

## Experiment – 1

### Aim:

Pin configuration of raspberry pi , Arduino UNO , and node MSU.

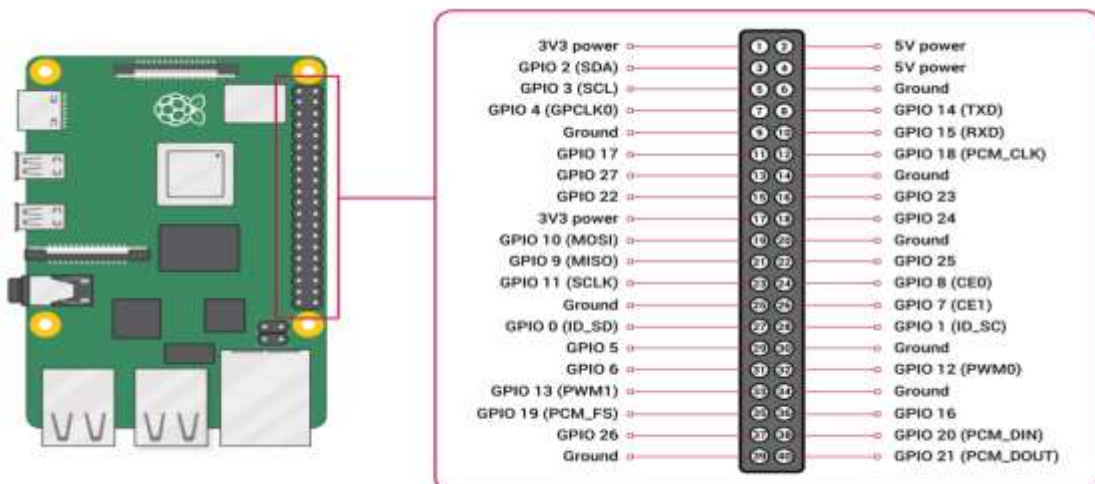
### Materials Required:

- Arduino Uno
- Raspberry Pi
- Node MSU

### Raspberry Pi Pin Configuration:

The Raspberry Pi pin configuration varies depending on the model of the Raspberry Pi board. Here is a summary of the pin configuration for the Raspberry Pi 3 Model B+:

- 40-pin GPIO header
- 5V power pins (2)
- 3.3V power pins (2)
- Ground pins (6)
- SPI pins (2)
- I2C pins (2)
- UART pins (2)
- PWM pins (2)
- A



ins (1)

A Raspberry Pi 3 board has 40 pins on it. Among these pins, we have four power pins on the Raspberry Pi, two of which are 5v pins and another two are 3.3v pins. The 5v power pins are connected directly to the Raspberry Pi's power input and we can use these pins to run low power applications.

Then there are the ground pins. There are eight ground pins and all of these are connected to each other; you can use any of these ground pins for your projects.

That leaves us with 28 GPIO pins, labeled starting from GPIO 0 and going up to GPIO 27.

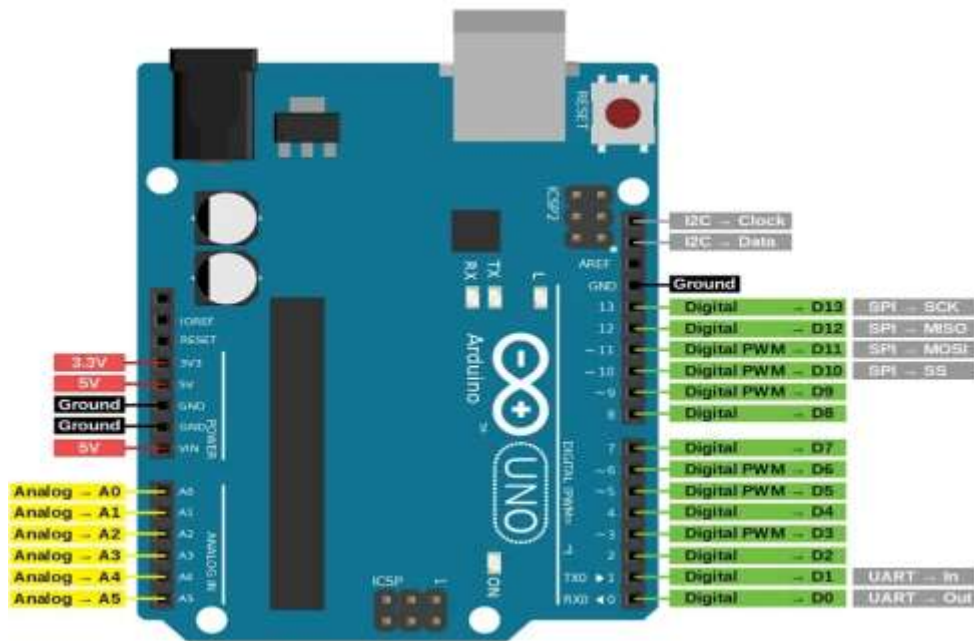
The GPIO pins, as indicated by their full form, can be programmed to be output pins or input pins. So we can set values of output pins and we can even read values of input pins. The GPIO pins can be digitally programmed so that they can be turned ON or OFF. The output of any GPIO pin is 3.3v and can be used to control output components like an LED or a motor. These ON/OFF conditions can also be interpreted as a Boolean True/False, 1/0 or HIGH/LOW.

These are the common types of pins on a Raspberry Pi 3 board. Some of these pins also have a dual function. For example, pin 3 or GPIO 2 also acts like an I2C pin.

### **Arduino UNO Pin Configuration:**

The Arduino UNO has a total of 20 input/output pins, including:

- 14 digital pins (0-13)
- 6 analog input pins
- 5V power pin
- 3.3V power pin
- Ground pins (3)



**Vin:** This is the input voltage pin of the Arduino board used to provide input supply from an external power source.

**5V:** This pin of the Arduino board is used as a regulated power supply voltage and it is used to give supply to the board as well as onboard components.

**3.3V:** This pin of the board is used to provide a supply of 3.3V which is generated from a voltage regulator on the board

**GND:** This pin of the board is used to ground the Arduino board.

**Reset:** This pin of the board is used to reset the microcontroller. It is used to Resets the microcontroller.

**Analog Pins:** The pins A0 to A5 are used as an analog input and it is in the range of 0-5V.

**Digital Pins:** The pins 0 to 13 are used as a digital input or output for the Arduino board.

**Serial Pins:** These pins are also known as a UART pin. It is used for communication between the Arduino board and a computer or other devices. The transmitter pin number 1 and receiver pin number 0 is used to transmit and receive the data resp.

**External Interrupt Pins:** This pin of the Arduino board is used to produce the External interrupt and it is done by pin numbers 2 and 3.

**PWM Pins:** This pins of the board is used to convert the digital signal into an analog by varying the width of the Pulse. The pin numbers 3,5,6,9,10 and 11 are used as a PWM pin.

**SPI Pins:** This is the Serial Peripheral Interface pin, it is used to maintain SPI communication with the help of the SPI library. SPI pins include:

1. SS: Pin number 10 is used as a Slave Select
2. MOSI: Pin number 11 is used as a Master Out Slave In
3. MISO: Pin number 12 is used as a Master In Slave Out
4. SCK: Pin number 13 is used as a Serial Clock

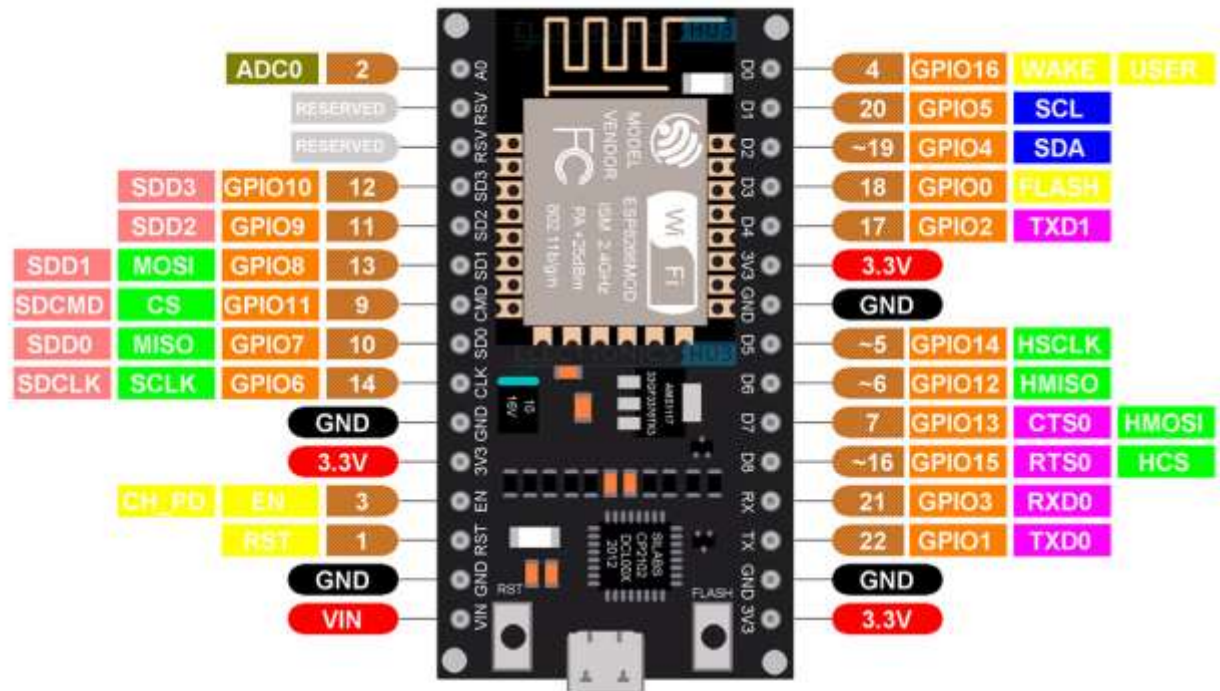
**LED Pin:** The board has an inbuilt LED using digital pin-13. The LED glows only when the digital pin becomes high.

**AREF Pin:** This is an analog reference pin of the Arduino board. It is used to provide a reference voltage from an external power supply.

### **Node MCU Pin Configuration:**

Node MCU is an open-source development board based on the ESP8266 chip. Here is the pin configuration for Node MCU:

- 11 digital pins (D0-D10)
- 1 analog input pin (A0)
- 3V3 power pin
- Ground pins (2)



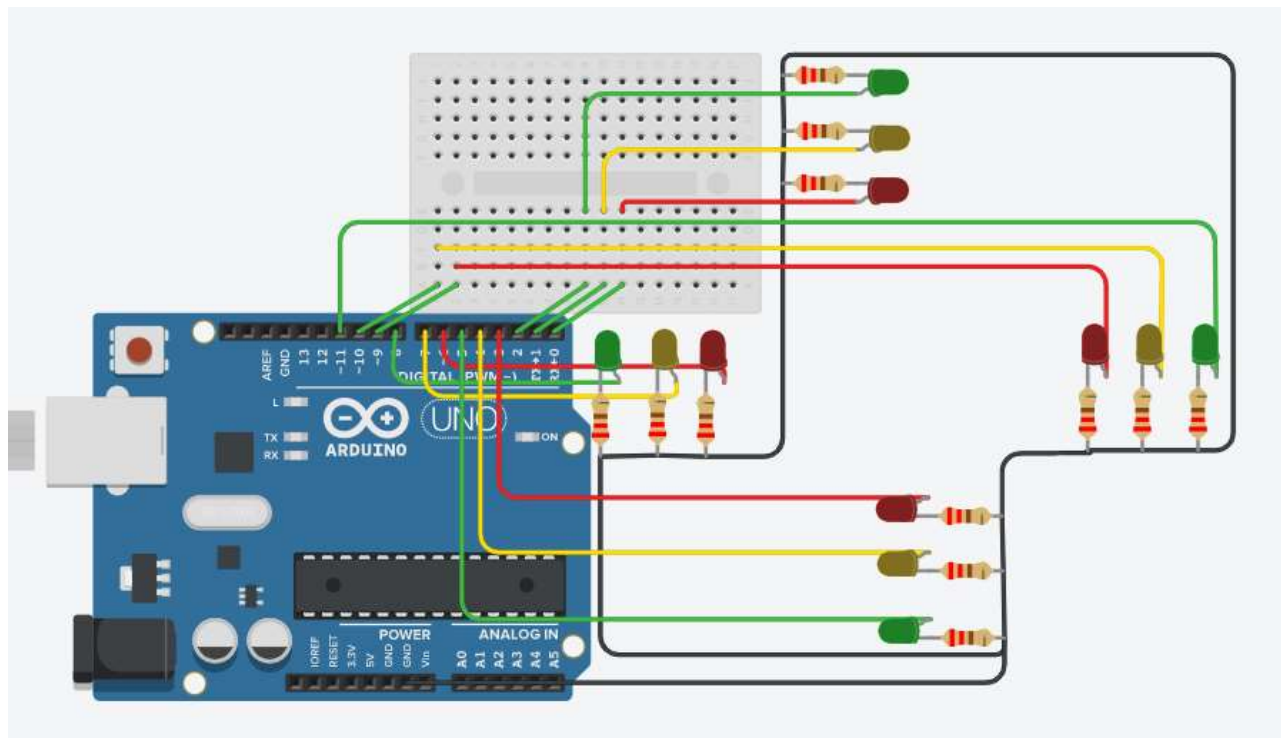
## Experiment - 2

### Aim:

WAP to blink LED light and single pole traffic light system.

### Material requirements:

- Microcontroller (Arduino Uno, for example)
- LED lights (for both the LED light and the single pole traffic light system)
- Resistors (220 ohms)
- Jumper wires
- Breadboard (optional)



### Block Diagram:

**Code:**

Start by defining variables so that you can address the lights by name rather than a number. Start a new Arduino project, and begin with these lines:

```
int red = 10;  
int yellow = 9;  
int green = 8;
```

Next, let's add the setup function, where you'll configure the red, yellow and green LEDs to be outputs. Since you have created variables to represent the pin numbers, you can now refer to the pins by name instead:

```
void setup(){  
    pinMode(red, OUTPUT);  
    pinMode(yellow, OUTPUT);  
    pinMode(green, OUTPUT);  
}
```

The `pinMode` function configures the Arduino to use a given pin as an output. You have to do this for your LEDs to work at all. Now for the actual logic of the traffic light. Here's the code you need. Add this below your variable definitions and setup function:

```
void loop(){  
    changeLights();  
    delay(15000);  
}  
  
void changeLights(){  
    // green off, yellow on for 3 seconds  
    digitalWrite(green, LOW);
```

```
digitalWrite(yellow, HIGH);
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);
delay(3000);
}
```

You should have a working traffic light that changes every 15 seconds

### **Output:**

For the blinking LED light, the LED will turn on and off at 1 second intervals. For the single pole traffic light system, the red light will turn on for 5 seconds, followed by the yellow light for 2 seconds, then the green light for 5 seconds, followed by the yellow light for 2 seconds, and the cycle will repeat.



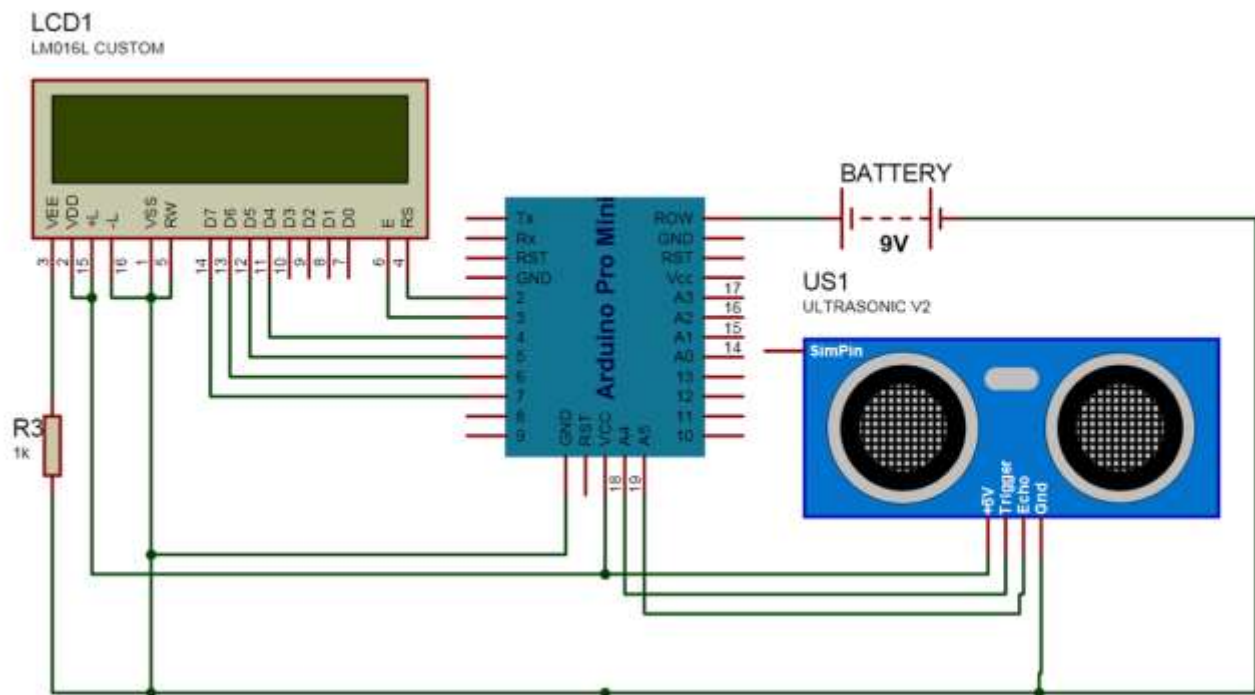
## Experiment - 3

### Aim:

To measure the distance between the Ultrasonic Sensor and an object using an Arduino board and an LED Light.

### Materials Required:

- Arduino Uno or similar board
- Breadboard
- HC-SR04 Ultrasonic Sensor
- LED Light
- 220 ohm resistor
- Jumper wires



### Block Diagram:

**Code :**

```
/*  
 * created by Rui Santos, https://randomnerdtutorials.com  
 *  
 * Complete Guide for Ultrasonic Sensor HC-SR04  
 *  
Ultrasonic sensor Pins:  
    VCC: +5VDC  
    Trig : Trigger (INPUT) - Pin11  
  
    Echo: Echo (OUTPUT) - Pin 12  
    GND: GND  
*/  
  
int trigPin = 11;  // Trigger  
int echoPin = 12;  // Echo  
long duration, cm, inches;  
  
void setup() {  
  
    //Serial Port begin  
    Serial.begin (9600);  
    //Define inputs and outputs  
    pinMode(trigPin, OUTPUT);
```

```
pinMode(echoPin, INPUT);
}

void loop() {
  // 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:

  digitalWrite(trigPin, LOW);
  delayMicroseconds(5);
  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

  cm = (duration/2) / 29.1; // Divide by 29.1 or multiply by 0.0343
  inches = (duration/2) / 74; // Divide by 74 or multiply by 0.0135

  Serial.print(inches);
```

```
Serial.print("in, ");  
Serial.print(cm);  
Serial.print("cm");  
Serial.println();  
  
delay(250);  
}
```

**Output:**

The ultrasonic sensor emits a high-frequency sound pulse and calculates the distance depending upon the time taken by the echo signal to travel back after reflecting from the desired target. The speed of sound is 341 meters per second in air. After the distance is calculated, it will be display

2in , 9cm

5in , 3cm on the LCD display.

## Experiment – 4

### Aim:

Humidity sensor (DHT11/DHT22) with code diagram and output.

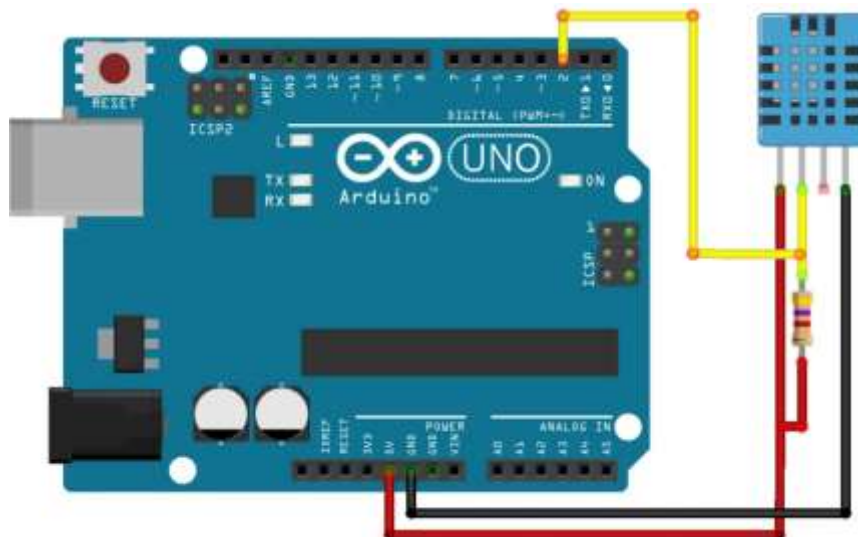
### Materials Required:

- Arduino Uno board
- DHT11 or DHT22 humidity sensor
- Breadboard
- Jumper wires

### Theory of operation:

Both the DHT11 and DHT22 sensors use a capacitive humidity sensor and a thermistor to measure humidity and temperature, respectively. The capacitive humidity sensor measures the capacitance of a thin film of polymer that absorbs water molecules from the air. The amount of water absorbed changes the capacitance of the polymer, which is proportional to the relative humidity of the air. The thermistor measures the temperature of the air, which is required to accurately calculate the relative humidity. The DHT11 sensor has a lower accuracy and range compared to the DHT22 sensor.

### Block Diagram:



**Code:**

We will be using the Adafruit DHT library to interface with the sensors. The library can be downloaded from the Arduino IDE library manager. The library provides functions to initialize the sensor, read the humidity and temperature, and handle any errors.

```
#include <DHT.h>

#define DHTPIN 2    // Digital pin connected to the DHT sensor
#define DHTTYPE DHT11 // DHT 11

DHT dht(DHTPIN, DHTTYPE);

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

void loop() {
  delay(2000); // Wait 2 seconds between measurements

  float h = dht.readHumidity();
  float t = dht.readTemperature();

  if (isnan(h) || isnan(t)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }
```

```
Serial.print("Humidity: ");
Serial.print(h);
Serial.print(" %\t");
Serial.print("Temperature: ");
Serial.print(t);
Serial.println(" *C");
}
#include <DHT.h>

#define DHTPIN 2    // Digital pin connected to the DHT sensor
#define DHTTYPE DHT11 // DHT 11

DHT dht(DHTPIN, DHTTYPE);

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

void loop() {
  delay(2000); // Wait 2 seconds between measurements

  float h = dht.readHumidity();
  float t = dht.readTemperature();

  if (isnan(h) || isnan(t)) {
    Serial.println("Failed to read from DHT sensor!");
```

```
    return;
}

Serial.print("Humidity: ");
Serial.print(h);
Serial.print(" %\t");
Serial.print("Temperature: ");
Serial.print(t);
Serial.println(" *C");
}
```

The code starts by including the DHT library and defining the pin number and type of sensor being used. In the setup function, the serial monitor is initialized and the sensor is initialized using the `dht.begin()` function. In the loop function, the humidity and temperature are read using the `dht.readHumidity()` and `dht.readTemperature()` functions, respectively. The `isnan()` function is used to check if the readings are valid. If the readings are valid, they are printed on the serial monitor using the `Serial.print()` function.

### Output:

The output on the serial monitor should look something like this:

```
Humidity: 50.00 %    Temperature: 25.00 *C
Humidity: 50.10 %    Temperature: 25.10 *C
Humidity: 50.20 %    Temperature: 25.20 *C
Humidity: 50.30 %    Temperature: 25.30 *C
```



## Experiment – 5

### Aim:

Motion Detection with ESP32 & PIR Sensor.

### Materials Required:

- ESP32 Development Board
- PIR motion sensor (HC-SR501)
- LED
- 1k Ohm resistor
- Jumper Wires
- Breadboard

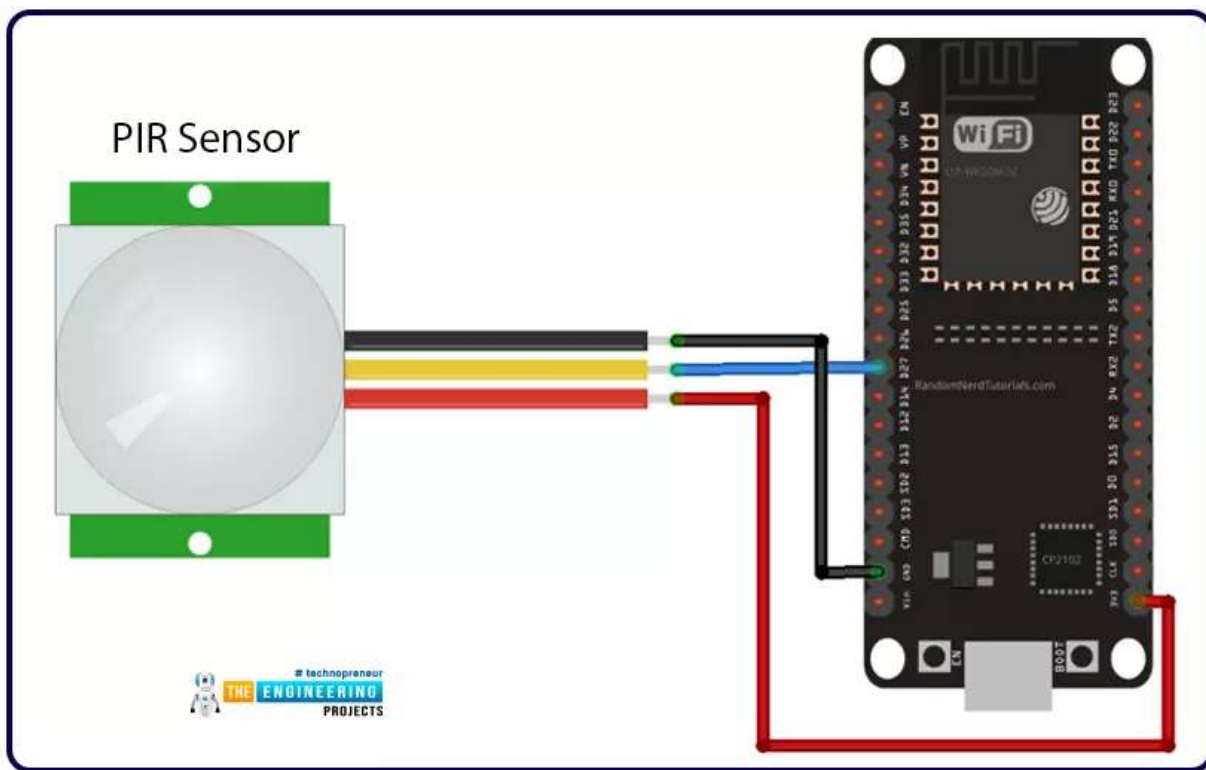
### Theory of operation:

PIR Sensor has two variable resistors on its back side, used to adjust the Sensitivity and Detection Range, explained below:

- Low sensitivity ignores the small motions i.e. a moving leaf or a small mouse. The sensitivity can be adjusted based on the installation location and project requirements.

- The second resistor is used to specify how long the detection output should be active. It can be set to turn on for as little as a few seconds or as long as a few minutes.

### Block Diagram:



### Code:

```
const int PIR_SENSOR_OUTPUT_PIN = 13; /* PIR sensor O/P pin */
int warm_up;

void setup() {
  pinMode(PIR_SENSOR_OUTPUT_PIN, INPUT);
  Serial.begin(115200); /* Define baud rate for serial communication */
  Serial.println("Waiting For Power On Warm Up");
  delay(20000); /* Power On Warm Up Delay */
}
```

```

    Serial.println("Ready!");
}

void loop() {
    int sensor_output;
    sensor_output = digitalRead(PIR_SENSOR_OUTPUT_PIN);
    if( sensor_output == LOW )
    {
        if( warm_up == 1 )
        {
            Serial.print("Warming Up\n\n");
            warm_up = 0;
            delay(2000);
        }
        Serial.print("No object in sight\n\n");
        delay(1000);
    }
    else
    {
        Serial.print("Object detected\n\n");
        warm_up = 1;
        delay(1000);
    }
}

```

**Output:**

```
Motion detected
Turning On the the LED
Turning OFF the the LED
00:04:56.721 -> Motion detected
00:04:56.721 -> Turning On the the LED
00:05:01.725 -> Turning OFF the the LED
00:05:11.720 -> Motion detected
00:05:11.720 -> Turning On the the LED
00:05:16.717 -> Turning OFF the the LED
00:05:26.726 -> Motion detected
00:05:26.726 -> Turning On the the LED
```

## Experiment – 6

### Aim:

ESP32 Boards with In-built Temperature Sensor

### Materials Required:

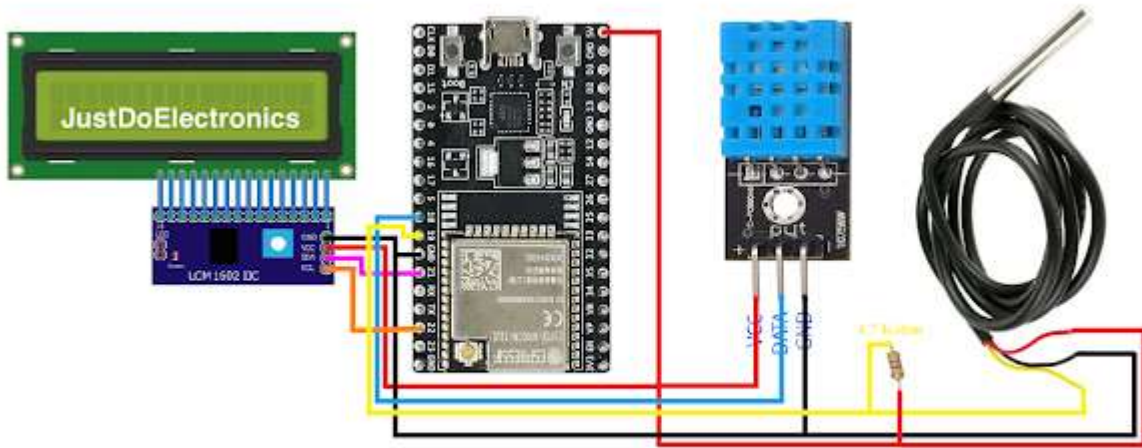
- ESP32 Board

- DHT11 Sensor
- DS18B20 sensor
- 4.7Kohm Resister
- 16x2 LCD Display With I2C
- Zero PCB

### Theory of operation:

ESP32 boards come in many configurations, and not all have an in-built temperature sensor. The ESP32-S3 is one board that has a built-in temperature sensor that measures the chip's internal temperature. The sensor module has an 8-bit Sigma-Delta ADC and a DAC to compensate for the temperature offset. The temperature sensor works on any ESP32-S3 board, regardless of which board the chip is embedded in.

### Block Diagram:



### Code:

```
#include <OneWire.h>
#include <DallasTemperature.h>

#define SENSOR_PIN 17 // ESP32 pin GPIO17 connected to DS18B20 sensor's
DATA pin

OneWire oneWire(SENSOR_PIN);
DallasTemperature DS18B20(&oneWire);

float tempC; // temperature in Celsius
float tempF; // temperature in Fahrenheit

void setup() {
  Serial.begin(9600); // initialize serial
  DS18B20.begin();    // initialize the DS18B20 sensor
}

void loop() {
  DS18B20.requestTemperatures(); // send the command to get tempera-
tures
  tempC = DS18B20.getTempCByIndex(0); // read temperature in °C
  tempF = tempC * 9 / 5 + 32; // convert °C to °F

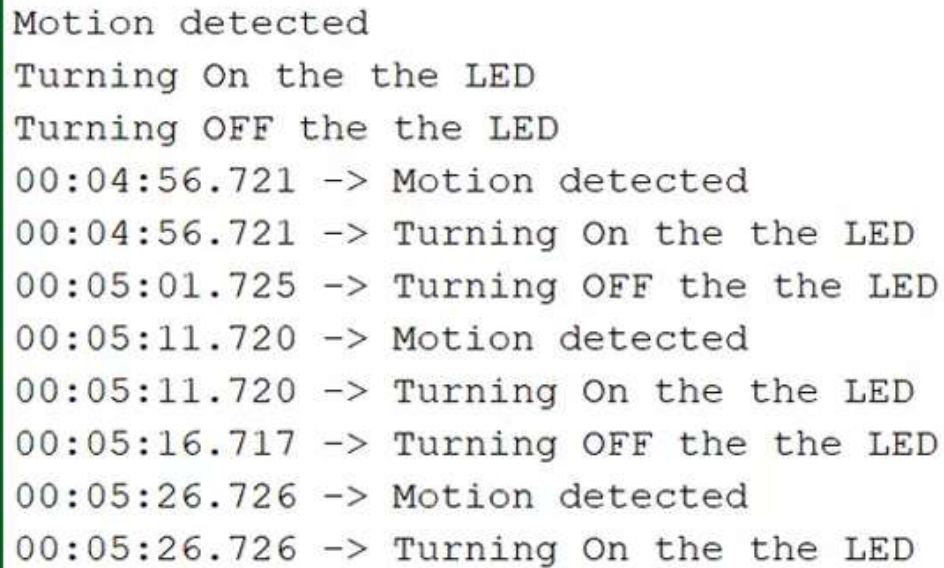
  Serial.print("Temperature: ");
  Serial.print(tempC); // print the temperature in °C
  Serial.print("°C");
  Serial.print(" ~ "); // separator between °C and °F
  Serial.print(tempF); // print the temperature in °F
```

```
Serial.println("°F");
```

```
delay(500);
```

```
}
```

### Output:



```
Motion detected
Turning On the the LED
Turning OFF the the LED
00:04:56.721 -> Motion detected
00:04:56.721 -> Turning On the the LED
00:05:01.725 -> Turning OFF the the LED
00:05:11.720 -> Motion detected
00:05:11.720 -> Turning On the the LED
00:05:16.717 -> Turning OFF the the LED
00:05:26.726 -> Motion detected
00:05:26.726 -> Turning On the the LED
```

## Experiment – 7

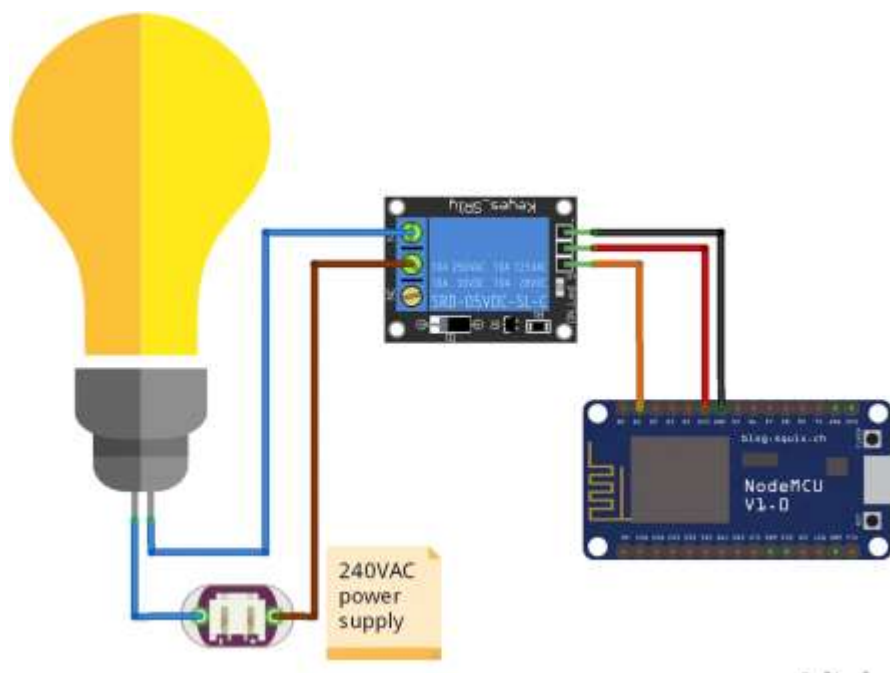
### Aim:

Switching light on/off remotely using ESP8266 (AP-IF)

### Materials Required:

- ESP8266 development board
- Breadboard
- Jumper wires
- LED light
- 220-ohm resistor
- USB cable
- Laptop/PC with Arduino IDE and Serial Monitor

### Circuit Diagram:



Note: Make sure to connect the resistor with the anode of the LED (positive leg).

### Code:



```
#include <ESP8266WiFi.h>

const char* ssid = "YourNetworkSSID";
const char* password = "YourNetworkPassword";

WiFiServer server(80);

void setup() {
    Serial.begin(115200);
    pinMode(LED_BUILTIN, OUTPUT);

    // Connect to Wi-Fi network
    Serial.println();
    Serial.println();
    Serial.print("Connecting to ");
    Serial.println(ssid);

    WiFi.begin(ssid, password);

    while (WiFi.status() != WL_CONNECTED) {
        delay(1000);
        Serial.print(".");
    }

    Serial.println("");
    Serial.println("WiFi connected");

    // Start the server
```

```
server.begin();
Serial.println("Server started");
}

void loop() {
    // Check if a client has connected
    WiFiClient client = server.available();
    if (!client) {
        return;
    }

    // Wait until the client sends some data
    Serial.println("New client");
    while (!client.available()) {
        delay(1);
    }

    // Read the first line of the request
    String request = client.readStringUntil('\r');
    Serial.println(request);
    client.flush();

    // Match the request
    int value = LOW;
    if (request.indexOf("/LED=ON") != -1) {
        digitalWrite(LED_BUILTIN, HIGH);

        value = HIGH;
    }
}
```

```

    } else if (request.indexOf("/LED=OFF") != -1) {
        digitalWrite(LED_BUILTIN, LOW);
value = LOW
    }

    // Set the LED according to the request
    //digitalWrite(LED_BUILTIN, value);

    // Return the response
    client.println("HTTP/1.1 200 OK");
    client.println("Content-Type: text/html");
    client.println("");
    client.println("<!DOCTYPE HTML>");
    client.println("<html>");
    client.println("<head><title>ESP8266 LED Control</title></head>");
    client.println("<body>");
    client.println("<h1>ESP8266 LED Control</h1>");
    client.println("<p>Click <a href=\"/LED=ON\">here</a> to turn the LED
on.</p>");
    client.println("<p>Click <a href=\"/LED=OFF\">here</a> to turn the LED
off.</p>");
    client.println("</body>");
    client.println("</html>");

    delay(1);
    Serial.println("Client disconnected");
    Serial.println("");
}

```

**Procedure:**

1. Connect the circuit as per the circuit diagram.
2. Connect the ESP8266 development board to the laptop/PC using a USB cable.
3. Open the Arduino IDE on the laptop/PC and copy-paste the code given above.
4. Update the code with your network SSID and password.
5. Compile and upload the code to the ESP8266 development board.
6. Open the Serial Monitor from the Arduino IDE and set the baud rate to 115200.
7. Wait for the ESP826

**Output:**

Connect to the AP created by the ESP8266 using your laptop/PC or mobile device.

1. Open a web browser and enter the IP address of the ESP8266 (default IP address is 192.168.4.1).
2. The web page should display two links/buttons to turn the LED on and off.
3. Click on the link/button to turn the LED on or off.
4. The LED on the breadboard should turn on or off accordingly.
5. The Serial Monitor in the Arduino IDE will display the status of the client connection and the status of the LED.

Note: The LED\_BUILTIN pin is used as the output pin for controlling the LED in this lab file. If you are using a different output pin, make sure to update the pin number in the code.

## Experiment - 8

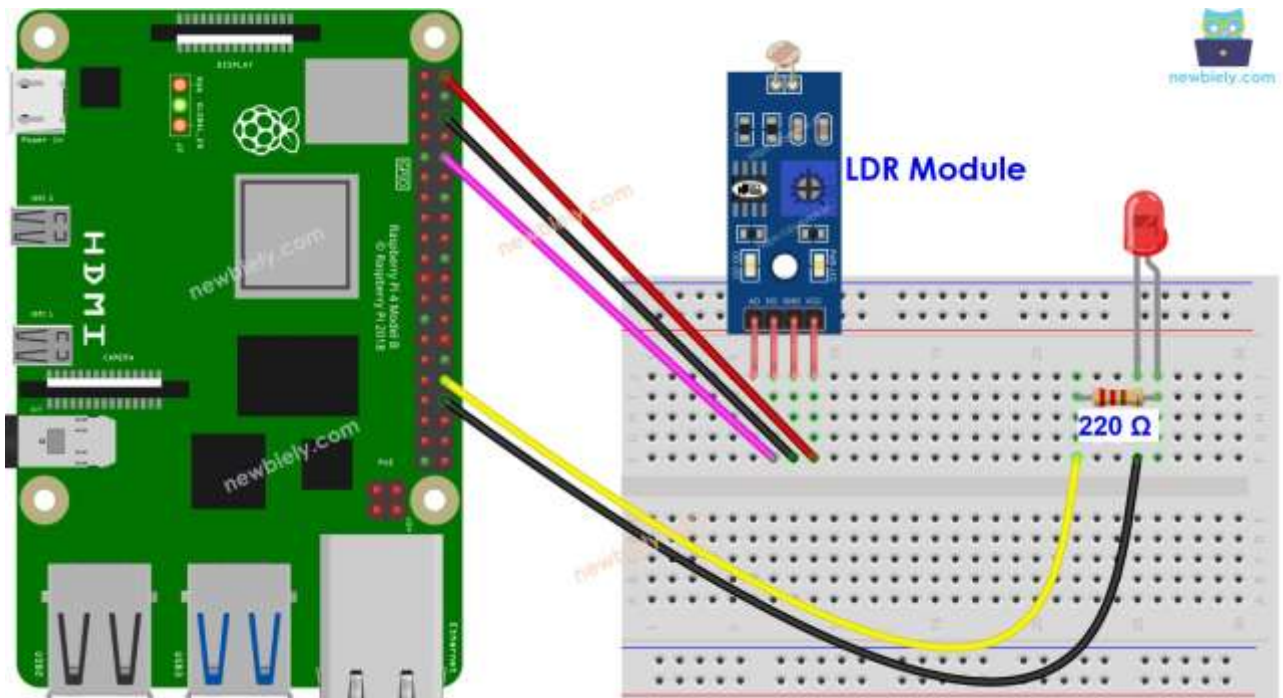
### Aim:

Raspberry Pi Light Sensor using an LDR

### Materials Required:

- Led
- 220 ohm resistor
- Breadboard
- Jumper Wires
- Raspberry Pi 4 Model B
- LDR Light Sensor Module
- 

### Circuit Diagram:



**Code:**

```
#!/usr/local/bin/python
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BOARD)

#define the pin that goes to the circuit
pin_to_circuit = 7
def rc_time (pin_to_circuit):
    count = 0

    #Output on the pin for
    GPIO.setup(pin_to_circuit, GPIO.OUT)
    GPIO.output(pin_to_circuit, GPIO.LOW)
    time.sleep(0.1)

    #Change the pin back to input
    GPIO.setup(pin_to_circuit, GPIO.IN)

    #Count until the pin goes high
    while (GPIO.input(pin_to_circuit) == GPIO.LOW):
        count += 1

    return count

#Catch when script is interrupted, cleanup correctly
try:
    # Main loop
    while True:
```

```

        print(rc_time(pin_to_circuit))
except KeyboardInterrupt:
    pass
finally:
    GPIO.cleanup()
git clone https://github.com/pimylifeup/Light_Sensor/
cd ./Light_Sensor

```

### Procedure:

- Make sure you have Raspbian or any other Raspberry Pi compatible operating system installed on your Pi.

```


```

- Make sure your Raspberry Pi is connected to the same local network as your PC.

```


```

- Make sure your Raspberry Pi is connected to the internet if you need to install some libraries.

```


```

- If this is the first time you use Raspberry Pi, See how to set up the Raspberry Pi

```


```

- Connect your PC to the Raspberry Pi via SSH using the built-in SSH client on Linux and macOS or PuTTY on Windows. See to how connect your PC to Raspberry Pi via SSH.
- Make sure you have the RPi.GPIO library installed.

### Output:

234

245

256

248

## **Experiment – 9**

**Aim:**

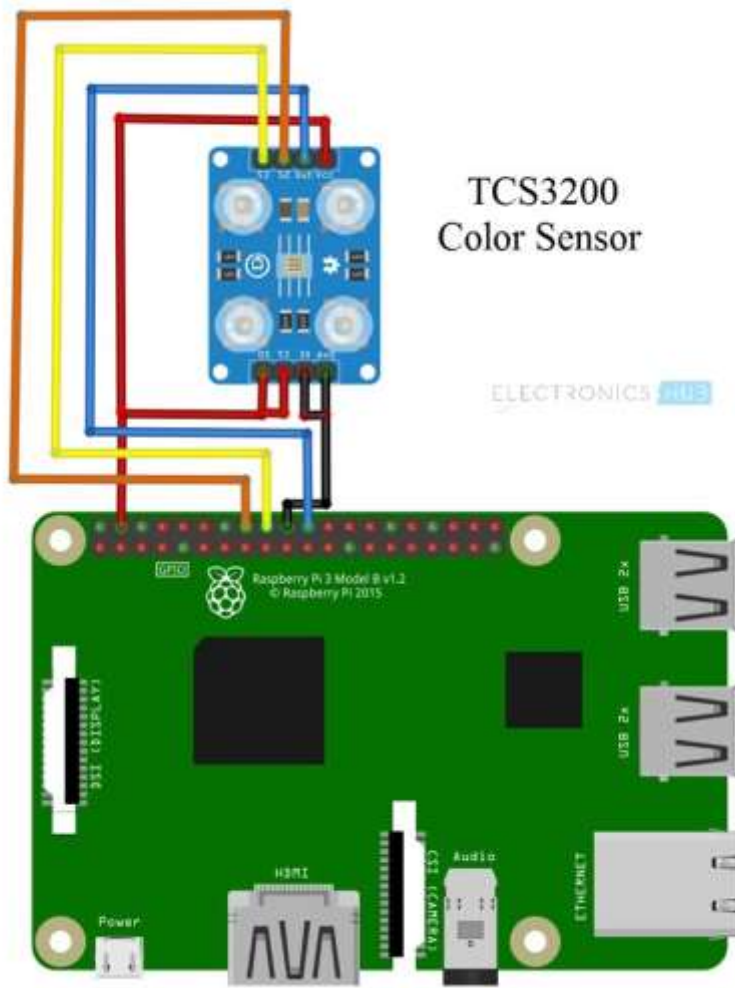
Raspberry Pi Colour Sensor using an LDR

**Materials Required:**

- Raspberry Pi
- TCS3200 Color Sensor
- Mini Breadboard
- Connecting Wires
- Power Supply
- Computer

**Circuit Diagram:**





**Code:**

```
import
Rpi.GPIO
as GPIO

import time

s2 = 23
s3 = 24
signal = 25
NUM_CYCLES = 10

def setup():
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(signal,GPIO.IN, pull_up_down=GPIO.PUD_UP)
```

```

GPIO.setup(s2,GPIO.OUT)
GPIO.setup(s3,GPIO.OUT)
print("\n")

def loop():
    temp = 1
    while(1):

        GPIO.output(s2,GPIO.LOW)
        GPIO.output(s3,GPIO.LOW)
        time.sleep(0.3)
        start = time.time()
        for impulse_count in range(NUM_CYCLES):
            GPIO.wait_for_edge(signal, GPIO.FALLING)
            duration = time.time() - start
            red = NUM_CYCLES / duration

        GPIO.output(s2,GPIO.LOW)
        GPIO.output(s3,GPIO.HIGH)
        time.sleep(0.3)
        start = time.time()
        for impulse_count in range(NUM_CYCLES):
            GPIO.wait_for_edge(signal, GPIO.FALLING)
            duration = time.time() - start
            blue = NUM_CYCLES / duration

        GPIO.output(s2,GPIO.HIGH)
        GPIO.output(s3,GPIO.HIGH)
        time.sleep(0.3)
        start = time.time()
        for impulse_count in range(NUM_CYCLES):
            GPIO.wait_for_edge(signal, GPIO.FALLING)
            duration = time.time() - start
            green = NUM_CYCLES / duration

    if green<7000 and blue<7000 and red>12000:
        print("red")
    temp=1

```

```

elif red<12000 and blue<12000 and green>12000:
print("green")
temp=1
elif green<7000 and red<7000 and blue>12000:
print("blue")
temp=1
elif red>10000 and green>10000 and blue>10000 and
temp==1:
print("place the object.....")
temp=0

def endprogram():
GPIO.cleanup()

if __name__=='__main__':

setup()

try:
loop()

except KeyboardInterrupt:
endprogram()

```

## Procedure:

The aim of this simple project is to understand the Raspberry Pi Color Sensor Interface and how can we make a Color Detection application using Raspberry Pi and TCS3200 Color Sensor.

Now, coming to the working, since both the S0 and S1 inputs of the TCS3200 Color Sensor are connected to +5V, the output frequency is scaled to 100% i.e. the output frequency will be in the range of 500 KHz to 600 KHz.

As S2 and S3 pins of the TCS3200 Color Sensor are used to select the Photo Diode, they are set in three different combination one after the other to get the RAW data of the Red, Blue and Green values.

Keeping these values as reference, the color detection program is written, where the Raspberry Pi correctly displays the name of the color placed in front of the sensor.

**Output:**

- If the detected color is primarily red: red
- If the detected color is primarily green: green
- If the detected color is primarily blue: blue
- If all three colors are detected in significant amounts and it's ready to place an object: place the object.....

The code continuously loops and checks for color readings from the sensor. Depending on the detected colors and their intensities, it will print out one of the above msg. If none of the conditions are met, it will continue to loop and check for color readings.

## Experiment – 10

### Aim:

OS installation in Raspberry pi.

### Materials Required:

- Raspberry Pi board (any model)
- Micro SD card (minimum 8GB)
- Micro SD card reader
- Laptop/PC with internet connectivity
- Power supply for Raspberry Pi
- HDMI cable (optional)
- USB keyboard and mouse (optional)

### Software Required:

- Raspberry Pi Imager
- Operating System image file (e.g., Raspbian, Ubuntu, etc.)

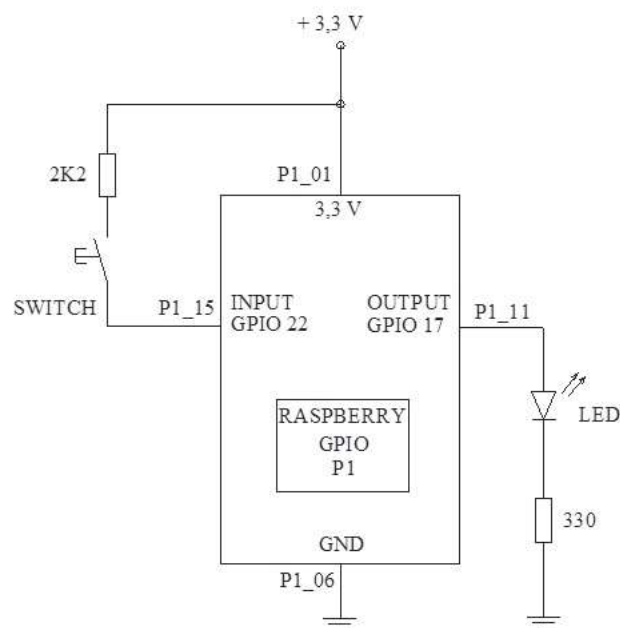
### Procedure:

1. Insert the Micro SD card into the Micro SD card reader and connect it to your laptop/PC.
2. Launch the Raspberry Pi Imager software and select the operating system you want to install on your Raspberry Pi. If the desired operating system is not listed, you can select "Use Custom" and select the image file.
3. Select the Micro SD card as the target device for the operating system installation.
4. Click on "Write" and wait for the image to be written to the Micro SD card.
5. Once the image is written to the Micro SD card, safely eject the Micro SD card and insert it into the Raspberry Pi.
6. Connect the Raspberry Pi to a display (HDMI) and power supply.
7. If required, connect a USB keyboard and mouse to the Raspberry Pi.

8. Power on the Raspberry Pi and wait for the operating system to boot up.
9. Follow the on-screen instructions to complete the initial setup of the operating system.

Note: If you do not have a display or keyboard/mouse, you can still install and configure the operating system on the Raspberry Pi by setting up SSH and connecting to it remotely using a laptop/PC.

### Circuit Diagram:



### Code:

EXAMPLE OUTPUT GPIO IN “C”

```
// led.c
// Example for output data at GPIO17 that is PIN P1_11
// Turn on and off every 2 secs a Led connected to this pin
// we need the library bcm2835.h and bcm2835.c
// gcc -o led led.c -l bcm2835 (for compile)
```

```
// ./led (For execution)
//
#include <bcm2835.h>
#define PIN RPI_GPIO_P1_11
int main(int argc, char **argv)
{
    if (!bcm2835_init())
        return 1;
    // Set the pin as Output
    bcm2835_gpio_fsel(PIN, BCM2835_GPIO_FSEL_OUTP);
    // Loop
    while (1)
    {
        // Turn Led on
        bcm2835_gpio_write(PIN, HIGH);
        // wait 2 seconds
        sleep(2);
        // turn Led off
        bcm2835_gpio_write(PIN, LOW);
        // wait 2 seconds
        sleep(2);
    }

    bcm2835_close();
    return 0;
}
```

```
//EXAMPLE OF INPUT GPIO IN "C"
#include <bcm2835.h>
#include <stdio.h>

// Input on pin GPIO 22
#define PIN RPI_GPIO_P1_15

int main(int argc, char **argv)
{
    if (!bcm2835_init())
        return 1;
    // Set GPIO 22 pin P1-15 as an input
    bcm2835_gpio_fsel(PIN, BCM2835_GPIO_FSEL_INPT);
    // with a pullup
    bcm2835_gpio_set_pud(PIN, BCM2835_GPIO_PUD_UP);

    while (1)
    {
        // Read data
        uint8_t value = bcm2835_gpio_lev(PIN);
        printf("Value of pin 15: %dn", value);
        // wait 2 seconds
        sleep(2);
    }
    bcm2835_close();
    return 0;
}
```



```
}
```