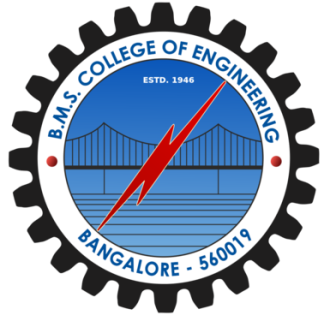
**Internet Of Things**

**ARDUINO PROGRAMMING AND SENSOR INTERFACING**

**BMS COLLEGE OF ENGINEERING**

****

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**AUG – DEC, 2016**

**1. Internet of Things**

The Internet of Things (IoT) describes the phenomenon of everyday devices connecting to the Internet through tiny embedded sensors and computing power

We’re entering a new era of computing technology that many are calling the Internet of Things (IoT). Machine to machine, machine to infrastructure, machine to environment, the Internet of Everything, the Internet of Intelligent Things, intelligent systems—call it what you want, but it’s happening, and its potential is huge.  We see the IoT as billions of smart, connected “things” that will encompass every aspect of our lives, and its foundation is the intelligence that embedded processing provides. The IoT is comprised of smart machines interacting and communicating with other machines, objects, environments and infrastructures.

**Applications of IoT**

* Machine-to-machine communication
* Machine-to-infrastructure communication
* Telehealth: remote or real-time pervasive monitoring of patients, diagnosis and drug delivery
* Continuous monitoring of, and firmware upgrades for, vehicles
* Asset tracking of goods on the move
* Automatic traffic management
* Remote security and control
* Environmental monitoring and control
* Home and industrial building automation
* “Smart” applications, including cities, water, agriculture, buildings, grid, meters, broadband, cars, appliances, tags, animal farming and the environment, to name a few.

**2. Introduction to Arduino**

The Arduino board is a small microcontroller board, which is a small circuit (the board) that contains a whole computer on a small chip (the microcontroller). Arduino is composed of two major parts: the Arduino board, which is the piece of hardware you work on when you build your objects; and the Arduino IDE, the piece of software you run on your computer. You use the IDE to create a sketch (a little computer program) that you upload to the Arduino board. The sketch tells the board what to do.

The Arduino software is easy-to-use for beginners, yet flexible enough for advanced users. It runs on Mac, Windows, and Linux. Teachers and students use it to build low cost scientific instruments, to prove chemistry and physics principles, or to get started with programming and robotics. It has the specific advantages such as

• **Inexpensive** - Arduino boards are relatively inexpensive compared to other microcontroller platforms. The least expensive version of the Arduino module can be assembled by hand, and even the pre-assembled Arduino modules cost less than $50

• **Cross-platform** - The Arduino Software (IDE) runs on Windows, Macintosh OSX, and Linux operating systems. Most microcontroller systems are limited to Windows.

• **Simple, clear programming environment** - The Arduino Software (IDE) is easy-to-use for beginners, yet flexible enough for advanced users to take advantage of as well. For teachers, it's conveniently based on the Processing programming environment, so students learning to program in that environment will be familiar with how the Arduino IDE works.

• **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, and people wanting to understand the technical details can make the leap from Arduino to the AVR C programming language on which it's based. Similarly, you can add AVR-C code directly into your Arduino programs if you want to.

• **Open source and extensible hardware** - The plans of the Arduino boards are published under a Creative Commons license, so experienced circuit designers can make their own version of the module, extending it and improving it. Even relatively inexperienced users can build the breadboard version of the module in order to understand how it works and save money.

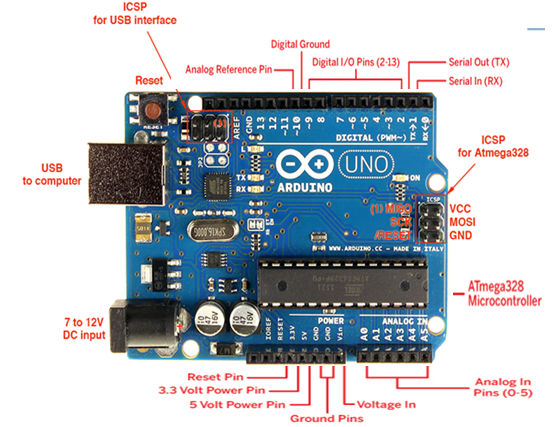
**Installing Arduino on Your Computer**

To program the Arduino board, you must first download the development environment (the IDE) from here: [www.arduino.cc/en/Main/Software](http://www.arduino.cc/en/Main/Software). Choose the right version for your operating system. Download the file and double-click on it to open it it; on Windows or Linux, this creates a folder named arduino-[version], such as arduino-1.0. Drag the folder to wherever you want it: your desktop, your Program Files folder (on Windows), etc. On the Mac, double-clicking it will open a disk image with an Arduino application (drag it to your Applications folder). Now whenever you want to run the Arduino IDE, you’ll open up the arduino (Windows and Linux) or Applications folder (Mac), and double-click the Arduino icon.

**Difference between Microprocessor and Microcontroller**

| **Microprocessor** | **Micro Controller** |
| --- | --- |
| 1G mobile phone | 1G mobile phone |
| Microprocessor is heart of Computer system. | Micro Controller is a heart of embedded system. |
| It is just a processor. Memory and I/O components have to be connected externally | Micro controller has external processor along with internal memory and i/O components |
| Since memory and I/O has to be connected externally, the circuit becomes large. | Since memory and I/O are present internally, the circuit is small. |
| Cannot be used in compact systems and hence inefficient | Can be used in compact systems and hence it is an efficient technique |
| Cost of the entire system increases | Cost of the entire system is low |
| Due to external components, the entire power consumption is high. Hence it is not suitable to used with devices running on stored power like batteries. | Since external components are low, total power consumption is less and can be used with devices running on stored power like batteries. |
| Most of the microprocessors do not have power saving features. | Most of the micro controllers have power saving modes like idle mode and power saving mode. This helps to reduce power consumption even further. |
| Since memory and I/O components are all external, each instruction will need external operation, hence it is relatively slower. | Since components are internal, most of the operations are internal instruction, hence speed is fast. |
| Microprocessor have less number of registers, hence more operations are memory based. | Micro controller have more number of registers, hence the programs are easier to write. |
| Microprocessors are based on von Neumann model/architecture where program and data are stored in same memory module | Micro controllers are based on Harvard architecture where program memory and Data memory are separate |
| Mainly used in personal computers | Used mainly in washing machine, MP3 players |

## 3. The Arduino Development board



### Digital Pins

The digital pins on an Arduino board can be used for general purpose input and output via the [pinMode()](https://www.arduino.cc/en/Reference/PinMode), [digitalRead()](https://www.arduino.cc/en/Reference/DigitalRead), and [digitalWrite()](https://www.arduino.cc/en/Reference/DigitalWrite) commands. Each pin has an internal pull-up resistor which can be turned on and off using digitalWrite() (w/ a value of HIGH or LOW, respectively) when the pin is configured as an input. The maximum current per pin is 40 mA.

Some other specific functions of pins are listed below:

* **Serial: 0 (RX) and 1 (TX).** Used to receive (RX) and transmit (TX) TTL serial data
* **External Interrupts: 2 and 3.** These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value.
* **PWM: 3, 5, 6, 9, 10, and 11.** Provide 8-bit PWM output with the [analogWrite()](https://www.arduino.cc/en/Reference/AnalogWrite) function. On boards with an ATmega8, PWM output is available only on pins 9, 10, and 11.
* **BT Reset: 7.** (Arduino BT-only) Connected to the reset line of the bluetooth module.
* **SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK).** These pins support SPI communication, which, although provided by the underlying hardware, is not currently included in the Arduino language.
* **LED: 13.** There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.

### Analog Pins

In addition to the specific functions listed below, the analog input pins support 10-bit analog-to-digital conversion (ADC) using the [analogRead()](https://www.arduino.cc/en/Reference/AnalogRead) function. Most of the analog inputs can also be used as digital pins: analog input 0 as digital pin 14 through analog input 5 as digital pin 19. Analog inputs 6 and 7 (present on the Mini and BT) cannot be used as digital pins.

### Power Pins

* **VIN**: The input voltage to the Arduino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.
* **5V.** The regulated power supply used to power the microcontroller and other components on the board. This can come either from VIN via an on-board regulator, or be supplied by USB or another regulated 5V supply.
* **3V3.**  A 3.3 volt supply generated by the on-board FTDI chip.
* **GND.** Ground pins.

### Other Pins

* **AREF.** Reference voltage for the analog inputs. Used with [analogReference](https://www.arduino.cc/en/Reference/AnalogReference)().
* **Reset.**  Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

**Analog and Digital Communication**

***Digital Write :*LED Blinking**

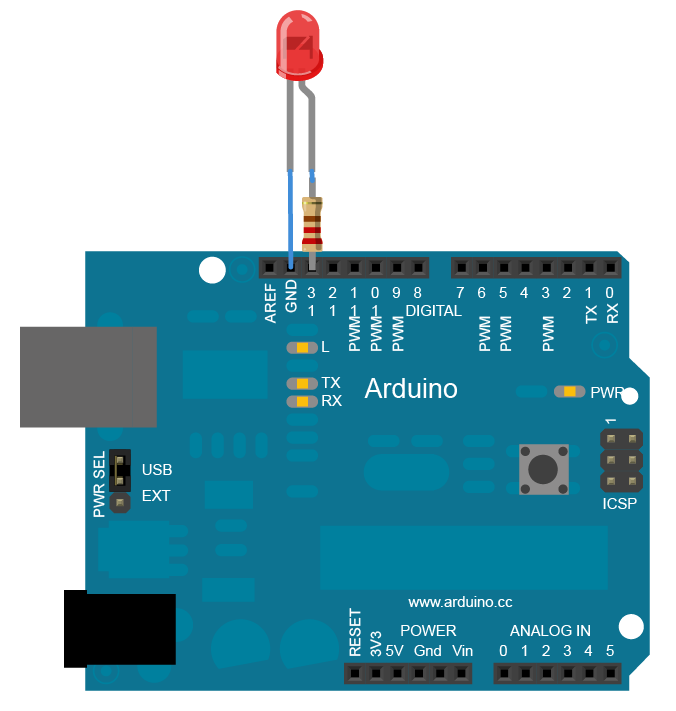
**Aim**

Turns on an LED on for one second, then off for one second, repeatedly

**Hardware Required**

* Arduino Board
* LEDs

**Circuit Diagram**



**Code**

// Pin 13 has an LED connected on most Arduino boards

int led = 13;  
  
void **setup**()  // the setup routine runs once when you press reset

{                  
// initialize the digital pin as an output.  
  pinMode(led, OUTPUT);       
}  
  
void **loop**() { // the loop routine runs over and over again forever  
  digitalWrite(led, HIGH);   // turn the LED on (HIGH is the voltage level)  
  delay(1000);               // wait for a second  
  digitalWrite(led, LOW);    // turn the LED off by making the voltage LOW

  delay(1000);               // wait for a second  
}

* 1. **Activate LEDs using pushbuttons**

***Digital Write & Digital Write:***

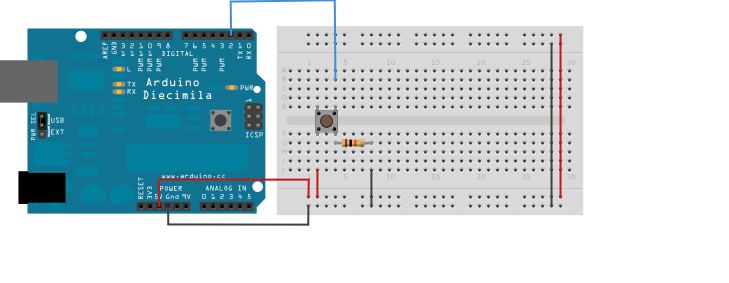
**Aim**

Push button to control LED (Digital Communication)

**Hardware Required**

* Arduino Board
* momentary button or switch
* 10K ohm resistor
* Breadboard

**Circuit Diagram**



**Code**

const int buttonPin = 2;     *// the number of the pushbutton pin*  
const int ledPin =  13;      *// the number of the LED pin*  
  
*// variables will change:*  
int buttonState = 0;         *// variable for reading the pushbutton status*  
  
void **setup**() {  
  *// initialize the LED pin as an output:*  
  pinMode(ledPin, OUTPUT);        
  *// initialize the pushbutton pin as an input:*  
  pinMode(buttonPin, INPUT);       
}  
  
void **loop**(){  
  *// read the state of the pushbutton value:*  
  buttonState = digitalRead(buttonPin);  
  
  *// check if the pushbutton is pressed.*  
  *// if it is, the buttonState is HIGH:*  
  if (buttonState == HIGH) {       
    *// turn LED on:*  
    digitalWrite(ledPin, HIGH);    
  }   
  else {  
    *// turn LED off:*  
    digitalWrite(ledPin, LOW);   
  }  
}

**3. *Analog Write :* LED fading**

Pin connection:

Attach Long leg of LED to any PWM pin. Here pin 9 is used.

int ledPin = 9; // LED connected to digital pin 9

void setup() {

// nothing happens in setup

}

void loop() {

// fade in from min to max in increments of 5 points:

for (int fadeValue = 0 ; fadeValue <= 255; fadeValue += 5) {

// sets the value (range from 0 to 255):

analogWrite(ledPin, fadeValue);

// wait for 30 milliseconds to see the dimming effect

delay(30);

}

// fade out from max to min in increments of 5 points:

for (int fadeValue = 255 ; fadeValue >= 0; fadeValue -= 5) {

// sets the value (range from 0 to 255):

analogWrite(ledPin, fadeValue);

// wait for 30 milliseconds to see the dimming effect

delay(30); }

**4. LED fading using potentiometer**

***Analog Read and Analog Write :***

**Aim**

Control brightness of led using potentiometer (Analog Communication)

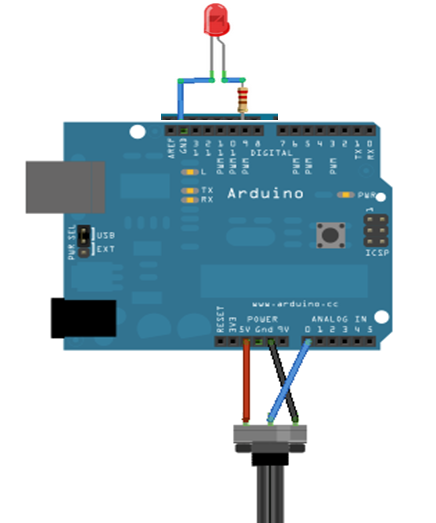
**Hardware Required**

* Arduino Board
* Potentiometer
* LED
* 220 ohm resistor
* hook-up wires

By turning the shaft of the potentiometer, you change the amount of resistance on either side of the wiper which is connected to the center pin of the potentiometer. This changes the voltage at the center pin. When the resistance between the center and the side connected to 5 volts is close to zero (and the resistance on the other side is close to 10 kilohms), the voltage at the center pin nears 5 volts. When the resistances are reversed, the voltage at the center pin nears 0 volts, or ground. This voltage is the analog voltage that you're reading as an input.

The Arduino has a circuit inside called an analog-to-digital converter that reads this changing voltage and converts it to a number between 0 and 1023. When the shaft is turned all the way in one direction, there are 0 volts going to the pin, and the input value is 0. When the shaft is turned all the way in the opposite direction, there are 5 volts going to the pin and the input value is 1023. In between,[analogRead](http://arduino.cc/en/Reference/AnalogRead)() returns a number between 0 and 1023 that is proportional to the amount of voltage being applied to the pin.

**Circuit Diagram**



**Code**

int ledPin = 9; // LED connected to digital pin 9

void setup() {

//do nothing

}

void loop() {

// read the input on analog pin 0:

int sensorValue = analogRead(A0);

// Convert the analog reading (which goes from 0 - 1023) to a voltage (0 - 255):

float voltage = sensorValue \* (255 / 1023.0);

// print out the value you read:

analogWrite(ledPin, voltage);

// wait for 30 milliseconds to see the dimming effect

delay(30);

}

**5. Traffic Controller**

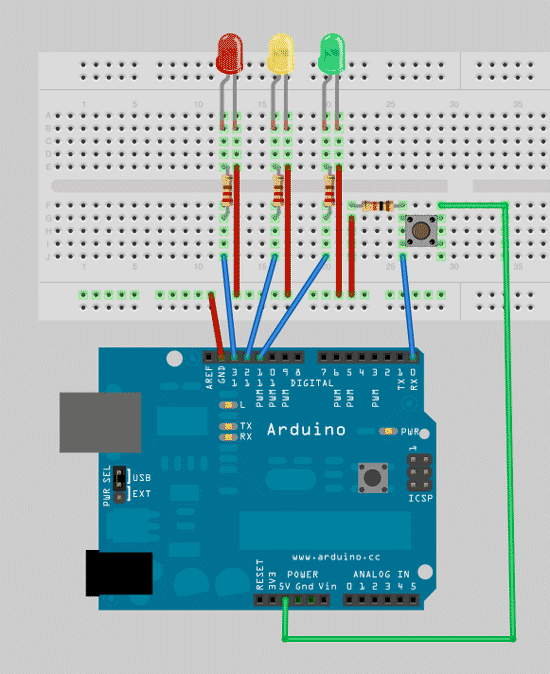
**Aim**

Traffic Signal Simulation

**Hardware Required**

* Arduino Board
* LEDs
* Resistors – 10 K
* Bread board

**Circuit Diagram**



**Code**

**// defining variables so that we can address the lights by name rather than a number**

int red = 13;

*int yellow = 12;*

*int green = 11;*

**//** **add the setup function, where’ll we define the red, yellow and green LEDs to be output mode**

void setup()

{

pinMode(red,OUTPUT);

pinMode(yellow,OUTPUT);

pinMode(green,OUTPUT);

}

void loop()

{

changeLights();

delay(15000);

}

void changeLights(){

// green off, yellow for 3 seconds

digitalWrite(green,HIGH);

digitalWrite(yellow,LOW);

delay(3000);

// turn off yellow, then turn red on for 5 seconds

digitalWrite(yellow,LOW);

digitalWrite(red,HIGH);

delay(5000);

// red and yellow on for 2 seconds (red is already on though)

digitalWrite(yellow,HIGH);

delay(2000);

// turn off red and yellow, then turn on green

digitalWrite(yellow,LOW);

digitalWrite(red,LOW);

digitalWrite(green,HIGH);

}

**Sensor Interfacing**

**6. Temperature Sensor**

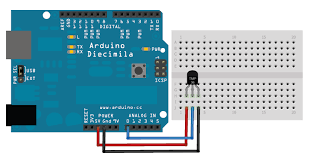
**Aim**

Read the temperature and print it in serial port

**Hardware Required**

* Arduino Board
* Temperature Sensor (LM35 temperature sensor)

**Circuit Diagram**



**Code**

int outputpin= 0;

//this sets the ground pin to LOW and the input voltage pin to high

void setup()

{

Serial.begin(9600);

}

//main loop

void loop()

{

int rawvoltage= analogRead(outputpin);

float millivolts= (rawvoltage/1024.0) \* 5000;

float celsius= millivolts/10;

Serial.print(celsius);

Serial.print(" degrees Celsius, ");

Serial.print((celsius \* 9)/5 + 32);

Serial.println(" degrees Fahrenheit");

delay(1000);

}

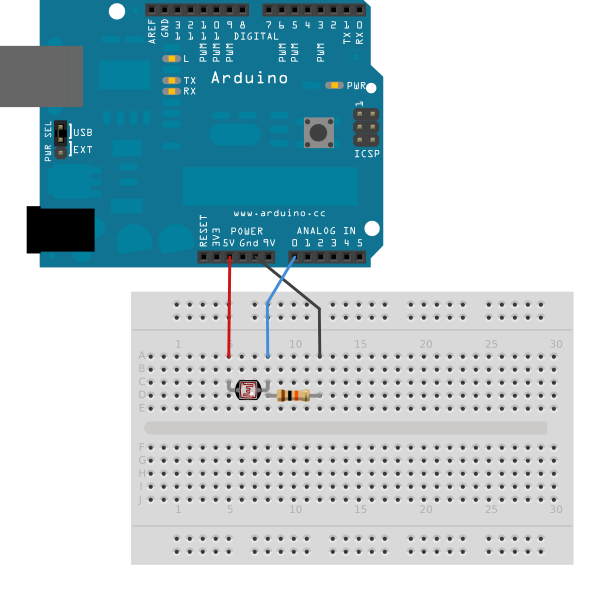
# 7.Nightlight Simulation

**Aim**

Simulating a night light using LDR

**Hardware Required**

* 1 LED
* 1 LDR
* 110K register

[](http://arduino.cc/en/uploads/Tutorial/switchCase_bb.png)

**Connection:**

1. Attach one leg of LDR to 5V and another leg to Arduino Analog pin A0
2. Attach one leg of 110K register with that leg of LDR connected to A0
3. Attach another leg of register to the ground
4. Connect the positive leg of LED to pin 11 and negative to GND

**Code:**

int LDR = 0; //analog pin to which LDR is connected, here we set it to 0 so it means A0

int LDRValue = 0; //that’s a variable to store LDR values

int light\_sensitivity = 500; //This is the approx value of light surrounding your LDR

void setup()

{

Serial.begin(9600); //start the serial monitor with 9600 buad

pinMode(11, OUTPUT); //attach positive leg of LED to pin 11

}

void loop()

{

LDRValue = analogRead(LDR); //reads the ldr’s value through LDR

Serial.println(LDRValue); //prints the LDR values to serial monitor

delay(50); //This is the speed by which LDR sends value to arduino

if (LDRValue < light\_sensitivity)

{

digitalWrite(11, HIGH);

}

else

{

digitalWrite(11, LOW);

}

delay(1000);

}

**Observation:**

While lights are switched off in the room, LED should switch ON, when lights are switched on in the room, LED should switch off immediately.

**8. PIR Sensor**

**Aim**

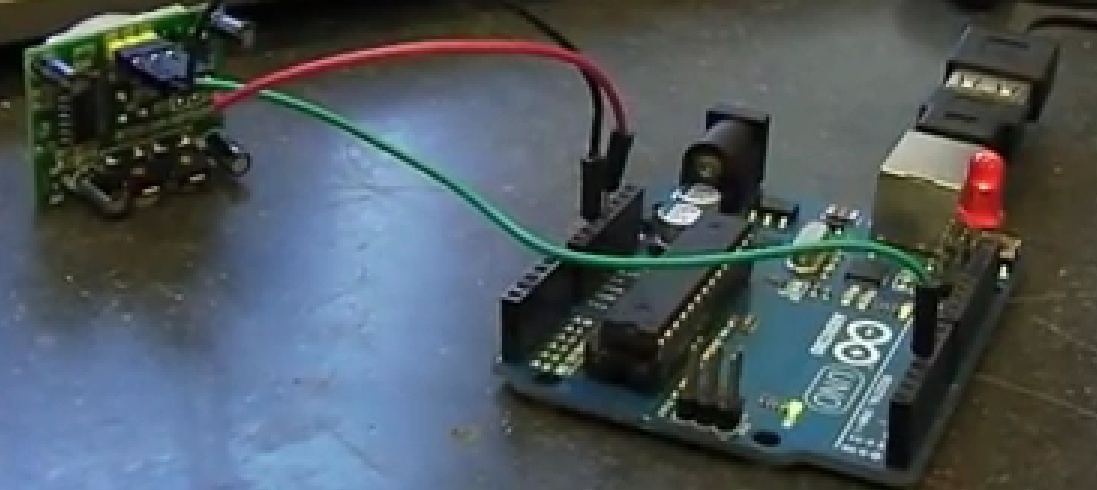
Developing motion detection System

**Hardware Requirement**

* Arduino Board
* PIR Motion Sensor

**Circuit Diagram**

****



**Code**

//the time we give the sensor to calibrate (10-60 secs according to the datasheet)

int calibrationTime = 30;

//the time when the sensor outputs a low impulse

long unsigned int lowIn;

//the amount of milliseconds the sensor has to be low

//before we assume all motion has stopped

long unsigned int pause = 5000;

boolean lockLow = true; // variable used to calculate start time of the motion

boolean takeLowTime; //variable used to calculate stop time of the motion

int pirPin = 3; //the digital pin connected to the PIR sensor's output

int ledPin = 13;

void setup(){ //setup

Serial.begin(9600);

pinMode(pirPin, INPUT);

pinMode(ledPin, OUTPUT);

digitalWrite(pirPin, LOW);

Serial.print("calibrating sensor "); //give the sensor some time to calibrate

for(int i = 0; i < calibrationTime; i++){

Serial.print(".");

delay(1000);

}

Serial.println(" done");

Serial.println("SENSOR ACTIVE");

delay(50);

}

void loop(){ //loop

if(digitalRead(pirPin) == HIGH){

digitalWrite(ledPin, HIGH);

if(lockLow){

//makes sure we wait for a transition to LOW before any further output is made:

lockLow = false;

Serial.println("---");

Serial.print("motion detected at ");

Serial.print(millis()/1000);

Serial.println(" sec");

delay(50);

}

takeLowTime = true;

}

if(digitalRead(pirPin) == LOW){

digitalWrite(ledPin, LOW); //the led visualizes the sensors output pin state

if(takeLowTime){

lowIn = millis(); //save the time of the transition from high to LOW

takeLowTime = false; //make sure this is only done at the start of a LOW phase

}

//if the sensor is low for more than the given pause,

//we assume that no more motion is going to happen

if(!lockLow && millis() - lowIn > pause){

//makes sure this block of code is only executed again after

//a new motion sequence has been detected

lockLow = true;

Serial.print("motion ended at "); //output

Serial.print((millis() - pause)/1000);

Serial.println(" sec");

delay(50);

}

}

}

**9. Moisture Sensor**

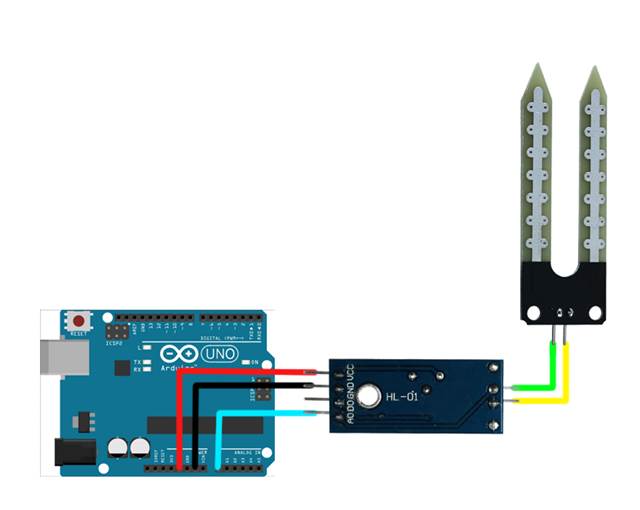
**Aim**

To find the Moisture level in the sand

**Hardware Requirement**

* Arduino Board
* Moisture Sensor

**Circuit Diagram**

****

**Code**

int sensorPin = A0; // select the input pin for the potentiometer

int sensorValue = 0; // variable to store the value coming from the sensor

void setup() {

Serial.begin(9600);

}

void loop() {

// read the value from the sensor:

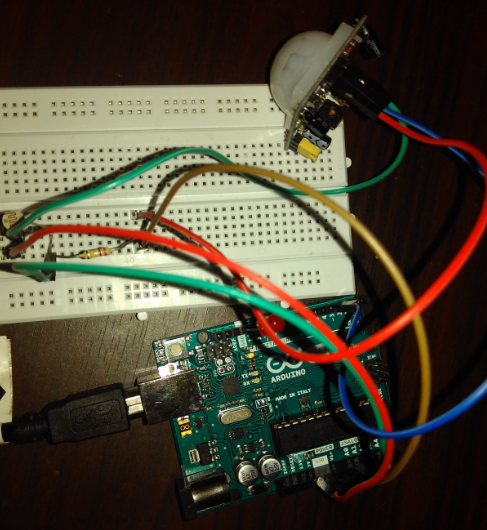
sensorValue = analogRead(sensorPin);

Serial.println (sensorValue);

delay (1000);

}

# 10.Nightlight Simulation with Human Presence Detection



**Hardware Required**

* 1 LED
* 1 LDR
* 110K register
* 1 PIR

**Understanding the configuration of PIR sensor:**

****

**Connection:**

1. Attach one leg of LDR to 5V and another leg to Arduino Analog pin A0
2. Attach one leg of 110K register with that leg of LDR connected to A0
3. Attach another leg of register to the ground
4. Connect the positive leg of LED to pin 11 and negative to GND
5. Connect positive leg of PIR to 5V and negative leg to GND
6. Connect output pin of PIR to digital pin 3

**Code:**

int LDR = 0; //analog pin to which LDR is connected, here we set it to 0 so it means A0

int LDRValue = 0; //that’s a variable to store LDR values

int light\_sensitivity = 500; //This is the approx value of light surrounding your LDR

//the time we give the sensor to calibrate (10-60 secs according to the datasheet)

int calibrationTime = 30;

//the time when the sensor outputs a low impulse

long unsigned int lowIn;

//the amount of milliseconds the sensor has to be low

//before we assume all motion has stopped

long unsigned int pause = 5000;

boolean lockLow = true;

boolean takeLowTime;

int pirPin = 3; //the digital pin connected to the PIR sensor's output

int ledPin = 11;

void setup()

{

Serial.begin(9600); //start the serial monitor with 9600 buad

pinMode(11, OUTPUT);

pinMode(pirPin, INPUT);

pinMode(ledPin, OUTPUT);

digitalWrite(pirPin, LOW);

Serial.print("calibrating sensor ");//give the sensor some time to calibrate

for(int i = 0; i<calibrationTime; i++){

Serial.print(".");

delay(1000);

}

Serial.println(" done");

Serial.println("SENSOR ACTIVE");

delay(50);

}

void loop()

{

LDRValue = analogRead(LDR); //reads the ldr’s value through LDR

// Serial.println(LDRValue); //prints the LDR values to serial monitor

if(digitalRead(pirPin) == HIGH && LDRValue < light\_sensitivity){

digitalWrite(ledPin, HIGH);

if(lockLow){

//makes sure we wait for a transition to LOW before any further output is made:

lockLow = false;

Serial.println("---");

Serial.print("motion detected at ");

Serial.print(millis()/1000);

Serial.println(" sec");

delay(50);

}

takeLowTime = true;

}

if(digitalRead(pirPin) == LOW || LDRValue >= light\_sensitivity){

digitalWrite(ledPin, LOW); //the led visualizes the sensors output pin state

if(takeLowTime){

lowIn = millis(); //save the time of the transition from high to LOW

takeLowTime = false; //make sure this is only done at the start of a LOW phase

}

//if the sensor is low for more than the given pause,

//we assume that no more motion is going to happen

if(!lockLow&&millis() - lowIn> pause){

//makes sure this block of code is only executed again after

//a new motion sequence has been detected

lockLow = true;

Serial.print("motion ended at "); //output

Serial.print((millis() - pause)/1000);

Serial.println(" sec");

delay(50);

}

delay(100);

}

}

**Steps and Observation:**

* Upload the program to Arduino and open Serial monitor
* It will show in Serial monitor: **calibrating sensor .............................. done**
* Wait until it is shown: **SENSOR ACTIVE**
* Caution: Avoid any movements near the sensor**. It should not show: “Motion detected at” in monitor**
* Switch off lights avoiding motion near sensor, LED should not glow
* Make some movement near PIR
* LED should keep glowing for some time
* While LED is ON, switch on the light, LED should switch off immediately.

**11. Reverse Alarm System**

**Aim**

Reverse Alarm System using Proximity Sensor

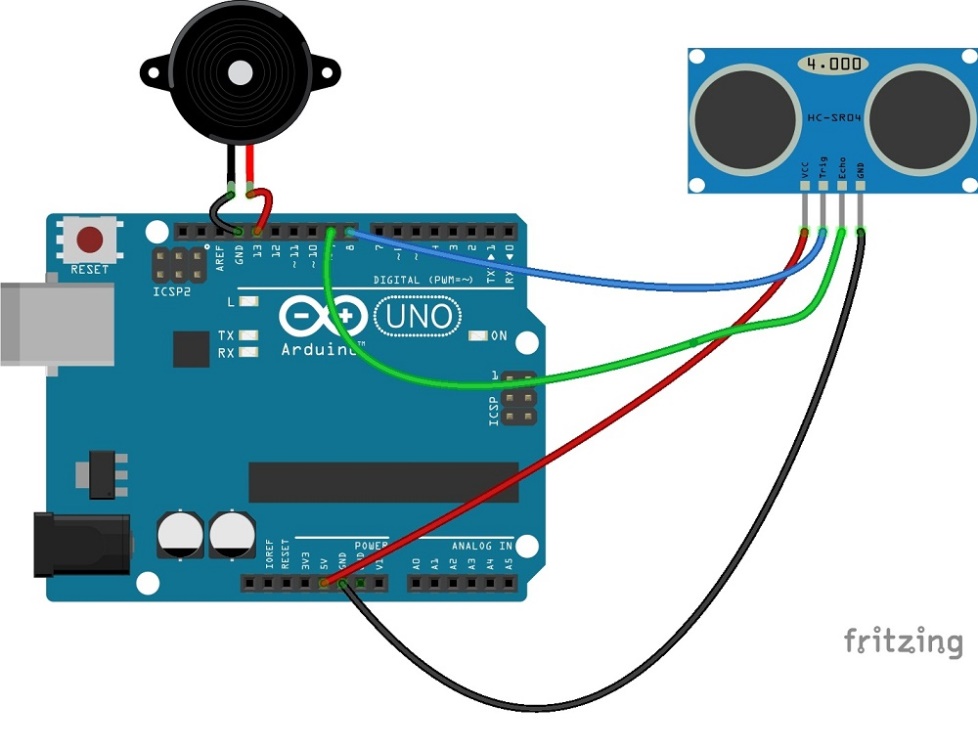
**Hardware Requirement**

* The HC-SR04 Ultrasonic Sensor

It is a very affordable proximity/distance sensor that has been used mainly for object avoidance in various robotics projects . It essentially gives your Arduino eyes / spacial awareness and can prevent your robot from crashing or falling off a table. It has also been used in turret applications, water level sensing, and even as a parking sensor. This simple project will use the HC-SR04 sensor with an Arduino and a Processing sketch to provide a neat little interactive display on your computer screen.

* Arduino Board
* Buzzer

**Circuit Diagram**

****

**Instructions:**

**Download NewPing library from here:**[**Download NewPing Library**](https://code.google.com/p/arduino-new-ping/downloads/detail?name=NewPing_v1.5.zip&can=2&q=)

Put the "NewPing" folder in "libraries\".

In the Arduino IDE, create a new sketch (or open one) and select from the menu bar "Sktech->Import Library->NewPing".

**Comment it all out like this in NewPing.cpp:**

/\*   
#if defined (\_\_AVR\_ATmega32U4\_\_) // Use Timer4 for ATmega32U4 (Teensy/Leonardo).  
ISR(TIMER4\_OVF\_vect) {  
#else  
ISR(TIMER2\_COMPA\_vect) {  
#endif  
        if(intFunc) intFunc(); // If wrapped function is set, call it.  
}\*/

**Code**

#include <NewPing.h>

#define TRIGGER\_PIN 12 // Arduino pin tied to trigger pin on the ultrasonic sensor.

#define ECHO\_PIN 11 // Arduino pin tied to echo pin on the ultrasonic sensor.

#define MAX\_DISTANCE 200 // Maximum distance we want to ping for (in centimeters). Maximum sensor distance is rated at 400-500cm.

NewPing sonar(TRIGGER\_PIN, ECHO\_PIN, MAX\_DISTANCE); // NewPing setup of pins and maximum distance.

int tonepin = 13;

int distance;

void setup() {

Serial.begin(9600); // Open serial monitor at 115200 baud to see ping results.

pinMode(tonepin,OUTPUT);

}

void loop() {

delay(500); // Wait 50ms between pings (about 20 pings/sec). 29ms should be the shortest delay between pings.

unsigned int uS = sonar.ping(); // Send ping, get ping time in microseconds (uS).

Serial.print("Ping: ");

distance=uS / US\_ROUNDTRIP\_CM;

Serial.print(distance); // Convert ping time to distance in cm and print result (0 = outside set distance range)

Serial.println("cm");

if(distance >0 && distance <10)

{

Serial.print("Less Distance");

tone(tonepin,2000,1000);

}

if (distance > 10)

{

noTone(8);

}

}

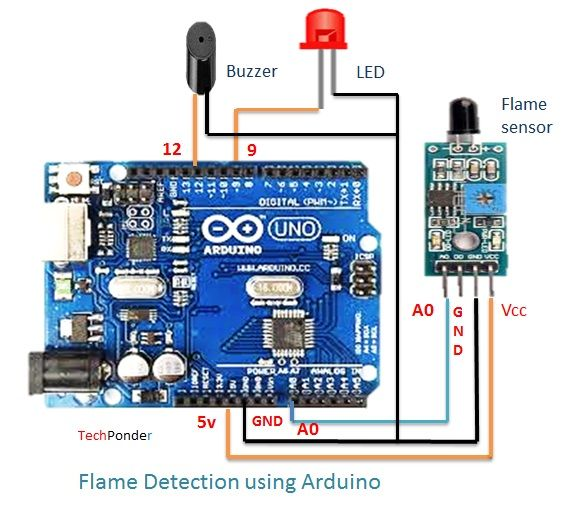
**12. Fire Alert**

**Aim**

Fire alarm simulation

**Hardware Required**

* Flame sensor (Analogue Output)
* Arduino
* Bread board
* LED
* Buzzer
* Connecting wires

**Circuit Diagram**

Flame sensor interfacing to Arduino

Flame sensor to Arduino

vcc -> vcc

gnd -> gnd

A0 -> A0

Led interfacing to Arduino

LED +ve is connected to 9th pin of Arduino

LED -ve is connected to gnd pin of arduino

Buzzer interfacing to Arduino

Buzzer +ve is connected to 12th pin of Arduino

Buzzer -ve is connected to GND pin of Arduino

**Code:**

#include<SoftwareSerial.h>

int sensorPin = A0; // select the input pin for the LDR

int sensorValue = 0; // variable to store the value coming from the sensor

int led = 9; // Output pin for LED

int buzzer = 12; // Output pin for Buzzer

void setup() {

// declare the ledPin and buzzer as an OUTPUT:

pinMode(led, OUTPUT);

pinMode(buzzer,OUTPUT);

Serial.begin(9600);

}

void loop()

{

Serial.println("Welcome to TechPonder Flame Sensor Tutorial");

sensorValue = analogRead(sensorPin);

Serial.println(sensorValue);

if (sensorValue < 100)

{

Serial.println("Fire Detected");

Serial.println("LED on");

digitalWrite(led,HIGH);

digitalWrite(buzzer,HIGH);

delay(1000);

}

digitalWrite(led,LOW);

digitalWrite(buzzer,LOW);

delay(sensorValue);

}

**13. Automatic irrigation controller simulation**

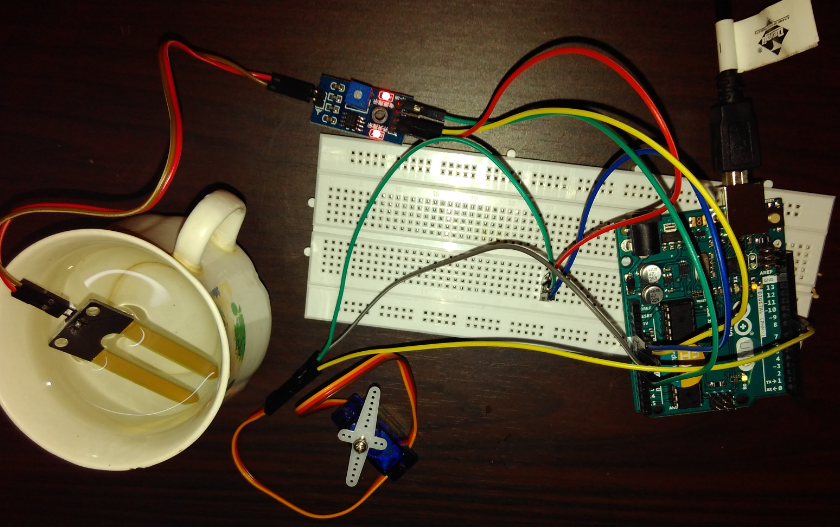
**Aim**

Sensing the soil moisture and sprinkling the Water simulation

**Hardware Required**

* Arduino
* Moisture Sensor
* Breadboard
* Min servo motor

**Circuit diagram**



Moisture sensor VCC to Arduino 5V

Moisture sensor GND to Arduino GND

Moisture sensor A0 to Arduino A0

Servo motor VCC to Arduino 5V

Servo motor GND to Arduino GND

Servo Motor Signal to Arduino digital pin 9

Code:

#include <Servo.h>

Servo myservo; // create servo object to control a servo

// twelve servo objects can be created on most boards

int pos = 0; // variable to store the servo position

int sensorPin = A0; // select the input pin for the potentiometer

int sensorValue = 0; // variable to store the value coming from the sensor

void setup() {

myservo.attach(9); // attaches the servo on pin 9 to the servo object

Serial.begin(9600);

}

void loop() {

// read the value from the sensor:

sensorValue = analogRead(sensorPin);

Serial.println (sensorValue);

if(sensorValue>500)

{

for (pos = 0; pos <= 180; pos += 1) { // goes from 0 degrees to 180 degrees

// in steps of 1 degree

myservo.write(pos); // tell servo to go to position in variable 'pos'

delay(15); // waits 15ms for the servo to reach the position

}

for (pos = 180; pos >= 0; pos -= 1) { // goes from 180 degrees to 0 degrees

myservo.write(pos); // tell servo to go to position in variable 'pos'

delay(15); // waits 15ms for the servo to reach the position

}

}

delay (1000);

}

**14. Reverse parking sensor (Using LCD)**

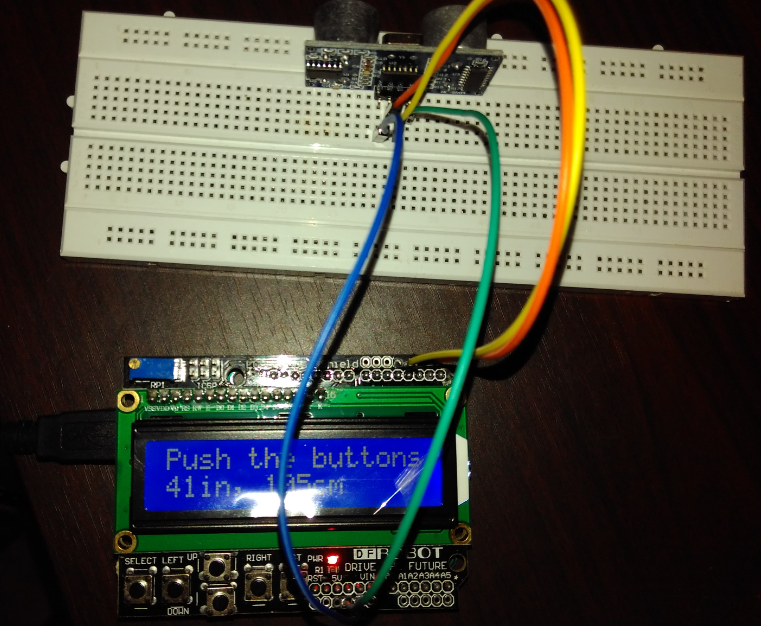
**Aim**

Measuring the distance using Ultrasonic sensor which helps in reverse parking system.

**Hardware Required**

* Arduino
* Ultrasonic Sensor
* Breadboard
* LCD

**Circuit diagram**

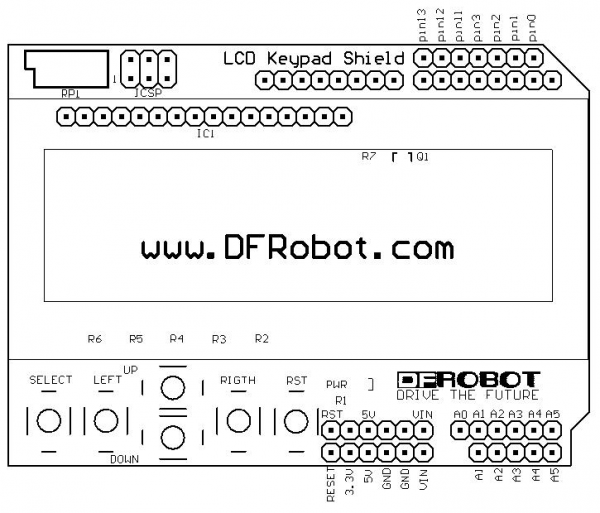
****

**Echo-pin 2**

**Trigger-pin 3**

**Vcc-5 V**

**Gnd-Gnd**



**Code:**

#include <LiquidCrystal.h>

const int trigPin = 3;

const int echoPin = 2;

LiquidCrystal lcd(8, 9, 4, 5, 6, 7); // select the pins used on the LCD panel

void setup() {

lcd.begin(16, 2); // start the library

lcd.setCursor(0,0); // set the LCD cursor position

}

void loop()

{

// establish variables for duration of the ping,

// and the distance result in inches and centimeters:

long duration, inches, cm;

lcd.setCursor(0,1); // move cursor to second line "1" and 0 spaces over

// The sensor is triggered by a HIGH pulse of 10 or more microseconds.

// Give a short LOW pulse beforehand to ensure a clean HIGH pulse:

pinMode(trigPin, OUTPUT);

digitalWrite(trigPin, LOW);

delayMicroseconds(2);

digitalWrite(trigPin, HIGH);

delayMicroseconds(10);

digitalWrite(trigPin, LOW);

// Read the signal from the sensor: a HIGH pulse whose

// duration is the time (in microseconds) from the sending

// of the ping to the reception of its echo off of an object.

pinMode(echoPin, INPUT);

duration = pulseIn(echoPin, HIGH);

// convert the time into a distance

inches = microsecondsToInches(duration);

cm = microsecondsToCentimeters(duration);

lcd.print(inches);

lcd.print("in, ");

lcd.print(cm);

lcd.print("cm");

delay(1000);

}

long microsecondsToInches(long microseconds)

{

return microseconds / 74 / 2;

}

long microsecondsToCentimeters(long microseconds)

{ return microseconds / 29 / 2;}

**15. Color recognition**

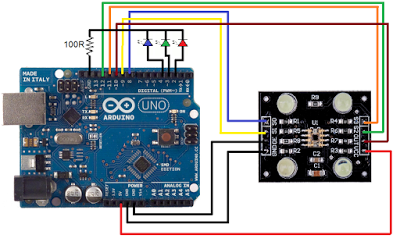
**Aim**

Identify the RGB Colors

**Hardware Required**

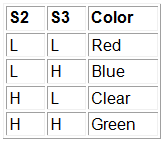
* RGB LED  
  Resistor 100 ohm  
  Color sensor TCS230  
  Arduino board

**Circuit diagram:**

****

Recognizes the basic colors RGB.

This is color sensor that used TCS230 and RGB LED.  
color detection in this module depend on S2 and S3 output as follow:

`  
L means low or 0 and the H means high or 1. you must use a 100ohm resistor to protect the LEDs.

**Code:**

/\*// TCS230 color recognition sensor

// Sensor connection pins to Arduino are shown in comments

Color Sensor      Arduino

-----------      --------

 VCC               5V

 GND               GND

 s0                8

 s1                9

 s2                12

 s3                11

 OUT               10

 OE                GND

\*/

const int s0 = 8;

const int s1 = 9;

const int s2 = 12;

const int s3 = 11;

const int out = 10;

// LED pins connected to Arduino

int redLed = 2;

int greenLed = 3;

int blueLed = 4;

// Variables

int red = 0;

int green = 0;

int blue = 0;

void setup()

{

  Serial.begin(9600);

  pinMode(s0, OUTPUT);

  pinMode(s1, OUTPUT);

  pinMode(s2, OUTPUT);

  pinMode(s3, OUTPUT);

  pinMode(out, INPUT);

  pinMode(redLed, OUTPUT);

  pinMode(greenLed, OUTPUT);

  pinMode(blueLed, OUTPUT);

  digitalWrite(s0, HIGH);

  digitalWrite(s1, HIGH);

}

void loop()

{

  color();

  Serial.print("R Intensity:");

  Serial.print(red, DEC);

  Serial.print(" G Intensity: ");

  Serial.print(green, DEC);

  Serial.print(" B Intensity : ");

  Serial.print(blue, DEC);

  //Serial.println();

  if (red < blue && red < green && red < 20)

  {

   Serial.println(" - (Red Color)");

   digitalWrite(redLed, HIGH); // Turn RED LED ON

   digitalWrite(greenLed, LOW);

   digitalWrite(blueLed, LOW);

  }

  else if (blue < red && blue < green)

  {

   Serial.println(" - (Blue Color)");

   digitalWrite(redLed, LOW);

   digitalWrite(greenLed, LOW);

   digitalWrite(blueLed, HIGH); // Turn BLUE LED ON

  }

  else if (green < red && green < blue)

  {

   Serial.println(" - (Green Color)");

   digitalWrite(redLed, LOW);

   digitalWrite(greenLed, HIGH); // Turn GREEN LED ON

   digitalWrite(blueLed, LOW);

  }

  else{

  Serial.println();

  }

  delay(300);

  digitalWrite(redLed, LOW);

  digitalWrite(greenLed, LOW);

  digitalWrite(blueLed, LOW);

 }

void color()

{

  digitalWrite(s2, LOW);

  digitalWrite(s3, LOW);

  //count OUT, pRed, RED

  red = pulseIn(out, digitalRead(out) == HIGH ? LOW : HIGH);

  digitalWrite(s3, HIGH);

  //count OUT, pBLUE, BLUE

  blue = pulseIn(out, digitalRead(out) == HIGH ? LOW : HIGH);

  digitalWrite(s2, HIGH);

  //count OUT, pGreen, GREEN

  green = pulseIn(out, digitalRead(out) == HIGH ? LOW : HIGH);

}

**16. Arduino with Vibration sensor**

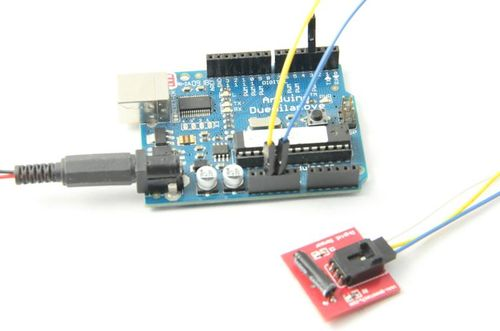
**Aim:**

Simulated the purpose of Vibration sensor by tapping

**Hardware Required**

* Arduino
* Vibration sensor

**Circuit diagram**



**Code:**

int ledPin = 13; // Connect LED to pin 13

int switcher = 3; // Connect Tilt sensor to Pin3

void setup()

{

pinMode(ledPin, OUTPUT); // Set digital pin 13 to output mode

pinMode(switcher, INPUT); // Set digital pin 3 to input mode

}

void loop()

{

if(digitalRead(switcher)==HIGH) //Read sensor value

{

digitalWrite(ledPin, HIGH); // Turn on LED when the sensor is tilted

delay(300);

}

else {

digitalWrite(ledPin, LOW); // Turn off LED when the sensor is not triggered } }