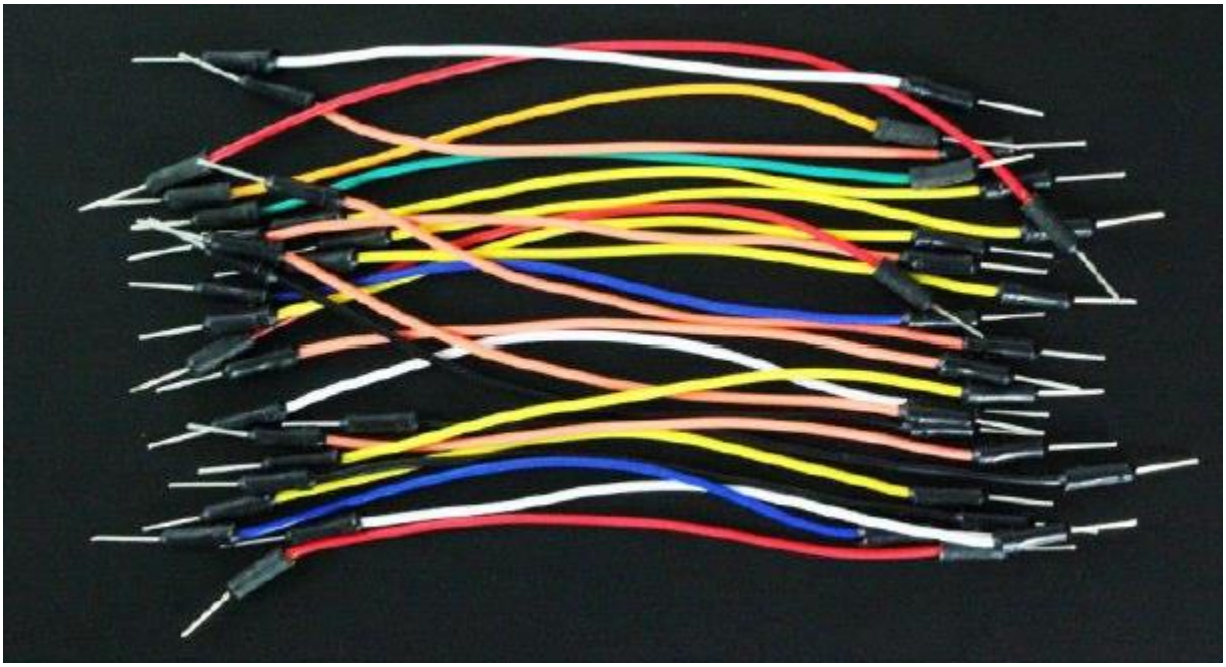
GPRS module

Arduino Breadboard

Another very important item when working with Arduino is a solderless breadboard. This device allows you to prototype your Arduino project without having to permanently solder the circuit together. Using a breadboard allows you to create temporary prototypes and experiment with different circuit designs. Inside the holes (tie points) of the plastic housing, are metal clips which are connected to each other by strips of conductive material.



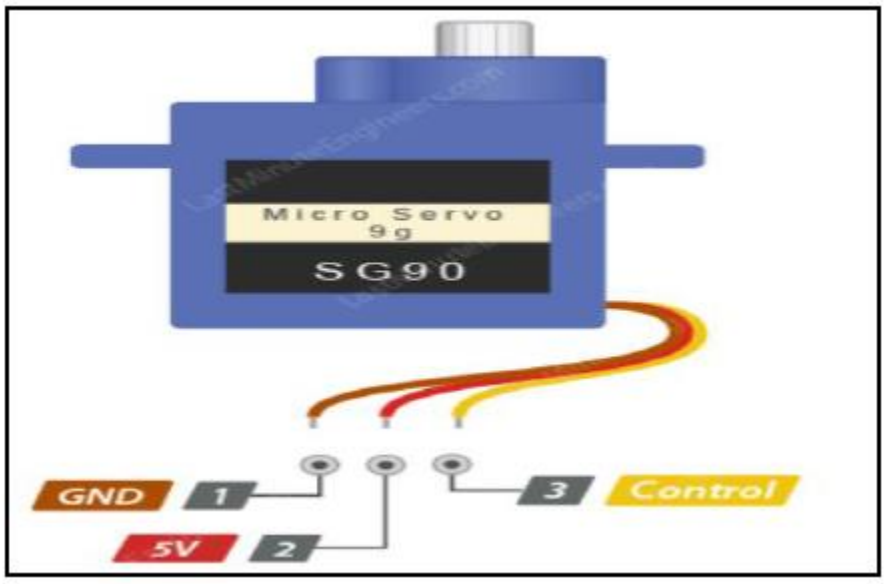
On a side note, the breadboard is not powered on its own and needs power brought to it from the Arduino board using jumper wires. These wires are also used to form the circuit by connecting resistors, switches and other components together.



Servo Motor

A Servo Motor is a small device that has an output shaft. This shaft can be positioned to specific angular positions by sending the servo a coded signal. As long as the coded signal exists on the input line, the servo will maintain the angular position of the shaft. If the coded signal changes, the angular position of the shaft

changes. Servo motors have three terminals – power, ground, and signal. The power wire is typically red, and should be connected to the 5V pin on the Arduino. The ground wire is typically black or brown as shown in figure:



Push button

Push-button is a very simple mechanism which is used to control electronic signal either by blocking it or allowing it to pass. This happens when mechanical pressure is applied to connect two points of the switch together. Push buttons or switches connect two points in a circuit when pressed. When the push-button is released, there is no connection between the two legs of the push-button.

Switch

How Arduino is programmed?

Using a software called Arduino IDE

Arduino Software (IDE)

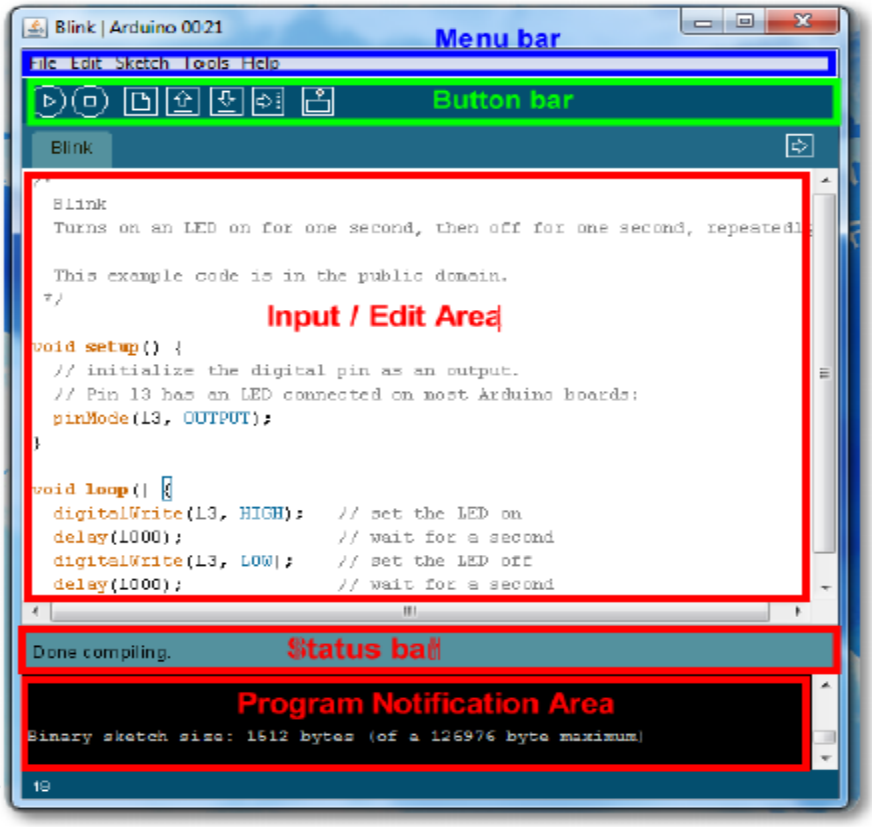
- The Arduino Integrated Development Environment - or Arduino Software (IDE) contains a text editor for writing code, a message area, a text console,

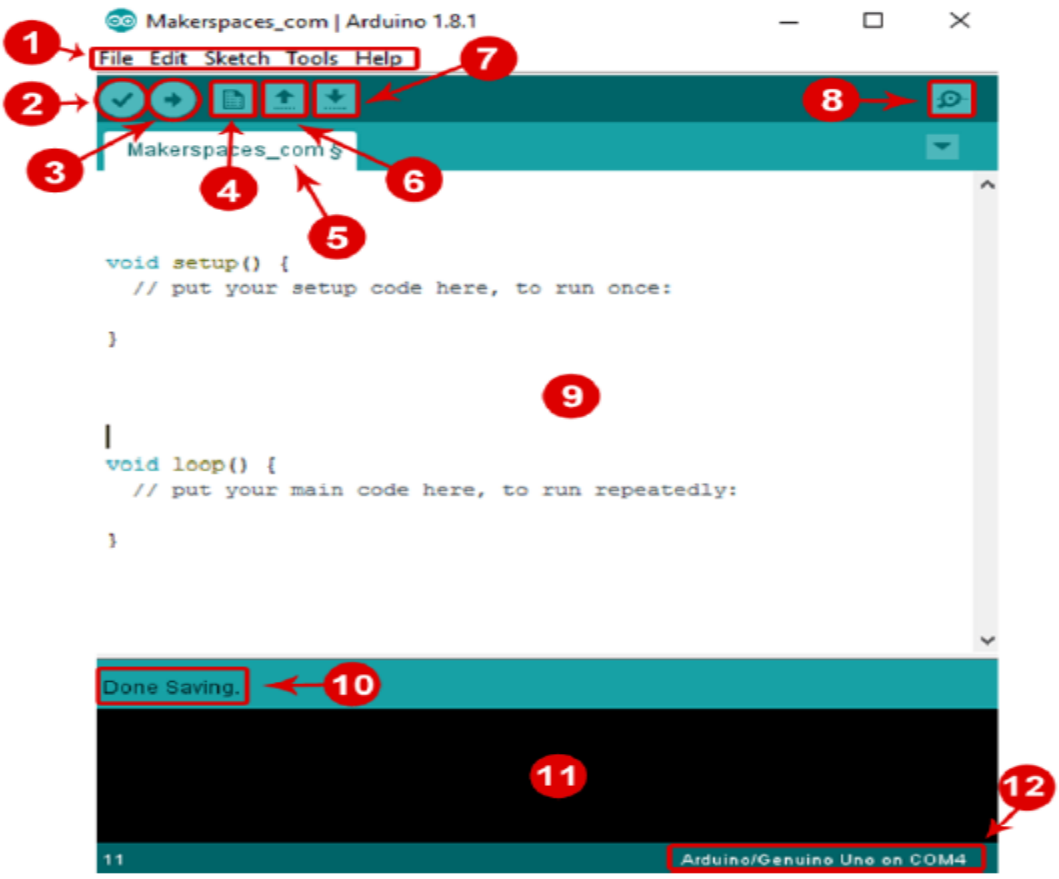
a toolbar with buttons for common functions and a series of menus. It connects to the Arduino hardware to upload programs and communicate with them.

Getting Started

- Visit: http://arduino.cc/en/Main/Software
- Download & install the Arduino environment (IDE) for Windows, Mac, or Linux. (Latest version: 1.6)
- Extract the ZIP file. (The extracted folder will contain both the Arduino program itself and also the drivers that allow the Arduino to be connected to your computer by a USB cable.
- Connect the board to your computer via the UBS cable.
- The power light on the LED will light up and you may get a 'Found New Hardware message from Windows.
- Ignore this message and cancel any attempts that Windows makes to try and install drivers automatically for you.
- Open Device Manager
    - Under the section “Other devices” you should see an icon for “unknown device”, write click on it and press update driver software.
    - Select the option: “Browse my computer for driver software”.
    - Navigate to:
    - arduino-1.0.2-windows\arduino1.0.2\drivers, in the extracted folder.
    - You should be done by successfully installing the Arduino driver.
- Launch the Arduino IDE
- Select your board (Tools>>board>>UNO)

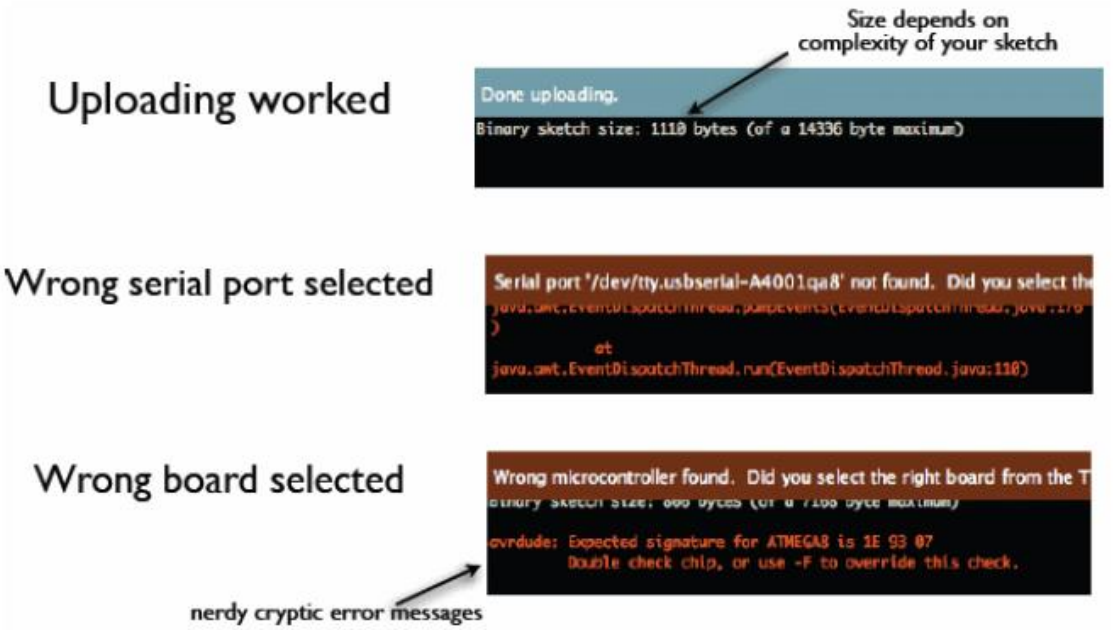
Arduino IDE


Blink | Arduino 0021
Menu bar
File Edit Sketch Tools Help
Button bar
Blink
Blink
Turns on an LED on for one second, then off for one second, repeatedly.
This example code is in the public domain.
Input / Edit Area
void setup() {
// initialize the digital pin as an output.
// Pin 13 has an LED connected on most Arduino boards:
pinMode(13, OUTPUT);
}
void loop() {
digitalWrite(13, HIGH); // set the LED on
delay(1000); // wait for a second
digitalWrite(13, LOW); // set the LED off
delay(1000); // wait for a second
Done compiling.
Status bar
Program Notification Area
Binary sketch size: 1512 bytes (of a 126976 byte maximum)
19

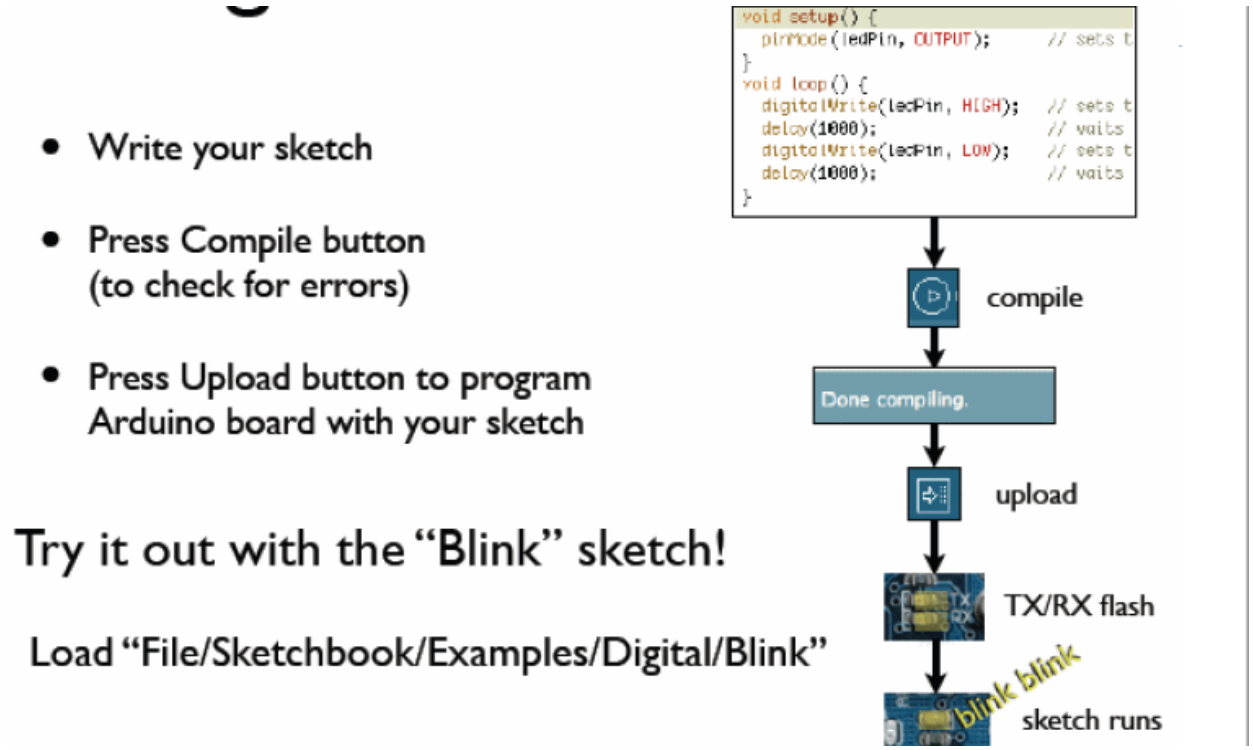

Makerspaces_com | Arduino 1.8.1
1
File Edit Sketch Tools Help
2
3
4
5
6
7
8
Makerspaces_com §
void setup() {
// put your setup code here, to run once:
}
9
void loop() {
// put your main code here, to run repeatedly:
}
Done Saving.
10
11
12
11
Arduino/Genuino Uno on COM4

1. **Menu Bar:** Gives you access to the tools needed for creating and saving Arduino sketches.
2. **Verify Button:** Compiles your code and checks for errors in spelling or syntax.
3. **Upload Button:** Sends the code to the board that's connected such as Arduino Uno in this case. Lights on the board will blink rapidly when uploading.
4. **New Sketch:** Opens up a new window containing a blank sketch.
5. **Sketch Name:** When the sketch is saved, the name of the sketch is displayed here.
6. **Open Existing Sketch:** Allows you to open a saved sketch or one from the stored examples.
7. **Save Sketch:** This saves the sketch you currently have open.
8. **Serial Monitor:** When the board is connected, this will display the serial information of your Arduino
9. **Code Area:** This area is where you compose the code of the sketch that tells the board what to do.
10. **Message Area:** This area tells you the status on saving, code compiling, errors and more.
11. **Text Console:** Shows the details of an error messages, size of the program that was compiled and additional info.
12. **Board and Serial Port:** Tells you what board is being used and what serial port it's connected to.

Status messages



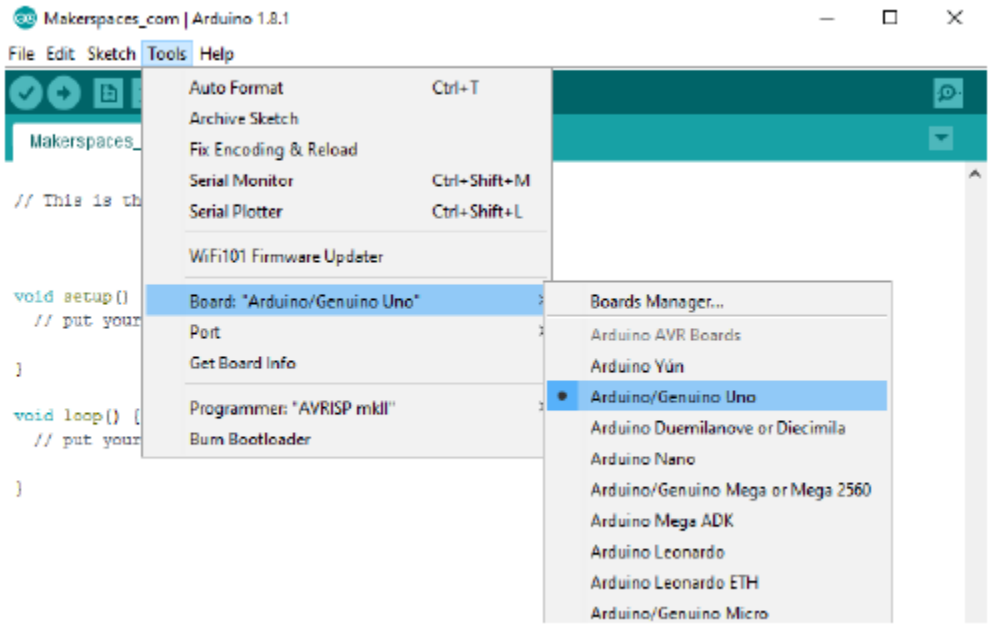
How to run and write your code



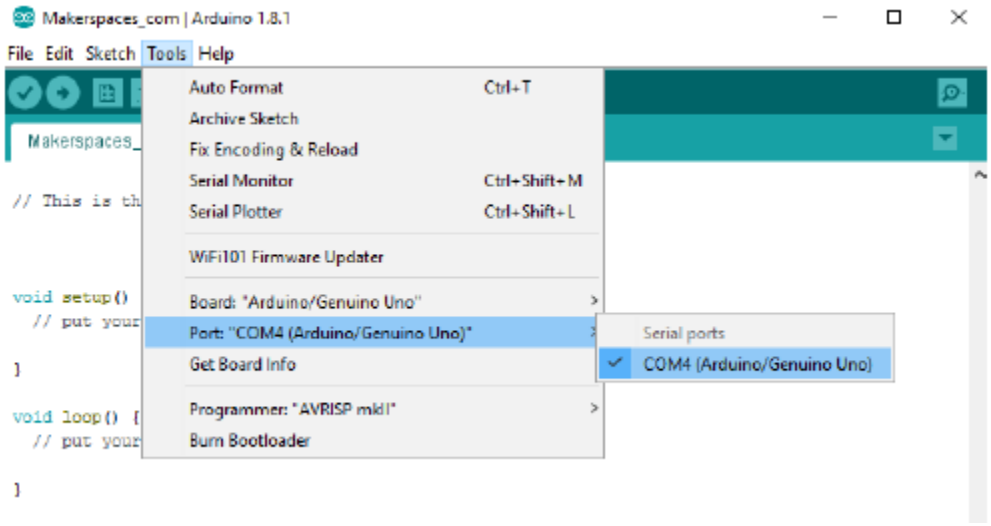
Connect Your Arduino Uno

At this point you are ready to connect your Arduino to your computer. Plug one end of the USB cable to the Arduino Uno and then the other end of the USB to your computer's USB port.

Once the board is connected, you will need to go to Tools then Board then finally select Arduino Uno.



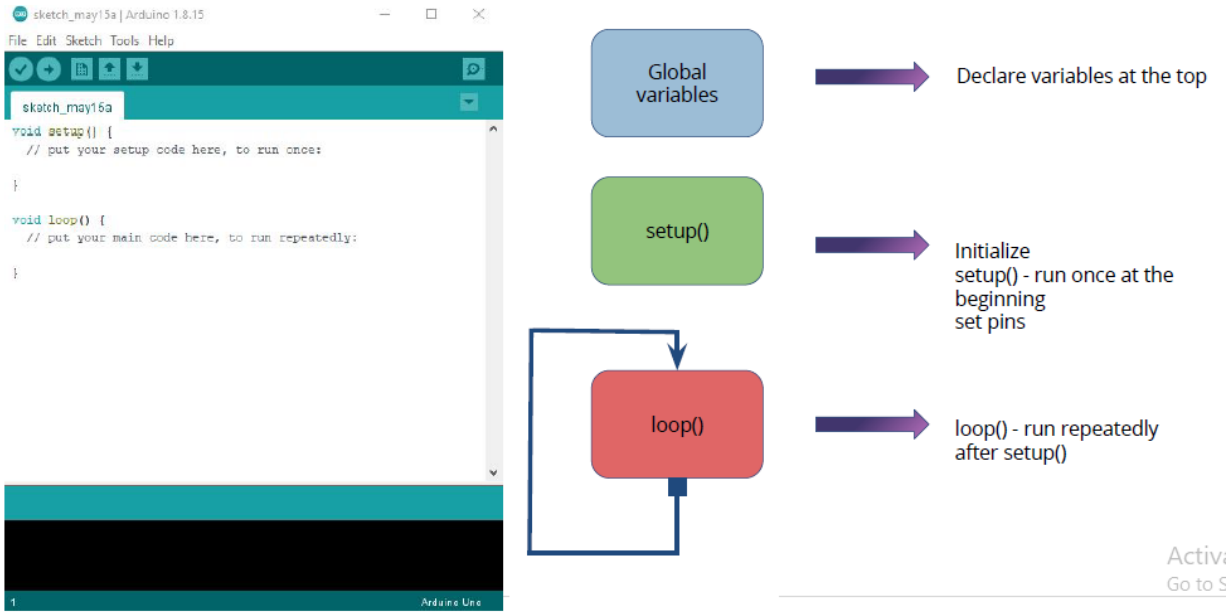
Next, you have to tell the Arduino which port you are using on your computer. To select the port, go to Tools then Port then select the port that says Arduino.



Useful functions

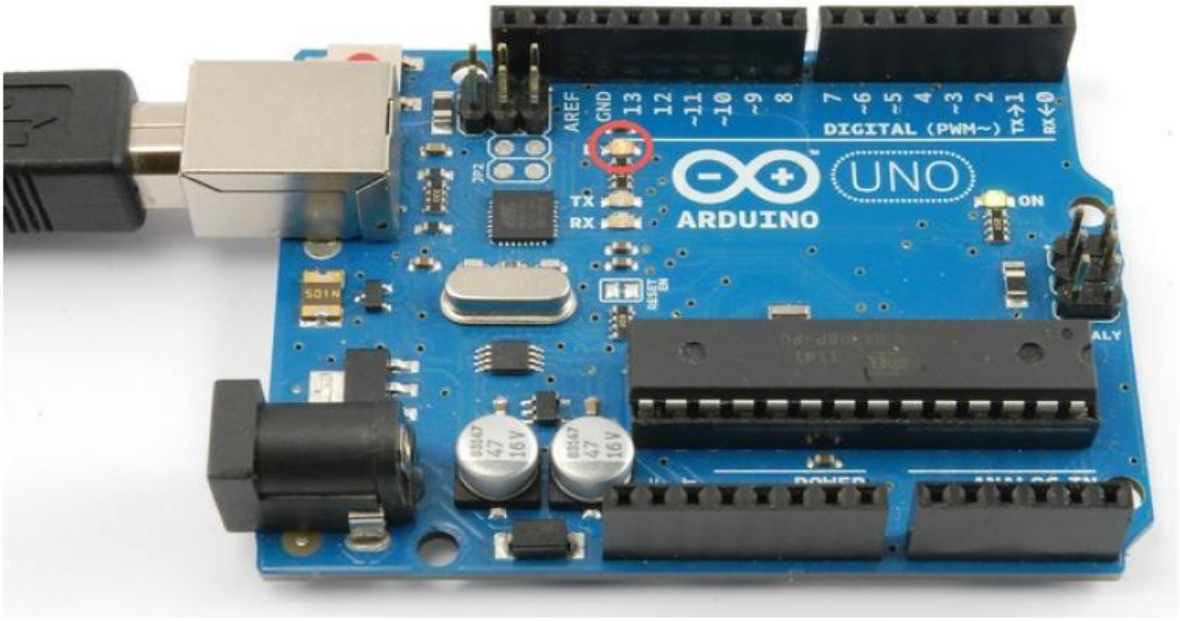
pinMode()	set pin as input or output
digitalWrite()	set a digital pin high/low
digitalRead()	read a digital pin's state
analogRead()	read an analog pin
analogWrite()	write an "analog" PWM value
delay()	wait an amount of time
millis()	get the current time

Sketch



Experiment 1 : blank build-in LED

Objective: learn how to make the built-in LED blink



- **You might notice that your Arduino board's built-in LED already blinks when you connect it to a USB.**
- **This is because Arduino boards are generally shipped with the 'Blink' sketch preinstalled.**
- **We will do a simple variation to the program by changing the rate of the blink.**

Code

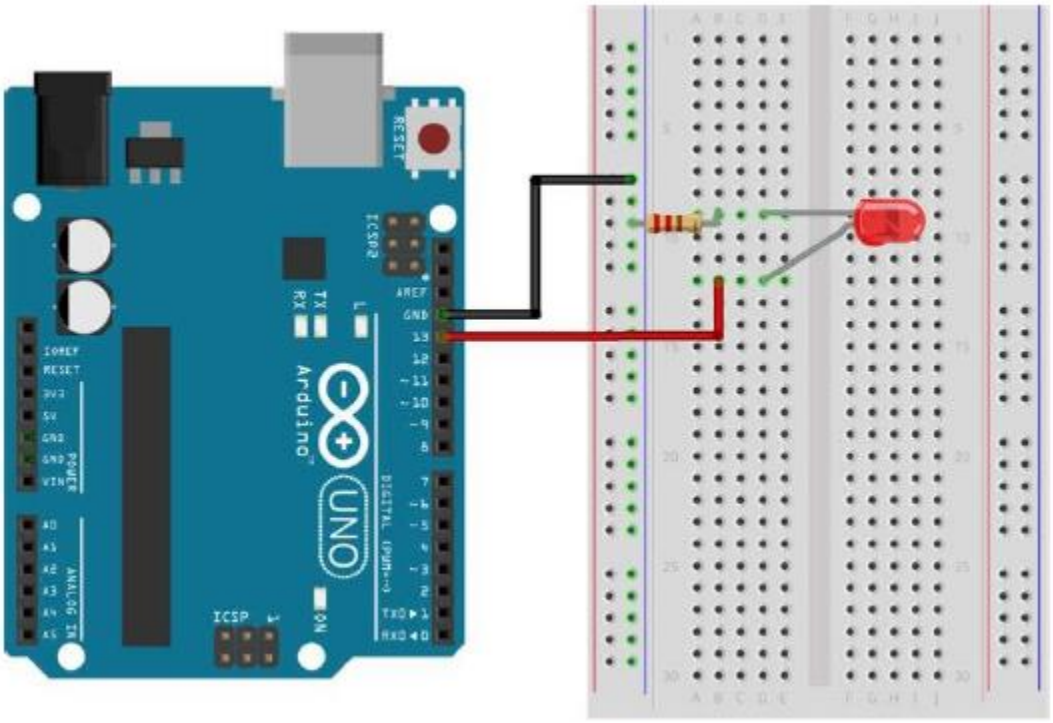
```
void setup() {
4     // initialize digital pin LED_BUILTIN as an output.
5     pinMode(LED_BUILTIN, OUTPUT);
6 }
7
8 // the loop function runs over and over again forever
9 void loop() {
10     digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage
level)
11     delay(1000);                       // wait for a second
12     digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage
LOW
```

```
13      delay(1000);                        // wait for a second
14  }
```

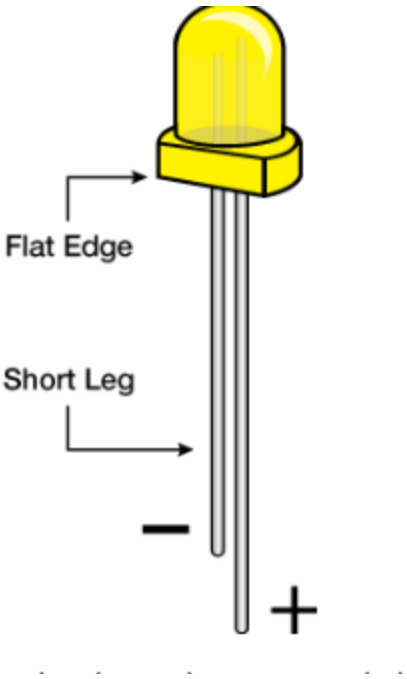
Experiment 2 : Blink an external LED

Objective: To understand how to turn on or off led

Circuit



Note – To find out the polarity of an LED, look at it closely. The shorter of the two legs, towards the flat edge of the bulb indicates the negative terminal.



Required Parts

- Arduino Uno Board
- Breadboard – half size
- Jumper Wires
- USB Cable
- LED (5mm)
- 220 Ohm Resistor

Connect The Parts

You can build your Arduino circuit by looking at the breadboard image above or by using the written description below. In the written description, we will use a letter/number combo that refers to the location of the component. If we mention H19 for example, that refers to column H, row 19 on the breadboard.

Step 1 – Insert black jumper wire into the GND (Ground) pin on the Arduino and then in the GND rail of the breadboard row 15

Step 2 – Insert red jumper wire into pin 13 on the Arduino and then the other end into F7 on the breadboard

Step 3 – Place the LONG leg of the LED into H7

Step 4 – Place the SHORT leg of the LED into H4

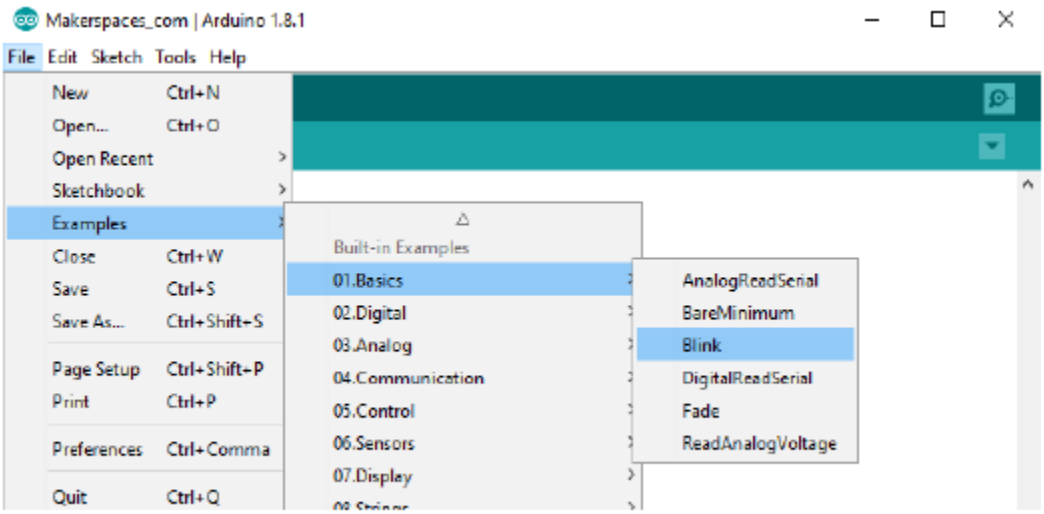
Step 5 – Bend both legs of a 220 Ohm resistor and place one leg in the GND rail around row 4 and other leg in I4

Step 6 – Connect the Arduino Uno to your computer via USB cable

Upload The Blink Sketch

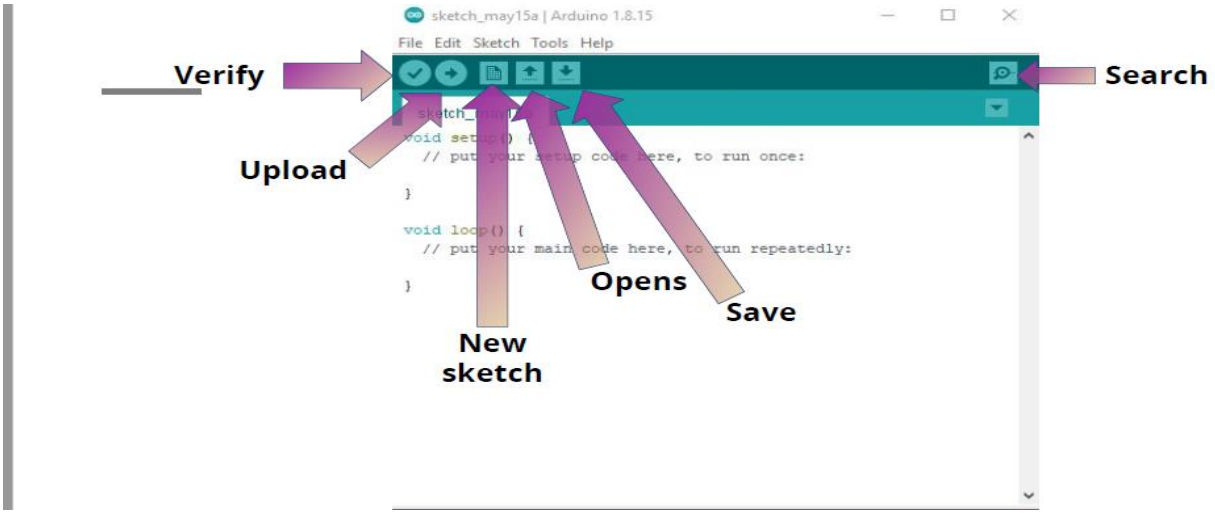
Now it's time to upload the sketch (program) to the Arduino and tell it what to do. In the IDE, there are built-in example sketches that you can use which make it easy for beginners.

To open the blink sketch, you will need to go to **File** > **Examples** > **Basics** > **Blink**



Now you should have a fully coded blink sketch that looks like the image below.

Next, you need to click on the verify button (check mark) that's located in the top left of the IDE box. This will compile the sketch and look for errors. Once it says "Done Compiling" you are ready to upload it. Click the upload button (forward arrow) to send the program to the Arduino board.



The built-in LEDs on the Arduino board will flash rapidly for a few seconds and then the program will execute. If everything went correctly, the LED on the breadboard should turn on for a second and then off for a second and continue in a loop.

NOTE – Arduino measures time in milliseconds and 1000 milliseconds = 1 second. The original code (1000) turns on the LED for 1 second and then off for 1 second. By adjusting the code from (1000) to (200) it shortens the time between on and off which makes it blink faster.

You must codded it by your self:

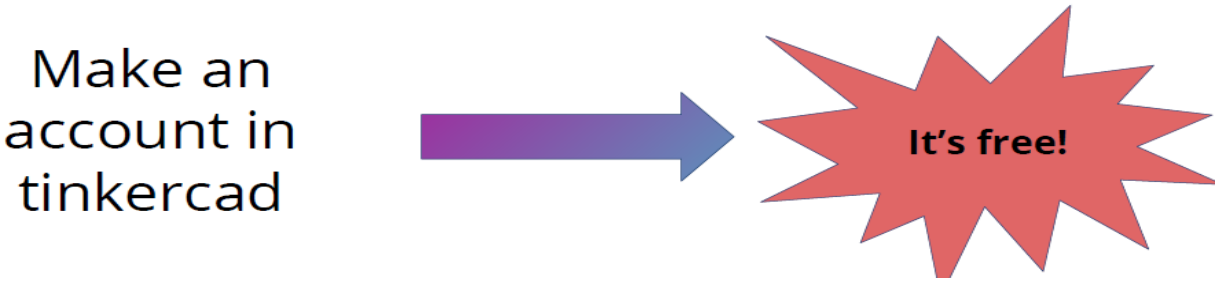
Code:

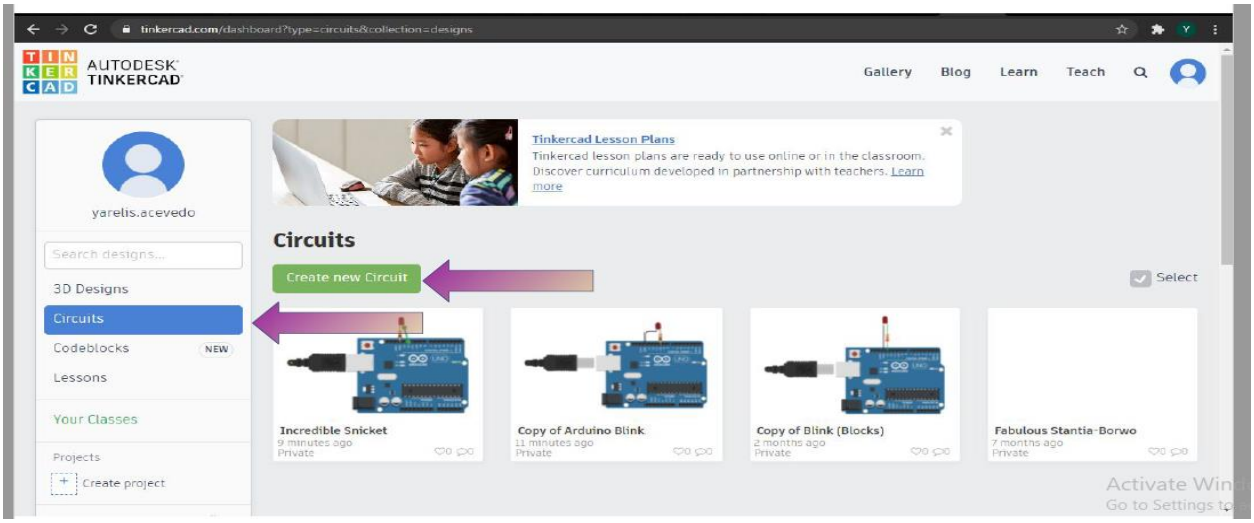
```
void setup() {  // initialize digital pin 13 as an
output.
   pinMode(2, OUTPUT);
}

// the loop function runs over and over again forever

void loop() {
   digitalWrite(2, HIGH); // turn the LED on (HIGH is the
voltage level)
   delay(1000); // wait for a second
   digitalWrite(2, LOW); // turn the LED off by making
the voltage LOW
   delay(1000); // wait for a second
}
```

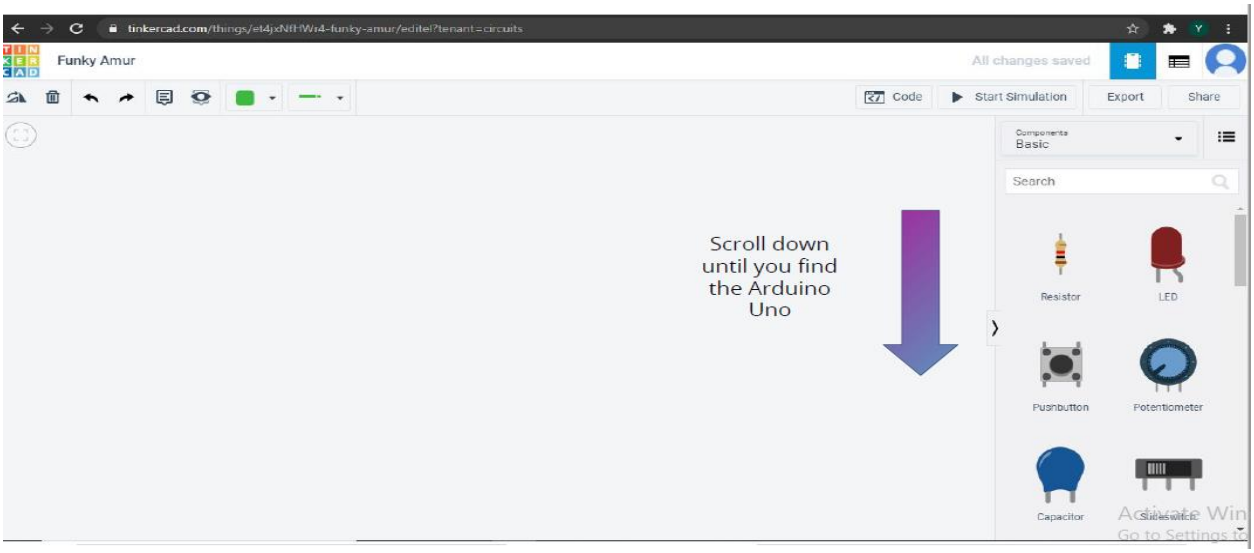
Experiment 3: Blink an external LED using any simulation platform such as tinkercad or proteus

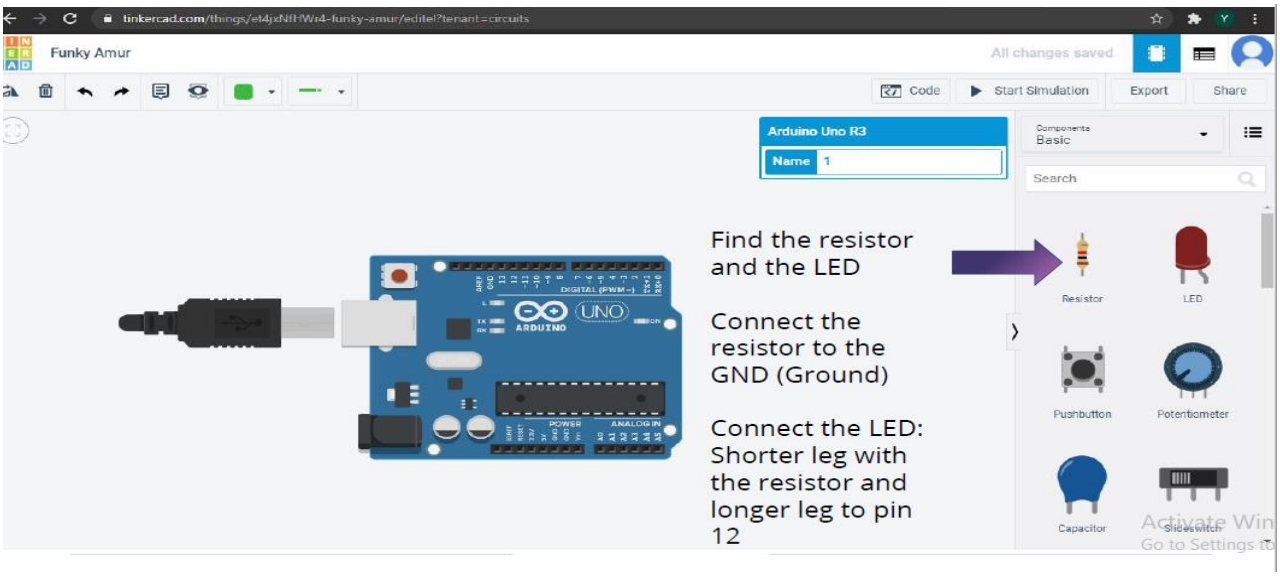
Objective: To understand how to turn on or off led using tinkercad website.

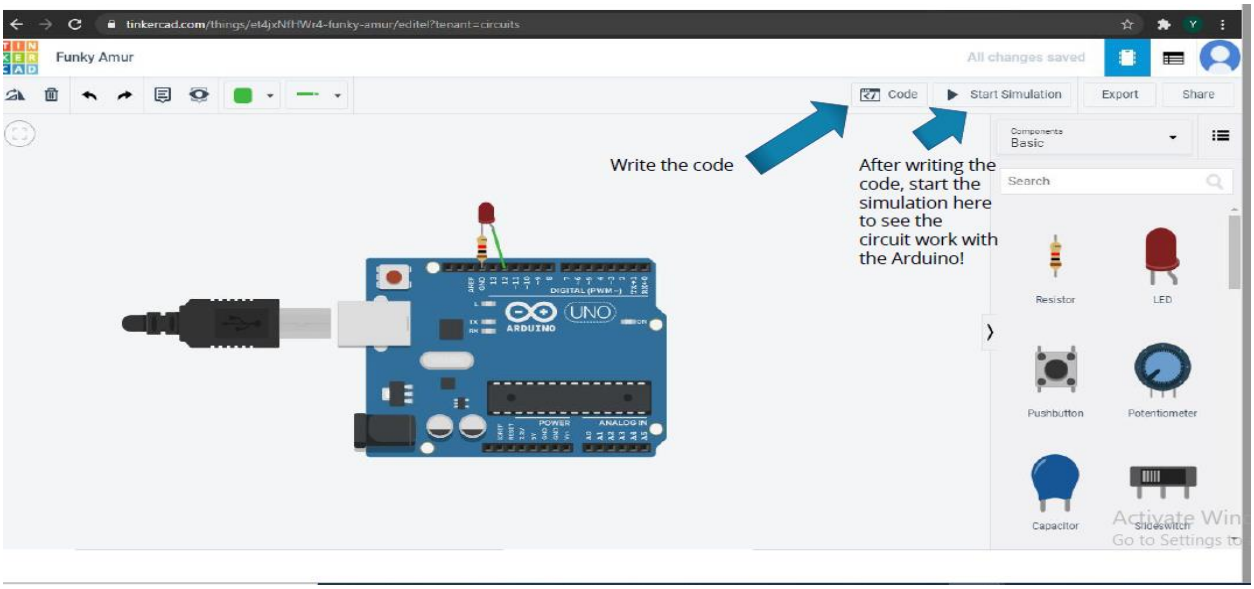
Tinkercad is a free, online 3D modeling program that runs in a web browser, known for its simplicity and ease of use. Since it became available in 2011 it has become a popular platform for creating models for 3D printing as well as an entry-level introduction to constructive solid geometry in schools.

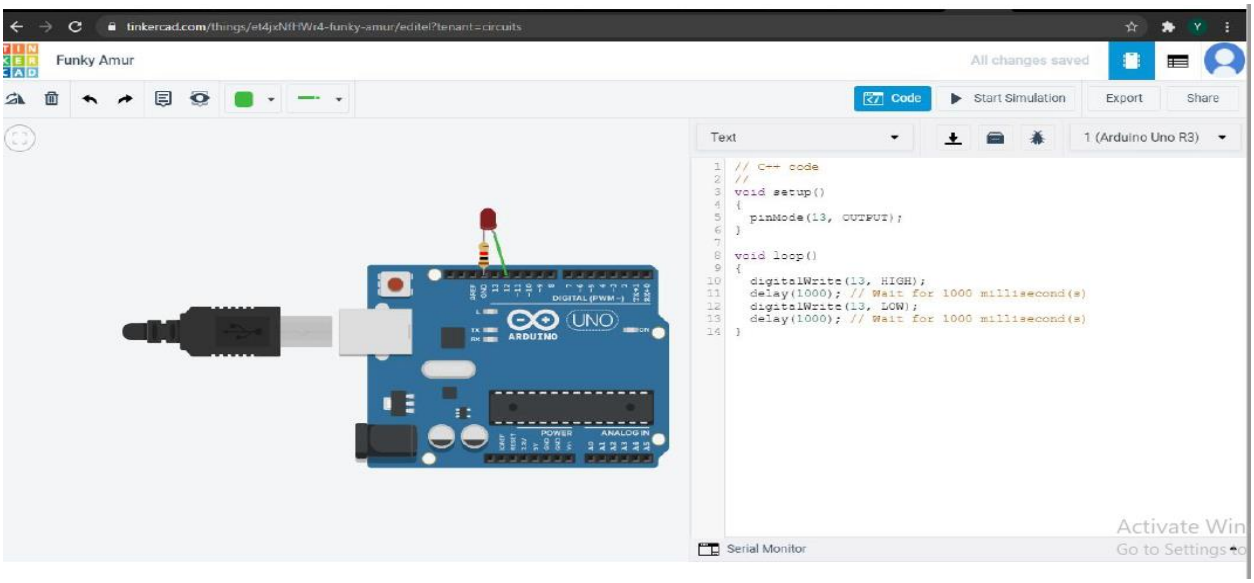







tinkercad.com/things/et4jxNfHWr4-funky-amur/editel?tenant=circuits
Funky Amur
All changes saved
Code
Start Simulation
Export
Share
Arduino Uno R3
Name 1
Components
Basic
Search
Find the resistor and the LED
Connect the resistor to the GND (Ground)
Connect the LED: Shorter leg with the resistor and longer leg to pin 12
Resistor
LED
Pushbutton
Potentiometer
Capacitor
Slide switch
Activate Win
Go to Settings to

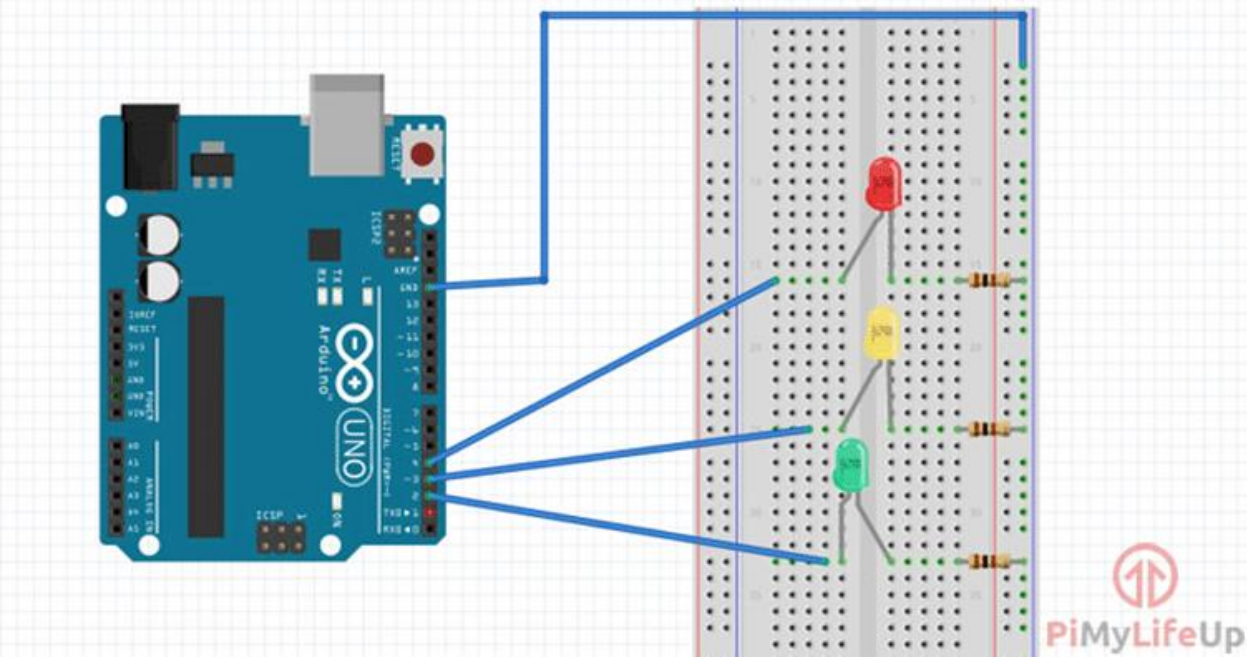

tinkercad.com/things/et4jxNfHWr4-funky-amur/editel?tenant=circuits
Funky Amur
All changes saved
Code
Start Simulation
Export
Share
Write the code
After writing the code, start the simulation here to see the circuit work with the Arduino!
Components
Basic
Search
Resistor
LED
Pushbutton
Potentiometer
Capacitor
Slide switch
Activate Win
Go to Settings to


tinkercad.com/things/et4jxNfHWr4-funky-amur/editel?tenant=circuits
Funky Amur
All changes saved
Code
Start Simulation
Export
Share
Text
1 (Arduino Uno R3)
// C++ code
//
void setup()
{
pinMode(13, OUTPUT);
}
void loop()
{
digitalWrite(13, HIGH);
delay(1000); // Wait for 1000 millisecond(s)
digitalWrite(13, LOW);
delay(1000); // Wait for 1000 millisecond(s)
}
Serial Monitor
Activate Win
Go to Settings to

Experiment 4: Traffic lights using LED's

Objective: To switch the traffic lights with a pre-defined time

Circuit Diagram



Component

- Arduino Uno
- Red LED, Yellow LED, and Green LED
- 3 x 100 Ohm resistor (Color = Brown Black Brown)
- Breadboard wire
- Breadboard

Code

```
int GREEN = 2;
int YELLOW = 3;
int RED = 4;
int DELAY_GREEN = 5000;
int DELAY_YELLOW = 2000;
int DELAY_RED = 5000;

void setup()
{
  pinMode(GREEN, OUTPUT);
  pinMode(YELLOW, OUTPUT);
  pinMode(RED, OUTPUT);
}

void loop()
```

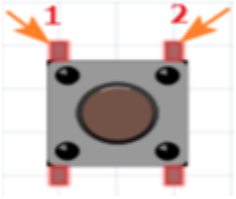
```
{
  green_light();
  delay(DELAY_GREEN);
  yellow_light();
  delay(DELAY_YELLOW);
  red_light();
  delay(DELAY_RED);
}

void green_light()
{
  digitalWrite(GREEN, HIGH);
  digitalWrite(YELLOW, LOW);
  digitalWrite(RED, LOW);
}

void yellow_light()
{
  digitalWrite(GREEN, LOW);
  digitalWrite(YELLOW, HIGH);
  digitalWrite(RED, LOW);
}

void red_light()
{
  digitalWrite(GREEN, LOW);
  digitalWrite(YELLOW, LOW);
  digitalWrite(RED, HIGH);
}
```

Experiment 5: Controlling the Light Emitting Diode (LED) with a push button.

Objective: control turn on and off the led using bush button switch

Push-button is a very simple mechanism which is used to control electronic signal either by blocking it or allowing it to pass. This happens when mechanical pressure is applied to connect two points of the switch together. Push buttons or switches connect two points in a circuit when pressed. When the push-button is released, there is no connection between the two legs of the push-button. Here it turns on the built-in LED on pin 11 when the button is pressed. The LED stays ON as long as the button is being pressed.

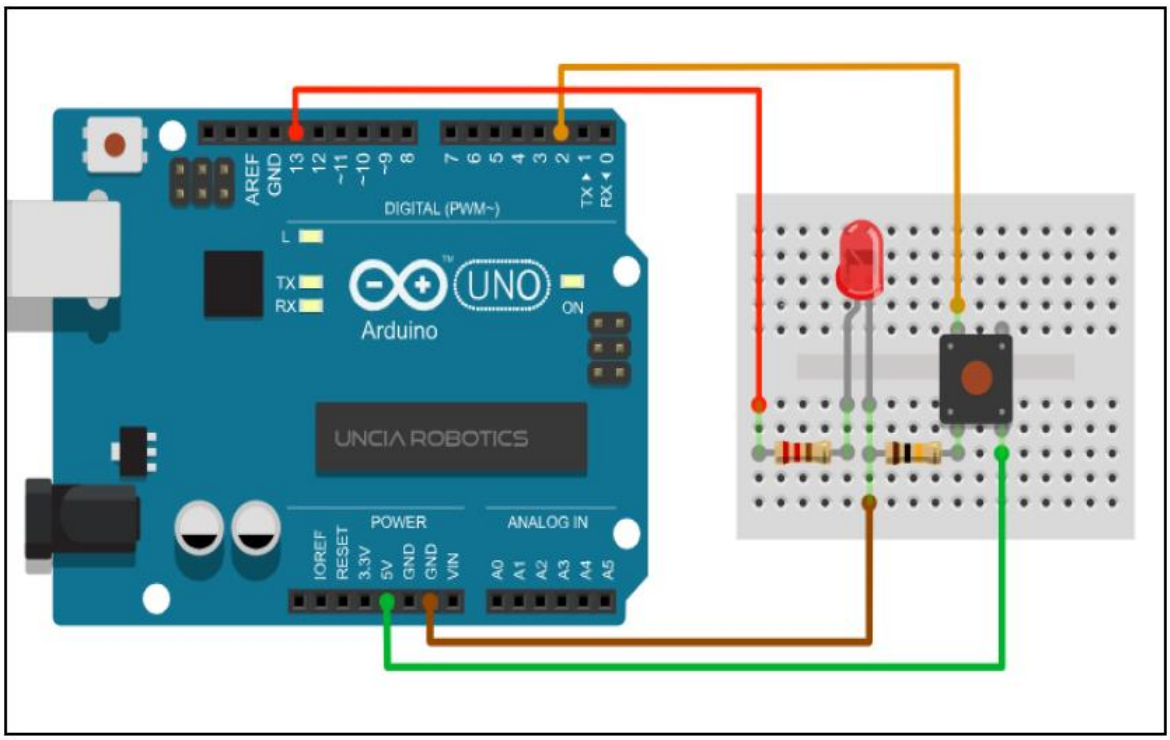
Push button



Hardware Required:

Component Name	Quantity
Arduino UNO	1
LED	1
Push Button	1
220Ω resistor	1
10KΩ resistor	1
USB Cable	1
Breadboard	1
Jumper wires	Several

Connection Diagram:



Steps of working

1. Insert the push button into your breadboard and connect it to the digital pin 7(D7) which act as INPUT.

2. Insert the LED into the breadboard. Attach the positive leg (the longer leg) to digital pin 11 of the Arduino Uno, and the negative leg via the 220-ohm resistor to GND. The pin D11 is taken as OUTPUT.
3. The 10kΩ resistor used as PULL-UP resistor and 220 Ω resistors is used to limit the current through the LED.
4. Upload the code as given below.
5. Press the push-button to control the ON state of LED.

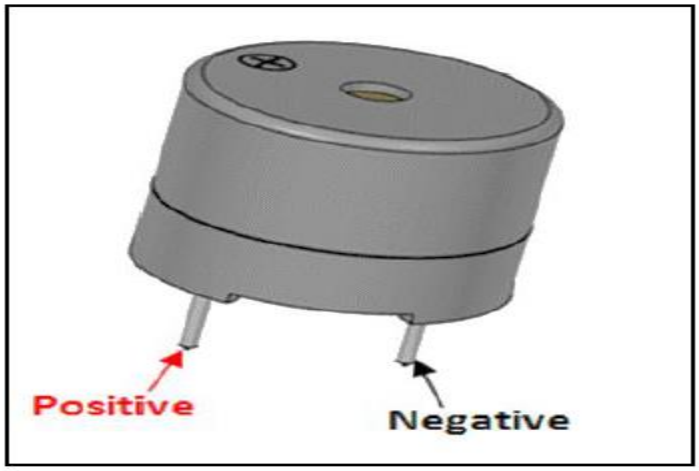
Code

```
const int buttonPin = 7; // choose the pin for the pushbutton
const int ledPin = 11; // choose the pin for a LED
int buttonState = 0; // variable for reading the pushbutton pin status
void setup()
{
pinMode(ledPin, OUTPUT); // declare LED as output
pinMode(buttonPin, INPUT); // declare pushbutton as input
}
void loop()
{
buttonState = digitalRead(button Pin); // read input value
if (buttonState == HIGH)
{ // check if the input is HIGH (button pressed)
digitalWrite(ledPin, HIGH); // turn LED ON
}
else
{
digitalWrite(ledPin, LOW); // turn LED OFF}}
```

Experiment 6: Interfacing of the Active Buzzer with Arduino.

Objective: to turn on or off the buzzer

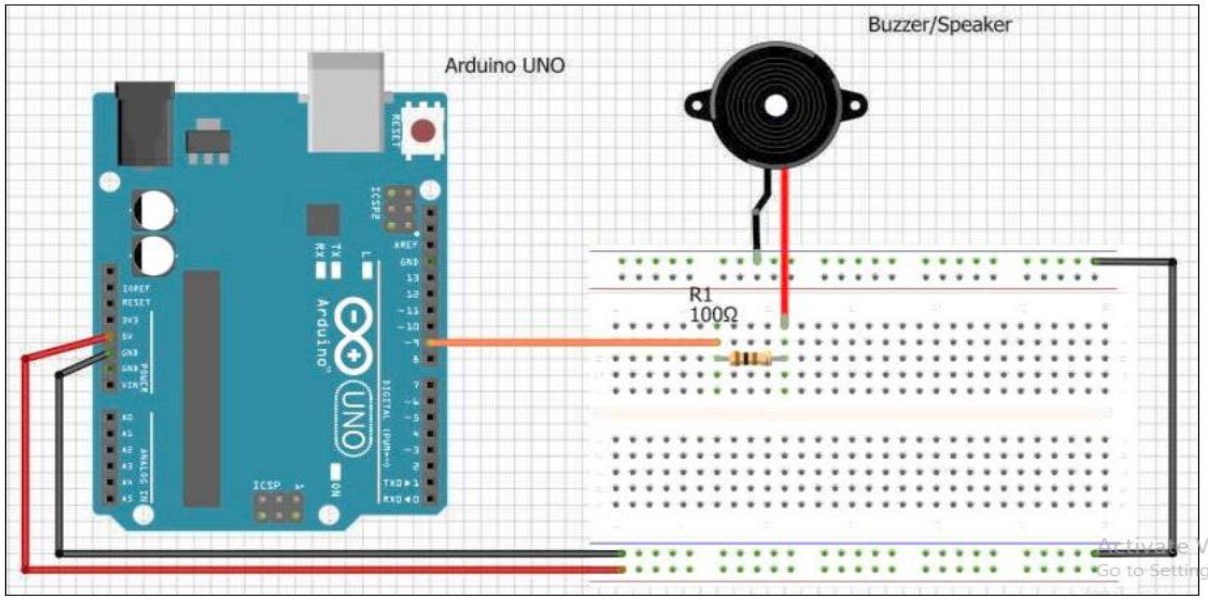
A piezo buzzer is a type of electronic device that's used to produce beeps and tones. The working principle of the device is piezoelectric effect. The main component of this device is a piezo crystal, which is a special material that changes shape when a voltage applied to it. The active buzzer will only generate sound when it will be electrified. It generates sound at only one frequency. This buzzer operates at an audible frequency of about 2 KHz.



Hardware Required:

Component Name	Quantity
Arduino UNO	1
Buzzer / piezo speaker	1
220-ohm resistors	1
USB Cable	1
Breadboard	1
Jumper wires	several

Connection Diagram:



Steps of working:

- Connect the Supply wire (RED) of the buzzer to the Digital Pin 9 of the Arduino through a 100-ohm resistor.
- Connect the Ground wire (BLACK) of the buzzer to any Ground Pin on the Arduino.
- Upload the code
- Observe the changes in the pitch and volume of the buzzer.

Code

```
int buzzer = 9; //the pin of the active buzzer
void setup()
{
pinMode (buzzer,OUTPUT); //initialize the buzzer pin as an output
}
void loop(){
unsigned char i;
while(1){
//output a frequency
for(i=0;i<80;i++)
{
digitalWrite(buzzer,HIGH);
delay(1); //wait for 1ms
digitalWrite(buzzer,LOW);
delay(1); //wait for 1ms
}
//output another frequency
for(i=0;i<100;i++){
digitalWrite(buzzer,HIGH);
delay(2); //wait for 2ms
digitalWrite(buzzer,LOW);
```

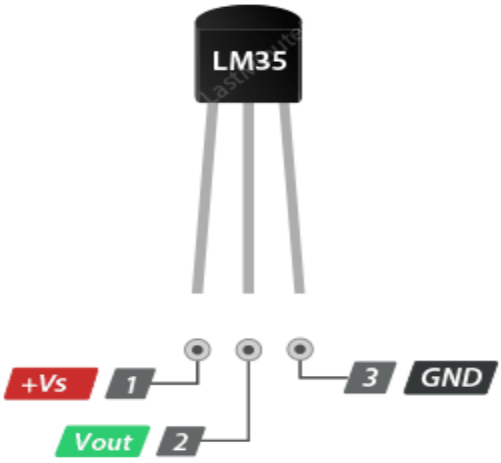
```
delay(2); //wait for 2ms
}
}
}
```

Experiment 7: Measuring temperature using LM-35

Objective: To measure the room temperature of a place through Arduino using LM-35.

LM-35 is an absolute temperature sensor which can measure the temperature of the surroundings within 100 to 500 feet. LM-35 output voltage is proportional to the Celsius/Centigrade temperature which increments the output by 1 on every 10-mV change in temperature. LM-35 can measure from -50 to 150 degree Celsius.

- Arduino analog pins work normally on +5 V.
- Resolution of analog pin starts from 0 to 1023.
- Maximum voltage of LM-35 is 1.5 V.
- Formula for converting the voltage into system input number = (V/5) *1023



+Vs is the power supply for the sensor which can be anywhere between 4V to 30V.

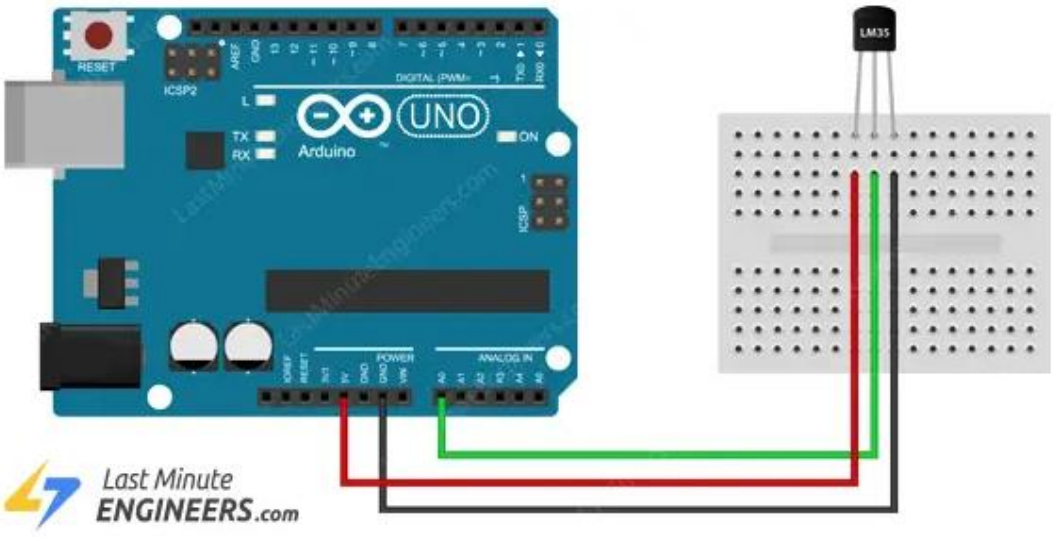
Vout pin produces an analog voltage that is directly proportional (linear) to the temperature. It should be connected to an Analog (ADC) input.

GND is a ground pin.

Hardware Required:

Component Name	Quantity
Arduino UNO	1
Lm35	1
USB Cable	1
Breadboard	1
Jumper wires	Several

Circuit Diagram



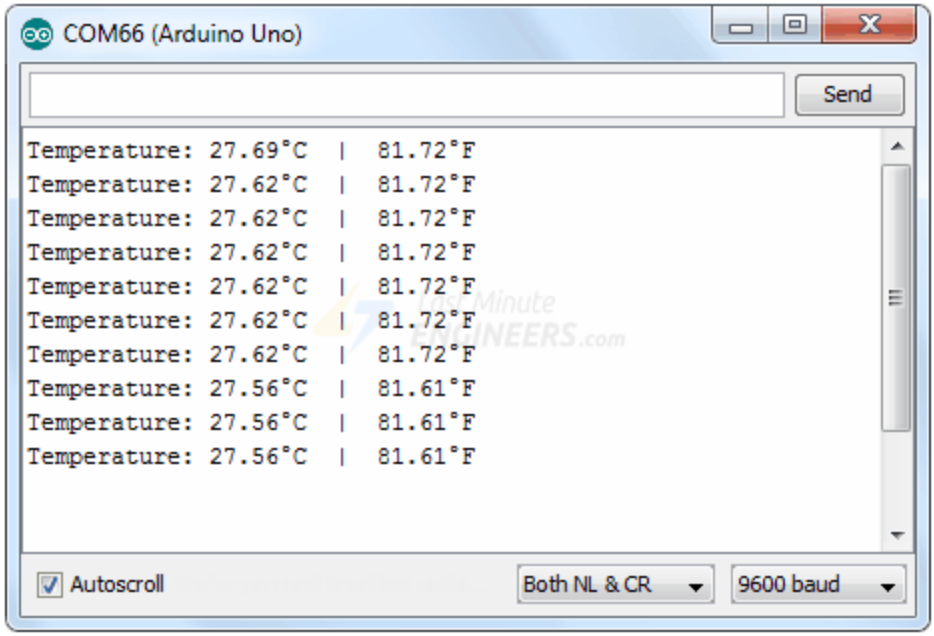
Steps of working

1. Insert the temperature sensor into your breadboard and connect its pin1 to the supply.
2. Connect its center pin to the analog pin A0 and the remaining pin3 to GND on the breadboard.
3. Upload the code as given below.
4. Vary the temperature and read the voltage changes.
5. Open the Arduino IDE's serial monitor to see the results.

Code

```
// Define the analog pin, the LM35's Vout pin is connected to
#define sensorPin A0

void setup() {
  // Begin serial communication at 9600 baud rate
  Serial.begin(9600);
}

void loop() {
  // Get the voltage reading from the LM35
  int reading = analogRead(sensorPin);

  // Convert that reading into voltage
  float voltage = reading * (5.0 / 1024.0);

  // Convert the voltage into the temperature in Celsius
  float temperatureC = voltage * 100;

  // Print the temperature in Celsius
  Serial.print("Temperature: ");
  Serial.print(temperatureC);
  Serial.print("\xC2\xB0"); // shows degree symbol
  Serial.print("C  |  ");

  // Print the temperature in Fahrenheit
  float temperatureF = (temperatureC * 9.0 / 5.0) + 32.0;
  Serial.print(temperatureF);
  Serial.print("\xC2\xB0"); // shows degree symbol
  Serial.println("F");

  delay(1000); // wait a second between readings
}
```

Sample output



Experiment 8: measure humidity and temperature using DHT11 and DHT22

Objective: To study and Interface the Temperature and Humidity with the Arduino

The Temperature Humidity sensor provides a pre-calibrated digital output. A unique capacitive sensor element measures relative humidity and the temperature is measured by a negative temperature coefficient (NTC) thermistor. It has excellent reliability and long term stability. Please note that this sensor will not work for temperatures below 0 degree.

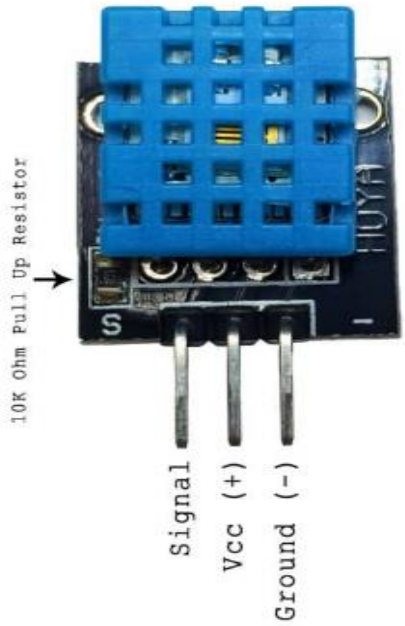
Hardware Required:

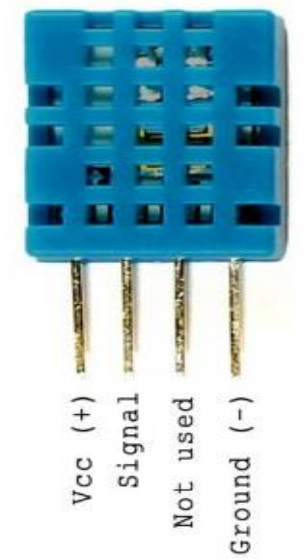
Component Name	Quantity
Arduino UNO	1
Temperature & Humidity sensor DHT11 OR DHT22	1
USB Cable	1

There are two sensors that can measure humidity and temperature.

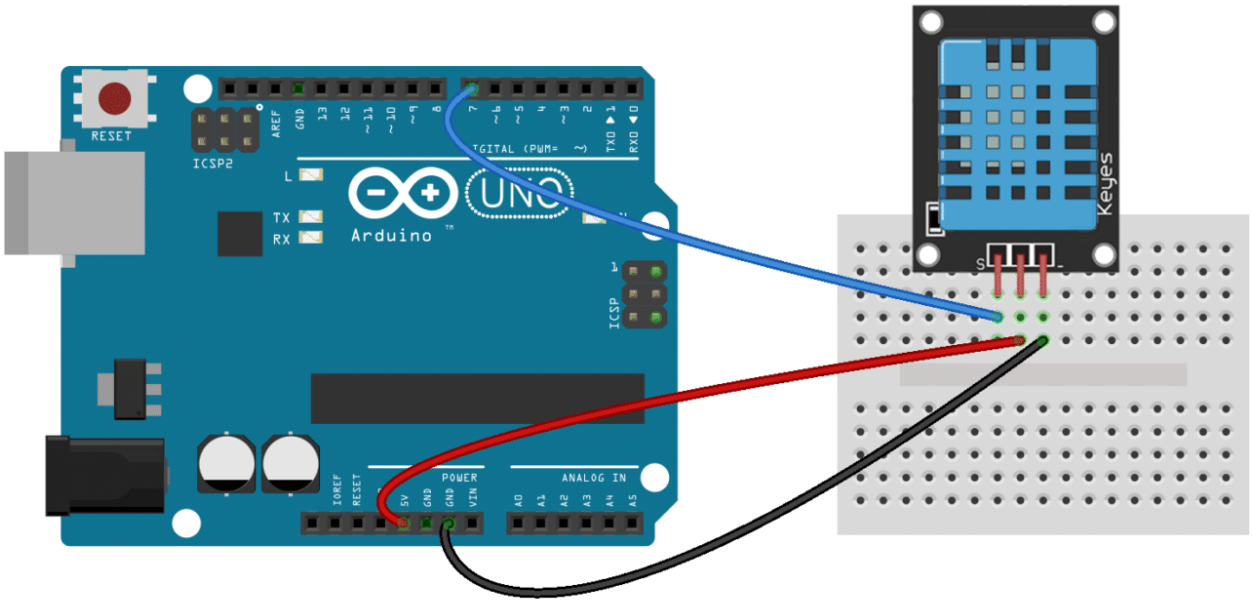
DHT11

There are two different versions of the DHT11 you might come across. One type has four pins, and the other type has three pins and is mounted to a small PCB. The PCB mounted version is nice because it includes a surface mounted 10K Ohm pull up resistor for the signal line. Here are the pin outs for both versions:

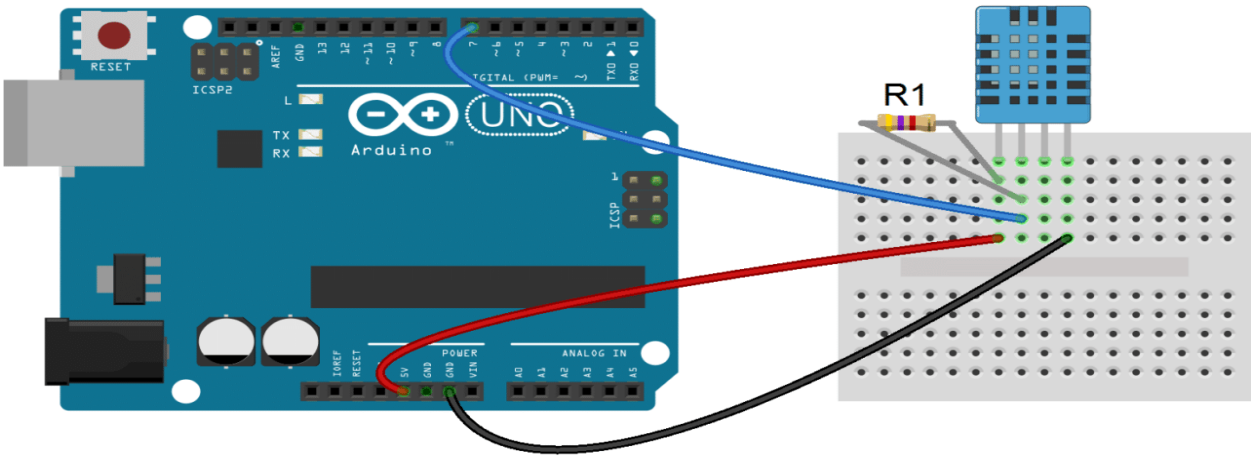




CONNECTING A THREE PIN DHT11:



CONNECTING A FOUR PIN DHT11:



- R1: 10K Ohm pull up resistor

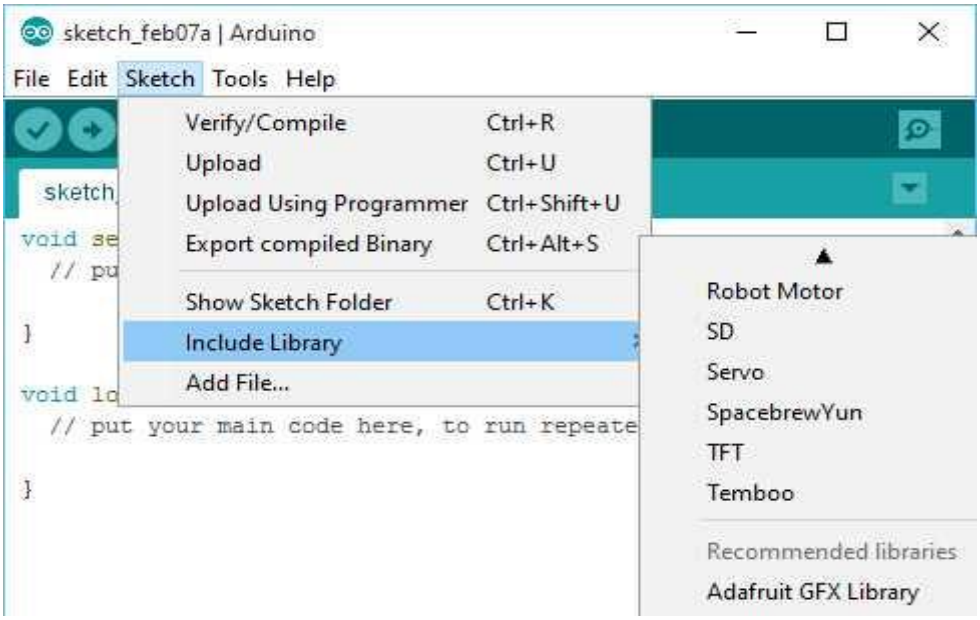
Before you code the DHT11 sensor, you must download library pf DHT11 to your aduino IDE.

Library link:

https://www.circuitbasics.com/wp-content/uploads/2015/10/DHTLib.zip

To include library, you must do the following steps:

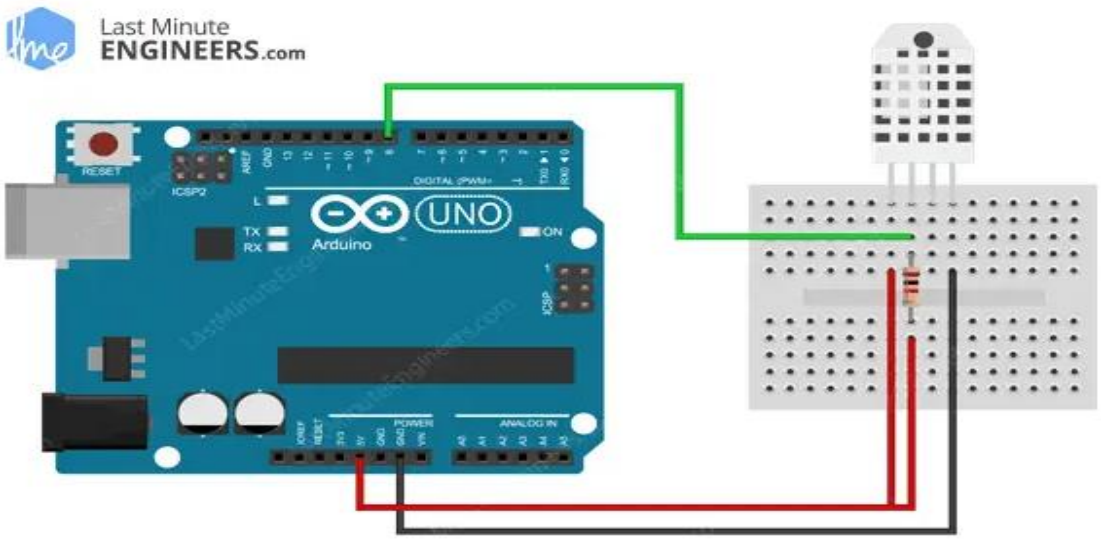
1. In the menu bar, go to Sketch > Include Library > Add .ZIP Library...
2. You will be prompted to select the library you want to add. Navigate to the .zip file's location and open it.
3. If you're using Arduino IDE 2, you may need to restart it for the library to be available.



Code:

```
#include <dht.h>

dht DHT;

#define DHT11_PIN 7

void setup(){

  Serial.begin(9600);

}


void loop(){

  int chk = DHT.read11(DHT11_PIN);

  Serial.print("Temperature = ");

  Serial.println(DHT.temperature);

  Serial.print("Humidity = ");

  Serial.println(DHT.humidity);

  delay(1000);

}
```

DHT22



- R1: 10K Ohm pull up resistor

You must include the same library in the previous code

Code

```
#include <dht.h>
#define dataPin 8 // Defines pin number to which the sensor is connected
dht DHT; // Creats a DHT object

void setup()
{
        Serial.begin(9600);
}
void loop()
{
        //Uncomment whatever type you're using!
        int readData = DHT.read22(dataPin); // DHT22/AM2302
        float t = DHT.temperature; // Gets the values of the temperature
        float h = DHT.humidity; // Gets the values of the humidity

        // Printing the results on the serial monitor
        Serial.print("Temperature = ");
        Serial.print(t);
        Serial.print(" ");
        Serial.print((char)176);//shows degrees character
        Serial.print("C | ");
        Serial.print((t * 9.0) / 5.0 + 32.0);//print the temperature in
Fahrenheit
        Serial.print(" ");
        Serial.print((char)176);//shows degrees character
        Serial.println("F ");
        Serial.print("Humidity = ");
        Serial.print(h);
        Serial.println(" % ");
        Serial.println("");

        delay(2000); // Delays 2 secods
}
```

What are the Differences: DHT11 vs DHT22?

Temperature range: With respect to the temperature range, for DHT11, it falls within -20 – 60℃ while for DHT22 it falls within -40 – 80℃.

Temperature accuracy: DHT11 has a temperature accuracy of ±2%, while that of DHT22 is ±0.5℃

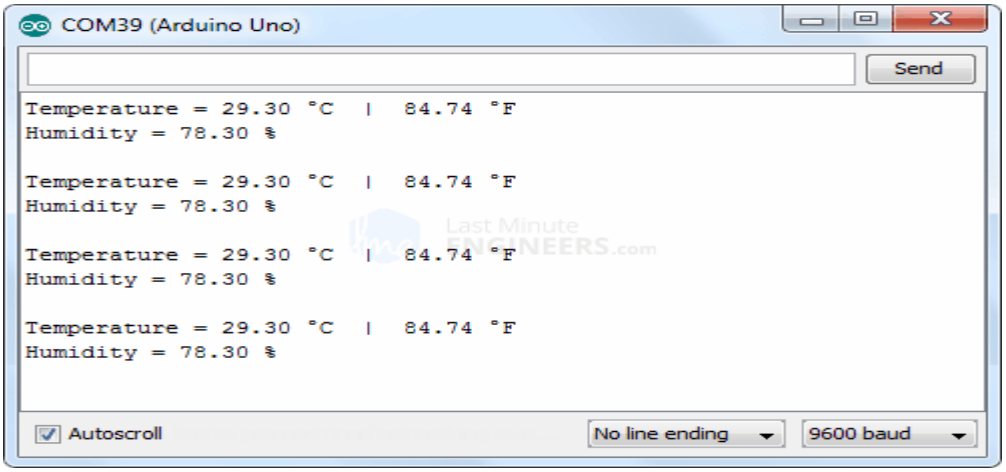
Humidity Range: the humidity range for DHT11 falls between 5 – 95% RH, while that of DHT22 falls within 0 – 100%RH.

Humidity Accuracy: DHT11 has a humidity accuracy of ±5%, in contrast to DHT22, which is ±2%.

Cost: The cost of DHT11 is $5.90 compared to that of DHT22, which is $9.90.

In conclusion, in all aspects, the DHT22 beats the DHT11. This includes humidity accuracy, humidity range temperature accuracy, and temperature range. DHT22 has just one downside, which is its higher price compared to that of DHT11. However, this is necessary, since you have to pay more to get the better deal.

Sample output



Comparison between IR and PIR

PIR stands for Passive Infrared Sensor, which is a sensor that uses infrared technology to remember the infrared image of the surrounding area and notices any changes, which would be caused by motion.

A passive infrared sensor, also known as a PIR sensor, is used in automatic doors or gates to detect when to open or close them. These sensors use infrared technology to detect a change in the energy around the surrounding area.

An IR sensor is an Infrared Sensor, which is a sensor that uses infrared technology, chips and a transmitter to determine whether the light that the transmitter is emitting is from an object or a person. This is also good for security purposes.

Infrared beam motion sensors are another type of sensor that uses infrared technology. They send out of a beam of light which can detect when someone or

something moves past it. You may not be able to see the light as you pass it, as the further away you are from it, the fainter it becomes.

These types of sensors are brilliant for security purposes, as they can pick up motion from the ground and upwards, as long as you position them in the right way. They can be beneficial to use for automatic doors and gates if necessary, as when the beam senses motion, this will send a signal to the doors, causing them to open.

Automatic doors use infrared through the sensors used to open and close the doors. When the passive infrared sensors detect a change in the temperature, which is caused by the human body, the sensors will send a signal to the automatic doors to open.

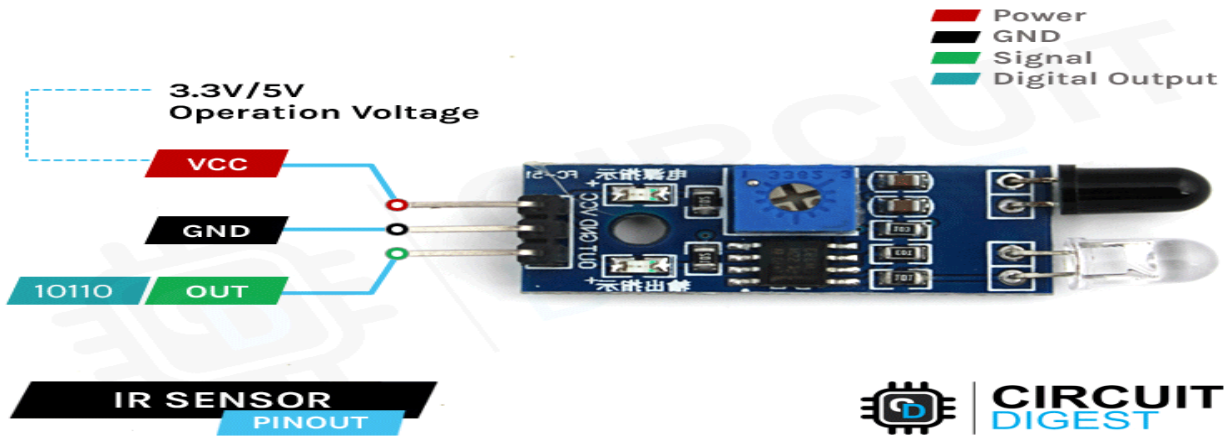
Their main differences include:

- The IR sensors detect whether the light from the transmitter is emitted by an object or a person. Whereas, the PIR sensors detect changes in the levels of energy around the area.
- If the PIR sensor is installed towards the changing environment, it may let off a false alarm due to the change in the way the air flows; however, IR sensors must be installed facing the motion.
- IR sensors are usually only installed on the outside of properties, whereas PIR sensors are usually installed inside.
- PIR sensors don't actually emit the infrared; objects give the sensor infrared rays. This differs from IR sensors, which do actually infrared.

Experiment 9: IR sensor with Arduino

Objective: To detected motion in room using IR sensor.

The IR sensor has a 3-pin connector that interfaces it to the outside world. The connections are as follows:

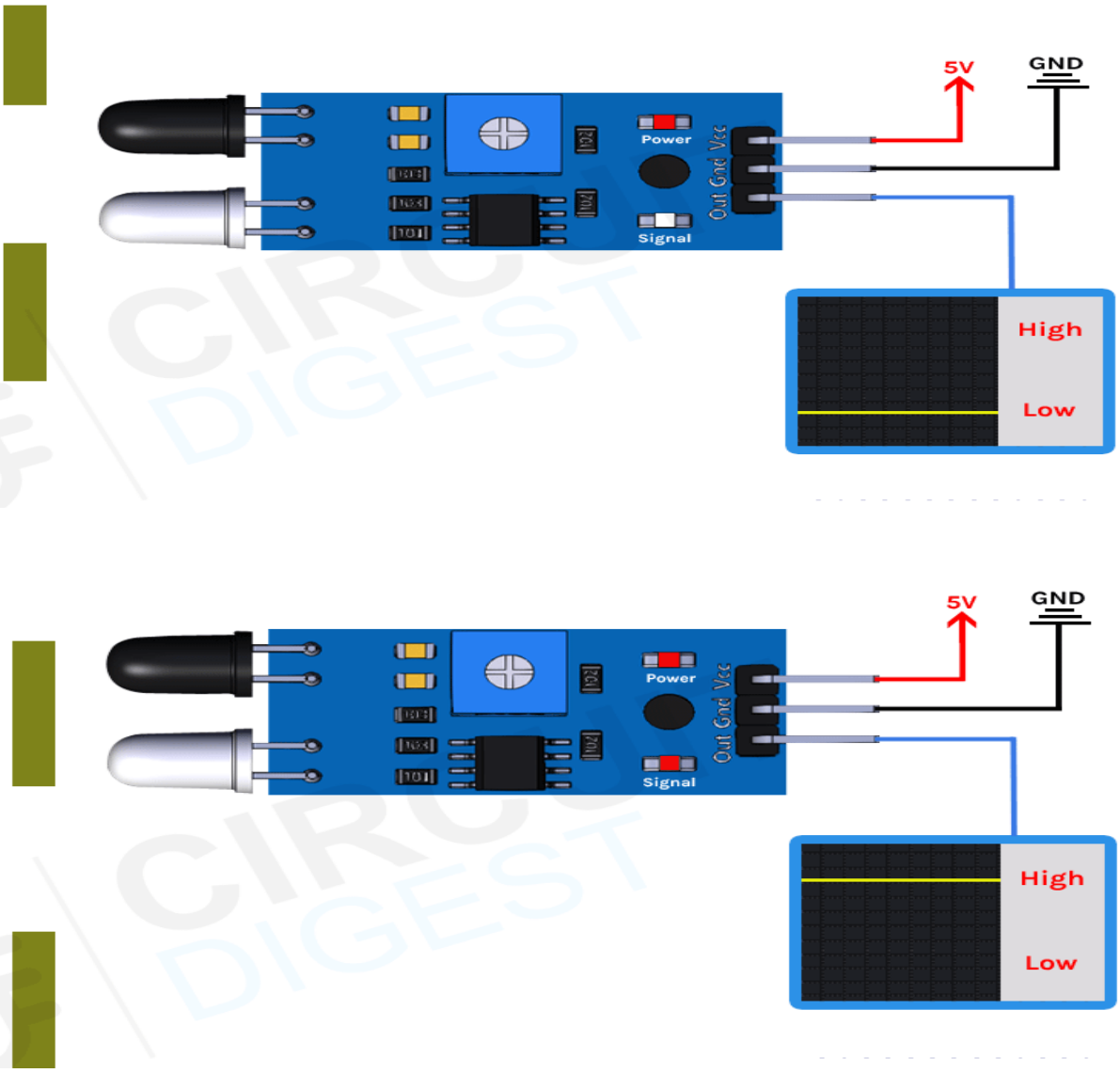


VCC is the power supply pin for the IR sensor which we connect to the 5V pin on the Arduino.

OUT pin is a 5V TTL logic output. LOW indicates no motion is detected; HIGH means motion is detected.

GND Should be connected to the ground of the Arduino.

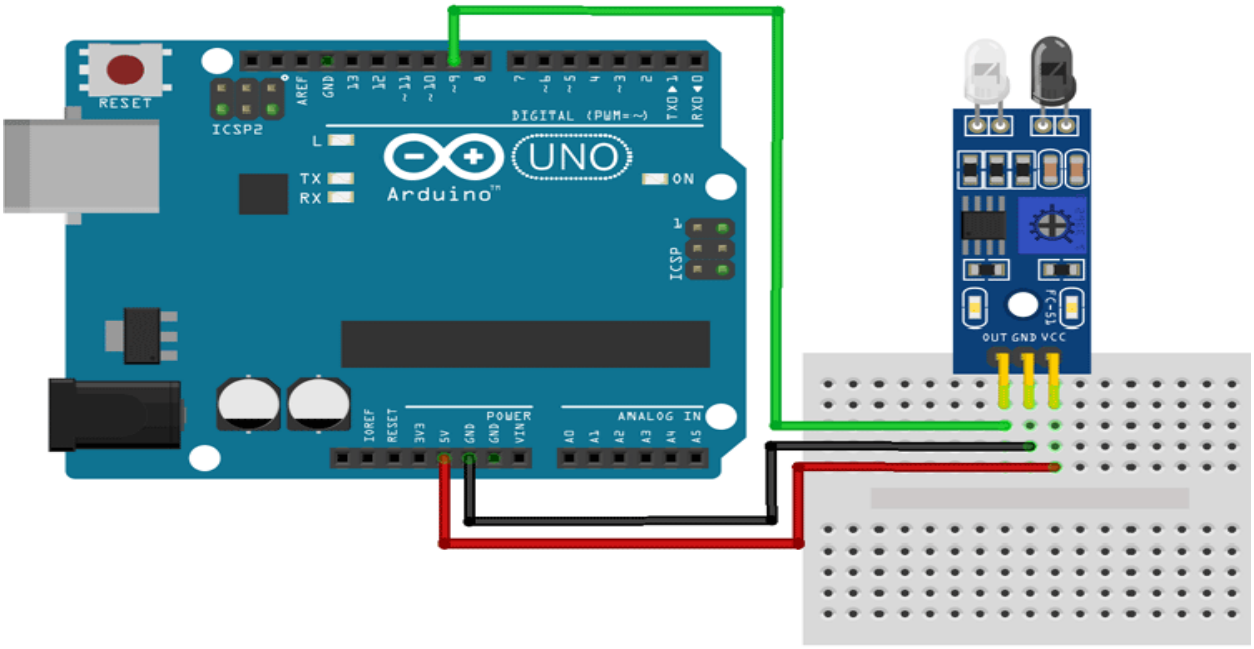
The **working of the IR sensor module** is very simple, it consists of two main components: the first is the IR transmitter section and the second is the IR receiver section. In the transmitter section, **IR led** is used and in the receiver section, a **photodiode** is used to receive infrared signal and after some signal processing and conditioning, you will get the output.



An IR proximity sensor works by applying a voltage to the onboard **Infrared Light Emitting Diode** which in turn emits infrared light. This light propagates through the

air and hits an object, after that the light gets reflected in the photodiode sensor. If the object is close, the reflected light will be stronger, if the object is far away, the reflected light will be weaker. If you look closely toward the module. When the sensor becomes active it sends a corresponding **Low signal** through the output pin that can be sensed by an Arduino or any kind of microcontroller to execute a particular task. The one cool thing about this module is that it has two onboard LEDs built-in, one of which lights on when power is available and another one turns on when the circuit gets triggered.

Hardware Circuit



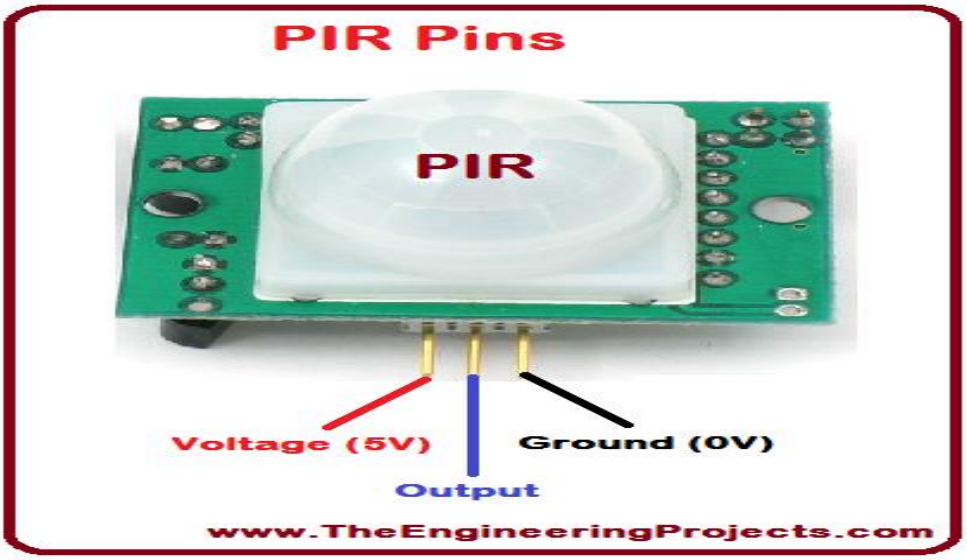
Code

```
int IRSensor = 9; // connect IR sensor module to Arduino pin D9
int LED = 13; // connect LED to Arduino pin 13
void setup(){
  Serial.begin(115200); // Init Serial at 115200 Baud Rate.
  Serial.println("Serial Working"); // Test to check if serial is working or not
  pinMode(IRSensor, INPUT); // IR Sensor pin INPUT
  pinMode(LED, OUTPUT); // LED Pin Output
}
void loop(){
  int sensorStatus = digitalRead(IRSensor); // Set the GPIO as Input
  if (sensorStatus == 1) // Check if the pin high or not
  {
    // if the pin is high turn off the onboard Led
```

```
    digitalWrite(LED, LOW); // LED LOW
   Serial.println("Motion Detected!");
// print Motion Detected! on the serial monitor window
  }
  else  {
    //else turn on the onboard LED
    digitalWrite(LED, HIGH); // LED High
    Serial.println("Motion Ended!");
// print Motion Ended! on the serial monitor window
  }
}
```

Experiment 10: Turn on led when PIR sensor detect motion

Objective: In this project you're going to create a simple circuit with an Arduino and PIR motion sensor that can detect movement. An LED will light up when movement is detected.



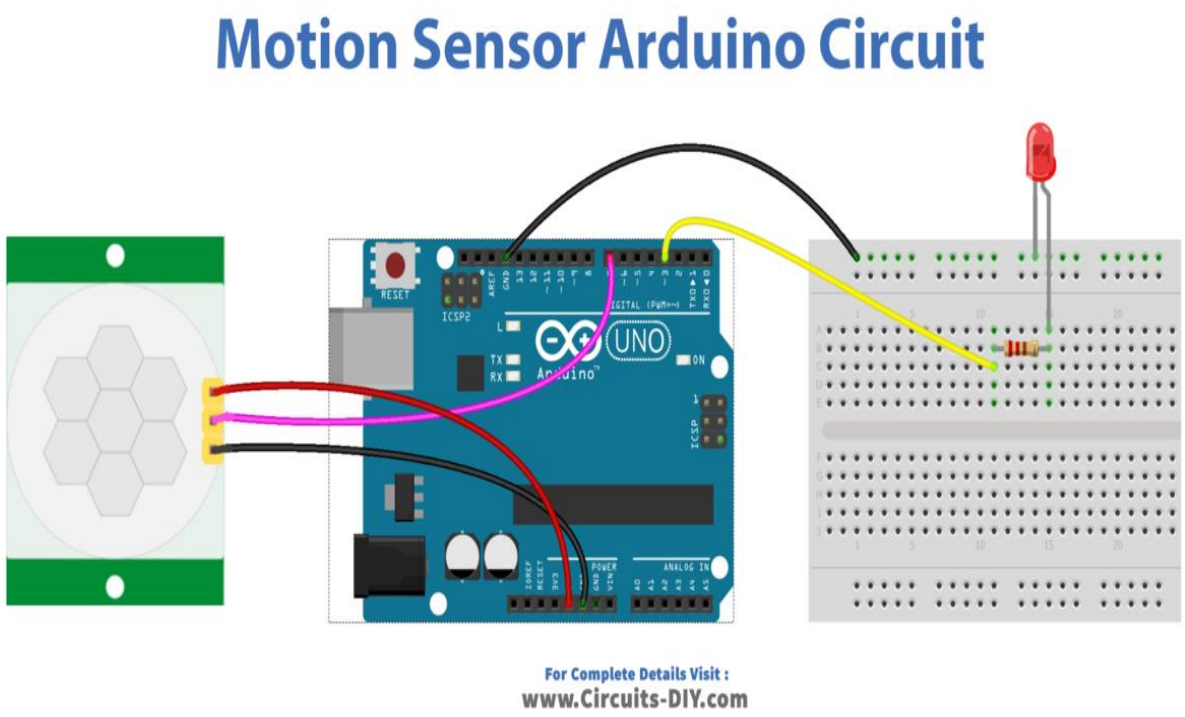
PIR Pins Description	
Pin Name	**Description**
Vcc	Voltage supply (5V)
OUT	Output pin for reading PIR data
GND	Ground (0V)

www.TheEngineeringProjects.com

Hardware component

S.no	Component	Value	Qty
1.	Arduino UNO	–	1
2.	USB Cable	–	1
3.	Motion Sensor	HC-SR501	1
4.	LED	–	1
5.	Resistor	220Ω	1
6.	Breadboard	–	1
7.	Jumper Wires	–	1

Hardware circuit



Code

```
Const int MOTION_SENSOR_PIN = 7;   // Arduino pin connected to the OUTPUT pin of motion sensor
const int LED_PIN          = 3;   // Arduino pin connected to LED's pin
int motionStateCurrent     = LOW; // current  state of motion sensor's pin
int motionStatePrevious    = LOW; // previous state of motion sensor's pin
```

```
void setup() {
  Serial.begin(9600);              // initialize serial
  pinMode(MOTION_SENSOR_PIN, INPUT); // set arduino pin to input mode
  pinMode(LED_PIN, OUTPUT);        // set arduino pin to output mode
}
void loop() {
  motionStatePrevious = motionStateCurrent;            // store old state
  motionStateCurrent  = digitalRead(MOTION_SENSOR_PIN); // read new state
  if (motionStatePrevious == LOW && motionStateCurrent == HIGH) { // pin state change: LOW -> HIGH
    Serial.println("Motion detected!");
    digitalWrite(LED_PIN, HIGH); // turn on
  }
  else
  if (motionStatePrevious == HIGH && motionStateCurrent == LOW) { // pin state change: HIGH -> LOW
    Serial.println("Motion stopped!");
    digitalWrite(LED_PIN, LOW);  // turn off
  }
}
```

Applications

- Security systems
- Home automation
- Presence detection
- Outdoor Lighting
- Automated guided vehicles

Experiment 11: ultrasonic sensor with Arduino

Objective: measure distance using SR04 Ultrasonic Sensor and Arduino Uno board.

An HC-SR04 ultrasonic distance sensor actually consists of two ultrasonic transducers. One acts as a transmitter that converts the electrical signal into 40 KHz

ultrasonic sound pulses. The other acts as a receiver and listens for the transmitted pulses. When the receiver receives these pulses, it produces an output pulse whose width is proportional to the distance of the object in front. This sensor provides excellent non-contact range detection between 2 cm to 400 cm (~13 feet) with an accuracy of 3 mm.

VCC supplies power to the HC-SR04 ultrasonic sensor. You can connect it to the 5V output from your Arduino.

Trig (Trigger) pin is used to trigger ultrasonic sound pulses. By setting this pin to HIGH for 10µs, the sensor initiates an ultrasonic burst.

Echo pin goes high when the ultrasonic burst is transmitted and remains high until the sensor receives an echo, after which it goes low. By measuring the time the Echo pin stays high, the distance can be calculated.

GND is the ground pin. Connect it to the ground of the Arduino.

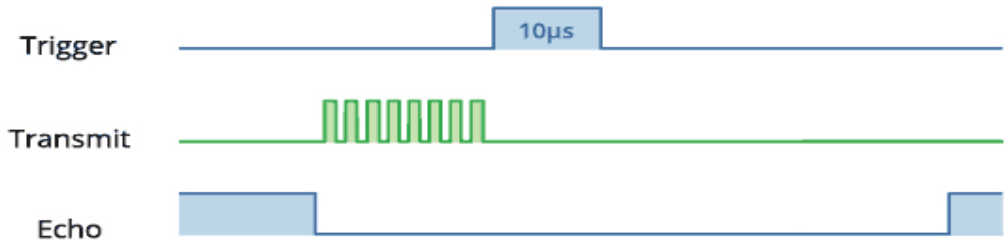
HC-SR04 Sensor		Arduino
VCC		5V
Trig		9
Echo		10
GND		GND

How Does HC-SR04 Ultrasonic Distance Sensor Work?

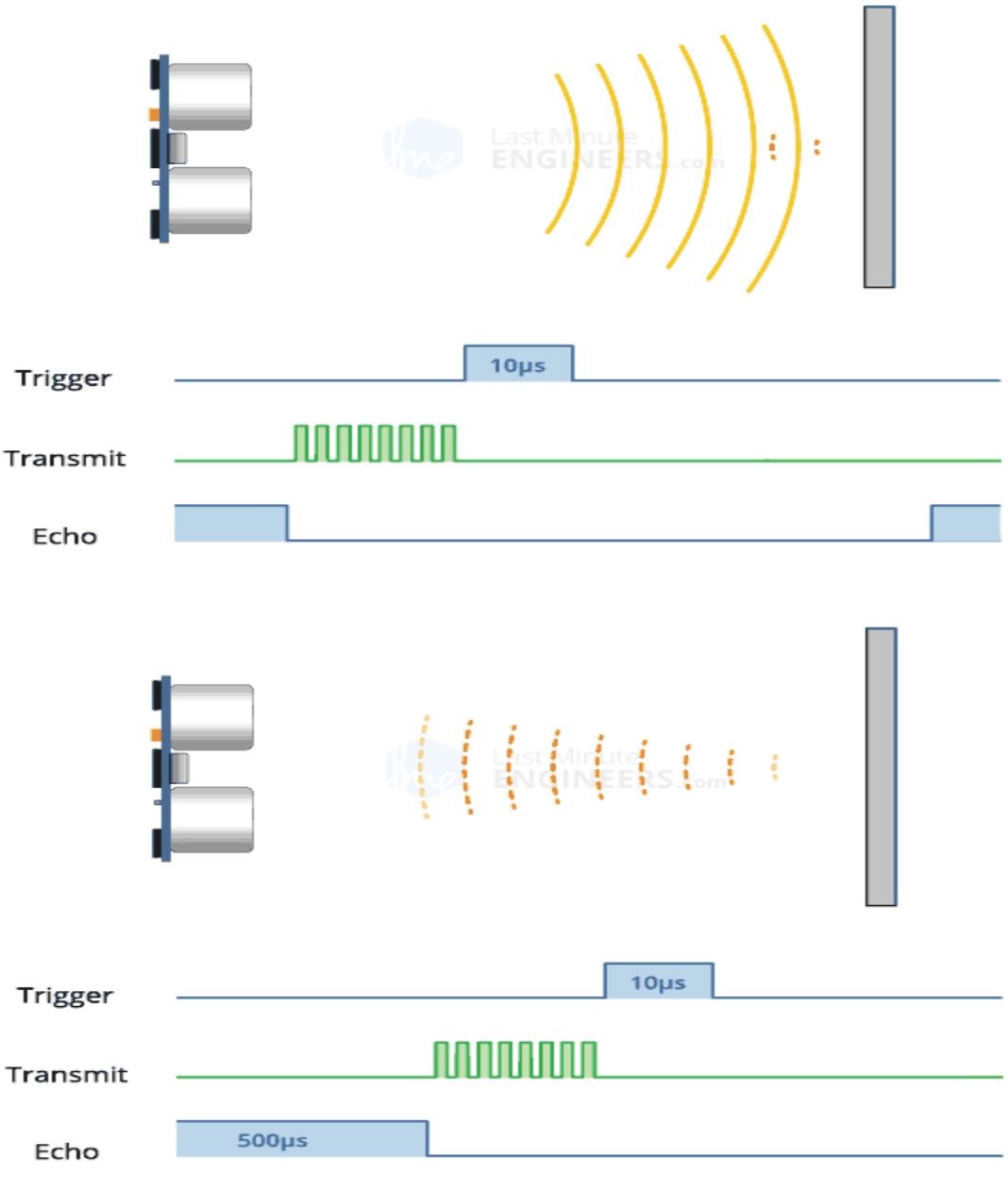
It all starts when the trigger pin is set HIGH for 10µs. In response, the sensor transmits an ultrasonic burst of eight pulses at 40 kHz. This 8-pulse pattern is specially designed so that the receiver can distinguish the transmitted pulses from ambient ultrasonic noise.

These eight ultrasonic pulses travel through the air away from the transmitter. Meanwhile the echo pin goes HIGH to initiate the echo-back signal.

If those pulses are not reflected back, the echo signal times out and goes low after 38ms (38 milliseconds). Thus a pulse of 38ms indicates no obstruction within the range of the sensor.



If those pulses are reflected back, the echo pin goes low as soon as the signal is received. This generates a pulse on the echo pin whose width varies from 150 µs to 25 ms depending on the time taken to receive the signal.



Calculating the Distance

Distance = Speed x Time

Let us take an example to make it more clear. Suppose we have an object in front of the sensor at an unknown distance and we receive a pulse of 500µs width on the echo pin. Now let's calculate how far the object is from the sensor. For this we will use the below equation.

Distance = Speed x Time

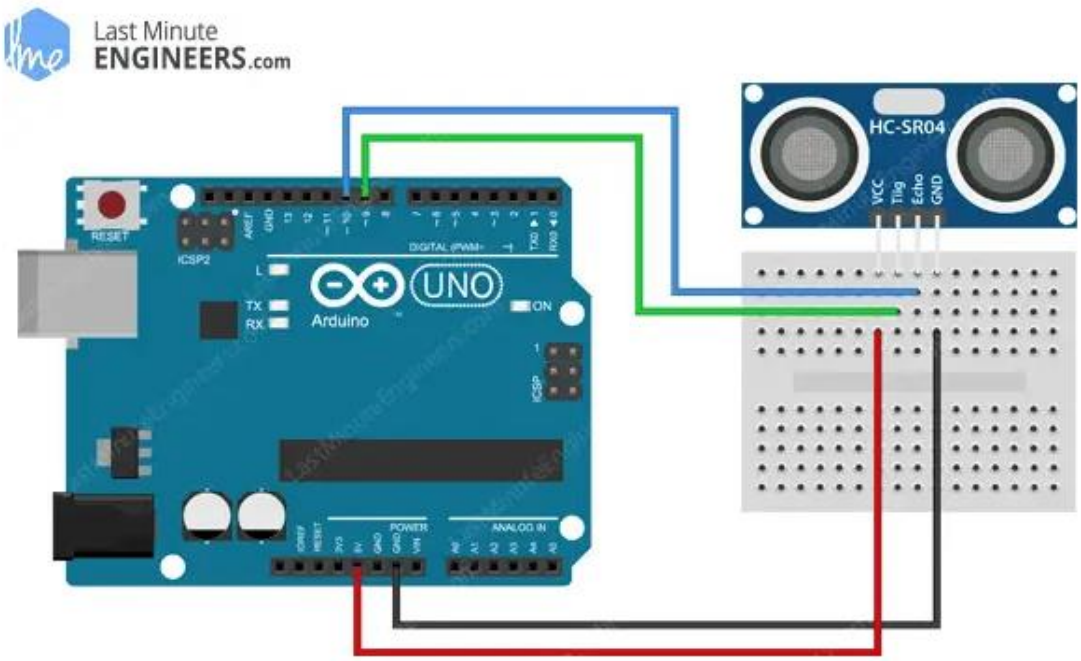
Here we have the value of time i.e. 500 µs and we know the speed. Of course it's the speed of sound! It is 340 m/s. To calculate the distance we need to convert the speed of sound into cm/µs. It is 0.034 cm/µs. With that information we can now calculate the distance!

Distance = 0.034 cm/µs x 500 µs

But we're not done yet! Remember that the echo pulse indicates the time it takes for the signal to be sent and reflected back. So to get the distance, you have to divide your result by two.

Distance = (0.034 cm/µs x 500 µs) / 2

Hardware circuit



Library Installation

Triggering the ultrasonic sensor and measuring the received signal pulse width manually is a lot of work but luckily there are many libraries available to us. One of the popular libraries is the NewPing library. This is the library we will use in our examples.

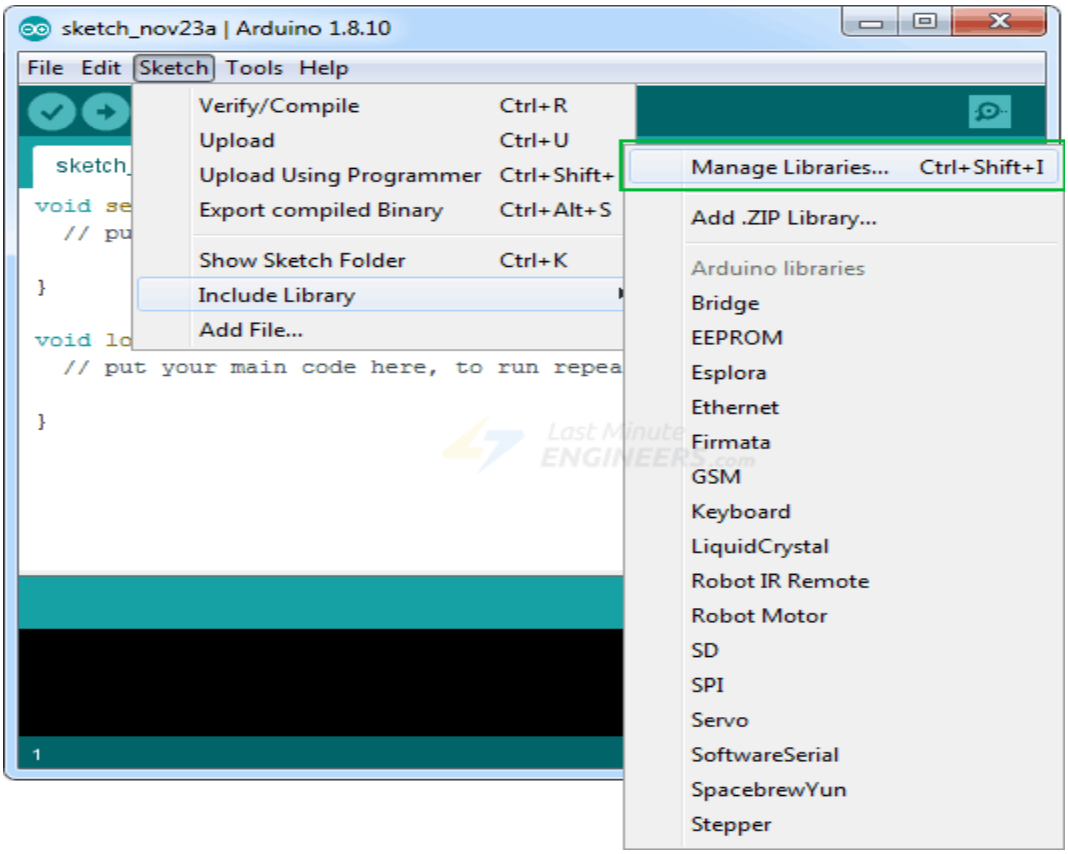
https://www.arduino.cc/reference/en/libraries/newping/

The NewPing library is quite advanced. It supports up to 15 ultrasonic sensors at once and can output directly in centimeters, inches, or time periods.

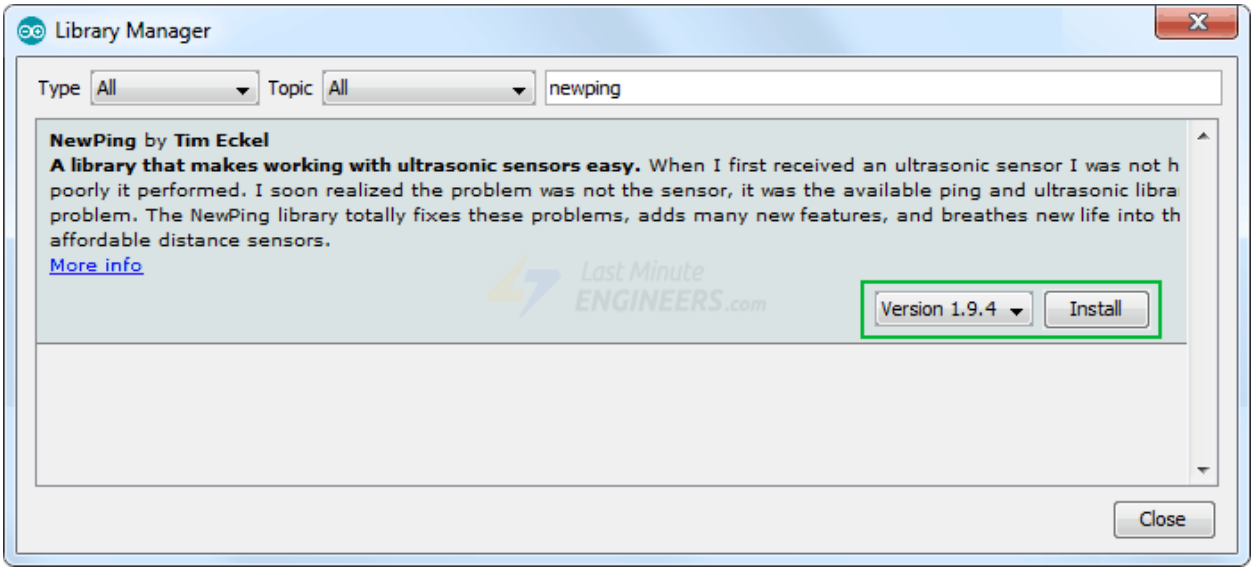
This library is not included in the Arduino IDE, so you will need to install it first.

To include a new library, you can download the library from the pervious link or use the next new way to add new library.

To install the library navigate to Sketch > Include Libraries > Manage Libraries… Wait for Library Manager to download the library index and update the list of installed libraries.



Filter your search by typing 'newping'. Click on the first entry and then select Install.



Code

You can measure distance using ultrasonic by two ways:

First way: depend on the equation that we discuss it to you

```
const int trigPin = 9;
const int echoPin = 10;
// defines variables
long duration;
int distance;
void setup() {
  pinMode(trigPin, OUTPUT); // Sets the trigPin as an Output
  pinMode(echoPin, INPUT); // Sets the echoPin as an Input
  Serial.begin(9600); // Starts the serial communication
}
void loop() {
  // Clears the trigPin
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  // Sets the trigPin on HIGH state for 10 micro seconds
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  // Reads the echoPin, returns the sound wave travel time in microseconds
  duration = pulseIn(echoPin, HIGH);
  // Calculating the distance
  distance = duration * 0.034 / 2;
  // Prints the distance on the Serial Monitor
  Serial.print("Distance: ");
  Serial.println(distance);
}
```

Second way: depend on the new library added

```
#include "NewPing.h"
// Hook up HC-SR04 with Trig to Arduino Pin 9, Echo to Arduino pin 10
#define TRIGGER_PIN 9
#define ECHO_PIN 10
// Maximum distance we want to ping for (in centimeters).
#define MAX_DISTANCE 400
// NewPing setup of pins and maximum distance.
NewPing sonar(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE);
void setup() {
        Serial.begin(9600);
}
```

```
void loop() {
        Serial.print("Distance = ");
        Serial.print(sonar.ping_cm());
        Serial.println(" cm");
        delay(500);
}
```

Experiment 12: how the Soil Moisture Sensor Works and Interface it with Arduino

Objective: To measure the moisture present in the soil.

When you hear the term “smart garden,” one of the first things that comes to mind is a system that monitors the moisture level of the soil and automatically supplies the necessary amount of water to the plants. With this system, plants can be watered only when required, avoiding over- or under-watering. If you want to build such a system, you will undoubtedly require a Soil Moisture Sensor.

How Does a Soil Moisture Sensor Work?

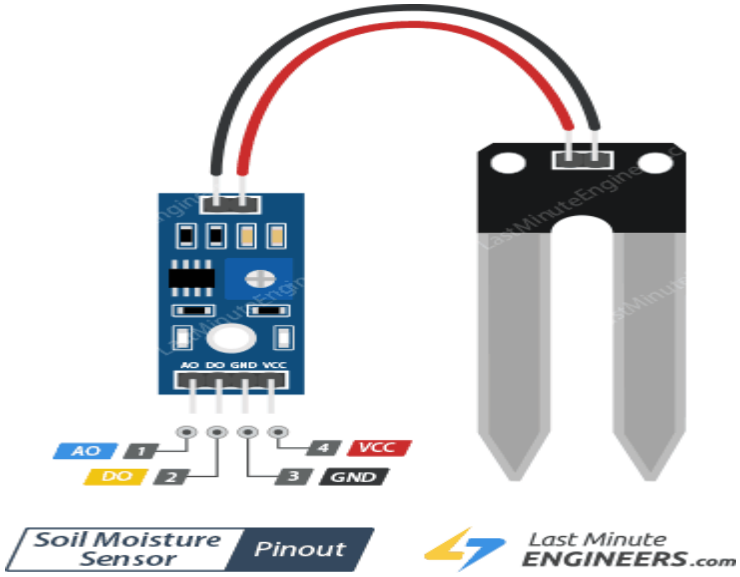
The soil moisture sensor operates in a straightforward manner.

The fork-shaped probe with two exposed conductors acts as a variable resistor (similar to a potentiometer) whose resistance varies with the soil’s moisture content. This resistance varies inversely with soil moisture:

- The more water in the soil, the better the conductivity and the lower the resistance.
- The less water in the soil, the lower the conductivity and thus the higher the resistance.

The sensor produces an output voltage according to the resistance, which by measuring we can determine the soil moisture level.

The soil moisture sensor is extremely simple to use and only requires four pins to connect.



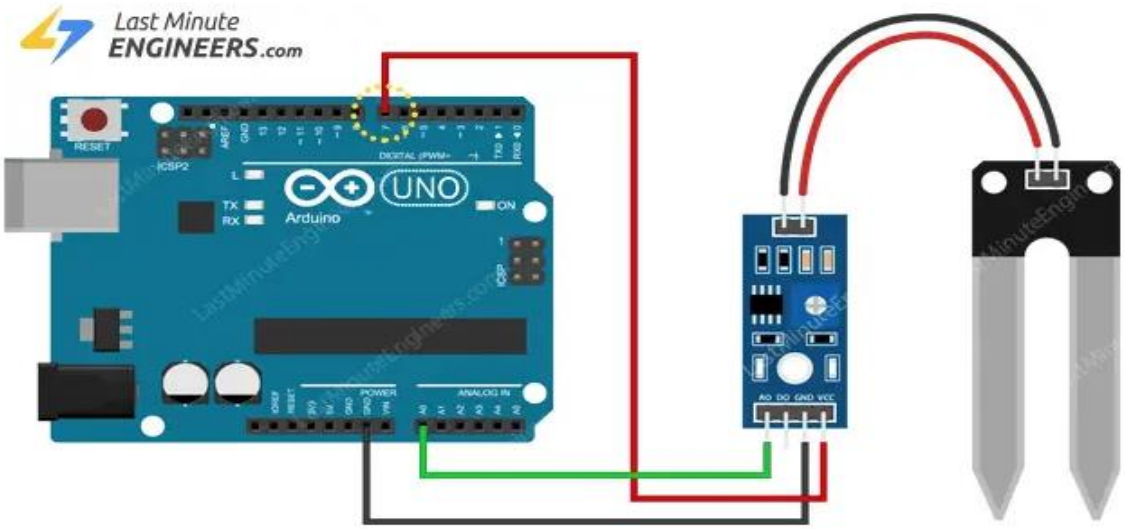
AO (Analog Output) generates analog output voltage proportional to the soil moisture level, so a higher level results in a higher voltage and a lower level results in a lower voltage.

DO (Digital Output) indicates whether the soil moisture level is within the limit. D0 becomes LOW when the moisture level exceeds the threshold value (as set by the potentiometer), and HIGH otherwise.

VCC supplies power to the sensor. It is recommended that the sensor be powered from 3.3V to 5V. Please keep in mind that the analog output will vary depending on the voltage supplied to the sensor.

GND is the ground pin.

Hardware circuit



Code

```
#define sensorPower 7
#define sensorPin A0

void setup() {
	pinMode(sensorPower, OUTPUT);

	// Initially keep the sensor OFF
	digitalWrite(sensorPower, LOW);

	Serial.begin(9600);
}

void loop() {
	//get the reading from the function below and print it
	Serial.print("Analog output: ");
	Serial.println(readSensor());

	delay(1000);
}

//  This function returns the analog soil moisture measurement
int readSensor() {
	digitalWrite(sensorPower, HIGH);	// Turn the sensor ON
	delay(10);				// Allow power to settle
	int val = analogRead(sensorPin); // Read the analog value form sensor
	digitalWrite(sensorPower, LOW);		// Turn the sensor OFF
```

```
      return val;          //Return analog moisture value
}
```

When you run the sketch, you should see readings similar to the ones below:

- When the soil is dry (around 850)
- When the soil is completely saturated (around 400)

Status: Dry
Test Reading: ~850

Status: Completely wet
Test Reading: ~400

Now, we will write a new code that estimates the level of soil moisture using the following threshold values:

- < 500 is too wet
- 500-750 is the target range
- > 750 is dry enough to be watered

Code

```
/* Change these values based on your calibration values */
#define soilWet 500   // Define max value we consider soil 'wet'
#define soilDry 750   // Define min value we consider soil 'dry'
// Sensor pins
#define sensorPower 7
#define sensorPin A0
void setup() {
```

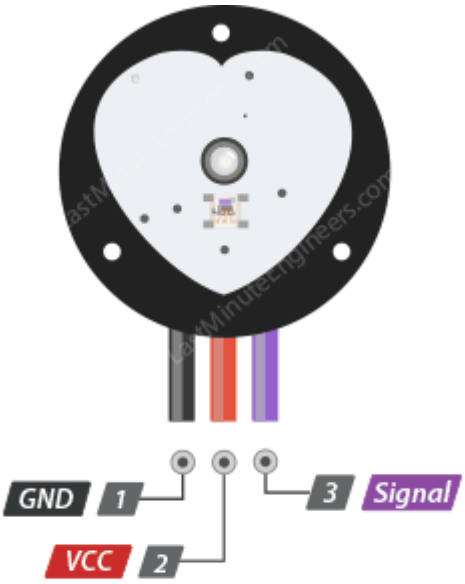
```
        pinMode(sensorPower, OUTPUT);
        // Initially keep the sensor OFF
        digitalWrite(sensorPower, LOW);
        Serial.begin(9600);
}

void loop() {
        //get the reading from the function below and print it
        int moisture = readSensor();
        Serial.print("Analog Output: ");
        Serial.println(moisture);
        // Determine status of our soil
        if (moisture < soilWet) {
                Serial.println("Status: Soil is too wet");
        } else if (moisture >= soilWet && moisture < soilDry) {
                Serial.println("Status: Soil moisture is perfect");
        } else {
                Serial.println("Status: Soil is too dry - time to water!");
        }

        delay(1000);   // Take a reading every second for testing
        // Normally you should take reading perhaps once or twice a day
        Serial.println();
}

//  This function returns the analog soil moisture measurement
int readSensor() {
        digitalWrite(sensorPower, HIGH);      // Turn the sensor ON
        delay(10);             // Allow power to settle
        int val = analogRead(sensorPin);// Read the analog value form sensor
        digitalWrite(sensorPower, LOW);       // Turn the sensor OFF
        return val;            // Return analog moisture value
}
```

Experiment 13 : heart beat sensor with Arduino Uno

Objectives: Monitor the Heart Rate using Pulse Sensor and Arduino

The Pulse Sensor is a well-designed low-power plug-and-play heart-rate sensor for the Arduino. Anyone who wants to incorporate real-time heart-rate data into their work—students, artists, athletes, makers, and game and mobile developers—can benefit from it.

The best part is that this sensor plugs right into Arduino and easily clips onto a fingertip or earlobe. It is also super small (button-shaped) and has holes for sewing into fabric. The front of the sensor, with the heart logo, is where you put your finger. You'll also notice a tiny circular opening through which the Kingbright's reverse mounted green LED shines.



S (Signal) is the signal output. Connects to analog input of an Arduino.

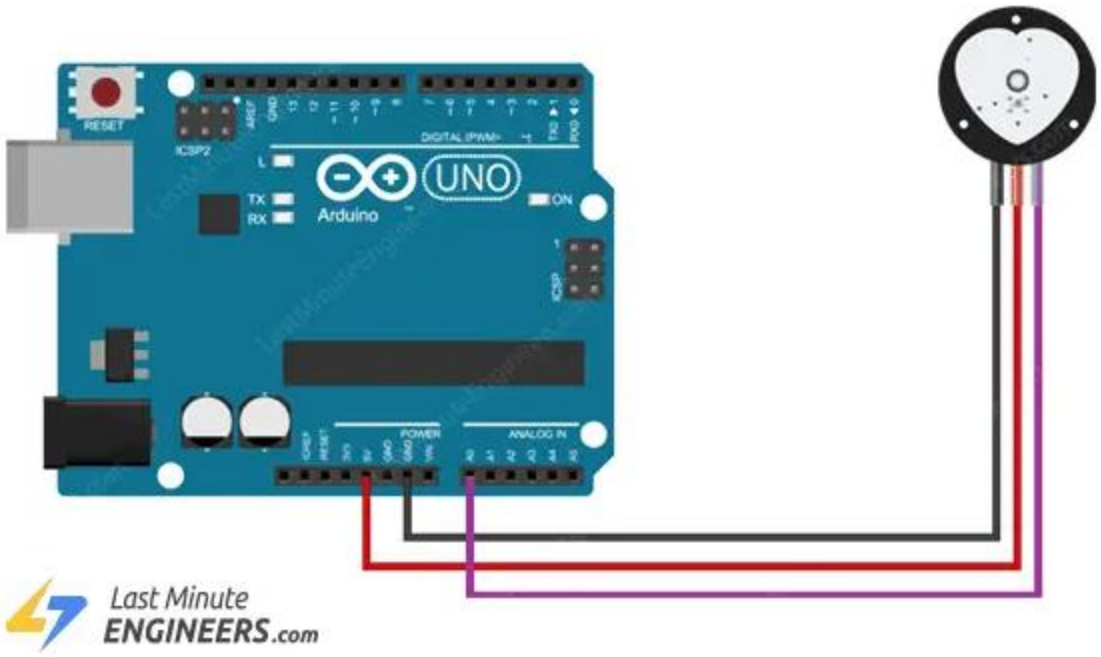
+ (VCC) is the VCC pin. Connects to 3.3 or 5V.

– (GND) is the Ground pin.

Connecting the Pulse Sensor to an Arduino is a breeze. You only need to connect three wires: two for power and one for reading the sensor value.

The module can be supplied with either 3.3V or 5V. Positive voltage is connected to '+,' while ground is connected to '-.' The third 'S' wire is the analog signal output from the sensor, which will be connected to the Arduino's A0 analog input.

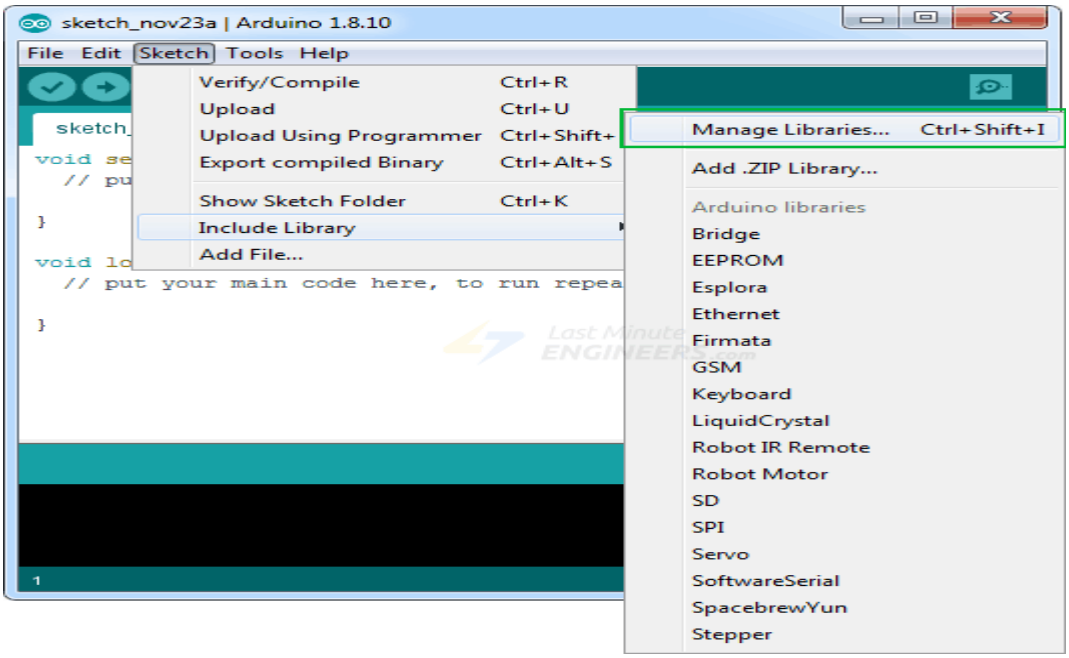
Hardware circuit



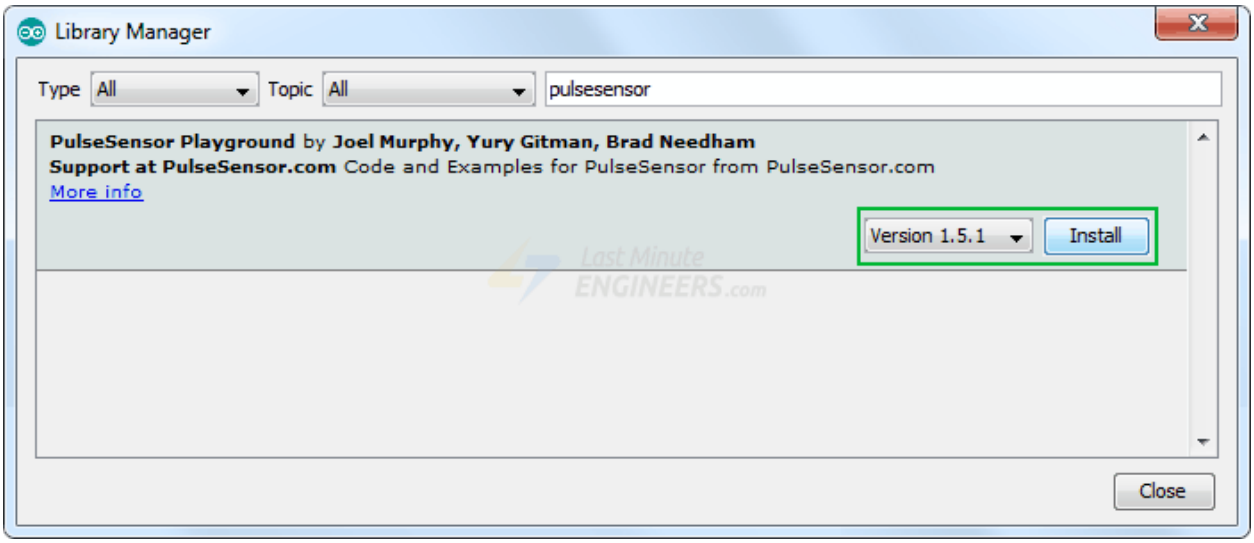
Library Installation

To run the following sketches, you must first install the 'PulseSensor Playground' library.

To install the library, navigate to Sketch > Include Library > Manage Libraries... Wait for the Library Manager to download the libraries index and update the list of installed libraries.



Filter your search by entering 'pulsesensor'.There should only be a single entry. Click on that and then choose Install.



Or you can download it from the following link and included it into the IDE

https://www.arduino.cc/reference/en/libraries/pulsesensor-playground/

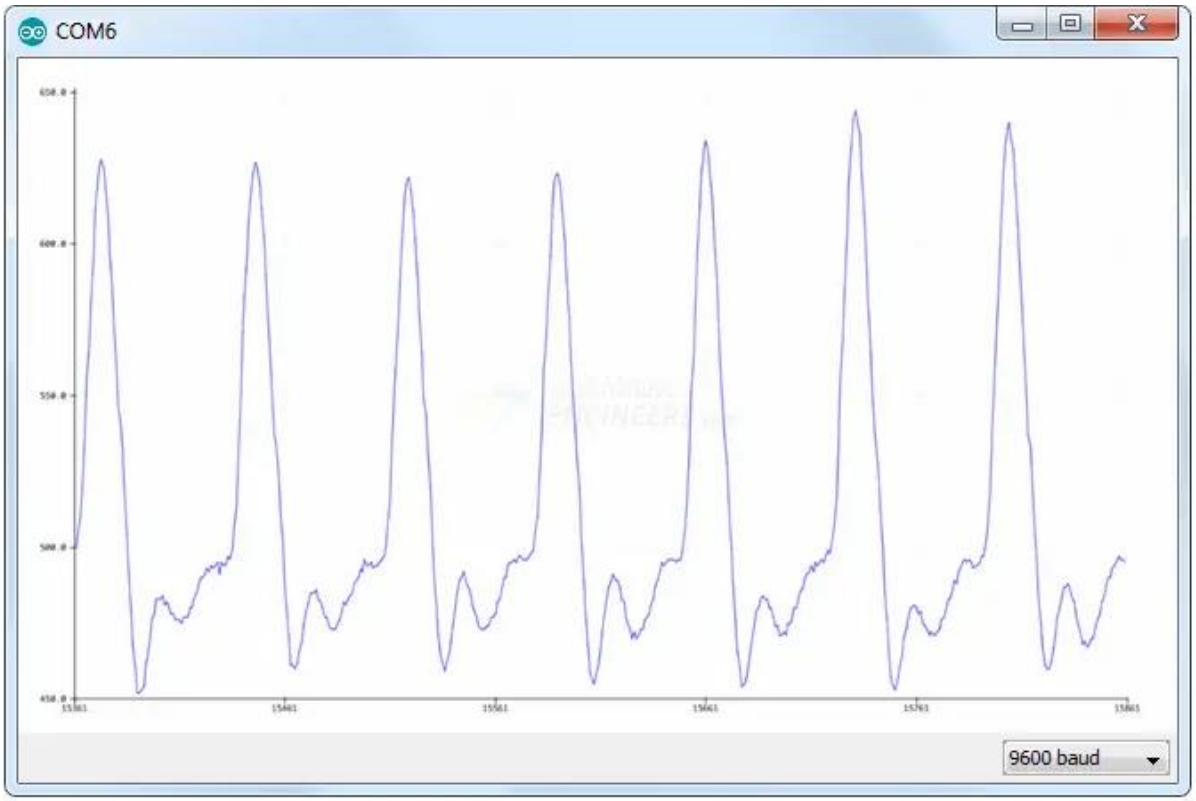
Code

The following code turn on the build in led in the sensor if the heart rate is high and you don't need for the pervious library.

```
int const PULSE_SENSOR_PIN = 0;   // 'S' Signal pin connected to A0
int Signal;          // Store incoming ADC data. Value can range from 0-1024
int Threshold = 550; // Determine which Signal to "count as a beat" and which to
ignore.

void setup() {
        pinMode(LED_BUILTIN,OUTPUT);  // Built-in LED will blink to your heartbeat
        Serial.begin(9600);           // Set comm speed for serial plotter window
}

void loop() {

        Signal = analogRead(PULSE_SENSOR_PIN); // Read the sensor value

        Serial.println(Signal);          // Send the signal value to serial plotter

        if(Signal > Threshold){// If the signal is above threshold, turn on the LED
                digitalWrite(LED_BUILTIN,HIGH);
```

```
        } else {
                digitalWrite(LED_BUILTIN,LOW);    // Else turn off the LED
        }
        delay(10);
}
```

The previous code sketch is designed to work with the Arduino Serial Plotter – a nice tool included with the Arduino IDE for visualizing analog signals in real-time. While the sketch is running and your Arduino board is connected to your computer via USB, navigate to Plotter > Serial Plotter The signal might take a while to stabilize, but once it does, you should see something similar.



Measuring Heart-Rate (BPM)

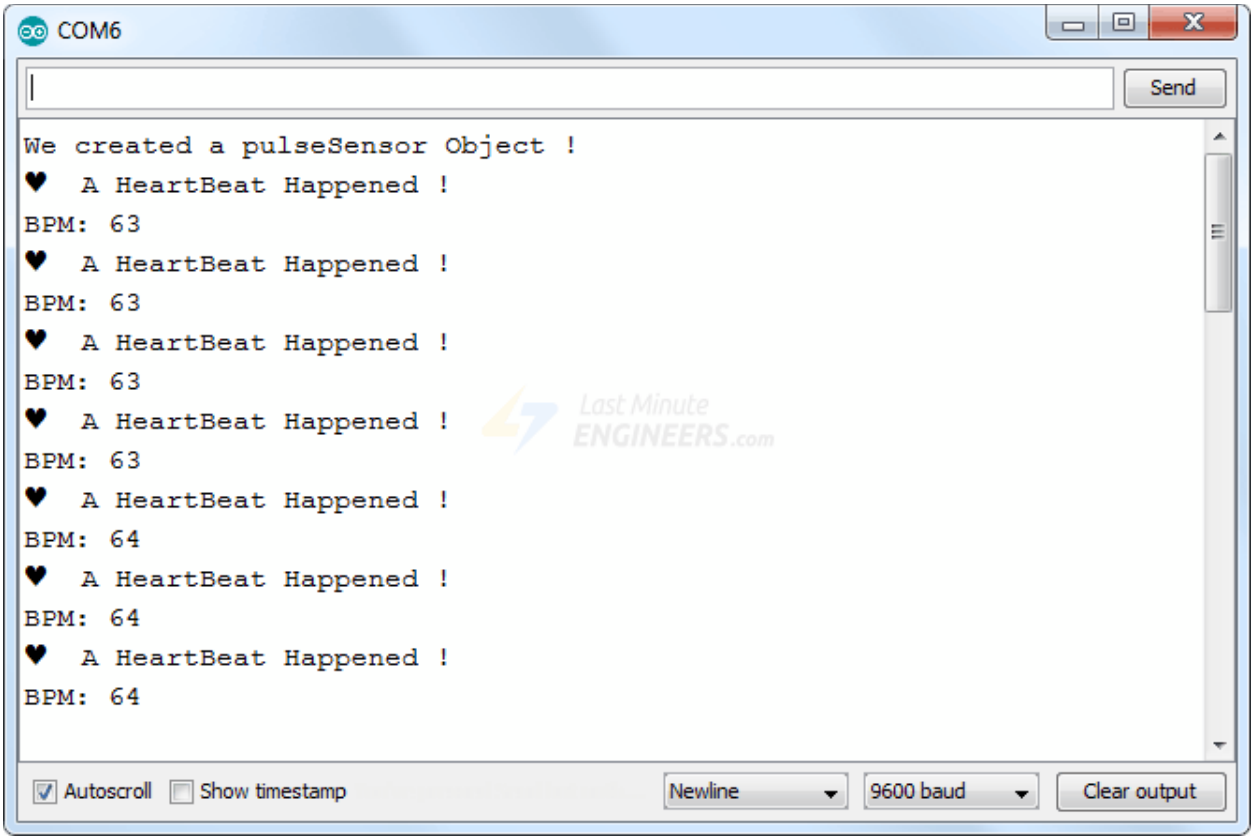
In this example, we will attempt to measure heart rate (Beats Per Minute or BPM). This sketch calculates the time between pulses to determine the heart rate and outputs the result to the Serial Monitor.

Code

```
#define USE_ARDUINO_INTERRUPTS true
// Set-up low-level interrupts for most acurate BPM math
#include <PulseSensorPlayground.h>     // Includes the PulseSensorPlayground Library
```

```
const int PulseWire = 0;       // 'S' Signal pin connected to A0
const int LED13 = 13;          // The on-board Arduino LED
int Threshold = 550;
// Determine which Signal to "count as a beat" and which to ignore
PulseSensorPlayground pulseSensor;  // Creates an object

void setup() {
        Serial.begin(9600);

        // Configure the PulseSensor object, by assigning our variables to it
        pulseSensor.analogInput(PulseWire);
        pulseSensor.blinkOnPulse(LED13);       // Blink on-board LED with heartbeat
        pulseSensor.setThreshold(Threshold);

        // Double-check the "pulseSensor" object was created and began seeing a signal
        if (pulseSensor.begin()) {
                Serial.println("PulseSensor object created!");
        }
}

void loop() {
        int myBPM = pulseSensor.getBeatsPerMinute();      // Calculates BPM

        if (pulseSensor.sawStartOfBeat()) {
// Constantly test to see if a beat happened
                Serial.println("♥  A HeartBeat Happened ! ");
// If true, print a message
                Serial.print("BPM: ");
                Serial.println(myBPM);
// Print the BPM value
                }
        delay(20);
}
```

Sample output



Experiment 14 MQ8 sensor with Arduino uno

Objective: Measure Gas percentage using MQ8 sensor that measure the percentage of Hydrogen.

What is a Gas Sensor

A gas sensor is an electronic device that is used to detect the presence and concentration of specific gases in the air. The most common types of gases that are detected by these sensors include carbon monoxide (CO), hydrogen (H2), methane (CH4), and propane (C3H8).

Gas sensors work by measuring changes in electrical resistance, conductivity, or voltage that are caused by the presence of a target gas. Each type of gas sensor is designed to detect a specific gas, and the output signal of the sensor will vary depending on the concentration of the target gas in the air.

In Arduino projects, gas sensors can be used to monitor indoor air quality, detect gas leaks in homes or buildings, or monitor the levels of specific gases in industrial environments. By connecting the gas sensor to an Arduino board, the output signal from the sensor can be read and used to trigger an alarm or other response in the event of a gas leak or other hazardous situation.

There are many different types of gas sensors available on the market, each with its own strengths and limitations. When selecting a gas sensor for an Arduino project, it is important to consider the type of gas that needs to be detected, the required sensitivity and accuracy, and the operating conditions (such as temperature and humidity) that the sensor will encounter.

What is an MQ8 Hydrogen H2 Gas Sensor?

MQ8 sensor is a sensor having high sensitivity to Hydrogen H2 gas.

The SnO2 semiconductor material is used in the MQ8 sensor for detecting the gases. It has lower conductivity in clean air. When hydrogen gas is present near the sensor, the sensor's conductivity rises with the rise in gas concentration. Users can convert the change of conductivity to gas concentration through a simple electronic circuit. MQ-8 gas sensor has a high sensitivity to hydrogen gas and it also has anti-interference to other gases. This sensor can detect hydrogen, especially city gas.

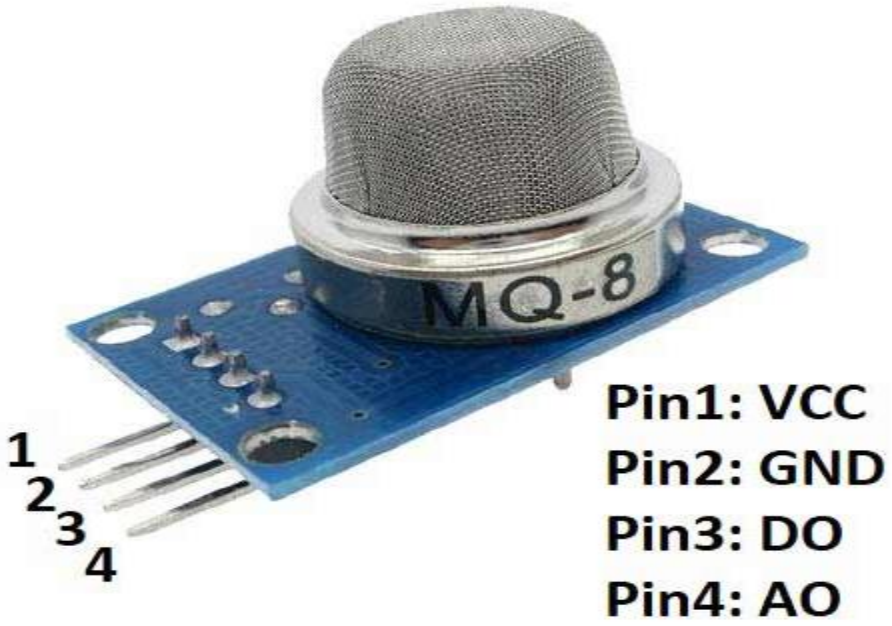
An MQ8 sensor has a wide array of applications such as:

- Domestic and industrial Hydrogen gas leakage detector
- Industrial flammable gas detector
- Portable gas detector

The applicability of an MQ8 sensor is enhanced by the following features:

- High sensitivity to Hydrogen (H2)
- Small sensitivity to alcohol, LPG, cooking fumes
- Highly Stable
- Long lifespan
- Low cost
- Simple drive circuit

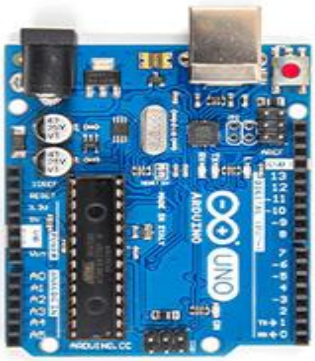
MQ8 Hydrogen H2 Gas Sensor pin diagram

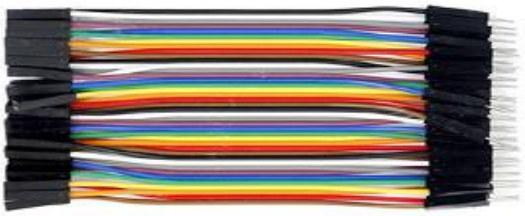


Pin1: It is VCC Pin. It is used to connect 5V to the sensor.
Pin2: It is GND Pin. It is used to connect GND to the sensor.
Pin3: It is a digital output Pin. From this pin, you will get digital data HIGH/LOW.
Pin4: It is an analog output Pin. From this pin, you will get analog data.

1- Experiment 1 : measure gas percentage

Hardware component

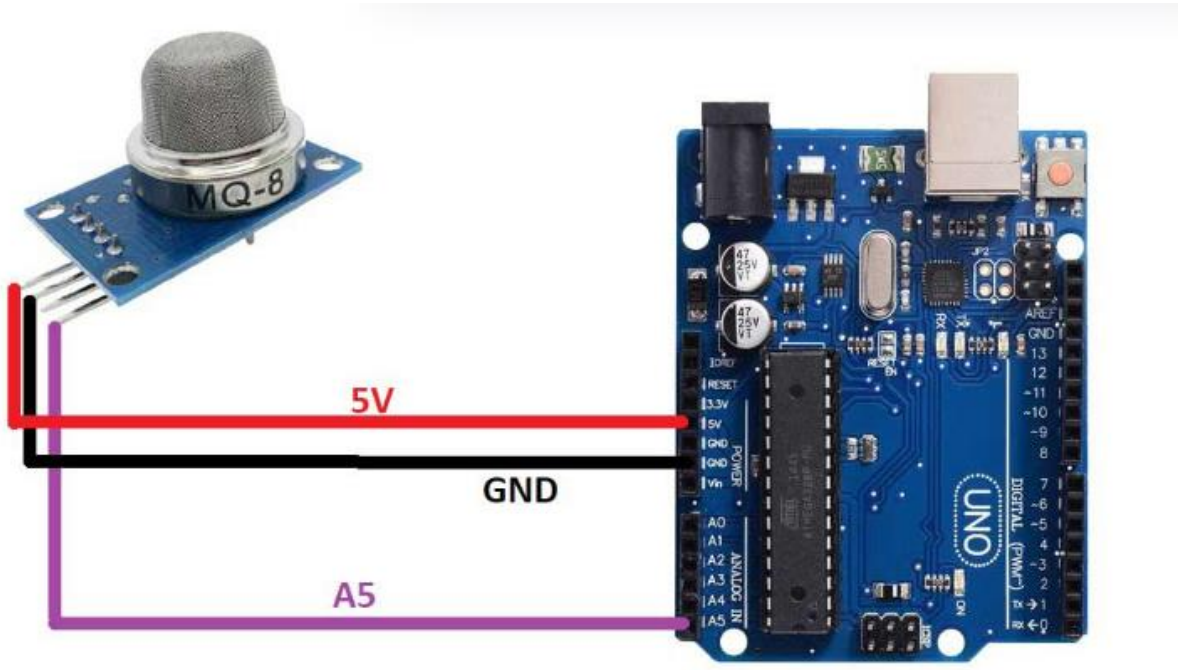




- **Arduino UNO R3**
- **MQ-8 Smoke Gas Module**

- **Male to Female jumper wire**

Circuit diagram of MQ8 Hydrogen H2 Gas Sensor with Arduino



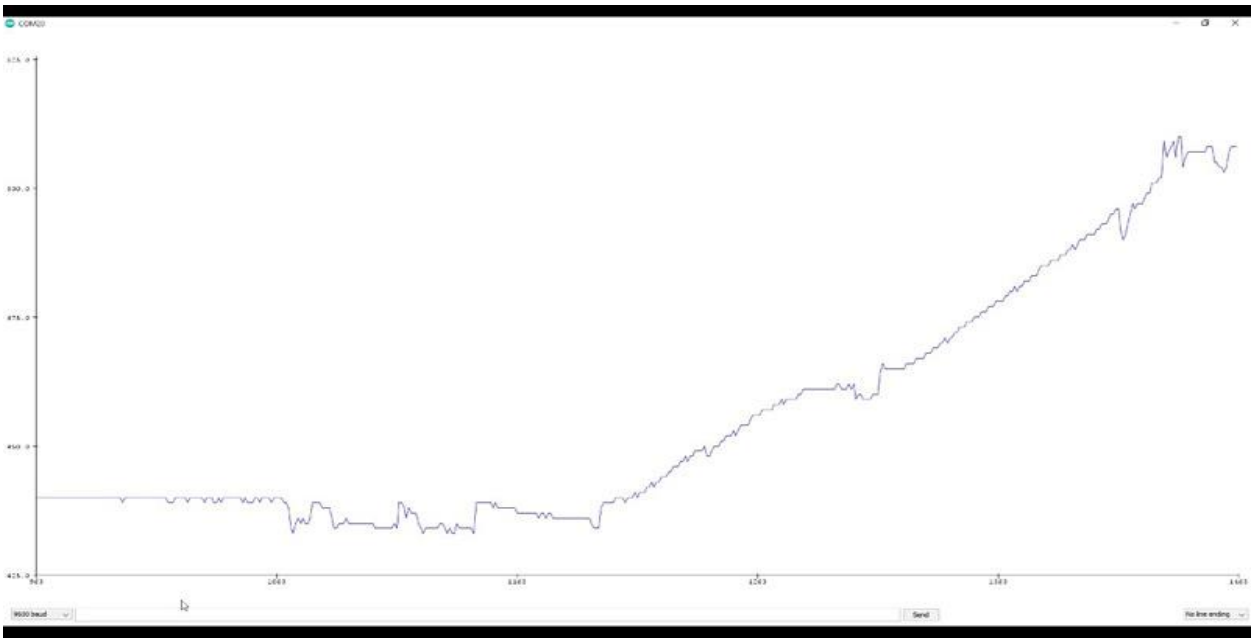
- Connect the VCC pin of the sensor to the 5V pin of the Arduino UNO board
- Connect the GND pin of the sensor to the GND pin of the Arduino UNO board
- Connect the AO pin of the sensor to the A5 pin of the Arduino UNO board

Code to detect gas leakage:

```
int sensorPin=A5;

int sensorData;

void setup()

{

  Serial.begin(9600);

  pinMode(sensorPin,INPUT);

 }

void loop()

{

  sensorData = analogRead(sensorPin);
```

```
  Serial.print("Sensor Data:");


  delay(100);
}
```

After uploading the code, you can see the output in the serial monitor.



Experiment 2 : Turn on led when

1 = Vcc
2 = AOUT
3 = DOUT
4 = GND

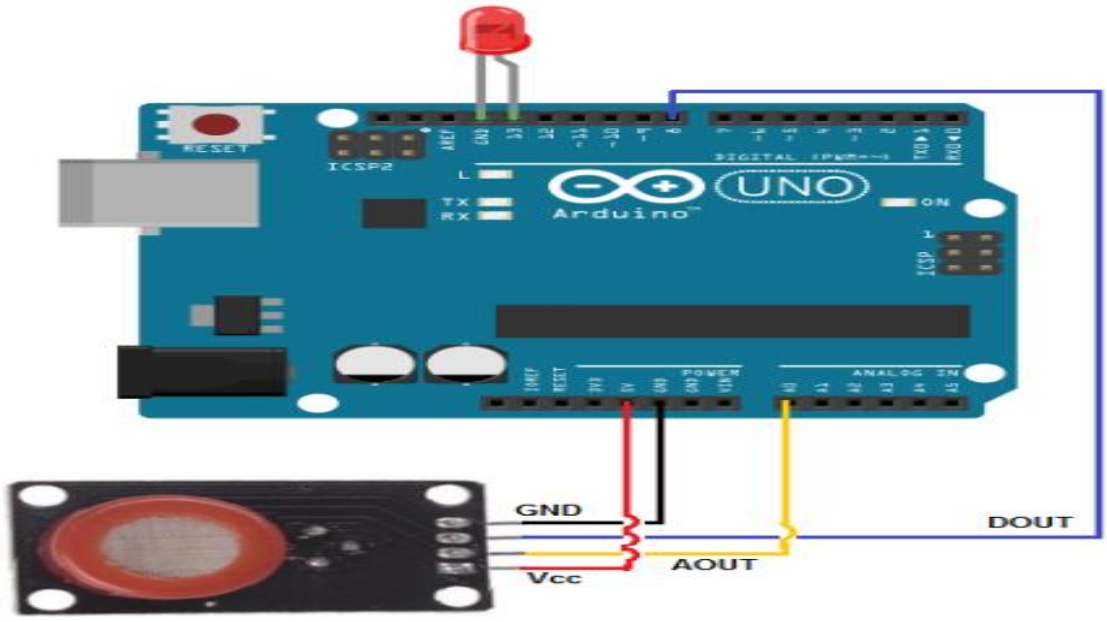
(bottom view)

There 4 leads are Vcc, AOUT, DOUT, and GND.

The Vcc and GND leads establishes power for the hydrogen sensor.

The other 2 leads are AOUT (analog output) and DOUT (digital output). How the sensor works is the terminal AOUT gives an analog voltage output in proportion to

the amount of methane the sensor detects. The more methane it detects, the greater the analog voltage it will output. Conversely, the less CO it detects, the less analog voltage it will output. If the analog voltage reaches a certain threshold, it will send the digital pin DOUT high. Once this DOUT pin goes high, the arduino will detect this and will trigger the LED to turn on, signaling that the methane threshold has been reached and is now over the limit. How you can change this threshold level is by adjusting the potentiometer to either raise or lower the level.



The connections are pretty basic.

To connect the sensor, there are 4 leads. 2 of them are for power. The Vcc terminal of the sensor connects into the 5V terminal of the arduino board. The GND terminal of the sensor connects into the GND terminal of the arduino. This establishes power for the sensor.

The other 2 connections are the analog and digital output of the sensor. These connect to analog pin A0 and digital pin D8, respectively.

Code

```
const int AOUTpin=0;//the AOUT pin of the hydrogen sensor goes into analog pin A0 of the arduino
const int DOUTpin=8;//the DOUT pin of the hydrogen sensor goes into digital pin D8 of the arduino
const int ledPin=13;//the anode of the LED connects to digital pin D13 of the arduino
```

```
int limit;
int value;

void setup() {
Serial.begin(115200);//sets the baud rate
pinMode(DOUTpin, INPUT);//sets the pin as an input to the arduino
pinMode(ledPin, OUTPUT);//sets the pin as an output of the arduino
}

void loop()
{
value= analogRead(AOUTpin);//reads the analaog value from the hydrogen sensor's AOUT pin
limit= digitalRead(DOUTpin);//reads the digital value from the hydrogen sensor's DOUT pin
Serial.print("Hydrogen value: ");
Serial.println(value);//prints the hydrogen value
Serial.print("Limit: ");
Serial.print(limit);//prints the limit reached as either LOW or HIGH (above or underneath)
delay(100);
if (limit == HIGH){
digitalWrite(ledPin, HIGH);//if limit has been reached, LED turns on as status indicator
}
else{
digitalWrite(ledPin, LOW);//if threshold not reached, LED remains off
}
}
```

Code Description

The first block of code defines all the pin connections of the sensor and the LED. Since the AOUTpin connects to analog pin A0, it is initialized to 0. Since the DOUTpin connects to digital pin D8, it is initialized to 8. Since the LED connects to digital pin D13, it is initialized to 13. 2 variables, limit and value, are also declared. These will be used to store the value of the analog pin AOUT and digital pin DOUT.

The next block of code sets the baud rate and declares the DOUTpin as input and the ledPin as output. This is because the sensor is an input to the arduino for the

arduino to read and process the sensor value. And the LED is an output will serves an indicator if the sensor has detected hydrogen.

The next block of code reads the sensor pin AOUT and stores the value in the integer *value*. It also reads the sensor pin DOUT and stores the value in the integer *limit*. We then print the hydrogen value, which will be a numeric value ranging from either 0 (no hydrogen detected) to 1023 (maximum level of hydrogen that can be read). We will aslo print the limit which will either be HIGH or LOW. If the hydrogen detected is under the threshold level, the value of limit returned will be low. If the hydrogen detected is above the threshold, the value of limit returned will be HIGH.

If the value is HIGH, the LED will turn on. If the value is low, the LED will remain off.

Experiment 15: MQ6 sensor with Arduino uno

Objective: measure gas percentage using mq6 sensor

MQ6 sensor is a sensor having high sensitivity to LPG, propane and isobutane. These sensors can also be used for the detection of different combustible gases, especially methane.

The SnO2 semiconductor material is used in the MQ6 for detecting the gases. It has lower conductivity in clean air. It helps in detecting the rising levels of gases through rise in its conductivity. Users can convert the change of conductivity to gas concentration through a simple electronic circuit.

An MQ6 gas sensor has a wide array of applications such as:

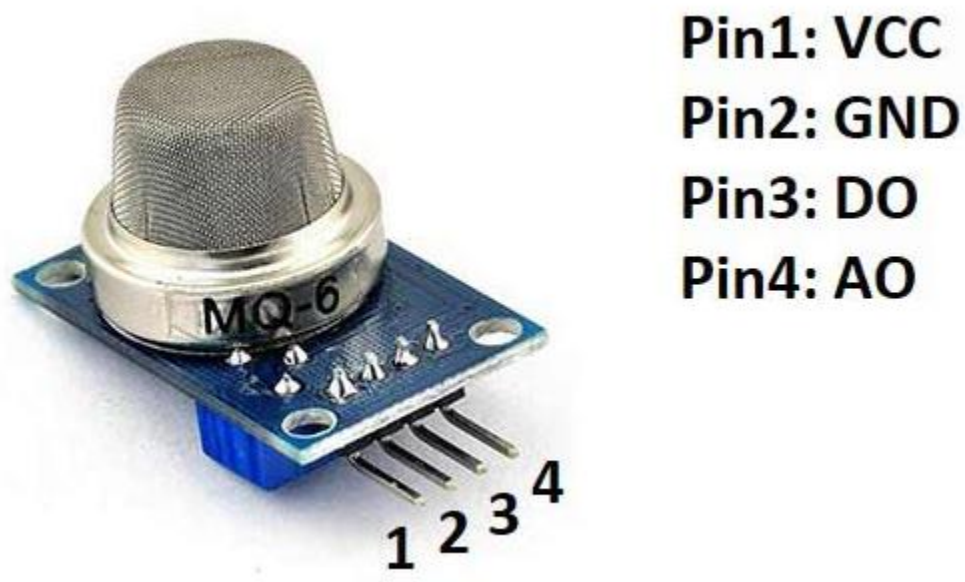
- Domestic and industrial gas leakage detector
- Industrial Combustible gas detector
- Portable gas detector

Applicability of the MQ6 gas sensor is enhanced by the following features:

- High sensitivity to LPG,iso-butane and propane
- Good sensitivity to combustible gases in a wide range
- Fast response time
- Long life
- Low cost

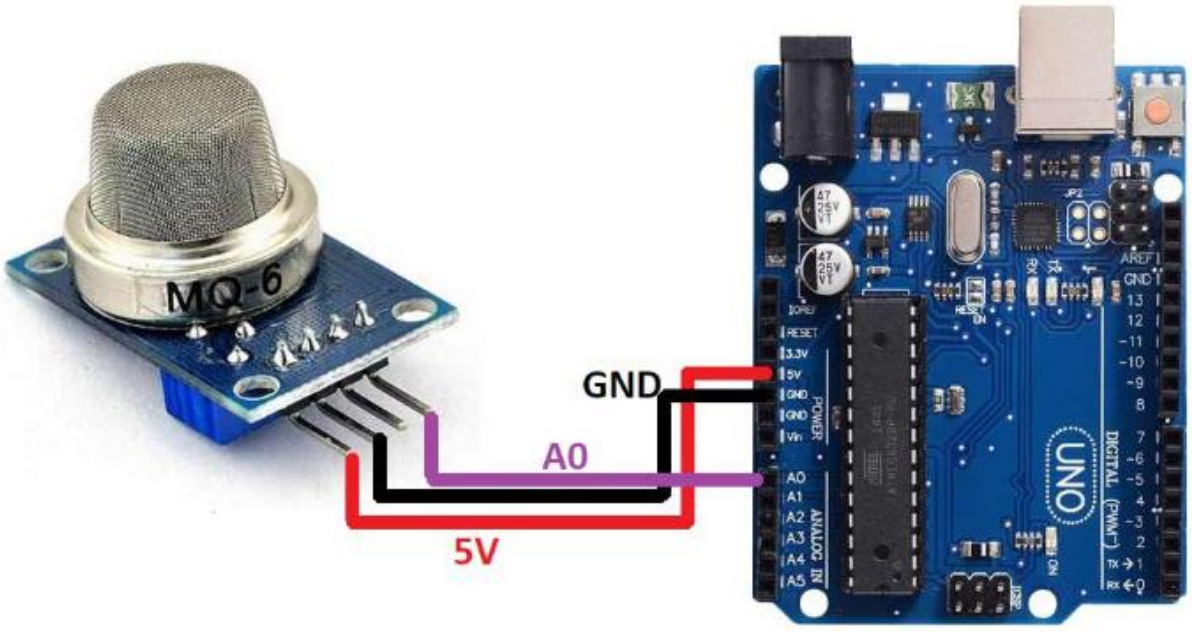
- Simple drive circuit

MQ6 Gas Sensor pin diagram



Pin1: It is VCC Pin. It is used to connect 5V to the sensor.
Pin2: It is GND Pin. It is used to connect GND to the sensor.
Pin3: It is a digital output Pin. From this pin, you will get digital data HIGH/LOW.
Pin4: It is analog output Pin. From this pin, you will get analog data.

Circuit diagram of MQ6 Gas Sensor with Arduino



- Connect the VCC pin of the sensor to the 5V pin of the Arduino UNO board

- Connect the GND pin of the sensor to the GND pin of the Arduino UNO board
- Connect the AO pin of the sensor to the A0 pin of the Arduino UNO board

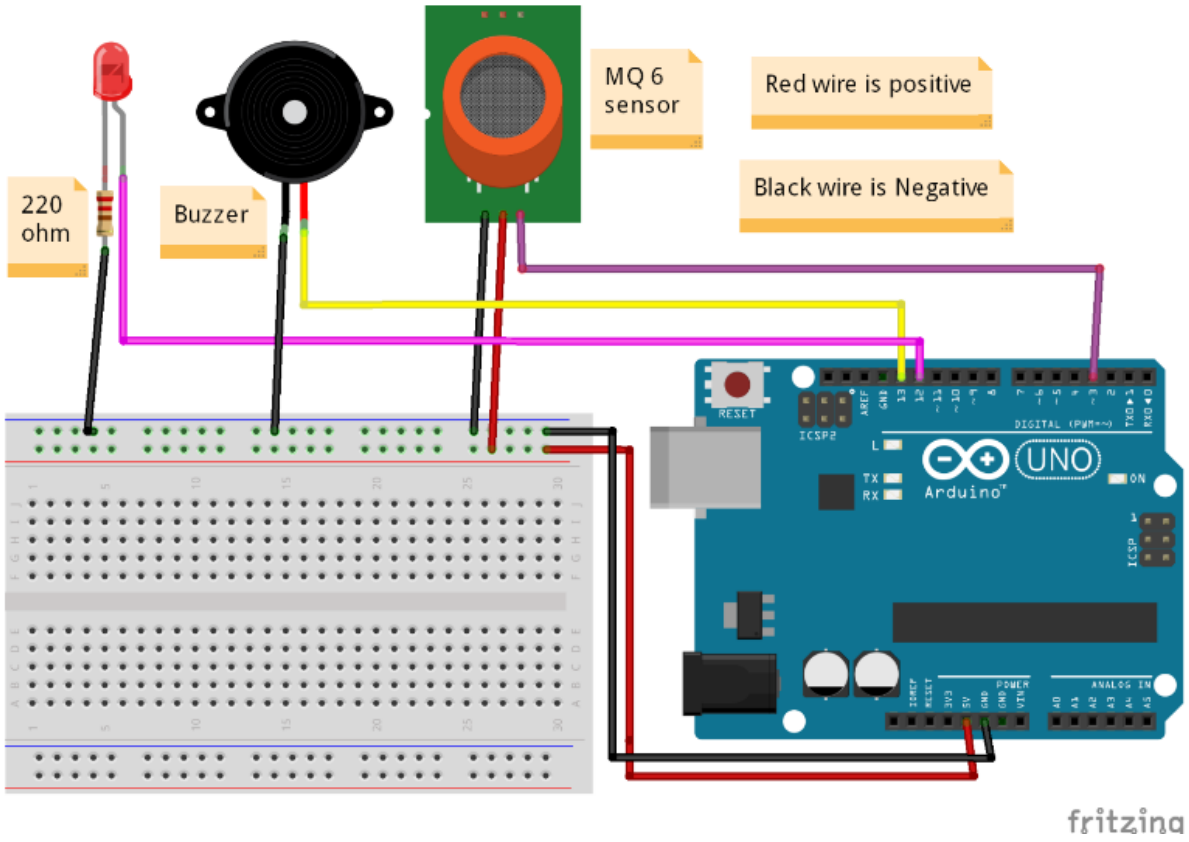
Code

```
int sensorPin=A0;
int sensorData;
void setup()
{
  Serial.begin(9600);
  pinMode(sensorPin,INPUT);
 }
void loop()
{
  sensorData = analogRead(sensorPin);
  Serial.print("Sensor Data:");

  delay(100);
}
```

Experiment 2 : if the gas percentage excceds specific value turn on led and buzzer

COMPONENTS NEEDED: –

- Any microcontroller preferably Arduino Uno for beginners.
- A red led
- An MQ-3 Alcohol sensor
- A breadboard
- Jumper wires
- 220ohm resistor
- A buzzer



Connection between Arduino uno and MQ6

Arduino UNO	**MQ-6 LPG Sensor**
(+5V)	VCC
GND	GND
D5 Pin	OUT Pin

Connection between buzzer and uno

Arduino UNO	**Buzzer**
D13 Pin	Positive Terminal
GND	Negative Terminal

Connection between led and uno

Arduino UNO	**LED**	**220 Ohm Resistor**

D12 Pin	Anode Pin	
	Cathode Pin	Terminal 1
GND		Terminal 2

Now connect ***D0 PIN OF SENSOR TO MICROCONTROLLER DIGITAL PIN 3.***

Code

```
int LED = 12;
int BUZZER = 13;
int LPG_sensor = 3;// MQ-6 SENSOR
int LPG_detected;
void setup()
{
 Serial.begin(9600);
 pinMode(LED, OUTPUT);
 pinMode(BUZZER, OUTPUT);
 pinMode(LPG_sensor, INPUT);
}
void loop()
{
  LPG_detected = digitalRead(LPG_sensor);
  Serial.println(LPG_detected);
 if (LPG_detected == 1)
 {
   Serial.println("LPG detected...");
   digitalWrite(LED, HIGH);
   digitalWrite(BUZZER, HIGH);
 }
 else
```

```
    {
      Serial.println("No LPG detected.");
      digitalWrite(LED, LOW);
      digitalWrite(BUZZER, LOW);
    }
  }
```

Experiment 16: sound sensor with Arduino

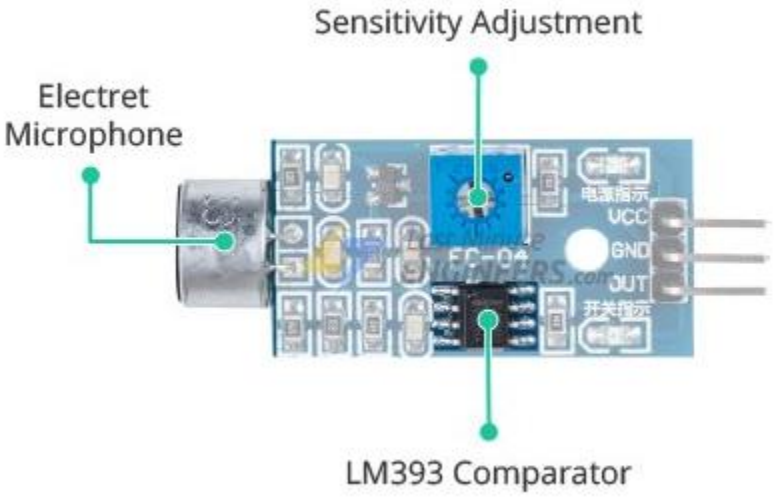
Objective: detect sound using sound sensor and Arduino uno

These sound sensors are inexpensive, simple to use, and capable of detecting voice, claps, or door knocks.

You can use them for a variety of sound-reactive projects, such as making your lights clap-activated or monitoring your pets while you're away.

The sound sensor is a small board that incorporates a microphone (50Hz-10kHz) and some processing circuitry to convert the sound wave into an electrical signal.

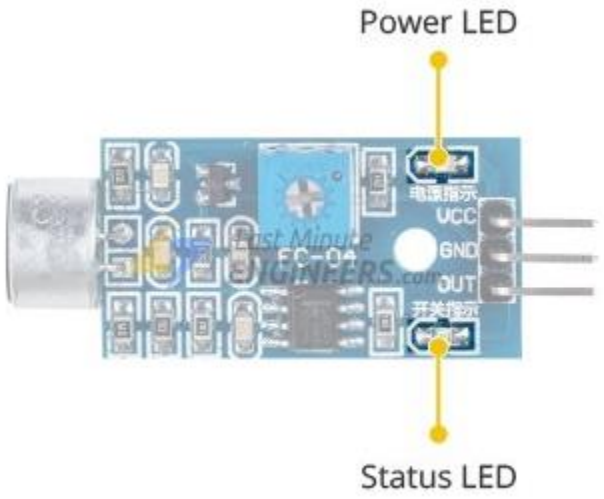
This electrical signal is fed to the on-board LM393 High Precision Comparator, which digitizes it and makes it available at the OUT pin.



The module includes a potentiometer for adjusting the sensitivity of the OUT signal.

You can use it to set a threshold, so that when the amplitude of the sound exceeds the threshold, the module outputs LOW, otherwise HIGH.

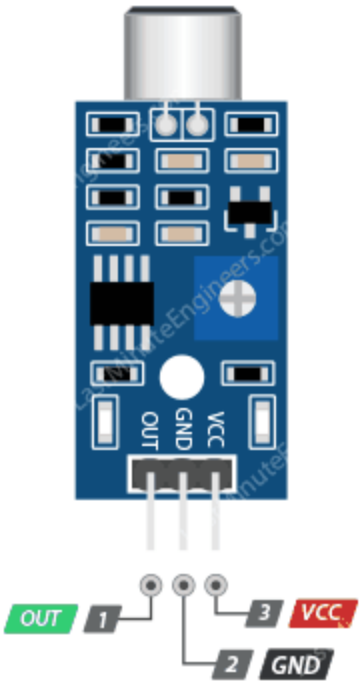
This setup is very useful for triggering an action when a certain threshold is reached. For example, when the amplitude of the sound exceeds a threshold (a knock is detected), you can activate a relay to control the light.



The module also includes two LEDs. The Power LED illuminates when the module is turned on, and the Status LED illuminates when the sound level exceeds the threshold value.

Sound Sensor Pinout

The sound sensor only has three pins:



VCC supplies power to the sensor. It is recommended that the sensor be powered from 3.3V to 5V.

GND is the ground pin.

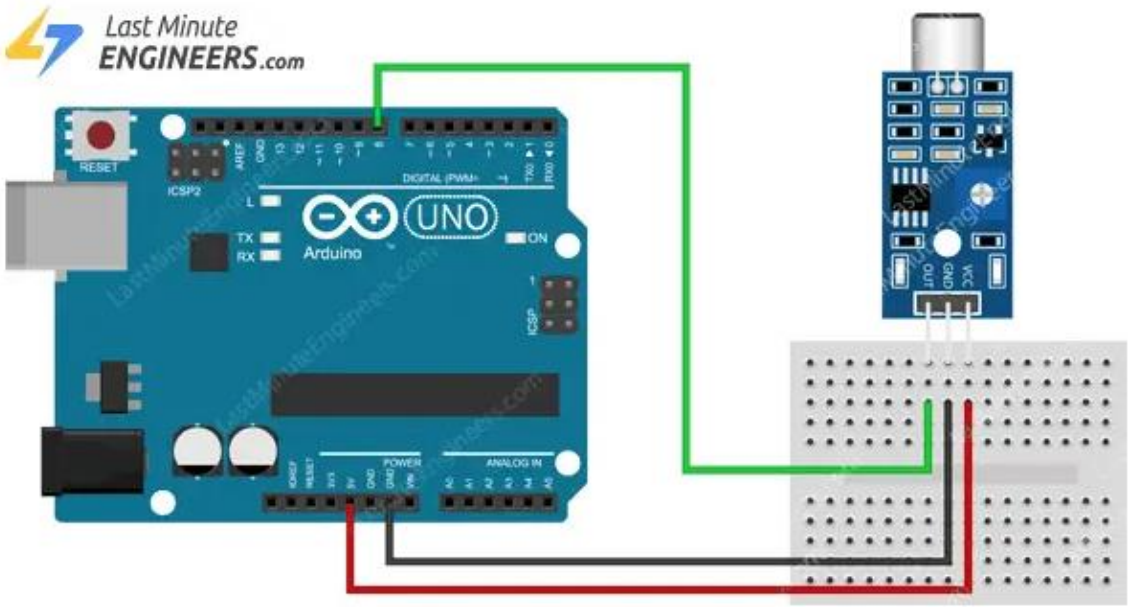
OUT pin outputs HIGH under quiet conditions and LOW when sound is detected. You can connect it to any digital pin on an Arduino or to a 5V relay directly.

Wiring a Sound Sensor to an Arduino

Connections are fairly simple. Start by connecting the module's VCC pin to the Arduino's 5V pin and the GND pin to ground.

Finally, connect the OUT pin to Arduino digital pin #8. That's it!

The wiring is shown in the image below.

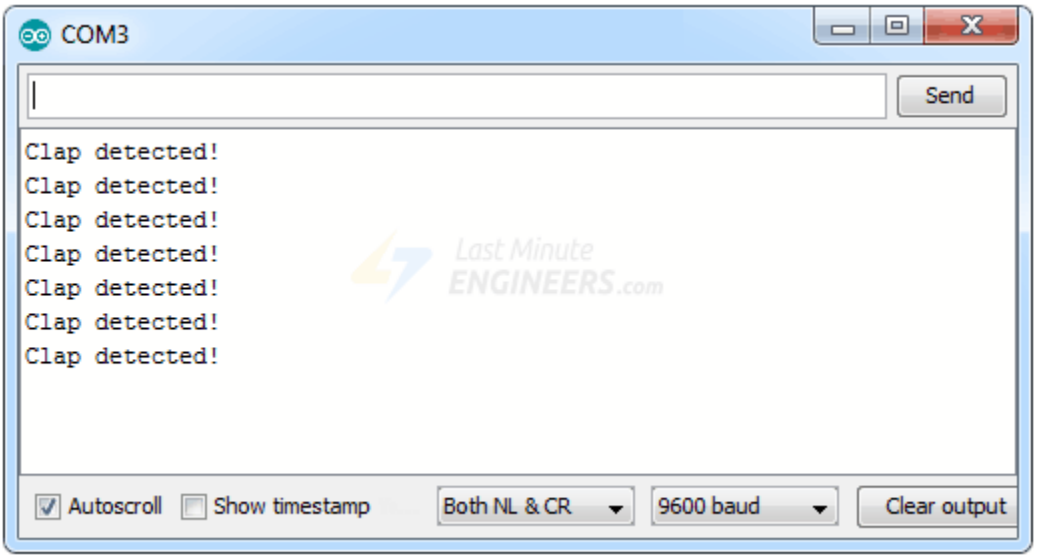


Code

```
#define sensorPin 8

// Variable to store the time when last event happened
unsigned long lastEvent = 0;

void setup() {
        pinMode(sensorPin, INPUT);     // Set sensor pin as an INPUT
        Serial.begin(9600);
}

void loop() {
        // Read Sound sensor
        int sensorData = digitalRead(sensorPin);

        // If pin goes LOW, sound is detected
        if (sensorData == LOW) {

                // If 25ms have passed since last LOW state, it means that
                // the clap is detected and not due to any spurious sounds
                if (millis() - lastEvent > 25) {
                        Serial.println("Clap detected!");
                }
```

```
                // Remember when last event happened
                lastEvent = millis();
        }
}
```

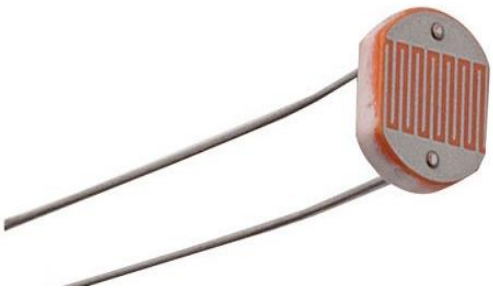
If everything is working properly, you should see the following output on the serial monitor when the clap is detected.



Experiment 17: photoresistor sensor with Arduino

Objective: Measure the amount of light using photoresistor sensor

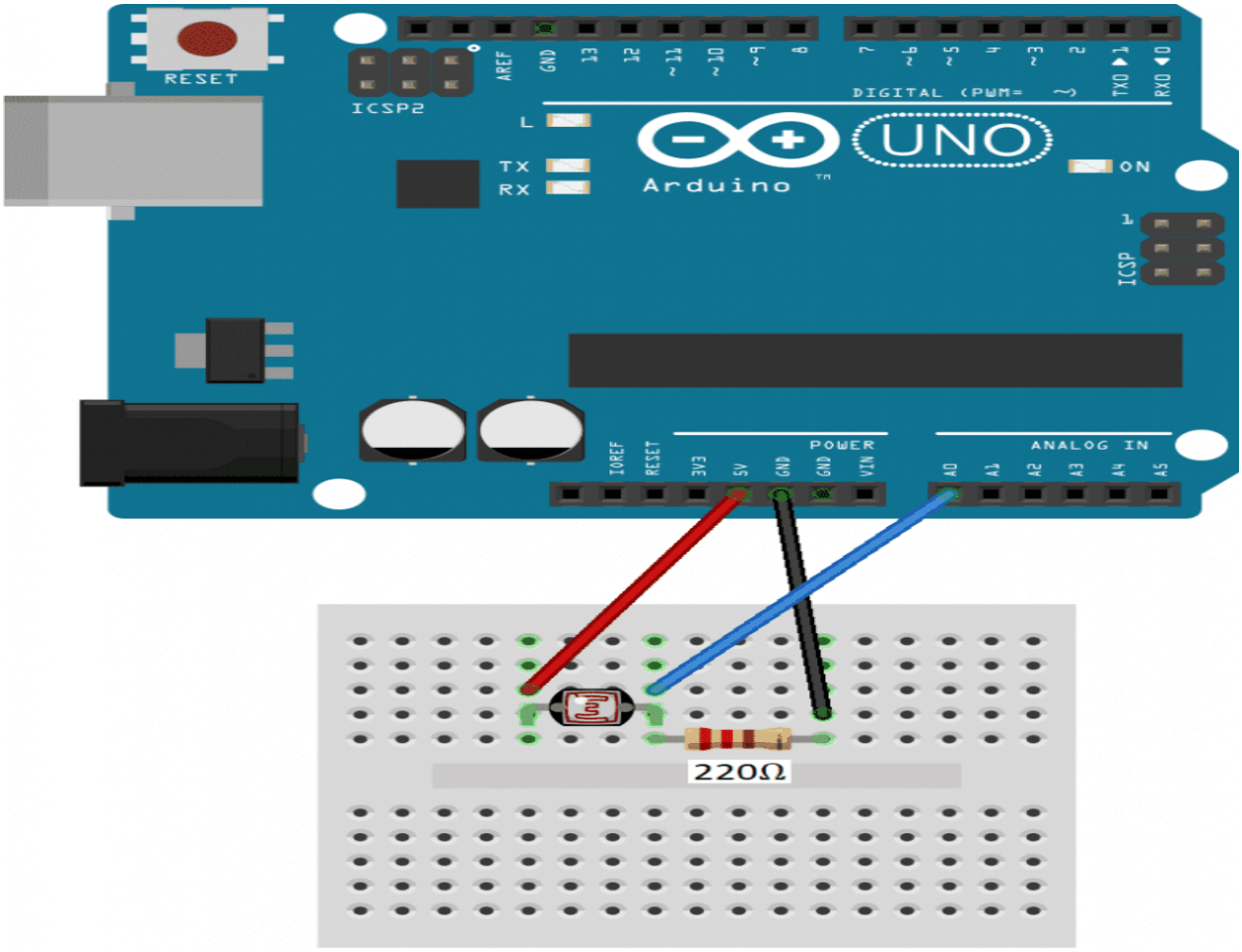
A photoresistor or photocell is a light-controlled variable resistor. The resistance of a photoresistor decreases with increasing incident light intensity. A photoresistor can be applied in light-sensitive detector circuits, and light- and dark-activated switching circuits. It's also called light-dependent resistor (LDR).



The resistance value becomes smaller when there is much light in the room. So in the dark the led remains off because the resistance has become very big.

Experiment 1 : Measure light percentage

Hardware Circuit

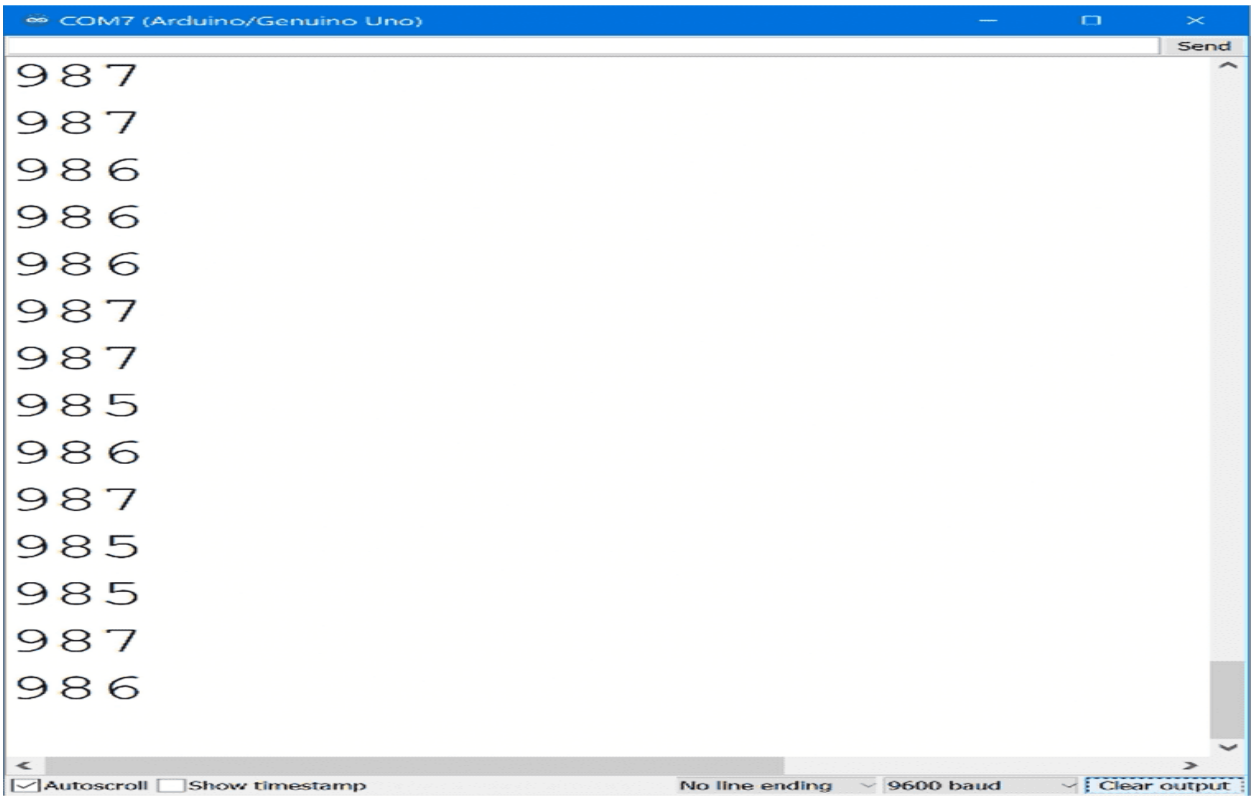


Code

```
int photoPin = A0;


void setup() {
  Serial.begin(9600);
}


void loop() {
  int light = analogRead(photoPin);
  Serial.println(light);
  delay(100);
```

```
}
```

Output



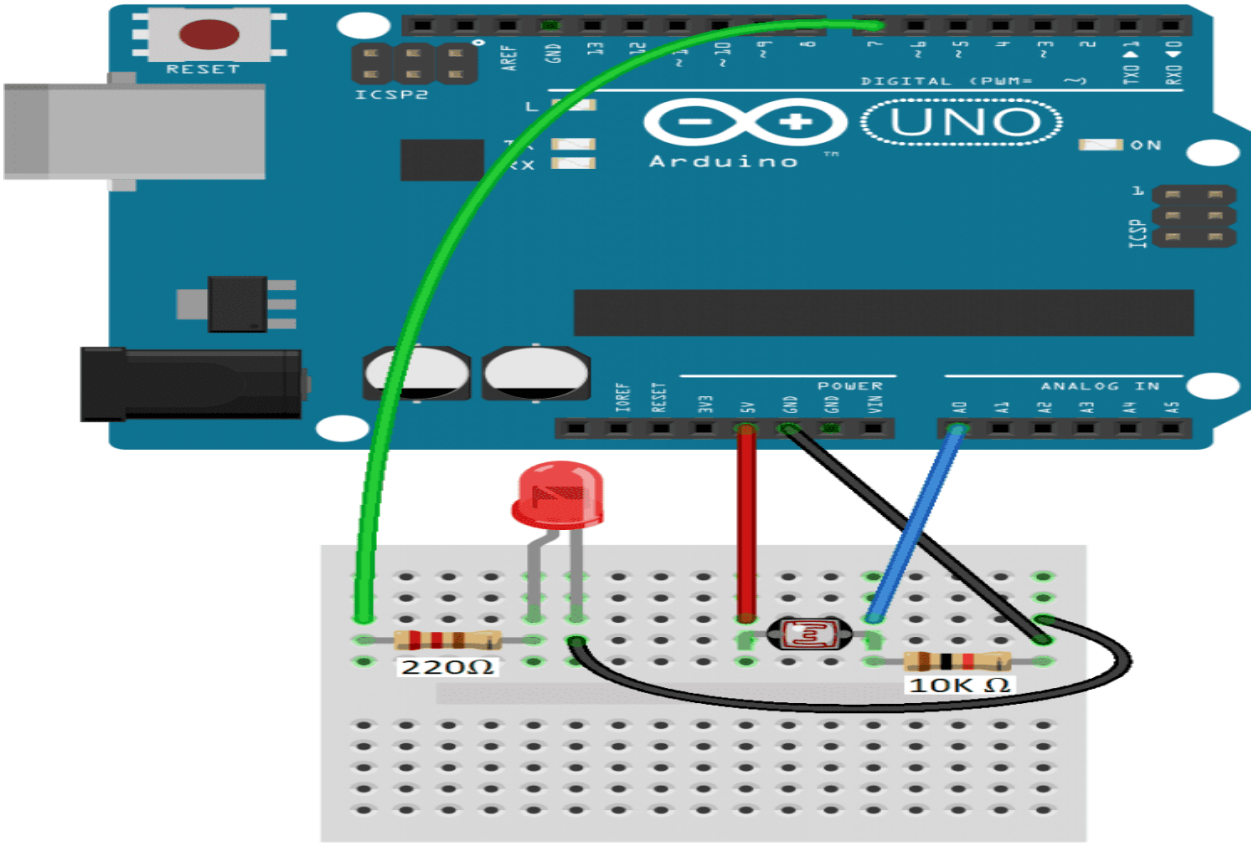
Experiment 2: HOW TO USE A PHOTORESISTOR TO CONTROL THINGS

Photoresistors are great for making light controlled switches. So now let's build a light controlled switch that turns on an LED when it's dark, and turns off the LED in the light.

Hardware component

These are the parts you will need:

- Arduino Uno
- Jumper wires
- Breadboard
- 220 Ohm resistor
- 10K Ohm resistor
- Photoresistor
- One LED



Code

```
const int pResistor = A0; // Photoresistor at Arduino analog pin A0
const int ledPin=7;       // Led pin at Arduino pin 7
//Variables
int value;                              // Store value from photoresistor (0-1023)
void setup(){
 pinMode(ledPin, OUTPUT);  // Set lepPin - 9 pin as an output
 pinMode(pResistor, INPUT);// Set pResistor - A0 pin as an input (optional)
}


void loop(){
  value = analogRead(pResistor);


  //You can change value "25"
```

```
if (value > 25){
  digitalWrite(ledPin, LOW);  //Turn led off
}
else{
  digitalWrite(ledPin, HIGH); //Turn led on
}
```

ESP8266

The ESP8266 is a low-cost Wi-Fi microchip, with built-in TCP/IP networking software, and microcontroller capability, produced by Espressif Systems in Shanghai, China.

The chip was popularized in the English-speaking maker community in August 2014 via the ESP-01 module, made by a third-party manufacturer Ai-Thinker. This small module allows microcontrollers to connect to a Wi-Fi network and make simple TCP/IP connections using Hayes-style commands. However, at first, there was almost no English-language documentation on the chip and the commands it accepted. The very low price and the fact that there were very few external components on the module, which suggested that it could eventually be very inexpensive in volume, attracted many hackers to explore the module, the chip, and the software on it, as well as to translate the Chinese documentation. The ESP8285 is a similar chip with a built-in 1 MiB flash memory, allowing the design of single-chip devices capable of connecting via Wi-Fi.

NodeMCU is an open source development board and firmware based in the widely used ESP8266 -12E WiFi module. It allows you to program the ESP8266 WiFi module with the simple and powerful LUA programming language or Arduino IDE.

With just a few lines of code you can establish a WiFi connection and define input/output pins according to your needs exactly like arduino, turning your ESP8266 into a web server and a lot more. It is the WiFi equivalent of ethernet module. Now you have internet of things (iot) real tool.

With its USB-TTL , the nodeMCU Dev board supports directly flashing from USB port. It combines features of WIFI accesspoint and station + microcontroller. These features make the NodeMCU extremly powerful tool for Wifi networking. It can be

used as access point and/or station, host a webserver or connect to internet to fetch or upload data

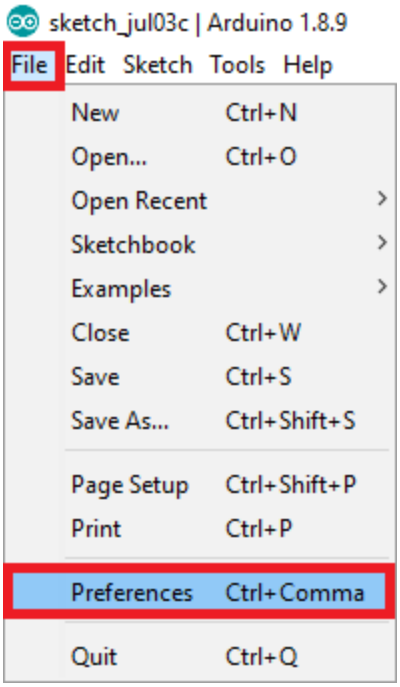
Features

- Finally, programable WiFi module.
- Arduino-like (software defined) hardware IO.
- Can be programmed with the simple and powerful Lua programming language or Arduino IDE.
- USB-TTL included, plug & play.
- 10 GPIOs D0-D10, PWM functionality, IIC and SPI communication, 1-Wire and ADC A0 etc. all in one board.
- Wifi networking (can be used as access point and/or station, host a web server), connect to internet to fetch or upload data.
- Event-driven API for network applications.
- PCB antenna.

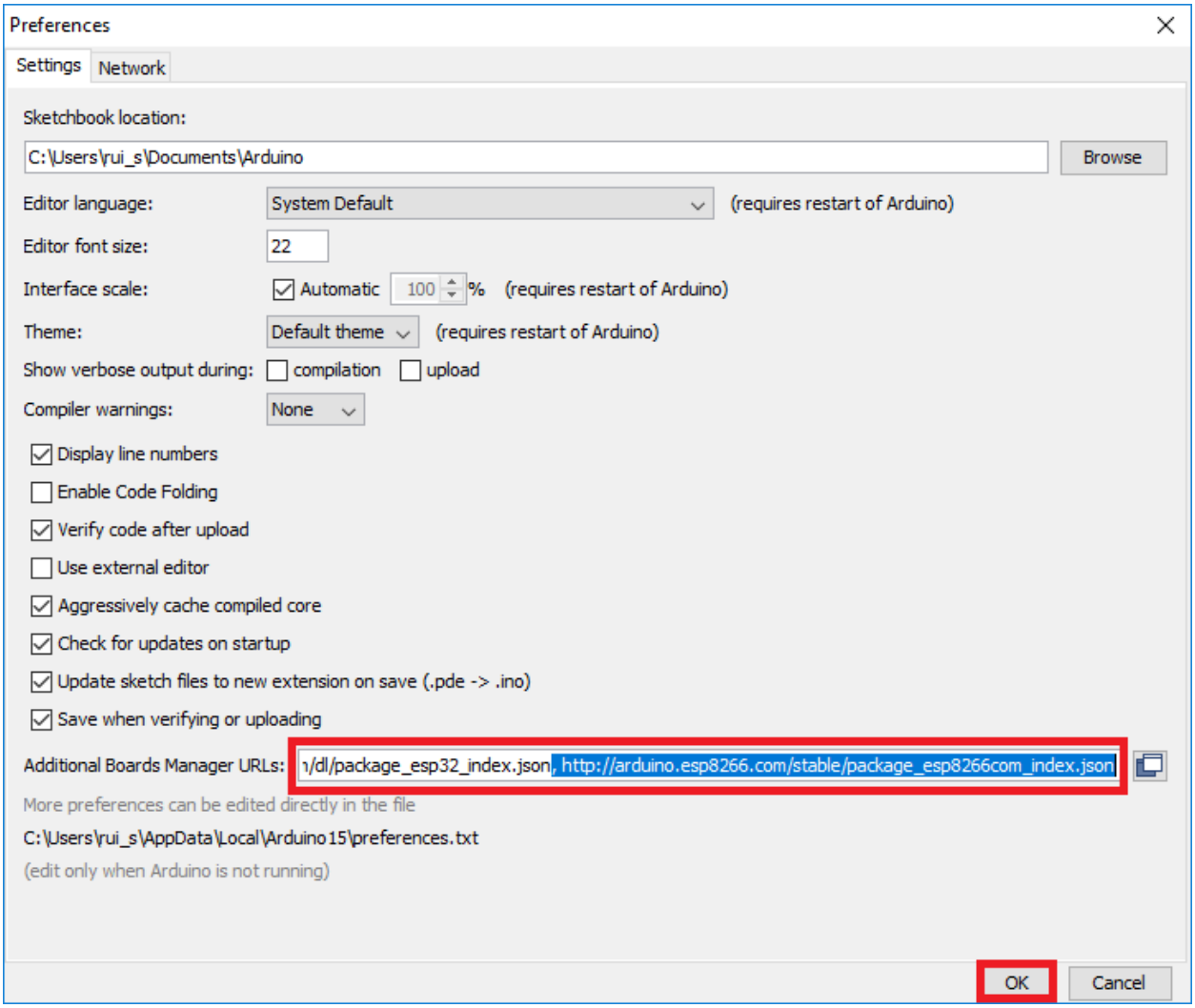
Install ESP8266 Add-on in Arduino IDE

To install the ESP8266 board in your Arduino IDE, follow these next instructions:

1. In your Arduino IDE, go to **File**> **Preferences**



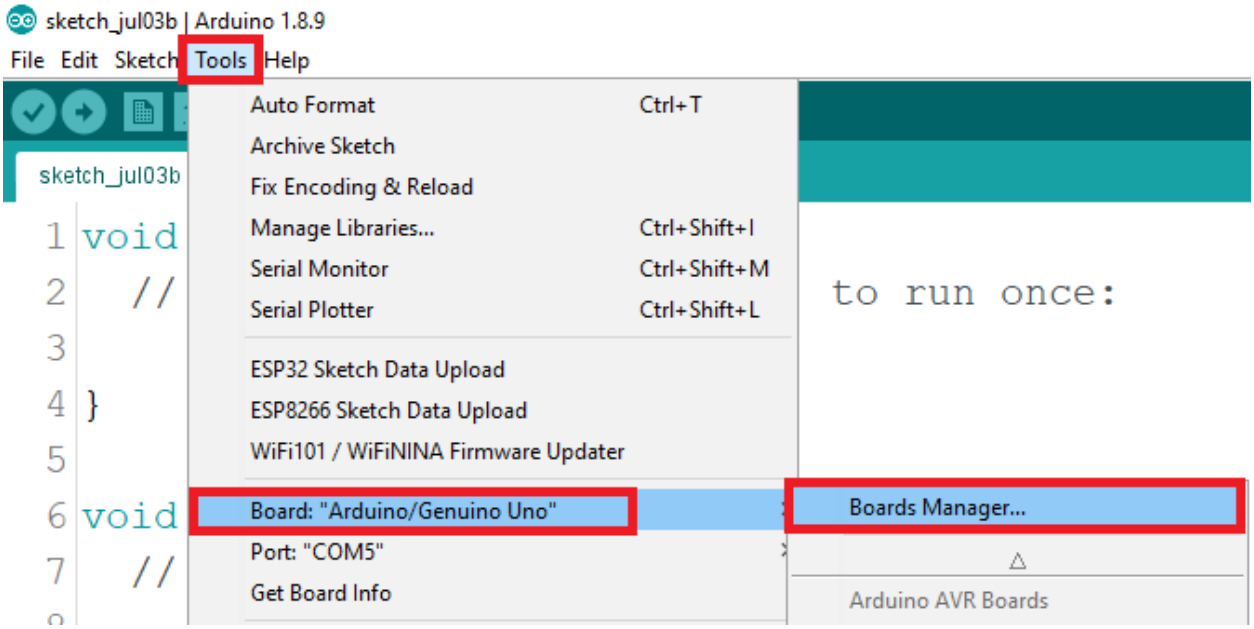
2. Enter **http://arduino.esp8266.com/stable/package_esp8266com_index.json** into the "Additional Boards Manager URLs" field as shown in the figure below. Then, click the "OK" button:



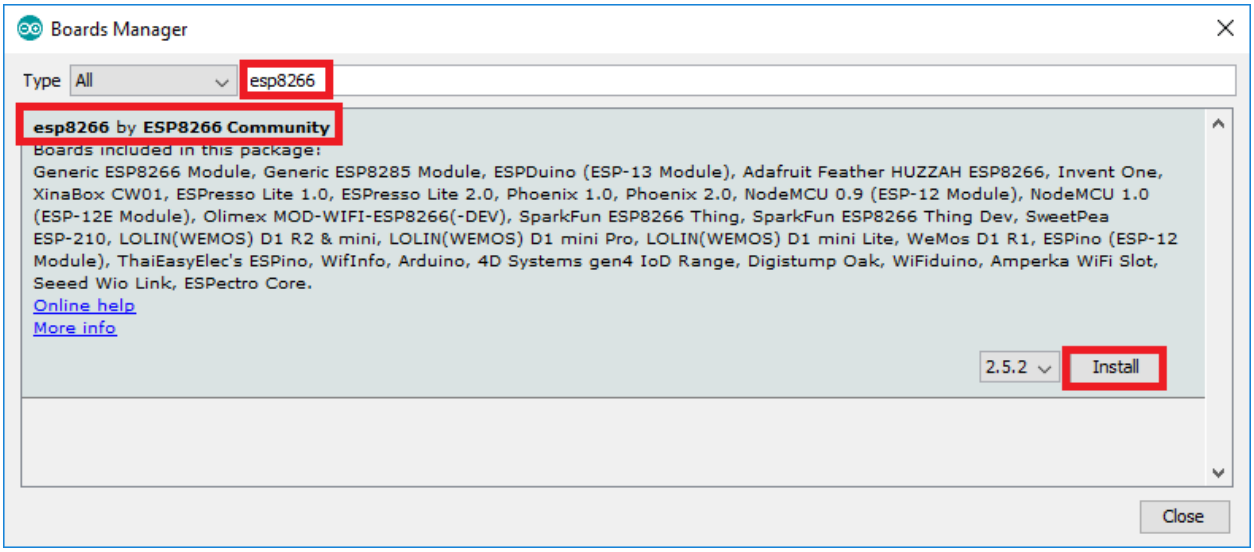
Note: if you already have the ESP32 boards URL, you can separate the URLs with a comma as follows:

https://dl.espressif.com/dl/package_esp32_index.json, http://arduino.esp8266.com/stable/package_esp8266com_index.json

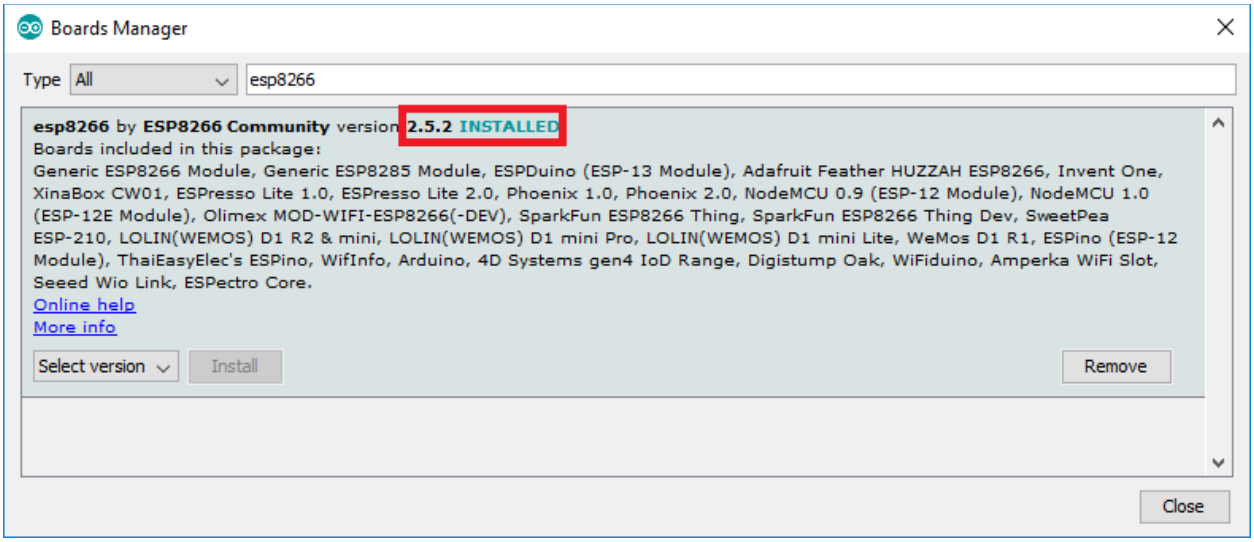
3. Open the Boards Manager. Go to **Tools** > **Board** > **Boards Manager…**



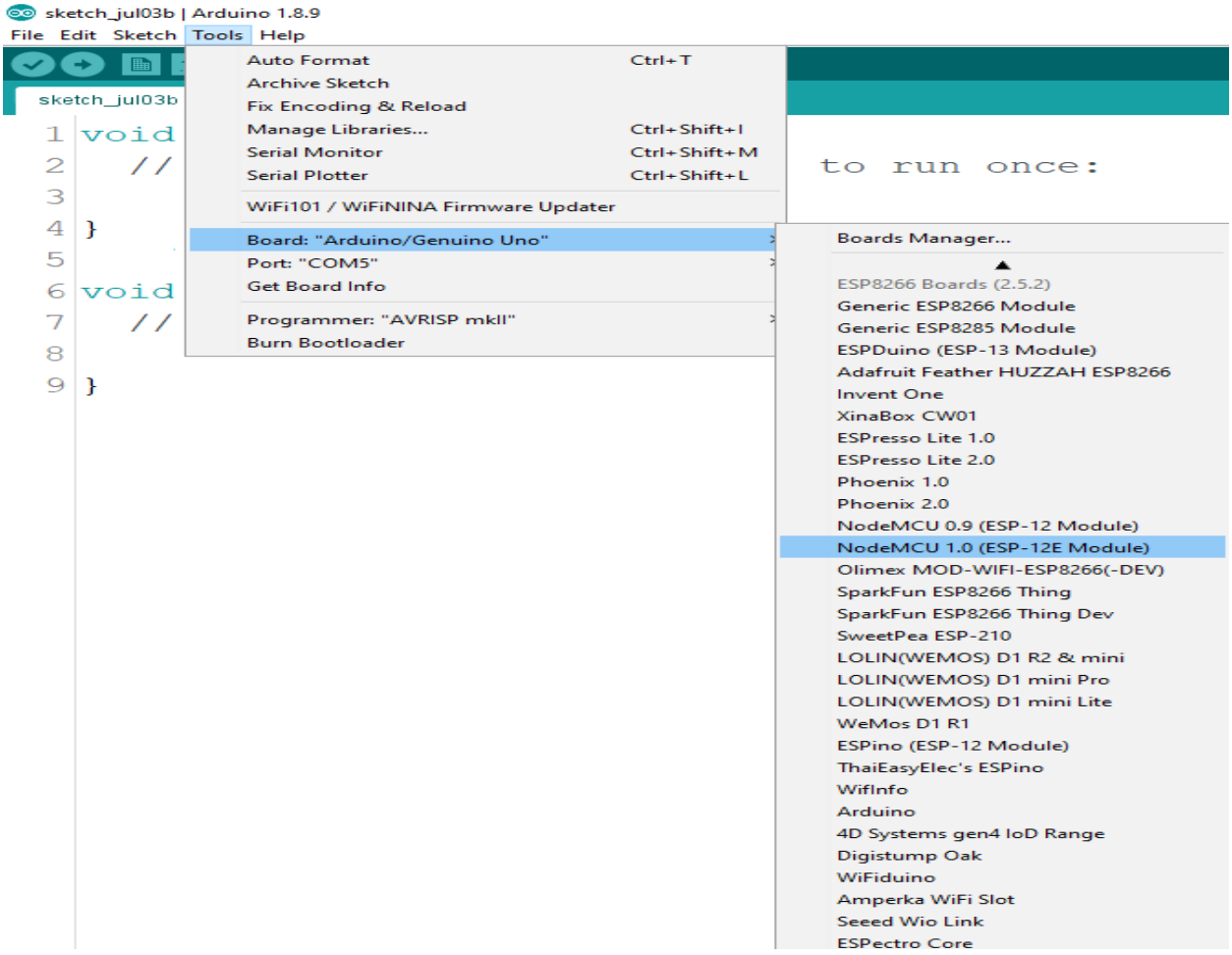
4. Search for ESP8266 and press install button for the “ESP8266 by ESP8266 Community“:



5. That’s it. It should be installed after a few seconds.



If you’re using an NodeMCU Kit, uploading the sketch is very simple, since it has built-in programmer. Plug your board to your computer. Make sure you have the right board selected:

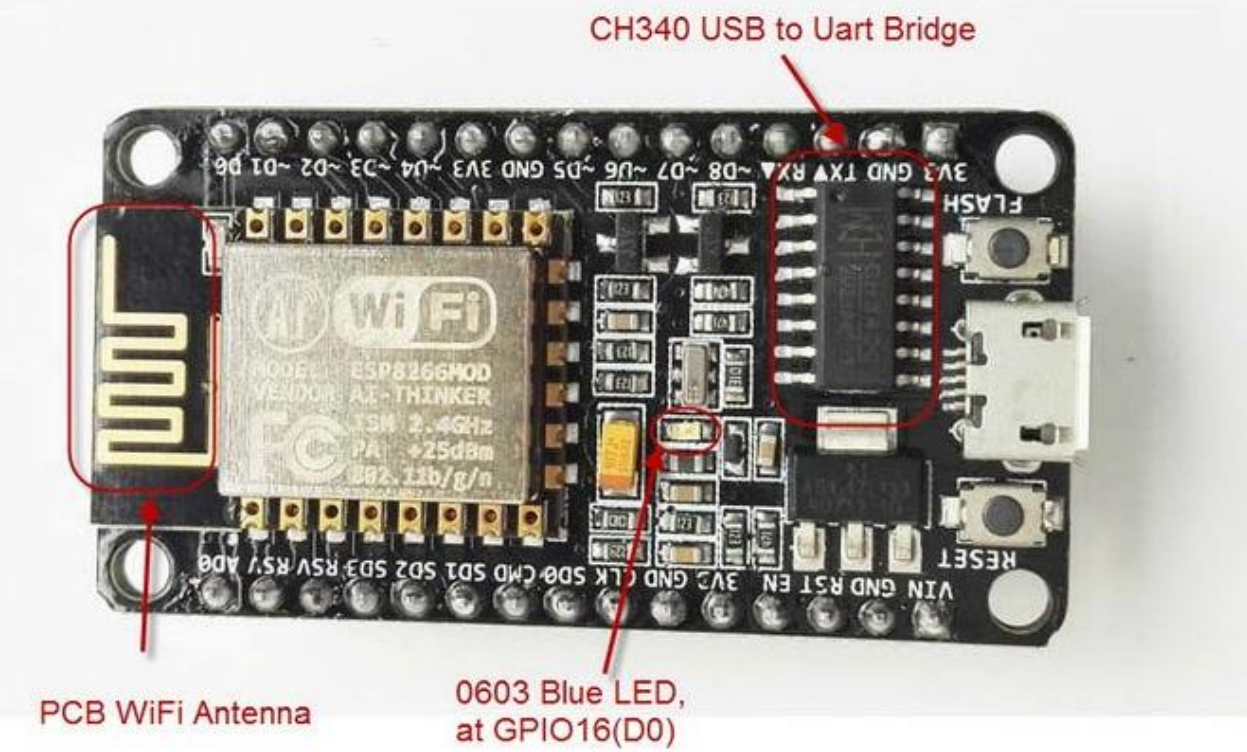


هنا ممكن نختار

Generic ESP8266 module

Experiment 18: Turn up built in LED using ESP module

In this part, we will learn how to use the Arduino IDE to blink the on-board LED, the blue led in the board was used to signal the execution of a particular procedure. Because the GPIO16 (DO) limitation to use the analogWrite() and looking to see my LED doing something else…



First,connect the NodeMCU to the PC,put below code to the Arduino IDE:

Code

```
#define LED D0                    // Led in NodeMCU at pin GPIO16 (D0).
void setup() {
pinMode(LED, OUTPUT);    // LED pin as output.
}
void loop() {
digitalWrite(LED, HIGH);// turn the LED off.(Note that LOW is the voltage level but actually
                        //the LED is on; this is because it is acive low on the ESP8266.
delay(1000);            // wait for 1 second.
digitalWrite(LED, LOW); // turn the LED on.
```

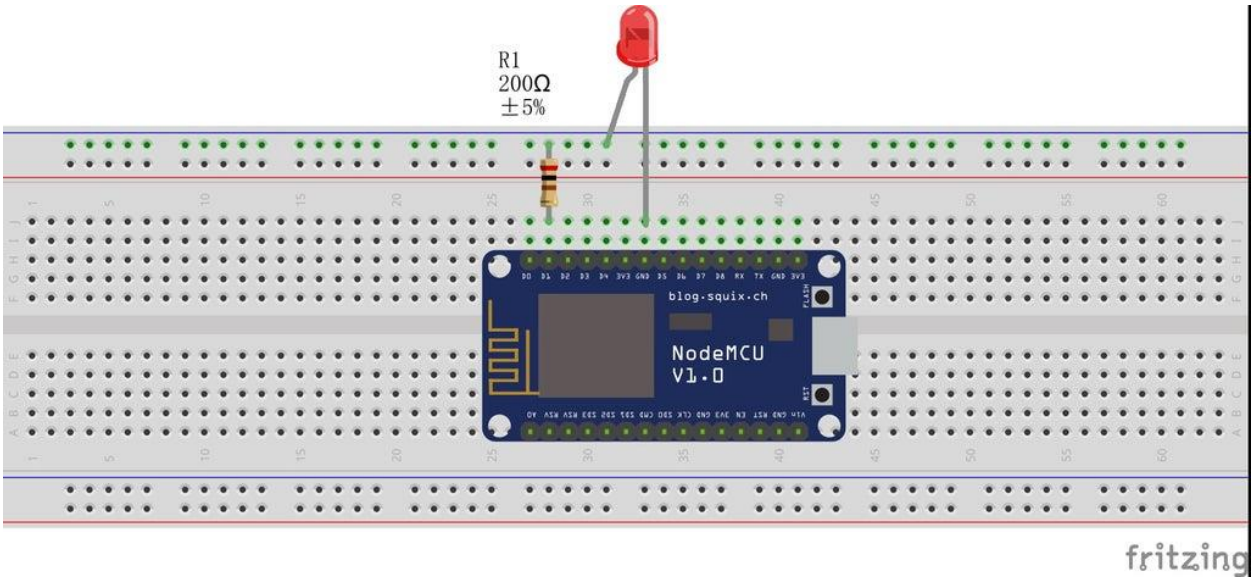
```
delay(1000); // wait for 1 second.

}
```

Light up external led using ESP 2866

Hardware Component

- NodeMCU x 1
- LED x 1
- 200 ohm resistor x 1
- Micro USB cable x 1

Hardware circuit



Connect one end of the resistor to the digital pin correspondent to the LED_BUILTIN constant.

Connect the long leg of the LED (the positive leg, called the anode) to the other end of the resistor.

Connect the short leg of the LED (the negative leg, called the cathode) to the GND. In the diagram we show a NodeMCU that has D1 as the LED_BUILTIN value.

The value of the resistor in series with the LED may be of a different value than 200 ohm; the LED will lit up also with values up to 1K ohm.

Upload below code to your NodeMCU:

Code

```
#define LED D1 // Led in NodeMCU at pin GPIO16 (D0).
void setup() {
```

```
pinMode(LED, OUTPUT); // set the digital pin as output.
}
void loop() {
digitalWrite(LED, HIGH);
delay(1000);          // wait for 1 second.
digitalWrite(LED, LOW); // turn the LED on.
delay(1000);          // wait for 1 second.
}
```

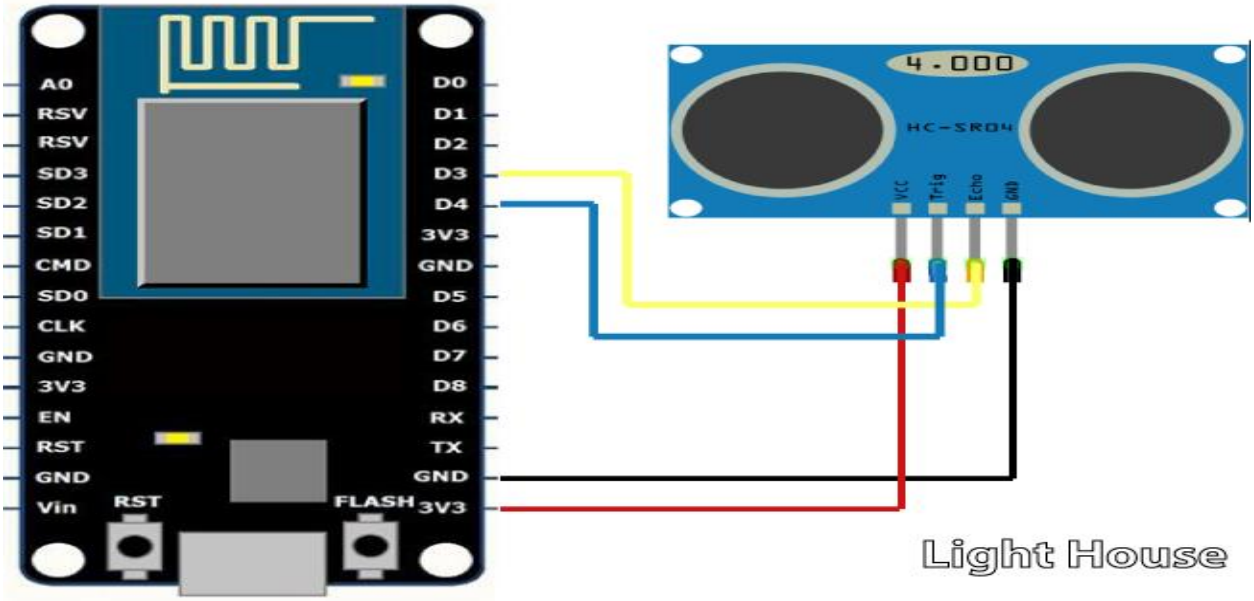
Ultrasonic sensor with ESP 2866

Objectives: measure distance using ultrasonic sensor and ESP 2866

Hardware component

- NodeMCU
- HC-SR04 (Ultra-sonic Sensor)
- Bread Board
- Jumper Wires
- Micro USB Cable

Hardware circuit



The circuit connections are made as follows:

- The HC-SR04 sensor attach to the Breadboard
- The sensor Vcc is connected to the NodeMCU +3.3v
- The sensor GND is connected to the NodeMCU GND
- The sensor Trigger Pin is connected to the NodeMCU Digital I/O D4
- The sensor Echo Pin is connected to the NodeMCU Digital I/O D3

Code

```
// defines pins numbers
const int trigPin = 2; //D4
const int echoPin = 0;  //D3
// defines variables
long duration;
int distance;
void setup() {
pinMode(trigPin, OUTPUT); // Sets the trigPin as an Output
pinMode(echoPin, INPUT); // Sets the echoPin as an Input
Serial.begin(9600); // Starts the serial communication
}
void loop() {
// Clears the trigPin
digitalWrite(trigPin, LOW);
delayMicroseconds(2);
// Sets the trigPin on HIGH state for 10 micro seconds
digitalWrite(trigPin, HIGH);
delayMicroseconds(10);
digitalWrite(trigPin, LOW);
// Reads the echoPin, returns the sound wave travel time in microseconds
duration = pulseIn(echoPin, HIGH);
// Calculating the distance
```

```
distance= duration*0.034/2;
// Prints the distance on the Serial Monitor
Serial.print("Distance: ");
Serial.println(distance);
delay(2000);
}
```

Build web server from ESP2866

The following video show the way

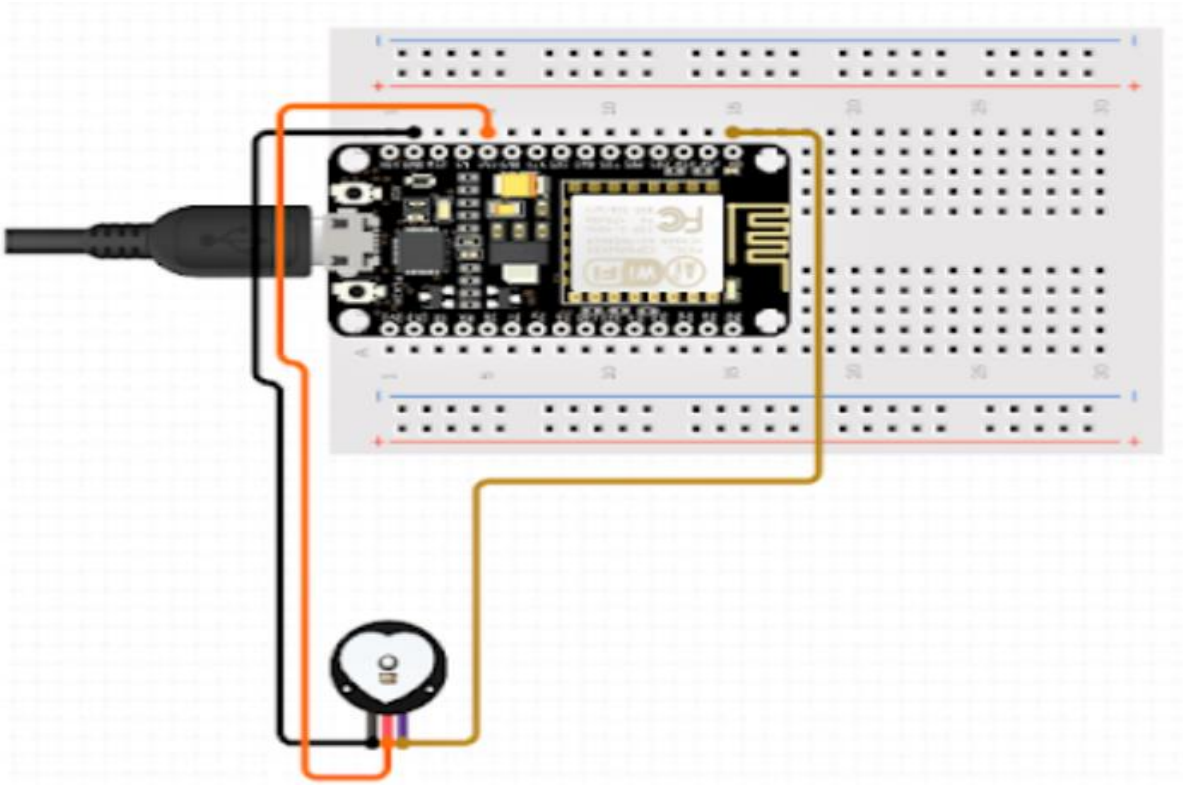
https://www.youtube.com/watch?v=dWM4p_KaTHY

Pulse heart beat sensor with ESP module

Hardware component

- An ESP8266 board
- A pulse sensor
- Arduino IDE
- USB cable

Hardware circuit



- Connect the pulse sensor to the **A0** pin of the ESP8266 board. Follow the **circuit diagram**.
- Connect the ESP8266 board to your computer using a USB cable.
- Open the Arduino IDE and select your ESP8266 board from the Tools menu.
- Install the PulseSensorPlayground library by navigating to Sketch > Include Library > Manage Libraries and searching for "PulseSensorPlayground." Click the Install button to install the library.

Code

```
// Include the necessary libraries
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>
#include <PulseSensorPlayground.h>
// Replace with your network credentials
const char* ssid = "your_SSID";
const char* password = "your_PASSWORD";
// Create an instance of the server
ESP8266WebServer server(80);
// Define the pin for the pulse sensor
```

```
const int pulsePin = A0;
// Create an instance of the pulse sensor
PulseSensorPlayground pulseSensor;
void setup() {
  // Start serial communication
  Serial.begin(115200);
  // Connect to Wi-Fi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }
  // Print the IP address
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
  // Initialize the pulse sensor
  pulseSensor.begin();
  pulseSensor.setPin(pulsePin);
  pulseSensor.setThreshold(550);
  // Set up the server
  server.on("/", handleRoot);
  server.on("/pulse", handlePulse);
  server.begin();
}
void loop() {
  // Update the pulse sensor
  pulseSensor.update();
  // Handle incoming client requests
  server.handleClient();
}
void handleRoot() {
  // Send the HTML page to the client
  String html = "<html><body><h1>ESP8266 Pulse Sensor</h1>";
  html += "<p>Visit /pulse to get the current pulse value.</p>";
  html += "</body></html>";
  server.send(200, "text/html", html);
}
void handlePulse() {
  // Get the current pulse value
  int pulse = pulseSensor.getBeatsPerMinute();
  // Send the pulse value to the client
```

```
  String pulseString = String(pulse);
  server.send(200, "text/plain", pulseString);
}
```

- Replace the SSID and PASSWORD placeholders with your network credentials.
- Upload the code to the ESP8266 board.

How It Works

Pulse sensors are small electronic devices that measure an individual's heart rate or pulse. These sensors are used in various applications, including fitness tracking, medical monitoring, and stress management. An ESP8266 board, or nodeMCU, a low-cost, low-power Wi-Fi microcontroller, can be used with a pulse sensor to create a web-connected device to monitor and display pulse data. In this article, we will show you how to use a pulse sensor with an ESP8266 board using Arduino.

Pulse Sensor with ESP8266

What You Will Need

- An ESP8266 board
- A pulse sensor
- Arduino IDE
- USB cable

Getting Started with ESP8266 and Heartbeat Sensor

- Connect the pulse sensor to the **A0** pin of the ESP8266 board. Follow the **circuit diagram**.
- Connect the ESP8266 board to your computer using a USB cable.
- Open the Arduino IDE and select your ESP8266 board from the Tools menu.
- Install the PulseSensorPlayground library by navigating to Sketch > Include Library > Manage Libraries and searching for "PulseSensorPlayground." Click the Install button to install the library.
- Copy and paste the following code into the Arduino IDE:

```
// Include the necessary libraries
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>
#include <PulseSensorPlayground.h>
// Replace with your network credentials
const char* ssid = "your_SSID";
const char* password = "your_PASSWORD";
// Create an instance of the server
```

```
ESP8266WebServer server(80);
// Define the pin for the pulse sensor
const int pulsePin = A0;
// Create an instance of the pulse sensor
PulseSensorPlayground pulseSensor;
void setup() {
  // Start serial communication
  Serial.begin(115200);
  // Connect to Wi-Fi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }
  // Print the IP address
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
  // Initialize the pulse sensor
  pulseSensor.begin();
  pulseSensor.setPin(pulsePin);
  pulseSensor.setThreshold(550);
  // Set up the server
  server.on("/", handleRoot);
  server.on("/pulse", handlePulse);
  server.begin();
}
void loop() {
  // Update the pulse sensor
  pulseSensor.update();
  // Handle incoming client requests
  server.handleClient();
}
void handleRoot() {
  // Send the HTML page to the client
  String html = "<html><body><h1>ESP8266 Pulse Sensor</h1>";
  html += "<p>Visit /pulse to get the current pulse value.</p>";
  html += "</body></html>";
  server.send(200, "text/html", html);
}
void handlePulse() {
  // Get the current pulse value
```

```
  int pulse = pulseSensor.getBeatsPerMinute();
  // Send the pulse value to the client
  String pulseString = String(pulse);
  server.send(200, "text/plain", pulseString);
}
```

- Replace the SSID and PASSWORD placeholders with your network credentials.
- Upload the code to the ESP8266 board.

How It Works

The code uses the PulseSensorPlayground library to read the pulse sensor value and the ESP8266WiFi library to connect to Wi-Fi and set up a web server. The code defines a handleRoot() function that sends an HTML page to the client and a handlePulse() function that sends the current pulse value to the client. The loop() function updates the pulse sensor and handles incoming client requests. When clients visit the server's IP address, they are shown an HTML page that displays a message and a link to the /pulse endpoint. When the /pulse endpoint is visited, the server sends the current pulse value to the client as plain text.

Arduino Code Explanation

```
// Include the necessary libraries
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>
#include <PulseSensorPlayground.h>
```

This section includes the necessary libraries for the ESP8266, including the WiFi library, the Web Server library, and the Pulse Sensor Playground library.

```
// Replace with your network credentials
const char* ssid = "your_SSID";
const char* password = "your_PASSWORD";
```

Here, you need to replace "your_SSID" and "your_PASSWORD" with the credentials for your Wi-Fi network.

```
// Create an instance of the server
ESP8266WebServer server(80);
```

This line creates an instance of the Web Server on port 80.

```
// Define the pin for the pulse sensor
const int pulsePin = A0;
```

This line defines the pin that the Pulse Sensor is connected to. In this case, it is connected to A0.

```
// Create an instance of the pulse sensor
PulseSensorPlayground pulseSensor;
```

This line creates an instance of the PulseSensorPlayground object.

```
void setup() {
  // Start serial communication
  Serial.begin(115200);
  // Connect to Wi-Fi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }
  // Print the IP address
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
  // Initialize the pulse sensor
  pulseSensor.begin();
  pulseSensor.setPin(pulsePin);
  pulseSensor.setThreshold(550);
  // Set up the server
  server.on("/", handleRoot);
  server.on("/pulse", handlePulse);
  server.begin();
}
```

The setup function sets up the ESP8266 by starting serial communication, connecting to Wi-Fi, initializing the pulse sensor, and setting up the server with two routes ("/" and "/pulse") and their corresponding handler functions ("handleRoot" and "handlePulse").

```
void loop() {
  // Update the pulse sensor
  pulseSensor.update();
  // Handle incoming client requests
```

```
  server.handleClient();
}
```

The loop function updates the pulse sensor and handles incoming client requests.

```
void handleRoot() {
  // Send the HTML page to the client
  String html = "<html><body><h1>ESP8266 Pulse Sensor</h1>";
  html += "<p>Visit /pulse to get the current pulse value.</p>";
  html += "</body></html>";
  server.send(200, "text/html", html);
}
```

The "handleRoot" function sends an HTML page to the client with instructions on how to get the current pulse value.

```
void handlePulse() {
  // Get the current pulse value
  int pulse = pulseSensor.getBeatsPerMinute();
  // Send the pulse value to the client
  String pulseString = String(pulse);
  server.send(200, "text/plain", pulseString);
}
```

The "handlePulse" function gets the current pulse value from the pulse sensor and sends it to the client.

Send data from Arduino to ESP2866

In this tutorial we will learn how to make serial communication between Arduino to ESP8266 & ESP8266 to Arduino. Serial communication is required when you want to transfer sensor data or any data from one device to another device, In our case it is ESP8266 NodeMCU and Arduino. Moreover, we will transfer DHT22 Sensor data from Arduino to NodeMCU and NodeMCU to Arduino. Apart from this we will also see how to use software serial library on Arduino and Serial1 on ESP8266 NodeMCU, So that you can transfer data with another serial port, which we will be helpful. If you are doing lots of things on both Arduino and ESP8266 NodeMCU. So let's get started.

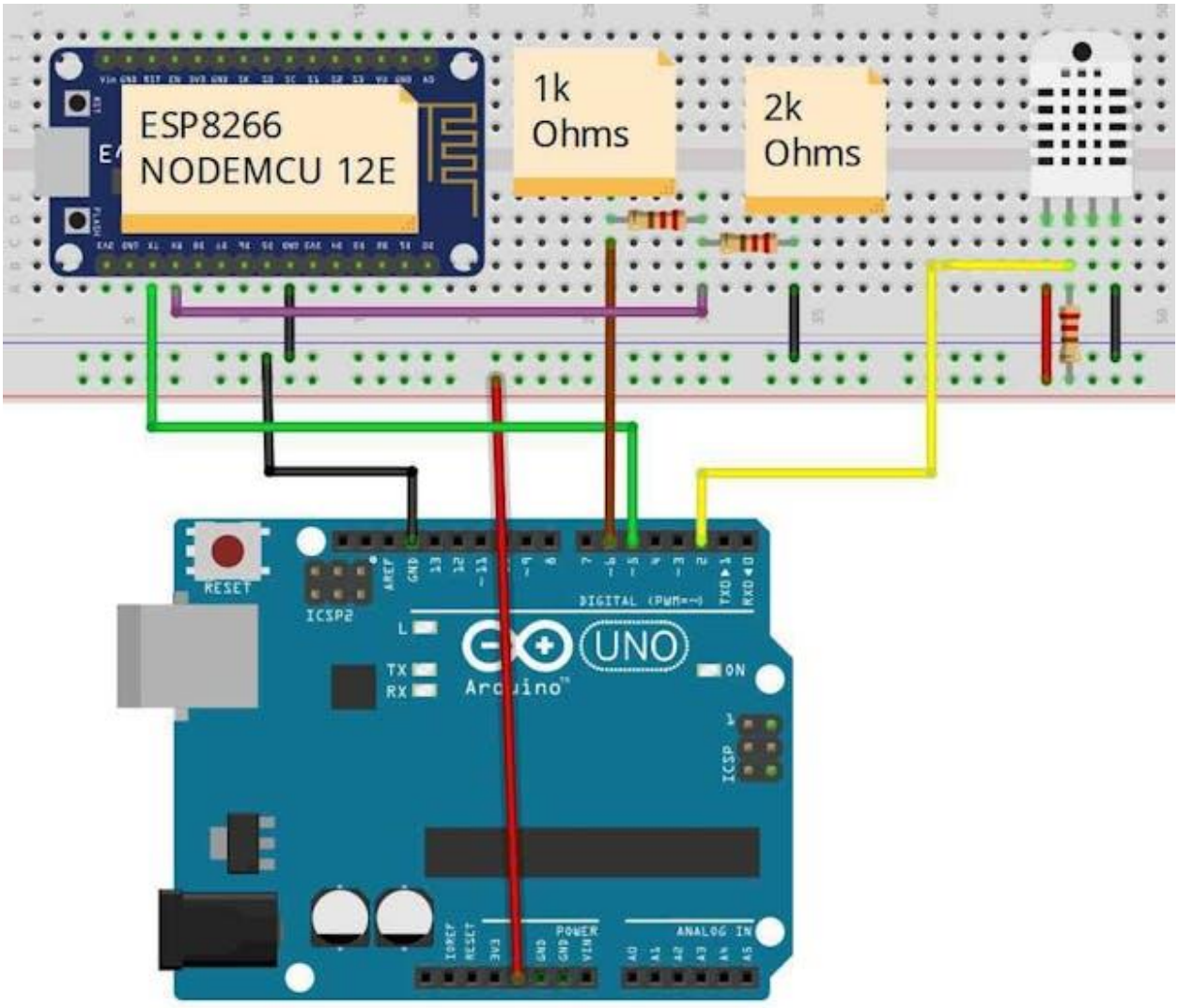
Hardware component

- NodeMCU ESP8266 12E,
- Arduino,
- DHT22 sensor,

- 10K ohms Resistor / 1k & 2k ohms resistor for voltage divider,
- Breadboard,
- Jumper Wire.

We have connected DHT22 sensor on Arduino Uno. It will transfer sensor data to NodeMCU via serial communication.

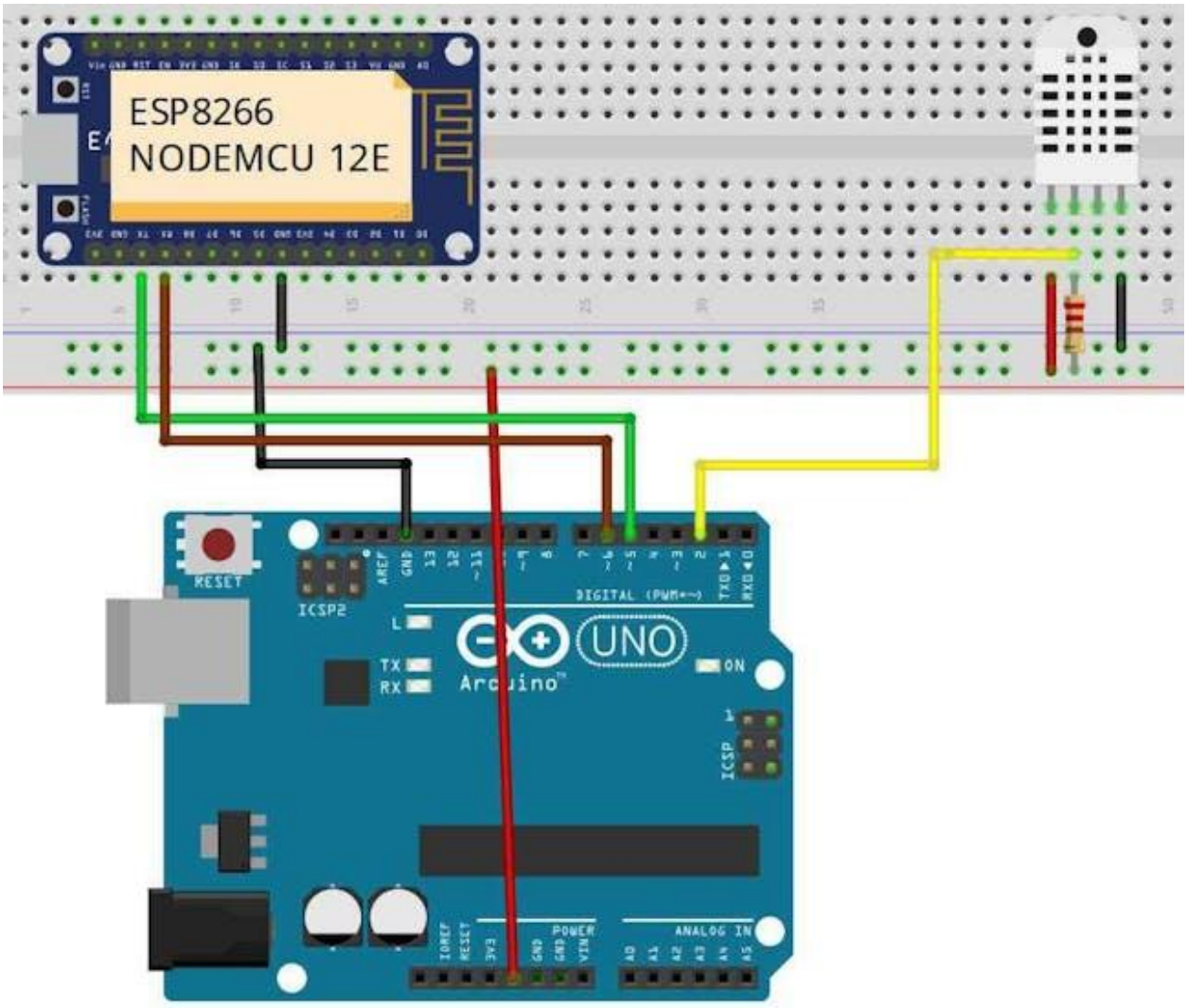
Circuit for Sending data from Arduino to ESP8266 NodeMCU with Voltage Divider:



Note: Before uploading the code, check you have selected board and port is correct. Remove Rx/Tx pins if connected.

If with voltage divider it doesn't work. You can try with direct connection Tx/Rx direct connection. But there is risk of damaging your NodeMCU because Arduino provide 5V and NodeMCU needs 3.3V.

Circuit for Sending data from Arduino to ESP8266 NodeMCU Direct Tx & Rx:



Code for Arduino:

Select Arduino Board and Arduino Port before uploading the code.

```
#include "DHT.h"
#include <SoftwareSerial.h>
#define DHTPIN 2
// Uncomment whatever type you're using!
//#define DHTTYPE DHT11 // DHT 11
#define DHTTYPE DHT22 // DHT 22 (AM2302), AM2321
//#define DHTTYPE DHT21 // DHT 21 (AM2301)
SoftwareSerial espSerial(5, 6);
DHT dht(DHTPIN, DHTTYPE);
String str;
void setup(){
Serial.begin(115200);
espSerial.begin(115200);
```

```
dht.begin();
delay(2000);
}
void loop()
{
float h = dht.readHumidity();
// Read temperature as Celsius (the default)
float t = dht.readTemperature();
Serial.print("H: ");
Serial.print(h);
Serial.print("% ");
Serial.print(" T: ");
Serial.print(t);
Serial.println("C");
str =String("coming from arduino: ")+String("H= ")+String(h)+String("T=
")+String(t);
espSerial.println(str);
delay(1000);
}
```

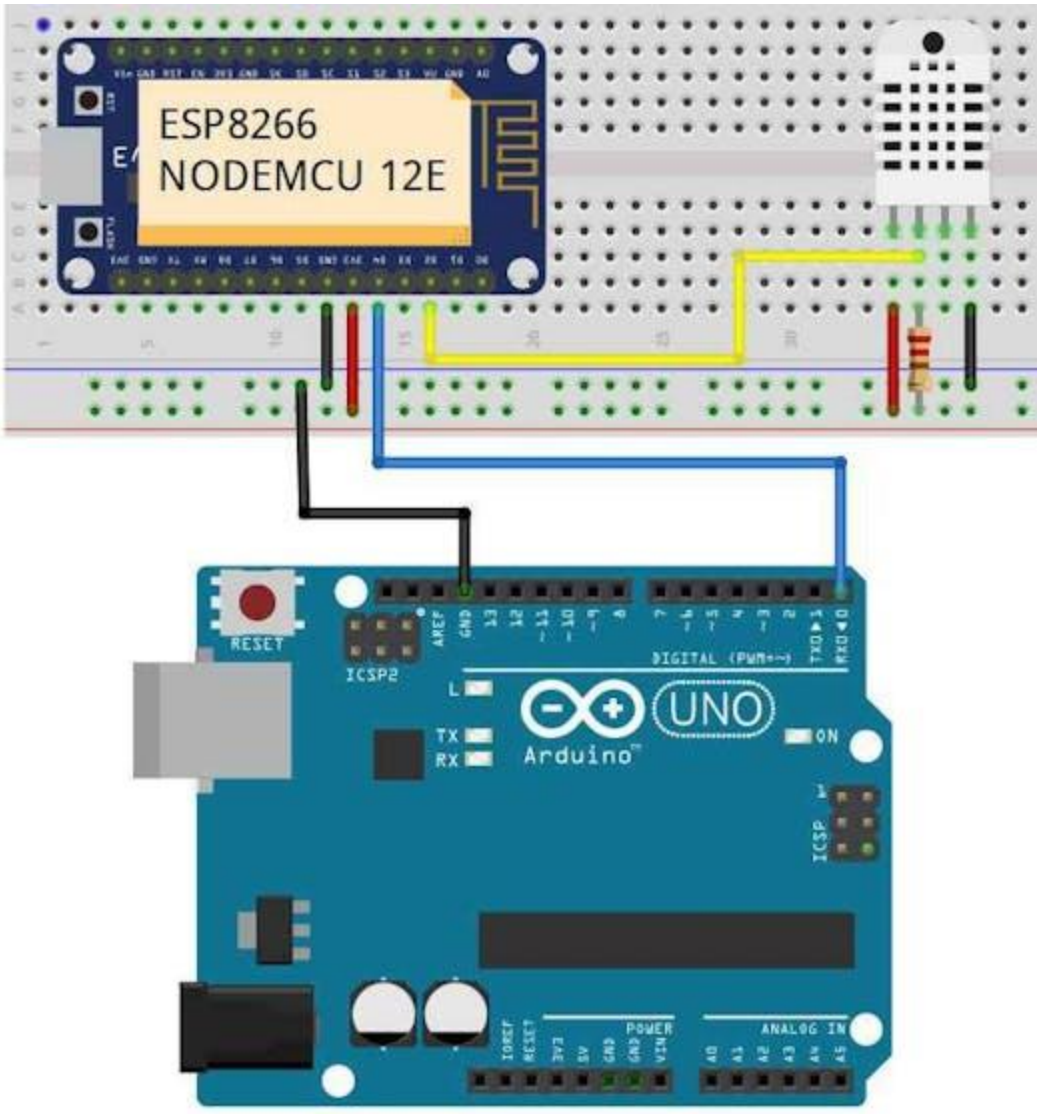
Code for ESP8266:

Select NodeMCU 1.0 Board and ESP8266 Port before uploading the code.

```
void setup() {
// Open serial communications and wait for port to open:
Serial.begin(115200);
while (!Serial) {
; // wait for serial port to connect. Needed for native USB port only
}
}
void loop() { // run over and over
if (Serial.available()) {
Serial.write(Serial.read());
}
}
```

Circuit for Sending Data from ESP8266 NodeMCU to Arduino:

Hardware Circuit



Note: Before uploading the code, check you have selected board and port is correct.

Code for Arduino:

Select Arduino Board and Arduino Port before uploading the code.

```
void setup() {
// Open serial communications and wait for port to open:
Serial.begin(115200);
while (!Serial) {
; // wait for serial port to connect. Needed for native USB port only
}
}
void loop() { // run over and over
if (Serial.available()) {
Serial.write(Serial.read());
}
}
```

Code for ESP8266:

Select NodeMCU 1.0 Board and ESP8266 Port before uploading the code.

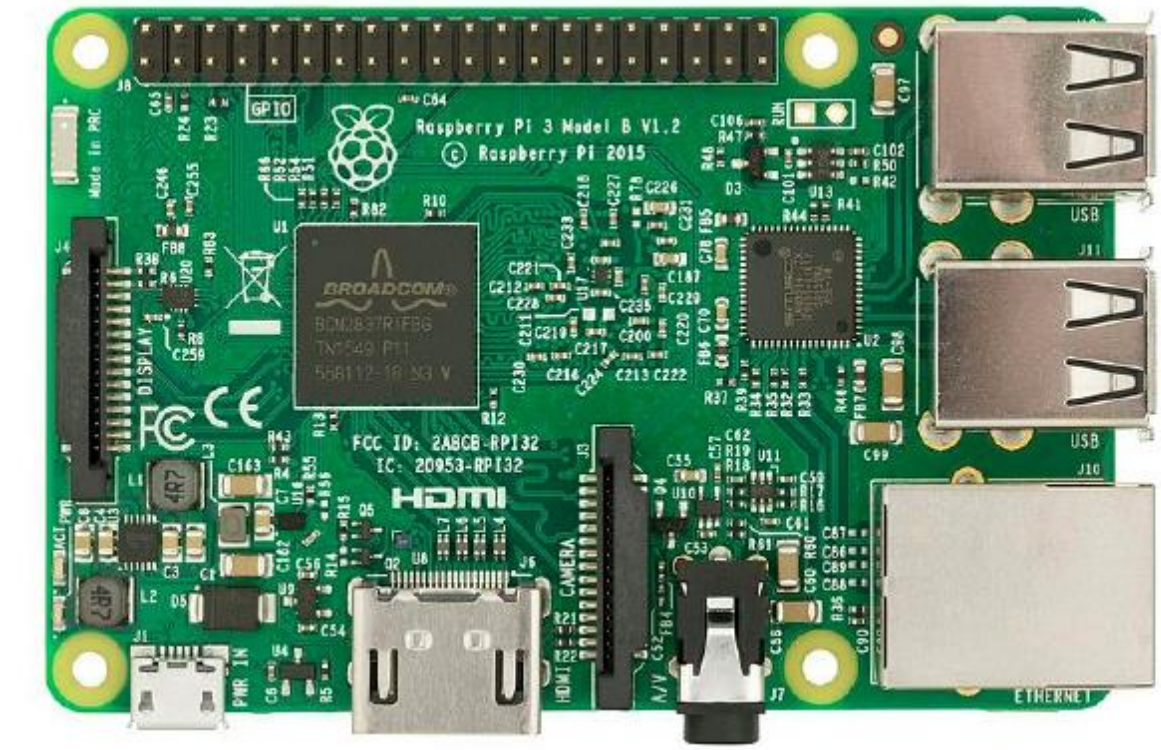
```
#include "DHT.h"
#include <SoftwareSerial.h>
#define DHTPIN 4
// Uncomment whatever type you're using!
//#define DHTTYPE DHT11 // DHT 11
#define DHTTYPE DHT22 // DHT 22 (AM2302), AM2321
//#define DHTTYPE DHT21 // DHT 21 (AM2301)
DHT dht(DHTPIN, DHTTYPE);
String str;
void setup(){
Serial.begin(115200);
Serial1.begin(115200);
dht.begin();
delay(2000);
}
void loop()
{
float h = dht.readHumidity();
// Read temperature as Celsius (the default)
float t = dht.readTemperature();
Serial.print("H: ");
Serial.print(h);
Serial.print("% ");
Serial.print(" T: ");
Serial.print(t);
Serial.println("C");
str =String("coming from ESP8266: ")+String("H= ")+String(h)+String("T= ")+String(t);
Serial1.println(str);
delay(1000);
}
```

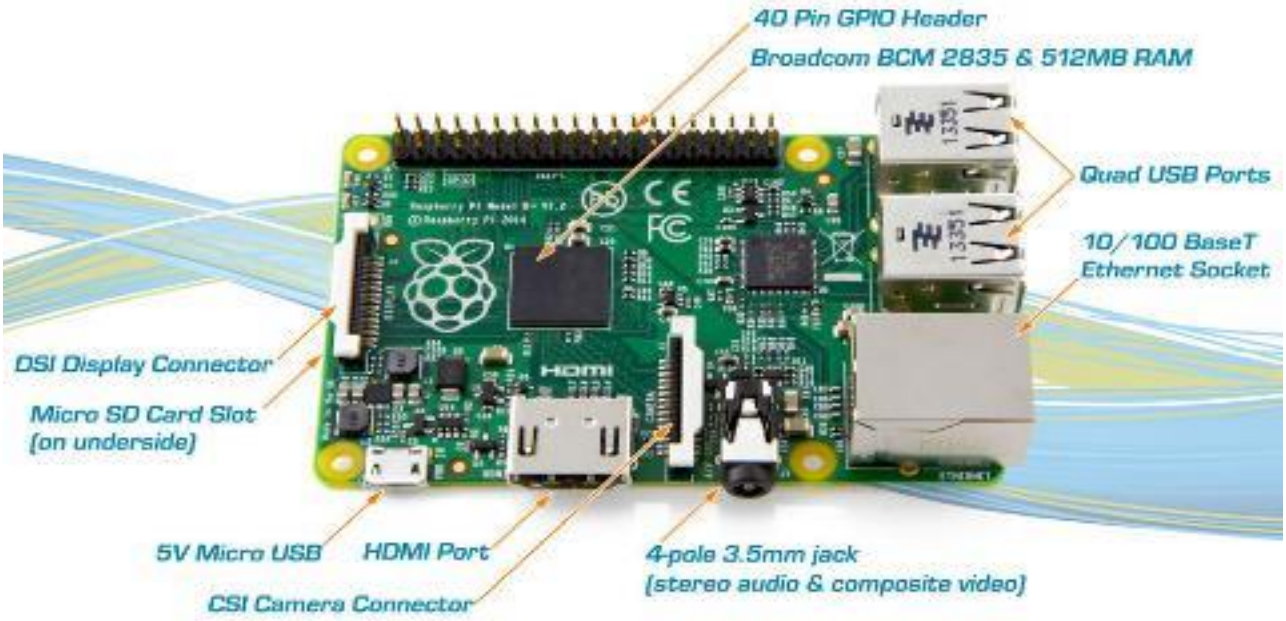
Part 2 : Raspberry pI

A Raspberry Pi is a small computer that lets you build programs that can run from the raspberry pi. It runs on several kinds of operating systems where Raspbien is the most used. The cool thing with the Raspberry Pi is that it can be used with a variety of sensors to gather, interpret, and react to data. I.e it can be connected to a humidity sensor that measures the water content within the soil of you plants, and based on the humidity, and future weather forcast it can make decisions on wheter or not to turn on a water pump that will water your plants.

The Raspberry Pi is a series of small single-board computers developed in the United Kingdom by the Raspberry Pi Foundation to promote the teaching of basic computer science in schools and in developing countries. The original model became far more popular than anticipated, selling outside of its target market for uses such as robotics. Over 5 million Raspberry Pis have been sold before February 2015, making it the best-selling British computer. By November 2016 they had sold 11 million units.

The following figure shows Raspberry Pi 3 Model B.





The first generation (Raspberry Pi 1 Model B) was released in February 2012. It was followed by a simpler and inexpensive model Model A. In 2014, the foundation released a board with an improved design in Raspberry Pi 1 Model B+. These boards are approximately credit-card sized and represent the standard

mainline form-factor. Improved A+ and B+ models were released a year later. A "compute module" was released in April 2014 for embedded applications, and a Raspberry Pi Zero with smaller size and reduced input/output (I/O) and general-purpose input/output (GPIO) capabilities was released in November 2015 for US$5.

The Raspberry Pi 2 which added more RAM was released in February 2015. Raspberry Pi 3 Model B released in February 2016, is bundled with on-board WiFi, Bluetooth and USB boot capabilities. As of January 2017, Raspberry Pi 3 Model B is the newest mainline Raspberry Pi. Raspberry Pi boards are priced between US$5–35. As of 28 February 2017, the Raspberry Pi Zero W was launched, which is identical to the Raspberry Pi Zero, but has the Wi-Fi and Bluetooth functionality of the Raspberry Pi 3 for US$10.

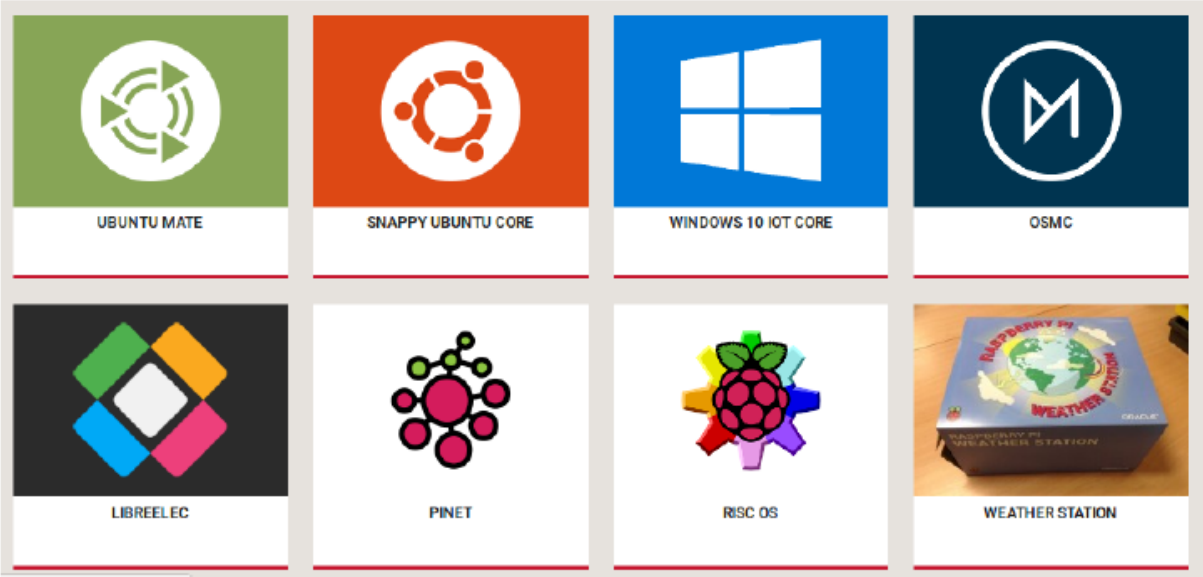
Raspberry feature

- All models feature a Broadcom system on a chip (SoC), which includes an ARM compatible central processing unit (CPU) and an on-chip graphics processing unit (GPU, a VideoCore IV).
- CPU speed ranges from 700 MHz to 1.2 GHz for the Pi 3 and on board memory range from 256 MB to 1 GB RAM.
- Secure Digital (SD) cards are used to store the operating system and program memory in either the SDHC or MicroSDHC sizes.
- Most boards have between one and four USB slots, HDMI and composite video output, and a 3.5 mm phono jack for audio.
- Lower level output is provided by a number of GPIO pins which support common protocols like I^2C.
- The B-models have an 8P8C Ethernet port and the Pi 3 and Pi Zero W have on board Wi-Fi 802.11n and Bluetooth.

Operating System

The Foundation provides Raspbian, a Debian based Linux distribution for download, as well as third party Ubuntu, Windows 10 IOT Core, RISC OS, and specialised media center distributions. It promotes Python and Scratch as the main programming language, with support for many other languages. The default firmware is closed source, while an unofficial open source is available.

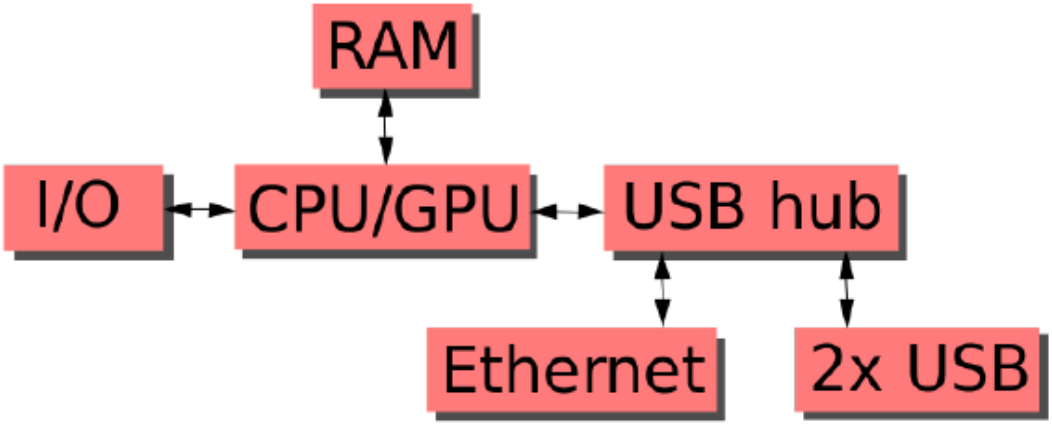
The Raspberry Pi Foundation recommends the use of Raspbian, a Debian-based Linux operating system. Other third party operating systems available via the official website include Ubuntu MATE, Snappy Ubuntu Core, Windows 10 IoT Core, RISC OS and specialised distributions for the Kodi media center and classroom management.



Linux Based Operating Systems

- Android Things
- Arch Linux
- OpenSuse
- Raspberry Pi Fedora Remix
- Pidora
- Gentoo Linux
- CentOS Raspberry Pi
- Kali Linux
- Slackware ARM
- Puppy Linux
- RISC OS Pi
- FreeBSD
- NetBSD
- Windows 10 IOT Core
- Haiku
- HelenOS

Raspberry pi Hardware



This block diagram depicts Models A, B, A+, and B+. Model A, A+, and the Pi Zero lack the Ethernet and USB hub components. The Ethernet adapter is internally connected to an additional USB port. In Model A, A+, and the Pi Zero, the USB port is connected directly to the system on a chip (SoC). On the Pi 1 Model B+ and later models the USB/Ethernet chip contains a fivepoint USB hub, of which four ports are available, while the Pi 1 Model B only provides two. On the Pi Zero, the USB port is also connected directly to the SoC, but it uses a micro USB (OTG) port.

Processor

The Broadcom BCM2835 SoC used in the first generation Raspberry Pi is somewhat equivalent to the chip used in first modern generation smartphones (its CPU is an older ARMv6 architecture), which includes a 700 MHz ARM1176JZF-S processor, VideoCore IV graphics processing unit (GPU), and RAM. It has a level 1 (L1) cache of 16 KB and a level 2 (L2) cache of 128 KB. The level 2 cache is used primarily by the GPU. The SoC is stacked underneath the RAM chip, so only its edge is visible.

The Raspberry Pi 2 uses a Broadcom BCM2836 SoC with a 900 MHz 32-bit quad-core ARM Cortex-A7 processor, with 256 KB shared L2 cache. The Raspberry Pi 3 uses a Broadcom BCM2837 SoC with a 1.2 GHz 64-bit quad-core ARM Cortex-A53 processor, with 512 KB shared L2 cache

The Raspberry Pi 3, with a quad-core Cortex-A53 processor, is described as 10 times the performance of a Raspberry Pi 1. This was suggested to be highly dependent upon task threading and instruction set use. Benchmarks showed the Raspberry Pi 3 to be approximately 80% faster than the Raspberry Pi 2 in parallelized tasks. Raspberry Pi 2 includes a quad-core Cortex-A7 CPU running at 900 MHz and 1 GB RAM. It is described as 4–6 times more powerful than its predecessor. The GPU is

identical to the original. In parallelized benchmarks, the Raspberry Pi 2 could be up to 14 times faster than a Raspberry Pi 1 Model B+.

RAM

The Raspberry Pi 2 and the Raspberry Pi 3 have 1 GB of RAM. The Raspberry Pi Zero and Zero W have 512 MB of RAM.

Networking

The Model A, A+ and Pi Zero have no Ethernet circuitry and are commonly connected to a network using an external user-supplied USB Ethernet or Wi-Fi adapter. On the Model B and B+ the Ethernet port is provided by a built-in USB Ethernet adapter using the SMSC LAN9514 chip. The Raspberry Pi 3 and Pi Zero W (wireless) are equipped with 2.4 GHz WiFi 802.11n (150 Mbit/s) and Bluetooth 4.1 (24 Mbit/s) based on Broadcom BCM43438 FullMAC chip with no official support for Monitor mode but implemented through unofficial firmware patching and the Pi 3 also has a 10/100 Ethernet port.

Peripherals

The Raspberry Pi may be operated with any generic USB computer keyboard and

mouse. It may also be used with USB storage, USB to MIDI converters, and virtually any other device/component with USB capabilities. Other peripherals can be attached through the various pins and connectors on the surface of the Raspberry Pi.

Video Capabilities

The video controller can emit standard modern TV resolutions, such as HD and Full HD, and higher or lower monitor resolutions and older standard CRT

TV resolutions.

The following figure shows the difference between raspberry pi generation

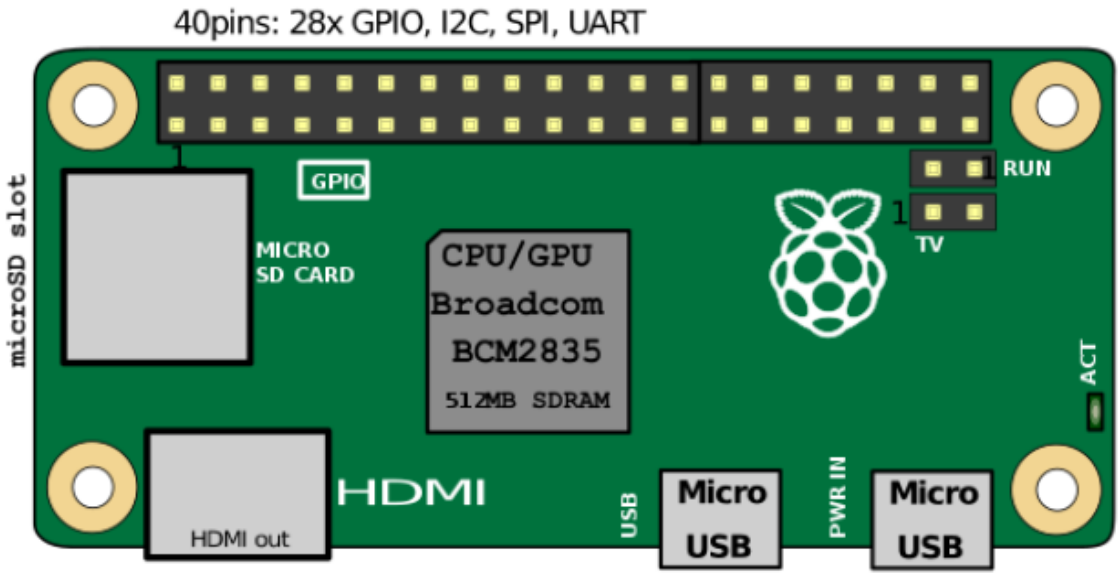
	Raspberry Pi 1 Model A	Raspberry Pi 1 Model A+	Raspberry Pi 1 Model B	Raspberry Pi 1 Model B+	Raspberry Pi 2 Model B	Raspberry Pi 3 Model B	Raspberry Pi Zero
USB 2.0 Ports	1	1	2	4	4	4	1 (Micro-USB)
Ethernet	None	None	10/100 Mbit/s	10/100 Mbit/s	10/100 Mbit/s	10/100 Mbit/s	None
Bluetooth	None	None	None	None	None	4.1	None
WiFi	None	None	None	None	None	802.11n	None
Audio In	I²S	I²S	I²S	I²S	I²S	I²S	I²S
Audio Out	I²S, analog (3.5mm jack), digital (HDMI)	I²S, analog (3.5mm jack), digital (HDMI)	I²S, analog (3.5mm jack), digital (HDMI)	I²S, analog (3.5mm jack), digital (HDMI)	I²S, analog (3.5mm jack), digital (HDMI)	I²S, analog (3.5mm jack), digital (HDMI)	Digital (mini-HDMI), analog GPIO PWM
Video In	CSI Camera Connector	CSI Camera Connector	CSI Camera Connector	CSI Camera Connector	CSI Camera Connector	CSI Camera Connector	None
Video Out	HDMI, Composite (RCA)	HDMI, Composite (TRRS)	HDMI, Composite (RCA)	HDMI, Composite (TRRS)	HDMI, Composite (TRRS)	HDMI, Composite (TRRS)	Mini-HDMI, GPIO Composite
External Storage	SD	MicroSD	SD	MicroSD	MicroSD	MicroSD	MicroSD

History of raspberry pi

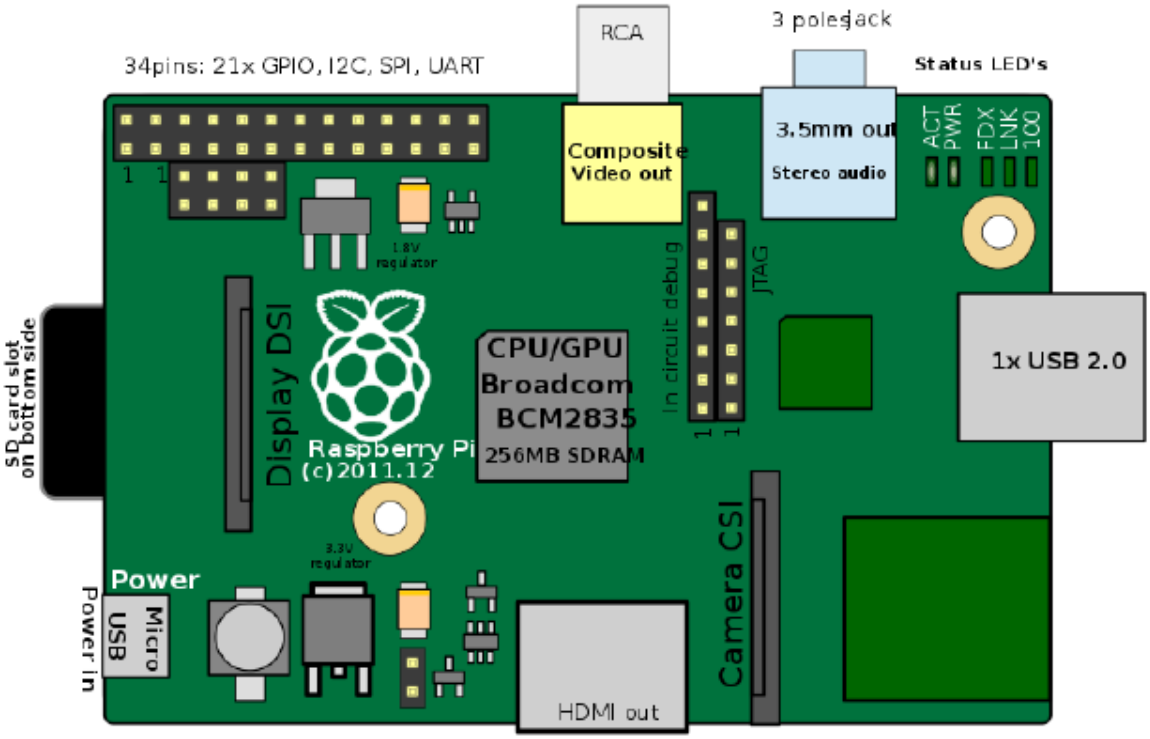
In 2006, early concepts of the Raspberry Pi were based on the Atmel ATmega644 microcontroller. Its schematics and PCB layout are publicly available. Foundation trustee Eben Upton assembled a group of teachers, academics and computer enthusiasts to devise a computer to inspire children. The computer is inspired by Acorn's BBC Micro of 1981. The Model A, Model B and Model B+ names are references to the original models of the British educational BBC Micro computer, developed by Acorn Computers. The first ARM prototype version of the computer was mounted in a package the same size as a USB memory stick. It had a USB port on one end and an HDMI port on the other.

The Foundation's goal was to offer two versions, priced at US$25 and $35. They started accepting orders for the higher priced Model B on 29 February 2012, the lower cost Model A on 4 February 2013. and the even lower cost (US$20) A+ on 10 November 2014. On 26 November 2015, the cheapest Raspberry Pi yet, the Raspberry Pi Zero, was launched at US$5 or £4.

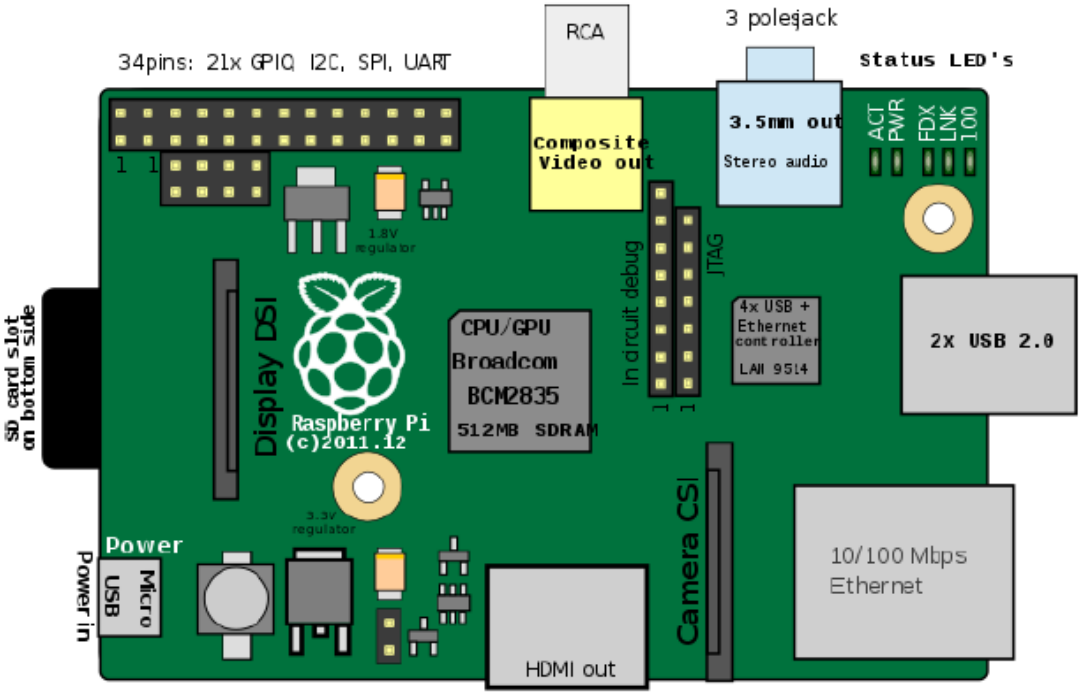
The following figure shows raspberry pi zero.



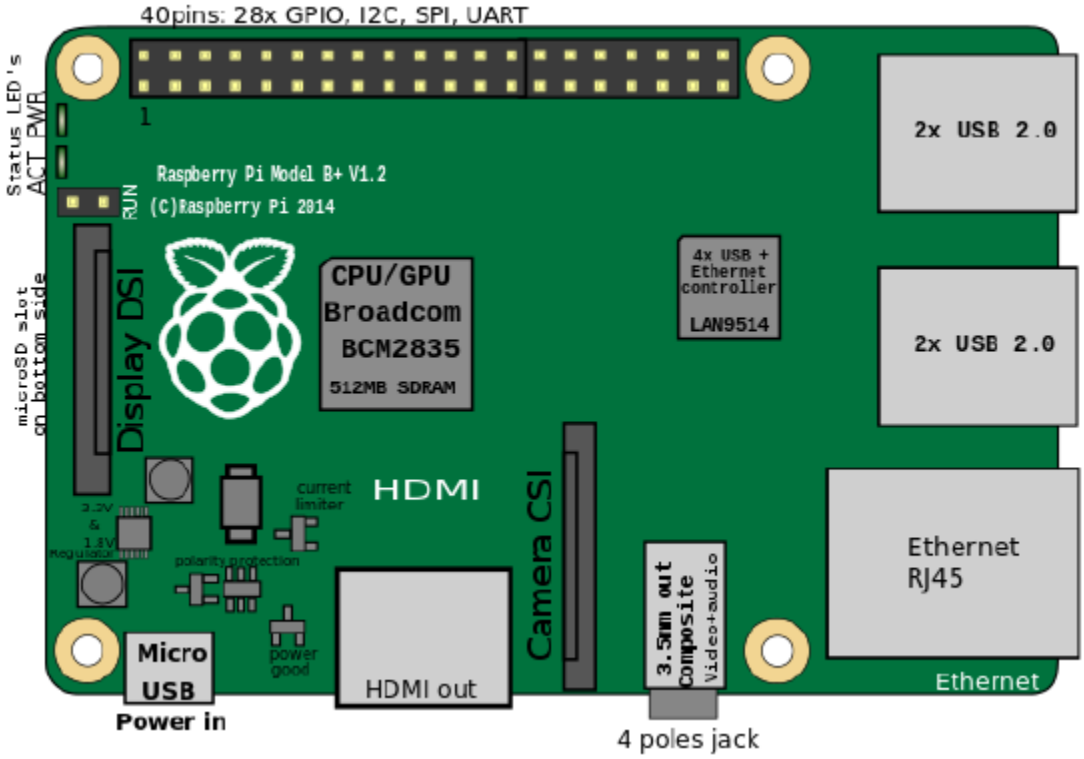
The following figure shows raspberry pi model A.



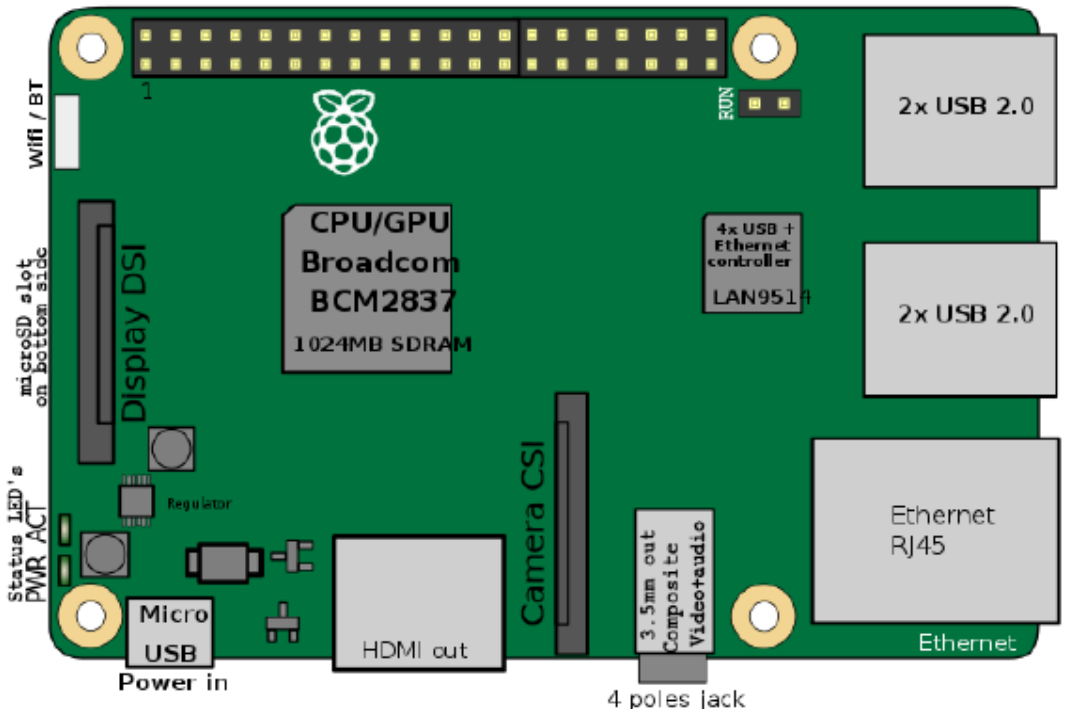
The following figure shows raspberry pi model B.



The following figure shows raspberry pi model B+ and 2B



The following figure shows raspberry pi model 3B

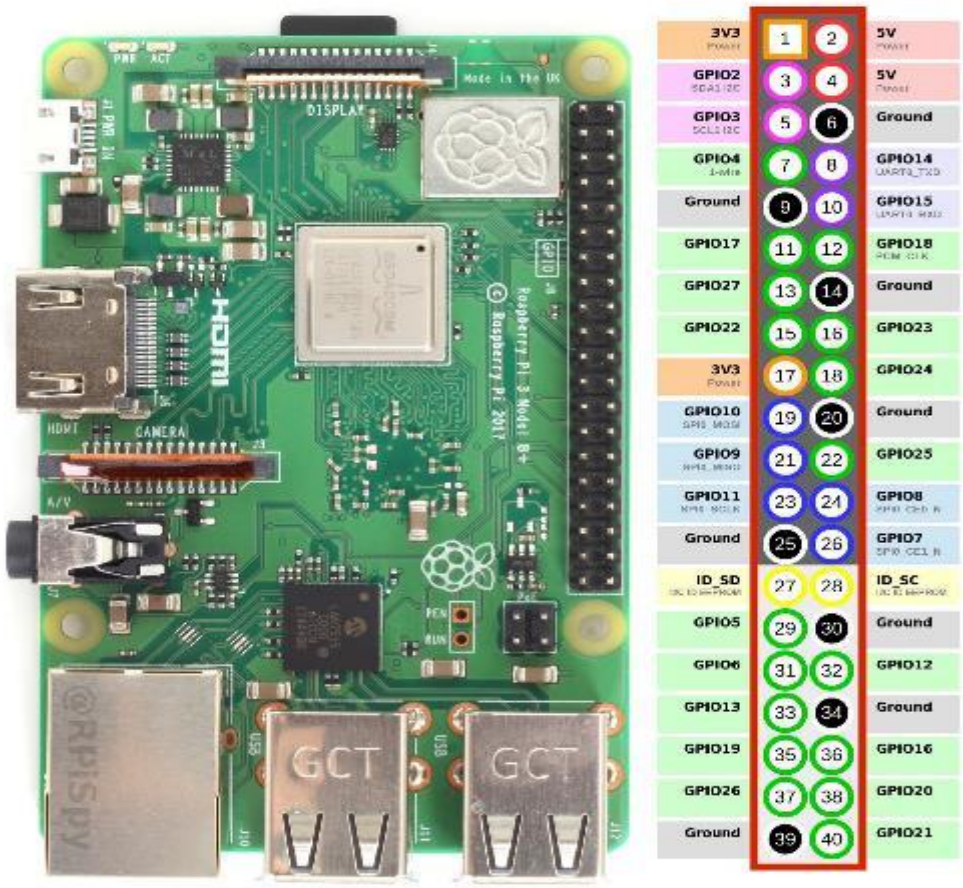


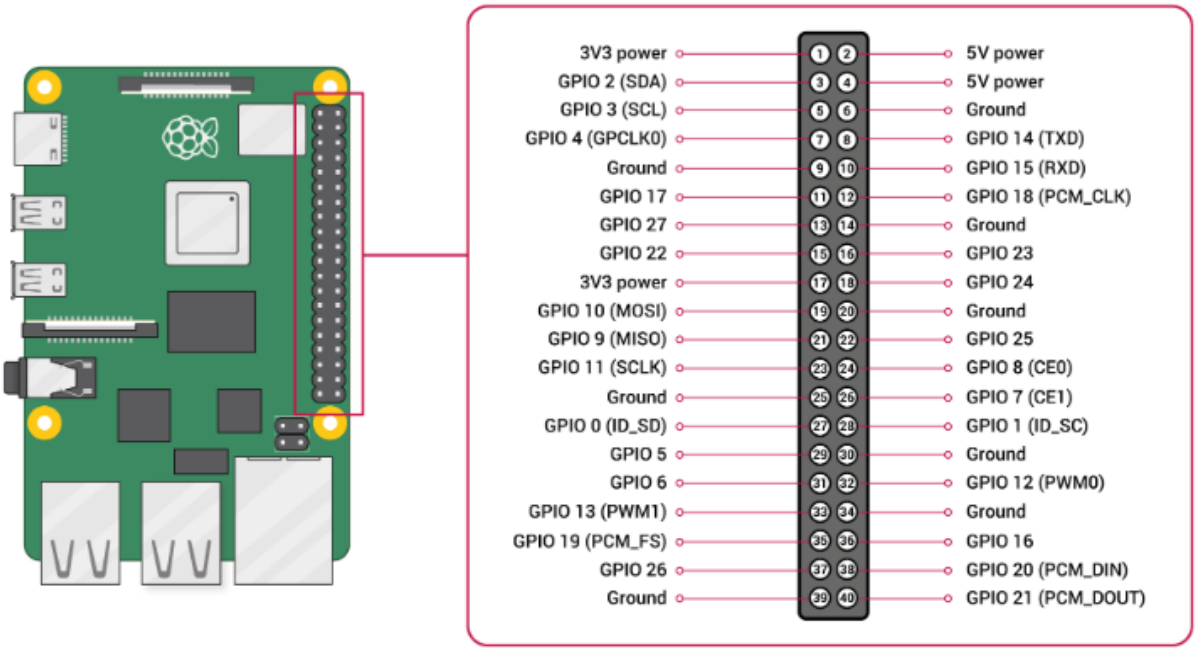
Accessories

- Camera – On 14 May 2013, the foundation and the distributors RS Components & Premier Farnell/Element 14 launched the Raspberry Pi camera board alongside a firmware update to accommodate it. The camera board is shipped with a flexible flat cable that plugs into the CSI connector which is located between the Ethernet and HDMI ports.
- Gertboard – A Raspberry Pi Foundation sanctioned device, designed for educational purposes, that expands the Raspberry Pi's GPIO pins to allow interface with and control of LEDs, switches, analog signals, sensors and other devices. It also includes an optional Arduino compatible controller to interface with the Pi.
- Infrared Camera – In October 2013, the foundation announced that they would begin producing a camera module without an infrared filter, called the Pi NoIR.
- HAT (Hardware Attached on Top) expansion boards – Together with the Model B+, inspired by the Arduino shield boards, the interface for HAT boards was devised by the Raspberry Pi Foundation.

Raspberry pi GPIO

The following figure shows the GPIO for raspberry pi 3 model A.





The following figure shows the GPIO for raspberry pi 3 model A.

To understand the GPIO, watch the following video

https://www.youtube.com/watch?v=7UVd-k-S7Sk&list=PLMYF6NkLrdN8ZB4VXJZNCNoBwOBEuGRcF&index=16

You can watch this video to conclude the pervious information

https://www.youtube.com/watch?v=Hatt8g9L9DA&list=PLIzymmJrCaIW0vezktmbTeXJtI6p03bCI

We will use python programming language and python 3 IDE to configure the sensor using raspberry pi. So, you can watch this video to undustrand some concepts of python.

https://www.youtube.com/watch?v=Hatt8g9L9DA&list=PLIzymmJrCaIW0vezktmbTeXJtI6p03bCI

Experiment 1 : How to setup Raspbian operating system on raspberry pi and how to connect the raspberry pi from laptop

Objective: setup Raspbian operating system on memory and connect to raspberry pi remotely.

We will setup Raspbian operating system. The following video shows how to setup Raspbian operating system. Also this video shows how to connect to raspberry bp remotely

https://www.youtube.com/watch?v=uYgNe9Kafck&list=PLIzymmJrCaIW0vezktmbTeXJtI6p03bCI&index=3

or you can watch the following video

https://www.youtube.com/watch?v=3cxal8JAjlY

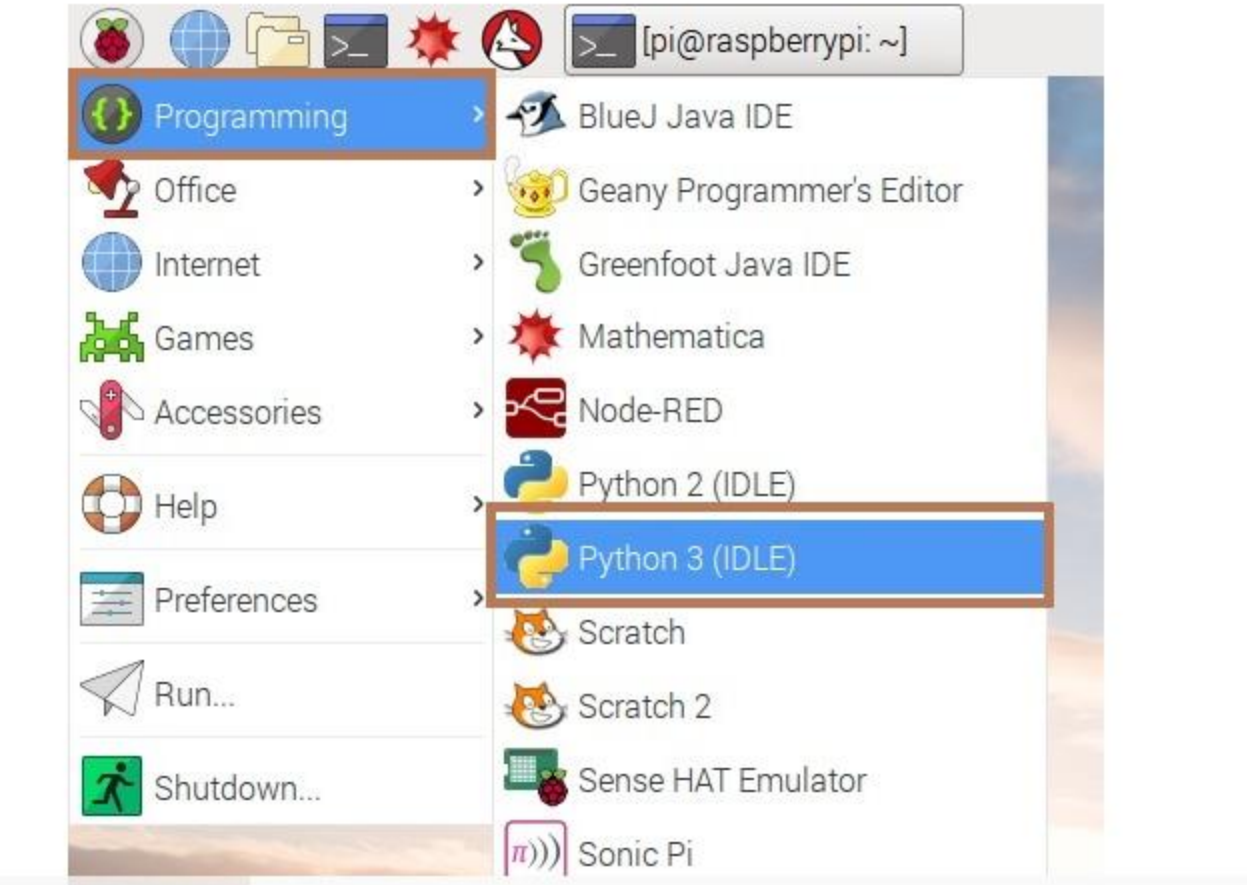
Note: you can watch the following videos to done the pervious step in two different steps.
The first step to setup Raspbian O.S. first using a monitor and raspberry pi. To do this shows the following videos (don't worry the videos are very short)
https://www.youtube.com/watch?v=ETVZOcstN2I&list=PLMYF6NkLrdN8ZB4VXJZNCNoBwOBEuGRcF&index=6
https://www.youtube.com/watch?v=XSsJZBjgHvw&list=PLMYF6NkLrdN8ZB4VXJZNCNoBwOBEuGRcF&index=8
https://www.youtube.com/watch?v=UMTMRCQiTCE&list=PLMYF6NkLrdN8ZB4VXJZNCNoBwOBEuGRcF&index=9
Therefore, you can connect to your rassperry pi from your pc remotely using the VNC program. To do this shows the following videos (don't worry the videos are very short)
https://www.youtube.com/watch?v=P7nNTb5yicM&list=PLMYF6NkLrdN8ZB4VXJZNCNoBwOBEuGRcF&index=10
https://www.youtube.com/watch?v=iB9UWr3dQng&list=PLMYF6NkLrdN8ZB4VXJZNCNoBwOBEuGRcF&index=11
https://www.youtube.com/watch?v=SVwxXa5D_50&list=PLMYF6NkLrdN8ZB4VXJZNCNoBwOBEuGRcF&index=12
https://www.youtube.com/watch?v=hPGOHQOBYxk&list=PLMYF6NkLrdN8ZB4VXJZNCNoBwOBEuGRcF&index=13
https://www.youtube.com/watch?v=6IpH1hGPQNo&list=PLMYF6NkLrdN8ZB4VXJZNCNoBwOBEuGRcF&index=14
https://www.youtube.com/watch?v=-EKMRbBGCak&list=PLMYF6NkLrdN8ZB4VXJZNCNoBwOBEuGRcF&index=15

Experiment 2 : control the LED using raspberry bi

Objective: Turn on or off the LED.

Before you make experiment open python 3 IDE as following figure.

Raspbian Menu Icon >> Programming >> Python 3 (IDLE).



To create a new file in IDLE, You can click on File and then New File in IDLE's menu bar.

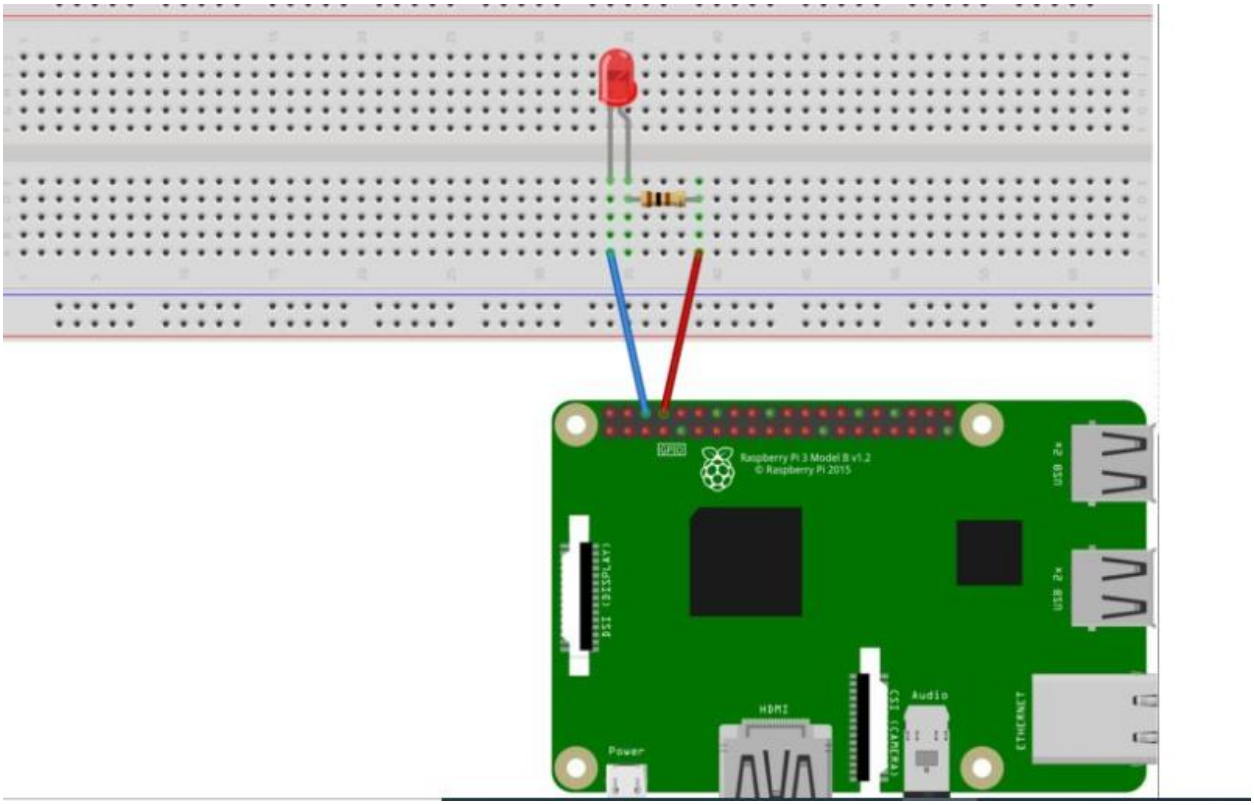
Create a new file by clicking File >> New File

Save the new file by clicking File >> Save. Save the file as blinking_led.py

We will need the following tools to complete the project:

- Raspberry Pi 3 setup with monitor and USB Mouse & Keyboard
- Bread board breadboard
- Jumper wires for easy hookup
- Resistor pack (80 Ohm)
- Red LED

Circuit diagram



To install the Python library open a terminal and execute the following.

```
$ sudo apt update && sudo apt upgrade


$ sudo apt-get install python-rpi.gpio python3-rpi.gpio
```

Code

```
import RPi.GPIO as GPIO # Import Raspberry Pi GPIO library
from time import sleep # Import the sleep function from the time module

GPIO.setwarnings(False) # Ignore warning for now
GPIO.setmode(GPIO.BOARD) # Use physical pin numbering
GPIO.setup(14, GPIO.OUT, initial=GPIO.LOW) # Set pin 18 to be an output pin and set
initial value to low (off)

while True: # Run forever
```

```
GPIO.output(14, GPIO.HIGH) # Turn on
sleep(1) # Sleep for 1 second
GPIO.output(18, GPIO.LOW) # Turn off
sleep(1) # Sleep for 1 second
```

With our program finished, save it as blinking_led.py and run it either inside your IDE or in the console with:

```
$ python blinking_led.py
```

To do the pervious program, you may watch one of the following videos:

1- https://www.youtube.com/watch?v=1cj1FBzf99w&list=PLIzymmJrCaIW0vezktmbTeXJtI6p03bCI&index=4

2- https://www.youtube.com/watch?v=XZNieNJMuhc&list=PLMYF6NkLrdN8ZB4VXJZNCNoBwOBEuGRcF&index=21

3- https://www.youtube.com/watch?v=GLhLPrFDkFA

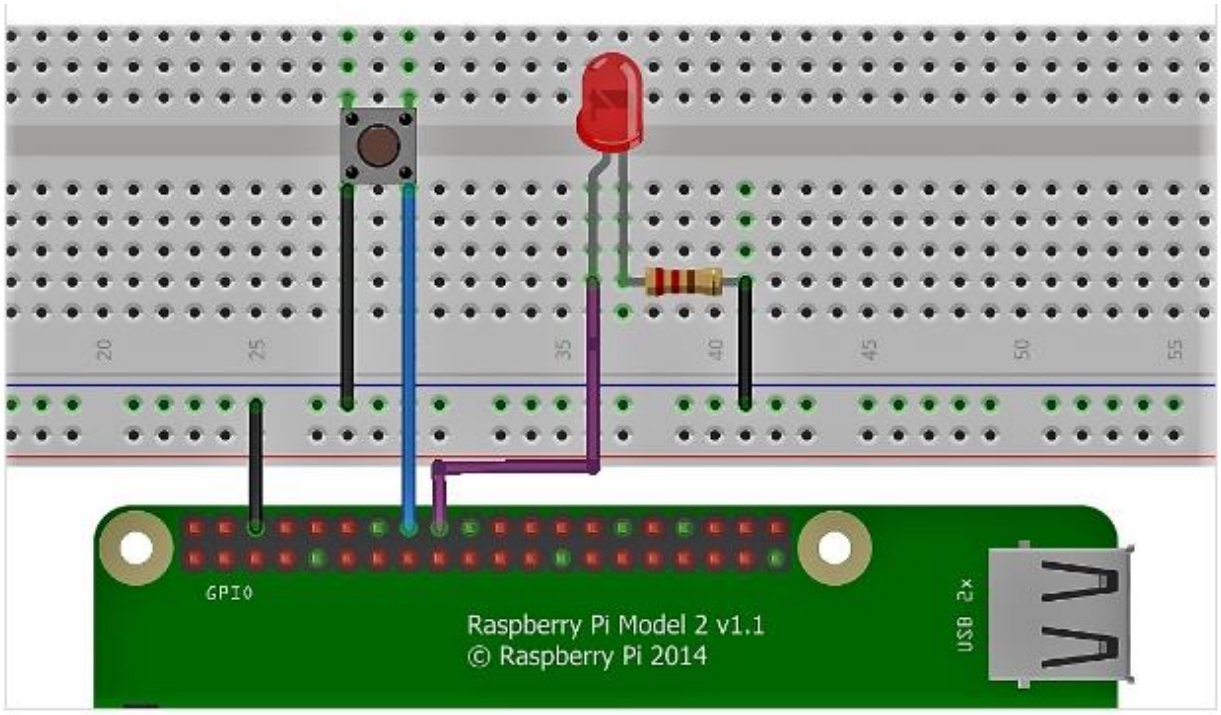
Experment 3 : control LED using push button

Objective: turn on or off led using push button.

We will need the following tools to complete the project:

- Raspberry Pi
- Led
- Pushbutton
- BreadBoard
- Jumper Wires

Circuit diagram



Step 2: Install Libraries for GPIO

```
$ sudo apt-get update

$ sudo apt-get install python-rpi.gpio python3-rpi.gpio
```

Step 3: Python Program

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)

GPIO.setup(23, GPIO.IN, pull_up_down=GPIO.PUD_UP)#Button to GPIO23
GPIO.setup(24, GPIO.OUT)  #LED to GPIO24

try:
    while True:
         button_state = GPIO.input(23)
         if button_state == False:
             GPIO.output(24, True)
             print('Button Pressed...')
             time.sleep(0.2)
         else:
             GPIO.output(24, False)
except:
    GPIO.cleanup()
```

Watch the following video.

https://www.youtube.com/watch?v=1N4jUr3m2as&list=PLIzymmJrCaIW0vezktmbTeXJtI6p03bCI&index=5

Experiment 4 : Control buzzer using raspberry bi

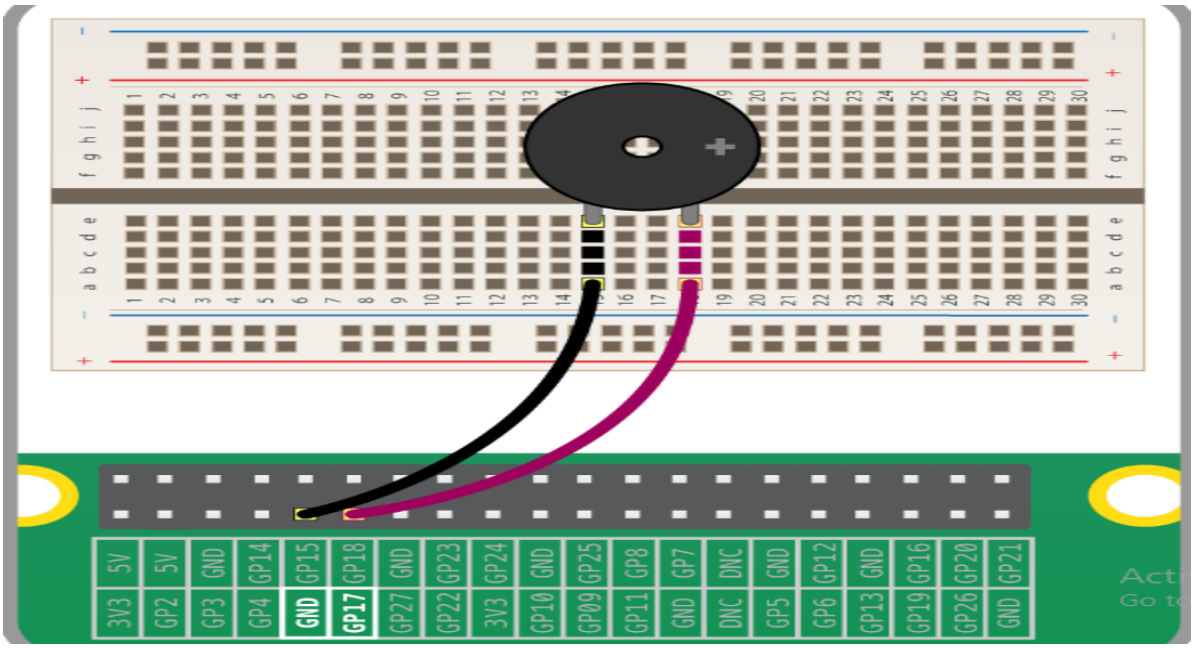
Objective: turn buzzer on and off.

An *active* buzzer can be connected just like an LED, but as they are a little more robust, you won't be needing a resistor to protect them.

We will need the following tools to complete the project:

- Raspberry Pi 3 setup with monitor and USB Mouse & Keyboard
- Buzzer

Circuit diagram:



Code

```
import RPi.GPIO as GPIO
from time import sleep
#Disable warnings (optional)
GPIO.setwarnings(False)
#Select GPIO mode
GPIO.setmode(GPIO.BCM)
#Set buzzer - pin 23 as output
buzzer=17
GPIO.setup(buzzer,GPIO.OUT)
#Run forever loop
while True:
      GPIO.output(buzzer,GPIO.HIGH)
      print ("Beep")
      sleep(0.5) # Delay in seconds
      GPIO.output(buzzer,GPIO.LOW)
      print ("No Beep")
      sleep(0.5)
```

watching this video

https://www.youtube.com/watch?v=VJkJSfvteGo

Experiment 5 : Traffic light using raspberry bi

Objective: use a 3 LEDs to make them blink as Traffic light.

We will need the following tools to complete the project:

- Raspberry Pi 3 setup with monitor and USB Mouse & Keyboard
- 3 LED
- 3 resistors (80 ohm)

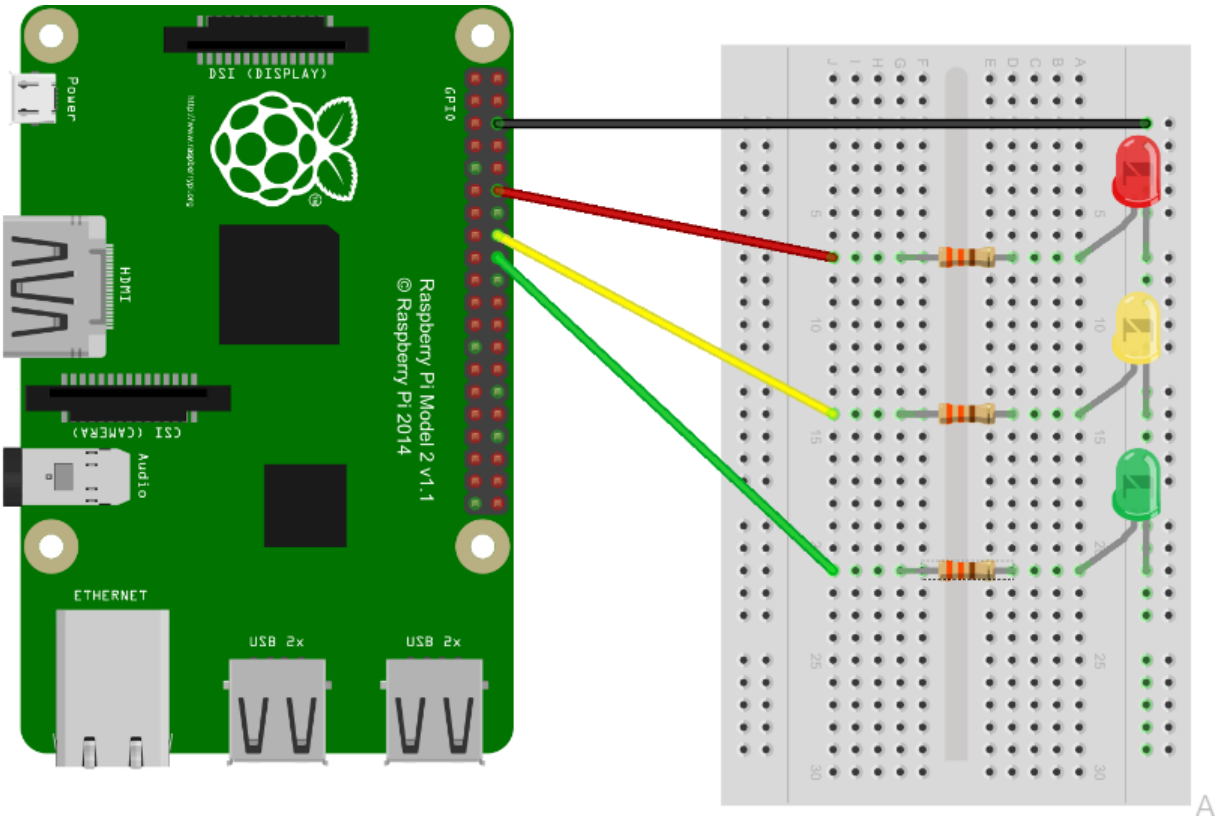
To get started, you'll need to place all the components on the breadboard and connect them to the appropriate GPIO pins on the Raspberry Pi.

First, you need to understand how each component is connected:

- An LED requires 1 ground pin and 1 GPIO pin, with a current limiting resistor
- A buzzer requires 1 ground pin and 1 GPIO pin

Each component requires its own individual GPIO pin, but components can share a ground pin. We will use the breadboard to enable this.

Place the components on the breadboard and connect them to the Raspberry Pi GPIO pins, according to the following diagram:



Observe the following table, showing which GPIO pin each component is connected to:

Component	GPIO pin
Red LED	18
Amber LED	23
Green LED	24

Code

Create a new file by clicking New.

Save the new file straight away by clicking Save; name the file trafficlights.py.

Enter the following code:

```
import RPi.GPIO as GPIO

import time

import random

#

# Use BCM e.g. identify pins by name

#


GPIO.setmode(GPIO.BCM)

#

# Set warnings off to suppress re-initialisation messages

#

GPIO.setwarnings(False)

#

# Set a variable to the GPIO Pin name we what to use

# one for each of the 3 leds

#

RED = 18

AMBER = 23

GREEN = 24

#

# Initialise the GPIO Ports to OUTPUT
```

```
#
GPIO.setup(RED , GPIO.OUT)
GPIO.setup(AMBER , GPIO.OUT)
GPIO.setup(GREEN , GPIO.OUT)
#
# Stop Function - Red light only
#
def Stop():
        global RED, AMBER, GREEN
        print "STOP"
        GPIO.output(RED , GPIO.HIGH)
        GPIO.output(AMBER, GPIO.LOW)
        GPIO.output(GREEN, GPIO.LOW)
#
# Get Ready to go Function - Red and amber lit
#
def GetReadyToGo():
        global RED, AMBER, GREEN
        print "Get Ready"
        GPIO.output(RED , GPIO.HIGH)
        GPIO.output(AMBER, GPIO.HIGH)
        GPIO.output(GREEN, GPIO.LOW)
#
# Go function - Green Only
#
def Go():
        global RED, AMBER, GREEN
        print "Go !"
        GPIO.output(RED , GPIO.LOW)
        GPIO.output(AMBER, GPIO.LOW)
```

```
        GPIO.output(GREEN, GPIO.HIGH)
#
# Get Ready To Stop function - Amber Only
#
def GetReadyToStop():
        global RED, AMBER, GREEN
        print "Warning lights changing"
        GPIO.output(RED , GPIO.LOW)
        GPIO.output(AMBER, GPIO.HIGH)
        GPIO.output(GREEN, GPIO.LOW)
#
# Run through the traffic light secquence forever
#
# If ctrl-C is pressed then quit
#
try:
        while (True):

                # stop and pause between 3 and 5 seconds
                Stop();
                pauseTime = random.randint(3,5)
                time.sleep(pauseTime)

                GetReadyToGo();
                time.sleep(1)

                # go and pause between 3 and 5 seconds
                Go();
                pauseTime = random.randint(3,5)
                time.sleep(pauseTime)
```

```
                #the amber light is lit for 2.9 seconds
                GetReadyToStop();
                time.sleep(2.9)

except KeyboardInterrupt:
        #
        # clean up on finish
        #
        print "clean Up"
        GPIO.cleanup()
```

you can also watch the following video

https://www.youtube.com/watch?v=Rp5gBbAIlW4&list=PLMYF6NkLrdN8ZB4VXJZNCNoBwOBEuGRcF&index=23

Experiment 6 : control Traffic light using push button

Objective: turn on a 3 LEDs and one buzzer when push button is pressed.

We will need the following tools to complete the project:

- Raspberry Pi 3 setup with monitor and USB Mouse & Keyboard
- 3 LED
- 3 resistors (80 ohm)
- Buzzer
- Button

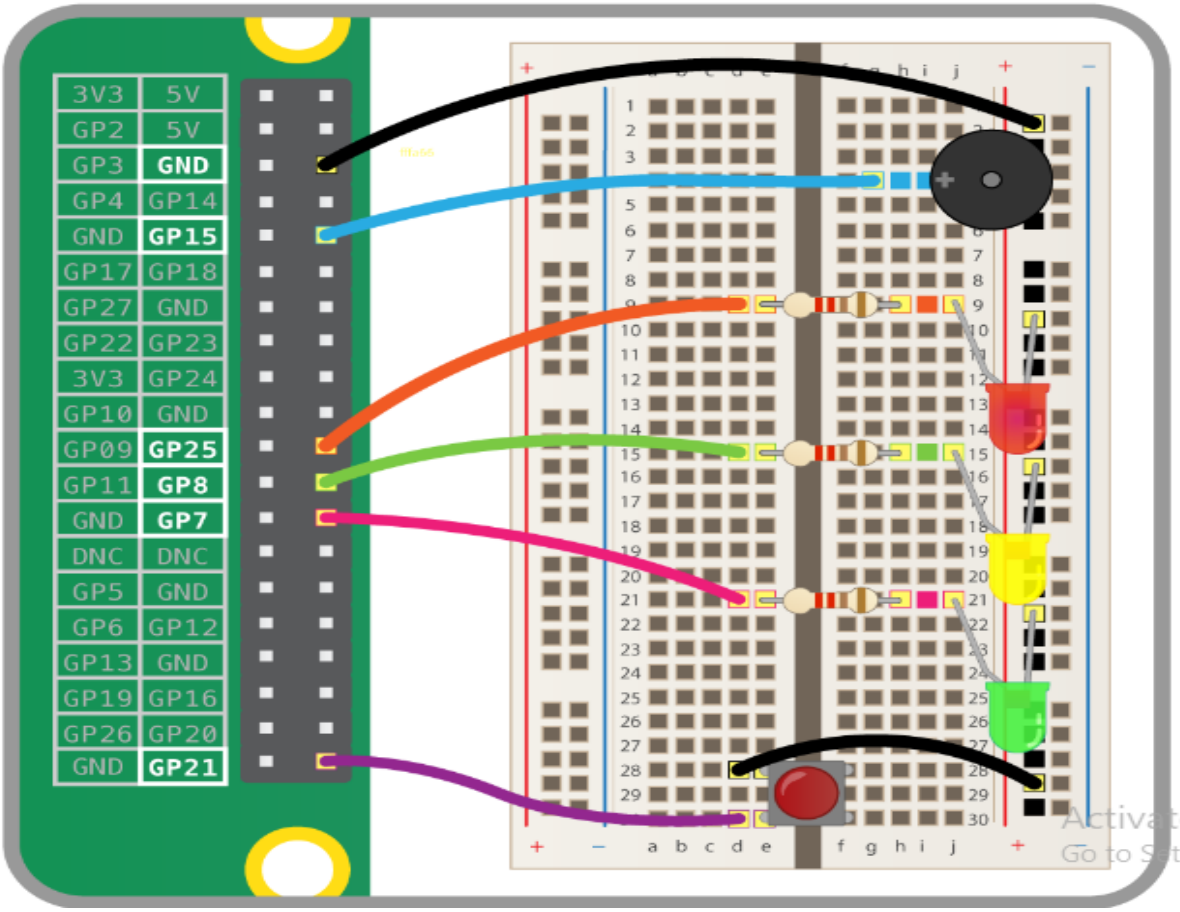
To get started, you'll need to place all the components on the breadboard and connect them to the appropriate GPIO pins on the Raspberry Pi.

First, you need to understand how each component is connected:

- A push button requires 1 ground pin and 1 GPIO pin
    - An LED requires 1 ground pin and 1 GPIO pin, with a current limiting resistor
    - A buzzer requires 1 ground pin and 1 GPIO pin

Each component requires its own individual GPIO pin, but components can share a ground pin. We will use the breadboard to enable this.

Place the components on the breadboard and connect them to the Raspberry Pi GPIO pins, according to the following diagram:



Observe the following table, showing which GPIO pin each component is connected to:

Component	GPIO pin
Button	21
Red LED	25
Amber LED	8
Green LED	7
Buzzer	15

Code

Create a new file by clicking New.

Save the new file straight away by clicking Save; name the file trafficlights.py.

Enter the following code:

```
import RPi.GPIO as GPIO
```

```
import time
import random
#
# Use BCM e.g. identify pins by name
#


GPIO.setmode(GPIO.BCM)
#
# Set warnings off to suppress re-initialisation messages
#
GPIO.setwarnings(False)
#
# Set a variable to the GPIO Pin name we what to use
# one for each of the 3 leds
#
RED = 18
AMBER = 23
GREEN = 24
Buzzer=15
Button=21
#
# Initialise the GPIO Ports to OUTPUT
#
GPIO.setup(RED , GPIO.OUT)
GPIO.setup(AMBER , GPIO.OUT)
GPIO.setup(GREEN , GPIO.OUT)
GPIO.setup(Buzzer, GPIO.OUT)
GPIO.setup(Button, GPIO.IN, pull_up_down=GPIO.PUD_UP)
#
# Stop Function - Red light only
```

```
#
def Stop():
        global RED, AMBER, GREEN
        print "STOP"
        GPIO.output(RED , GPIO.HIGH)
        GPIO.output(AMBER, GPIO.LOW)
        GPIO.output(GREEN, GPIO.LOW)
#
# Get Ready to go Function - Red and amber lit
#
def GetReadyToGo():
        global RED, AMBER, GREEN
        print "Get Ready"
        GPIO.output(RED , GPIO.HIGH)
        GPIO.output(AMBER, GPIO.HIGH)
        GPIO.output(GREEN, GPIO.LOW)
#
# Go function - Green Only
#
def Go():
        global RED, AMBER, GREEN
        print "Go !"
        GPIO.output(RED , GPIO.LOW)
        GPIO.output(AMBER, GPIO.LOW)
        GPIO.output(GREEN, GPIO.HIGH)
#
# Get Ready To Stop function - Amber Only
#
def GetReadyToStop():
        global RED, AMBER, GREEN
```

```
        print "Warning lights changing"
        GPIO.output(RED , GPIO.LOW)
        GPIO.output(AMBER, GPIO.HIGH)
        GPIO.output(GREEN, GPIO.LOW)
#
# Run through the traffic light secquence forever
#
# If ctrl-C is pressed then quit
#
try:
        while (True):
                button_state = GPIO.input(Button)
               if button_state == False:
                        GetReadyToGo();
                        time.sleep(1)
                        # go and pause between 3 and 5 seconds
                        Go();
                        pauseTime = random.randint(3,5)
                       GPIO.output(buzzer,GPIO.HIGH)
                        print ("Beep")
                        time.sleep(0.5) # Delay in seconds
                        GPIO.output(buzzer,GPIO.LOW)
                        print ("No Beep")
                        time.sleep(0.5)
                        time.sleep(pauseTime)
                        #the amber light is lit for 2.9 seconds
                        GetReadyToStop();
                        time.sleep(2.9)
                else:
                        # stop and pause between 3 and 5 seconds
```

```
                    Stop();
                    pauseTime = random.randint(3,5)
                    time.sleep(pauseTime)
except KeyboardInterrupt:
        #
        # clean up on finish
        #
        print "clean Up"
        GPIO.cleanup()
```

Experiment 7: Control ultrasonic sensor using raspberry pi

Objective: measure distance between two cars

Hardware component

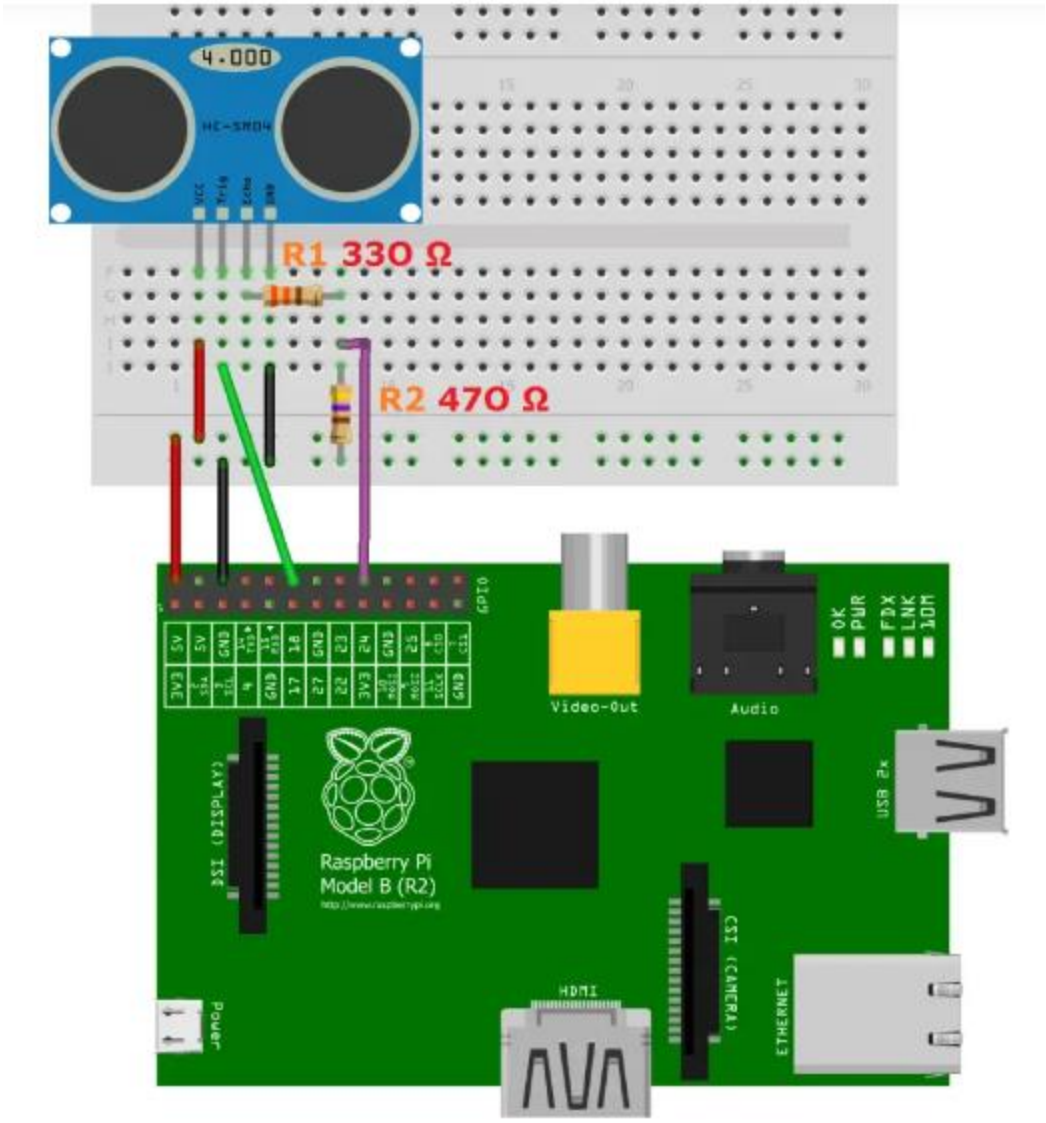
- HC-SR04 Module
- Resistors: 330Ω and 470Ω
- Jumper wire

Four pins on the ultrasound module are connected to the Raspberry:

- VCC to Pin 2 (VCC)
- GND to Pin 6 (GND)
- TRIG to Pin 12 (GPIO18)
- connect the 330Ω resistor to ECHO. On its end you connect it to Pin 18 (GPIO24) and through a 470Ω resistor you connect it also to Pin6 (GND).

We do this because the GPIO pins only tolerate maximal 3.3V. The connection to GND is to have a obvious signal on GPIO24. If no pulse is sent, the signal is 0 (through the connection with GND), else it is 1. If there would be no connection to GND, the input would be undefined if no signal is sent (randomly 0 or 1), so ambiguous.

Circuit diagram



Code

Create file name ultra.py and write the following code with it as follows:

```
#Libraries
import RPi.GPIO as GPIO
import time

#GPIO Mode (BOARD / BCM)
GPIO.setmode(GPIO.BCM)

#set GPIO Pins
GPIO_TRIGGER = 18
GPIO_ECHO = 24

#set GPIO direction (IN / OUT)
GPIO.setup(GPIO_TRIGGER, GPIO.OUT)
GPIO.setup(GPIO_ECHO, GPIO.IN)

def distance():
    # set Trigger to HIGH
    GPIO.output(GPIO_TRIGGER, True)

    # set Trigger after 0.01ms to LOW
```

```
    time.sleep(0.00001)
    GPIO.output(GPIO_TRIGGER, False)

    StartTime = time.time()
    StopTime = time.time()

    # save StartTime
    while GPIO.input(GPIO_ECHO) == 0:
        StartTime = time.time()

    # save time of arrival
    while GPIO.input(GPIO_ECHO) == 1:
        StopTime = time.time()

    # time difference between start and arrival
    TimeElapsed = StopTime - StartTime
    # multiply with the sonic speed (34300 cm/s)
    # and divide by 2, because there and back
    distance = (TimeElapsed * 34300) / 2

    return distance

if __name__ == '__main__':
    try:
        while True:
            dist = distance()
            print ("Measured Distance = %.1f cm" % dist)
            time.sleep(1)

        # Reset by pressing CTRL + C
    except KeyboardInterrupt:
        print("Measurement stopped by User")
        GPIO.cleanup()
```

After that we run:

```
sudo python ultra.py
```

watch the following video

https://www.youtube.com/watch?v=_7drIUmC8Zo

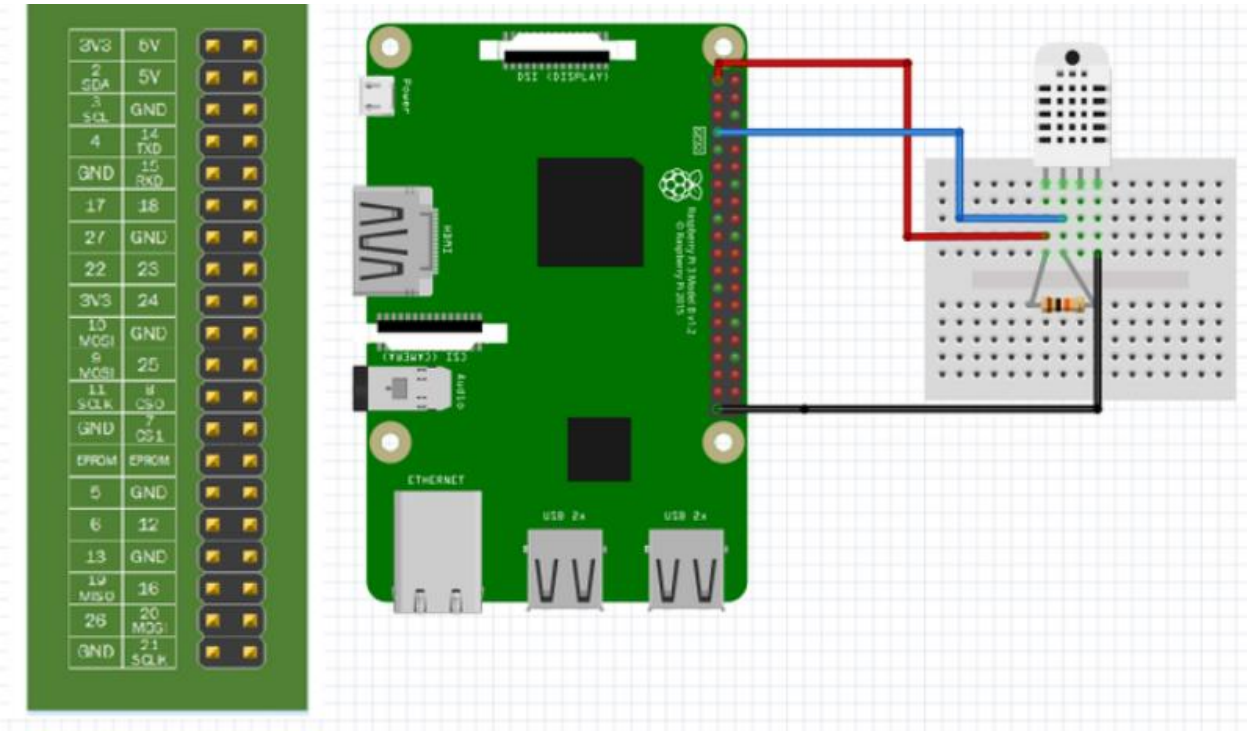
Experiment 8: control DHT11 or DHT22 using raspberry pi

Objective: measure temperature and humidity using DHT11 or DHT22

Hardware component

- Raspberry Pi board
- DHT11 or DHT22 temperature and humidity sensor
- 4.7k Ohm resistor or similar value
- Breadboard
- Jumper wires

Hardware circuit



Install Adafruit DHT Library

Before python code you need to download and install the DHT library in your Raspberry Pi. Open the terminal window and type:

```
git clone https://github.com/adafruit/Adafruit_Python_DHT.git
cd Adafruit_Python_DHT
sudo apt-get update
sudo apt-get install build-essential python-dev
sudo python setup.py install
```

Now you will have to reboot your Pi system to get the Adafruit driver.

Code

Open file called dh.py then write the following code on it as follows:

```
#Libraries
import Adafruit_DHT as dht
from time import sleep
#Set DATA pin
DHT = 4
while True:
    #Read Temp and Hum from DHT22
    h,t = dht.read_retry(dht.DHT22, DHT)
    #Print Temperature and Humidity on Shell window
    print('Temp={0:0.1f}*C  Humidity={1:0.1f}%'.format(t,h))
    sleep(5) #Wait 5 seconds and read again
```

you can watch the following video

https://www.youtube.com/watch?v=0jplzBkt3i0

Experiment 9: Control IR using raspberry pi

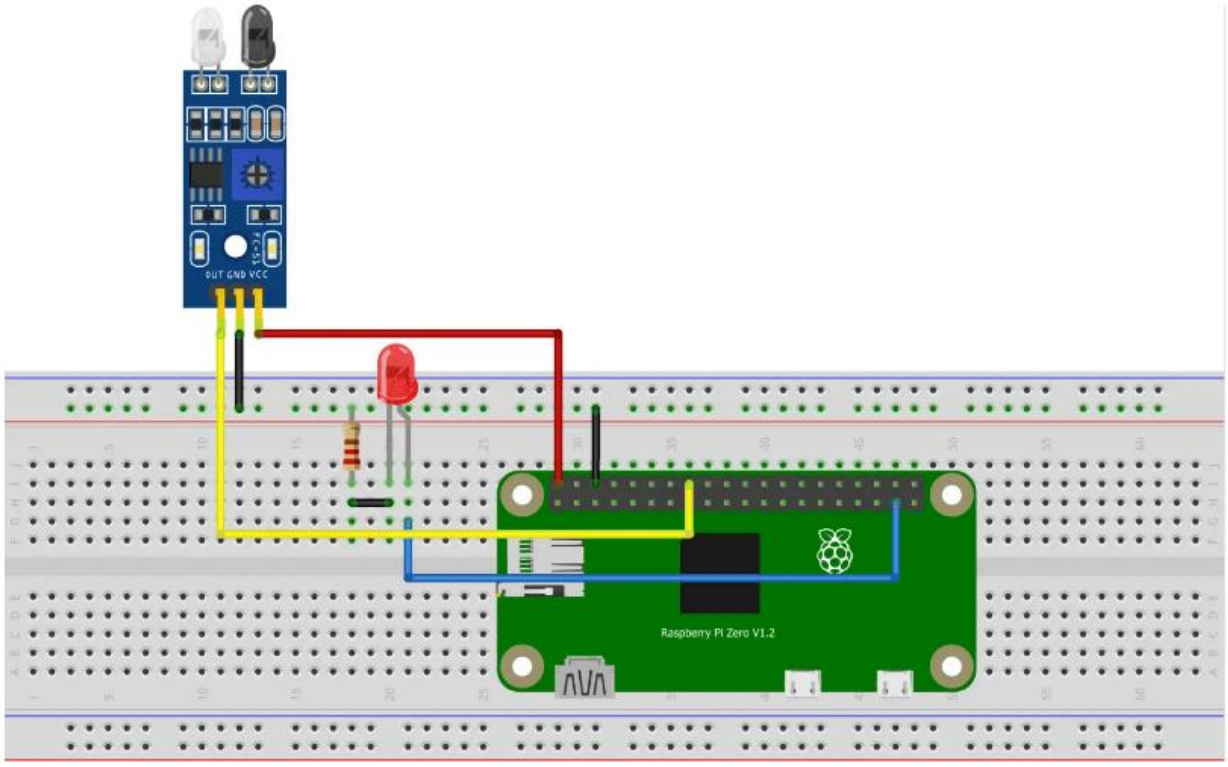
Objective: detect motion in room and then turn on LED if motion detected.

Hardware component

The following components are needed to follow along with this tutorial.

- Infrared (IR) Sensor
- Raspberry Pi
- LED
- Resistor (220 Ohms)
- Breadboard – Amazon
- Connecting Wires

Hardware circuit



Infrared (IR) Sensor	Raspberry Pi	LED
VCC	VCC	
GND	GN	
OUT	GPIO23	
	GPIO26	ANODE

Code

Create file called **ir_sensor.py then write the following code with it as follows:**

```
import RPi.GPIO as GPIO
import time
# declare the sensor and led pin
sensor_pin = 23
led_pin = 26
# GPIO setup
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.setup(sensor_pin, GPIO.IN)
GPIO.setup(led_pin, GPIO.OUT)
try:
    while True:
        if GPIO.input(sensor_pin):
            # If no object is near
            GPIO.output(led_pin, False)
            while GPIO.input(sensor_pin):
                time.sleep(0.2)
        else:
            # If an object is detected
            GPIO.output(led_pin, True)
except KeyboardInterrupt:
    GPIO.cleanup()
```

Run the code by writing

Python ir_sensor.py

watch the following video

https://www.youtube.com/watch?v=7i0dvQGM-Zk