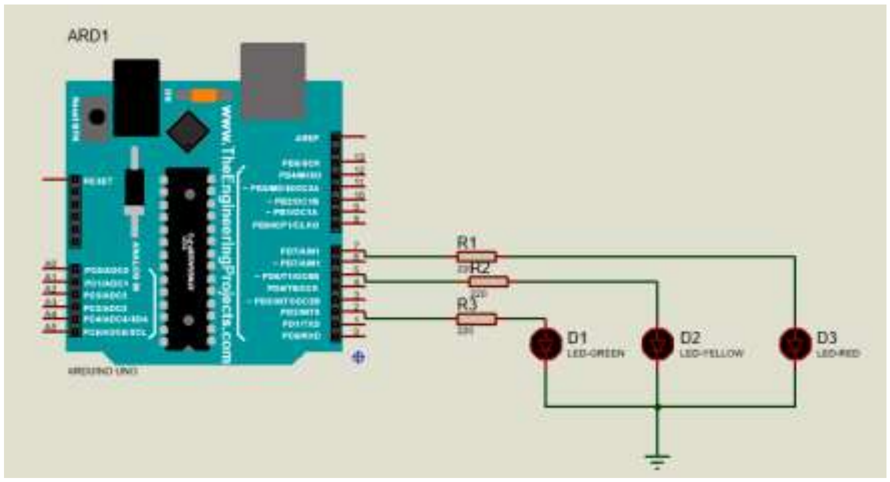
1. TRAFFIC LIGHT (LED)

A traffic light system controls the flow of vehicles at intersections using Red, Yellow, and Green LEDs.

Logic of operation:

Light	Duration	Purpose
Green	20 sec	Go
Yellow	6 sec	Prepare to stop
Red	15 sec	Stop



The LEDs have been powered by Arduino UNO (Board). It contains a code which uploaded to the board. And once it simulated LED Start's blinking like a traffic light. 15 Second will for Red Light, 6 Second for Yellow Light and 20 Second for Green Light.

CODE:

```
int red = 7;
int yellow = 5;
int green = 2;
void setup(){

 pinMode(red, OUTPUT);
 pinMode(yellow, OUTPUT);
 pinMode(green,  OUTPUT);
```

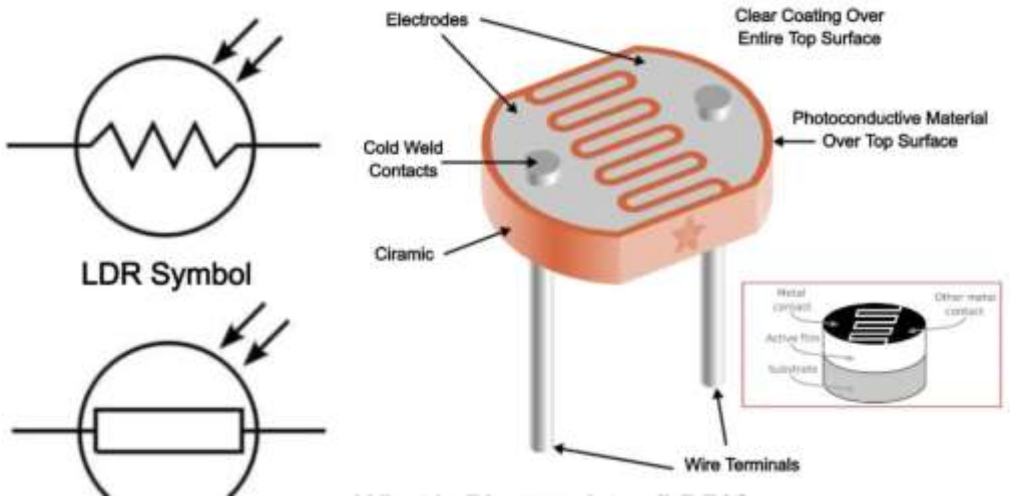
```
}
void loop(){
digitalWrite(red, HIGH);
 delay(15000);
digitalWrite(red,  LOW);
 digitalWrite(yellow, HIGH);
delay(1000);
  digitalWrite(yellow,  LOW);
delay(500);
digitalWrite(yellow, HIGH);
delay(1000);
  digitalWrite(yellow,  LOW);
delay(500);

  digitalWrite(yellow, HIGH);
delay(1000);
  digitalWrite(yellow,  LOW);
delay(500);

  digitalWrite(yellow, HIGH);
delay(1000);
  digitalWrite(yellow, LOW);
delay(500);

  digitalWrite(yellow, HIGH);
delay(1000);
  digitalWrite(yellow, LOW);
delay(500);

digitalWrite(green, HIGH);
delay(20000);
```

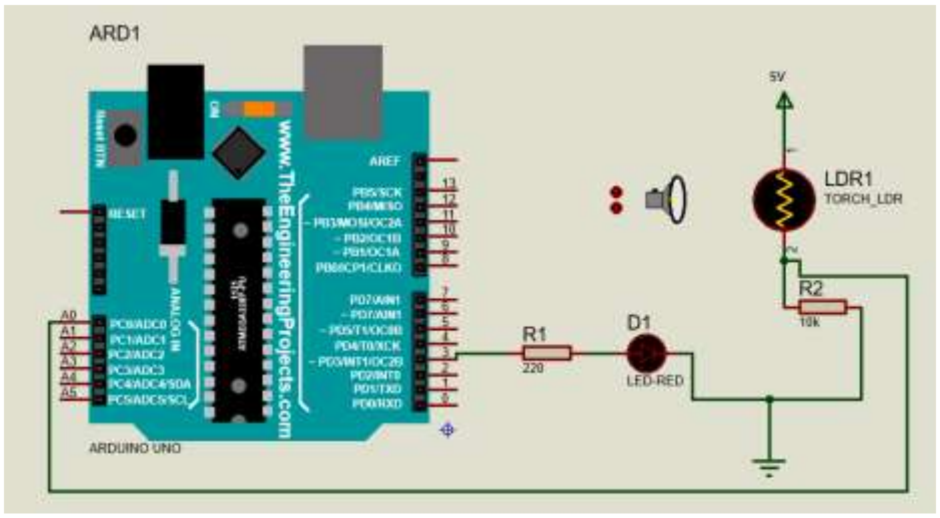
```
digitalWrite(green,  LOW);
//
digitalWrite(yellow, HIGH);
delay(1000);
  digitalWrite(yellow,  LOW);
delay(500);

  digitalWrite(yellow, HIGH);
delay(1000);
  digitalWrite(yellow,  LOW);
delay(500);

  digitalWrite(yellow, HIGH);
delay(1000);
  digitalWrite(yellow, LOW);
delay(500);

  digitalWrite(yellow, HIGH);
delay(1000);
  digitalWrite(yellow, LOW);
delay(500);

  digitalWrite(yellow, HIGH);
delay(1000);
  digitalWrite(yellow, LOW);
delay(500);
}
```

2. LIGHT DEPENDENT RESISTOR (LDR)

LDR is a special type of resistor that works on the photoconductivity principle means that resistance changes according to the intensity of light. Its resistance decreases with an increase in the intensity of light. It is often used as a light sensor, light meter, Automatic street light, and in areas where we need to have light sensitivity. LDR is also known as a Light Sensor.

Whenever the light falls on its photoconductive material, it absorbs its energy and the electrons of that photoconductive material in the valence band get excited and go to the conduction band and thus increasing the conductivity as per the increase in light intensity. Also, the energy in incident light should be greater than the bandgap gap energy so that the electrons from the valence band got excited and go to the conduction band.


LDR Symbol
Electrodes
Clear Coating Over Entire Top Surface
Photoconductive Material Over Top Surface
Cold Weld Contacts
Ciramic
Wire Terminals



CODE:

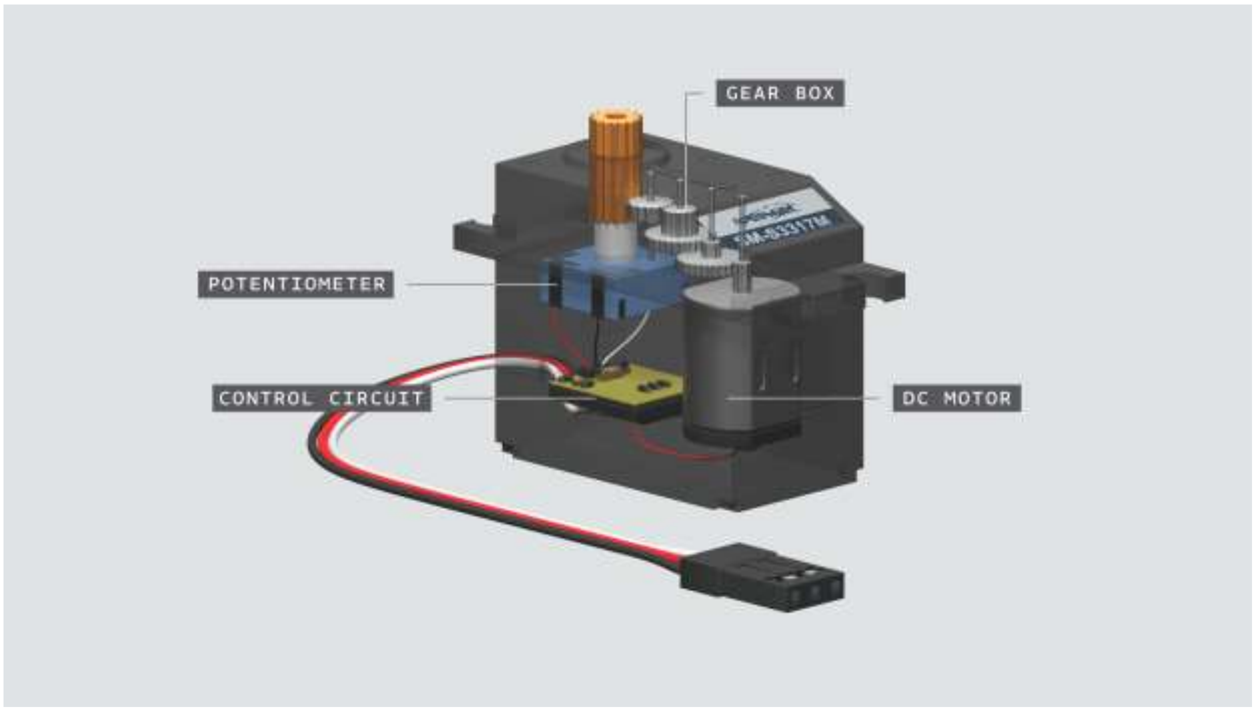
```
const int LIGHT_SENSOR_PIN = A0;
const int LED_PIN          = 3;
const int ANALOG_THRESHOLD = 500;
int analogValue;
void setup() {
  pinMode(LED_PIN, OUTPUT);
}
void loop() {
  analogValue = analogRead(LIGHT_SENSOR_PIN);
  if(analogValue < ANALOG_THRESHOLD)
    digitalWrite(LED_PIN, HIGH); // turn on LED
  else
    digitalWrite(LED_PIN, LOW);  // turn off LED
}
```

3. SERVO MOTOR

Servo motors are actuators that allow for precise control of position (angle). A typical characteristic is that the angle of the motor is 0 - 180 degrees. With other words, it can make one half of a rotation.

A standard servo motor, just as other motors, are essentially just a **DC motor**, but with some extra features:

- Control circuit for controlling the motor, e.g. setting the angle.
- Gears that transform speed into torque, which makes it capable of doing "heavy lifting" at a slower speed, as opposed to a regular DC motor that just spins very fast!
- Potentiometer that keeps track of its angle. This makes it possible for the servo to "know where it is".



PWM (Pulse Width Modulation) signals are used to control the angle such that: **0°** → ~1ms pulse, **90°** → ~1.5ms pulse, **180°** → ~2ms pulse.

Servo motors have three wires: power, ground, and signal. The power wire is typically red, and should be connected to positive pole (+) of the power source. The ground wire is typically black or brown and should be connected to the negative pole (-) of power source. The signal pin is typically yellow or orange and should be connected to PWM pin on the board (3,5,6,9,10,11).

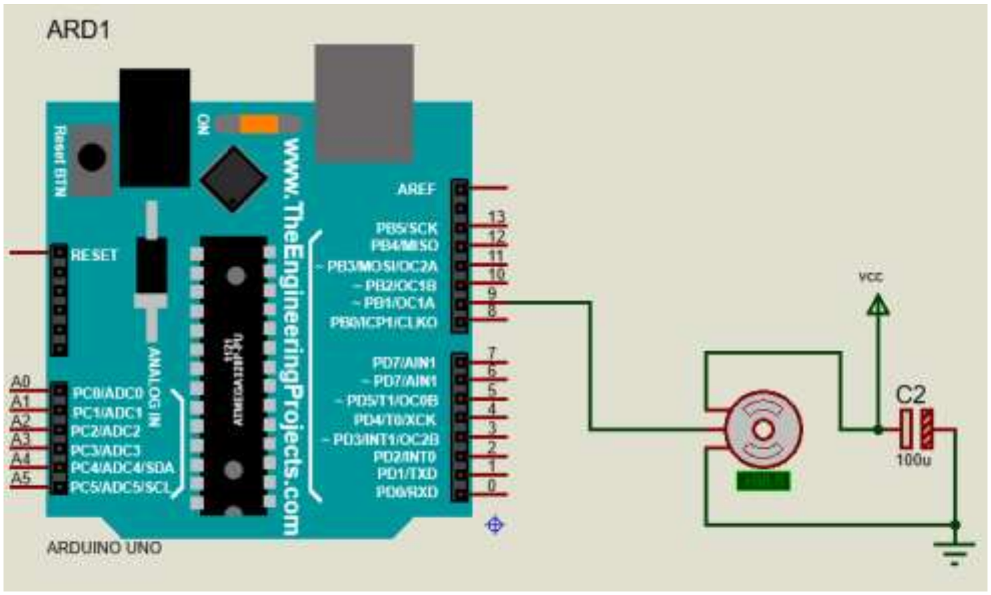


Figure: Sweeps the shaft of a RC servo motor back and forth across 180 degrees

CODE:

```
#include <Servo.h>
Servo myservo;
int pos = 0;
void setup() {
  myservo.attach(9);
}
void loop() {
  for (pos = 0; pos <= 180; pos += 1) {

    myservo.write(pos);
    delay(15);
  }
  for (pos = 180; pos >= 0; pos -= 1) {
    myservo.write(pos);
    delay(15);
  }
}
```

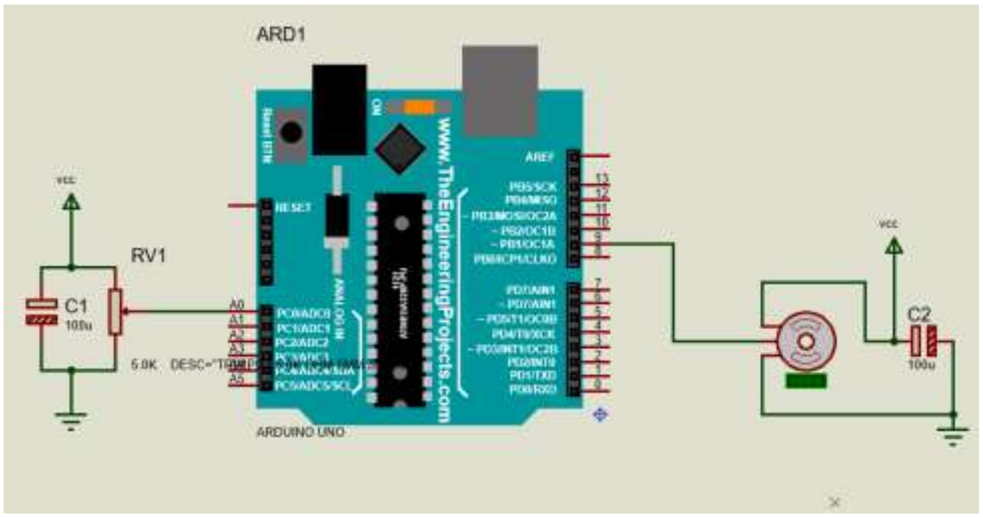


Figure: Controlling a servo position using a potentiometer (variable resistor).

CODE:

```
#include <Servo.h>
Servo myservo;
int potpin = 0;
int val;
void setup() {
  myservo.attach(9);
}
void loop() {
  val = analogRead(potpin);
  val = map(val, 0, 1023, 0, 180);
  myservo.write(val);
  delay(15);
}
```

4. Servo Motor with Light Sensor (LDR)

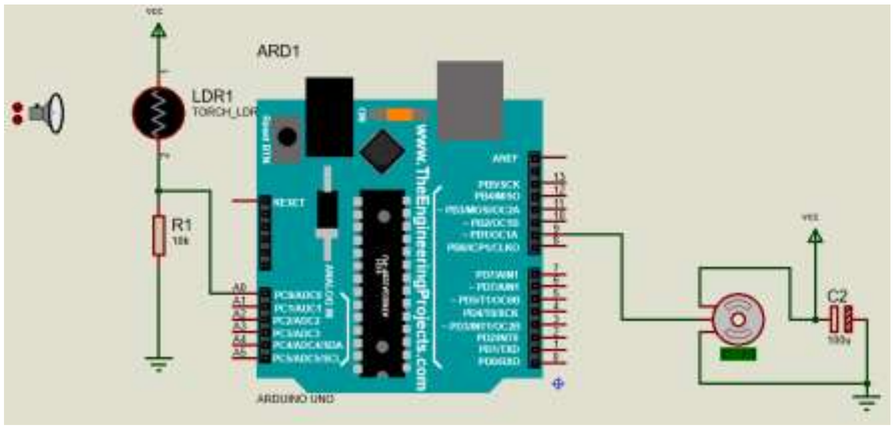


Figure: Schematic diagram

CODE:

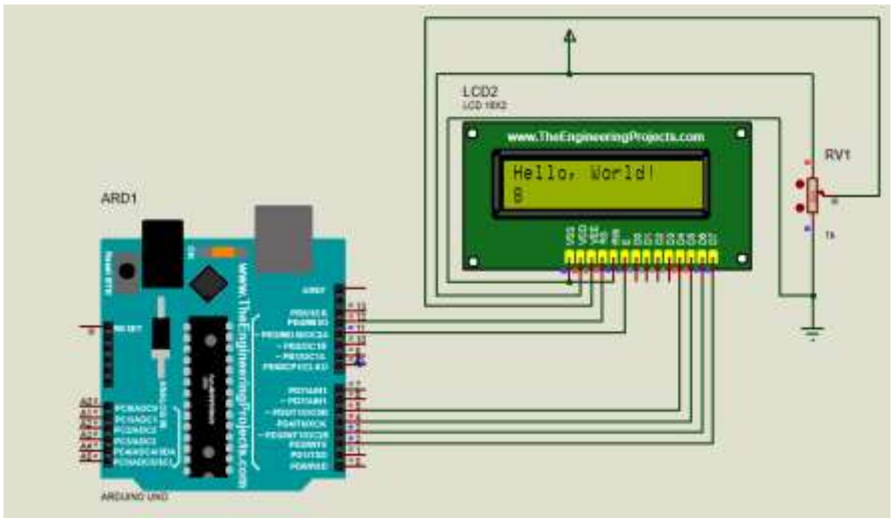
```
#include <Servo.h>
const int LIGHT_SENSOR_PIN = A0;
const int SERVO_PIN = 9;
const int ANALOG_THRESHOLD = 500;
Servo servo;
void setup() {
  servo.attach(SERVO_PIN);
  servo.write(0);
}
void loop() {
  int analogValue = analogRead(LIGHT_SENSOR_PIN);
  if (analogValue > ANALOG_THRESHOLD)
    servo.write(90);
  else
    servo.write(0);
}
```

5. **LCD (Liquid Crystal Display)**

The LCDs have a parallel interface, meaning that the microcontroller has to manipulate several interface pins at once to control the display. The interface consists of the following pins:

- A **register select (RS) pin** that controls where in the LCD's memory you're writing data to. You can select either the data register, which holds what goes on the screen, or an instruction register, which is where the LCD's controller looks for instructions on what to do next.
- A **Read/Write (R/W) pin** that selects reading mode or writing mode
- An **Enable pin** that enables writing to the registers
- 8 **data pins (D0 -D7)**. The states of these pins (high or low) are the bits that you're writing to a register when you write, or the values you're reading when you read

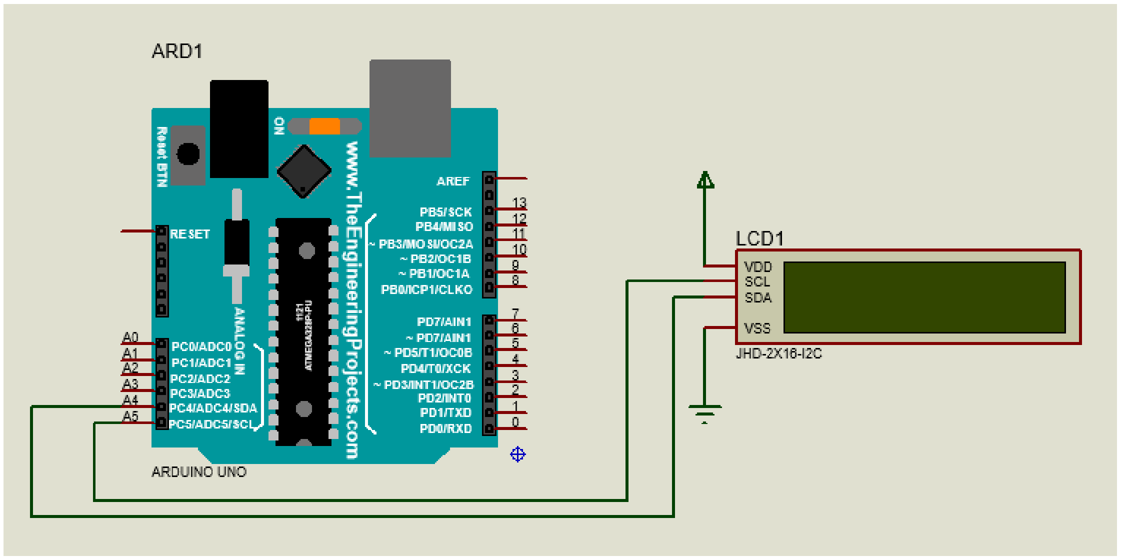
There's also a **display contrast pin (Vo)**, **power supply pins (+5V and GND)** and **LED Backlight (Bklt+ and BKlt-)** pins that can be used to power the LCD, control the display contrast, and turn on and off the LED backlight, respectively.


LCD2
LCD 16X2
www.TheEngineeringProjects.com
Hello, World!
8
RV1
ARD1
www.TheEngineeringProjects.com
ARDUINO UNO

CODE:

```
#include <LiquidCrystal.h>
const int rs = 12, en = 11, d4 = 5, d5 = 4, d6 = 3, d7 = 2;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);  // Adjust pins based on your wiring
```

```
void setup() {
  lcd.begin(16, 2);  // Initialize 16x2 LCD
  lcd.print("Hello, World!");  // Print a message on the first row
}
void loop() {

  lcd.setCursor(0, 1);  // Set cursor to the first line
  lcd.print(millis() / 1000);
}
```



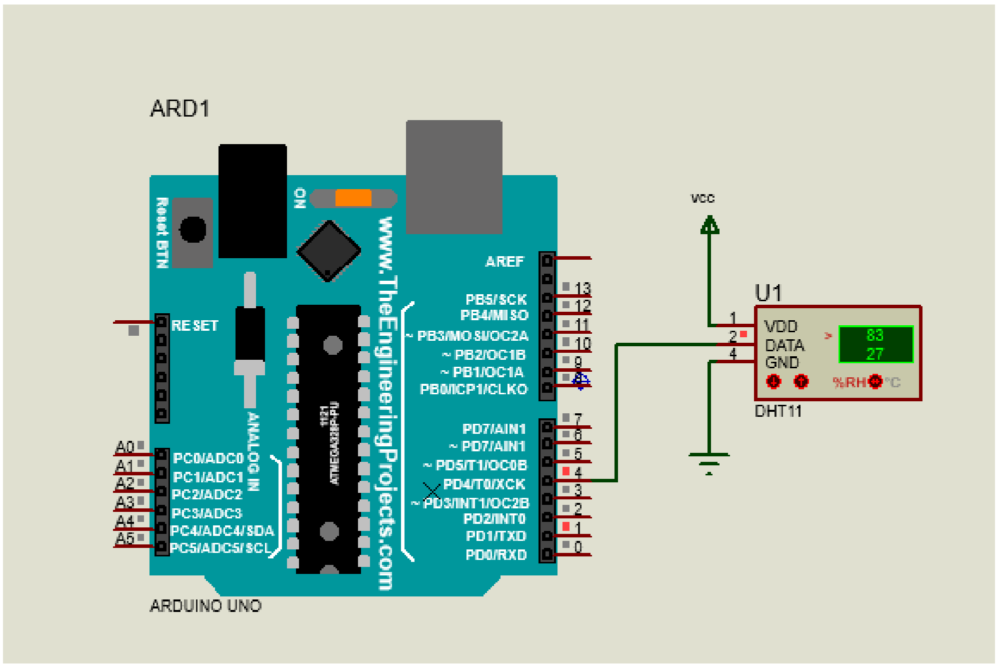
I2C LCD

CODE:

```
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27,16,2);
void setup() {
  lcd.init();
  lcd.clear();
  lcd.backlight();      // Make sure backlight is on

  // Print a message on both lines of the LCD.
  lcd.setCursor(2,0);   //Set cursor to character 2 on line 0
  lcd.print("Hello world!");

  lcd.setCursor(2,1);   //Move cursor to character 2 on line 1
  lcd.print("LCD Tutorial");
}
void loop() {
}
```

6. DHT Sensor

DHT (Digital Humidity and Temperature) sensors, like the DHT11 and DHT22, are commonly used for measuring temperature and humidity levels. They offer a simple and cost-effective way to gather environmental data. These sensors use a capacitive humidity sensor and a thermistor to measure the surrounding air, and then output a digital signal.



DHT11 test

CODE:

```
#include <DHT.h>
#define DHTPIN 4
#define DHTTYPE DHT11
```

```
DHT dht(DHTPIN, DHTTYPE);
void setup() {
  Serial.begin(9600);
  dht.begin();
}
void loop() {
  float hum = dht.readHumidity();
  float temp = dht.readTemperature();
  if (isnan(hum) || isnan(temp)) {
    Serial.println("Failed to read from DHT sensor!");
    delay(10000);
    return;
  }
  Serial.print("Temperature: ");
  Serial.print(temp);
  Serial.println(" °C");
  Serial.print("Humidity: ");
  Serial.print(hum);
  Serial.println(" %");
  delay(10000);
}
```

7. DHT22 with LCD

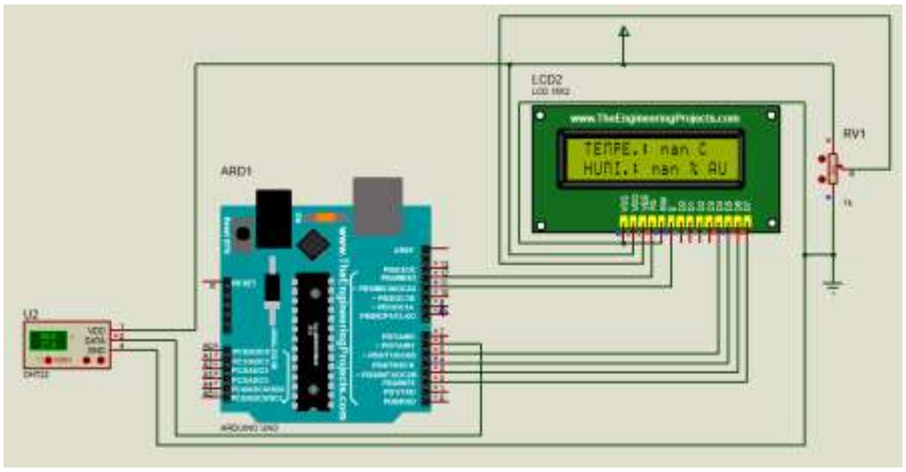


Figure: Schematic diagram

CODE:

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <DHT.h>
#define DHTPIN 6
#define DHTTYPE DHT22

DHT dht(DHTPIN, DHTTYPE);
LiquidCrystal_I2C lcd(0x27, 16, 2); // Adjust I2C address if needed (commonly 0x27 or 0x3F)

void setup() {
  lcd.init();            // Initialize the LCD
  lcd.backlight();        // Turn on the backlight
  dht.begin();           // Start the DHT sensor
}
void loop() {
  lcd.clear();
  float hum = dht.readHumidity();
  float temp = dht.readTemperature();
  if (isnan(hum) || isnan(temp)) {
    lcd.setCursor(0, 0);
    lcd.print("Sensor error!");
    lcd.setCursor(0, 1);
    lcd.print("Reconfiguring...");
    delay(10000);
    return;
  }
  lcd.setCursor(0, 0);
  lcd.print("TEMPE.: ");
  lcd.print(temp);
  lcd.print(" C");
  lcd.setCursor(0, 1);
  lcd.print("HUMI.: ");
  lcd.print(hum);
  lcd.print(" %");
  delay(10000);
}
```

8. **ULTRASONIC SENSOR** (HC-SR04)

The ultrasonic sensor is a device that can measure distances using sound waves . It works in a similar way as bats and dolphins by emitting sound waves and listening them bound back.

The sensor consists of two primary components: a transmitter and a receiver . The transmitter is responsible for emitting a high-frequency sound. In essence, ultrasonic refers to frequencies beyond the range of the human hearing - so something higher than 20kHz.

When the sound wave hits an object, it bounces back like echo. This returning wave is detected by the receiver. The sensor will use the micro-controller (Arduino) internal clock to find out how much it took for the sound to bounce back. This small clock turns on when a high-frequency wave is emitted and turns off when its echo is detected

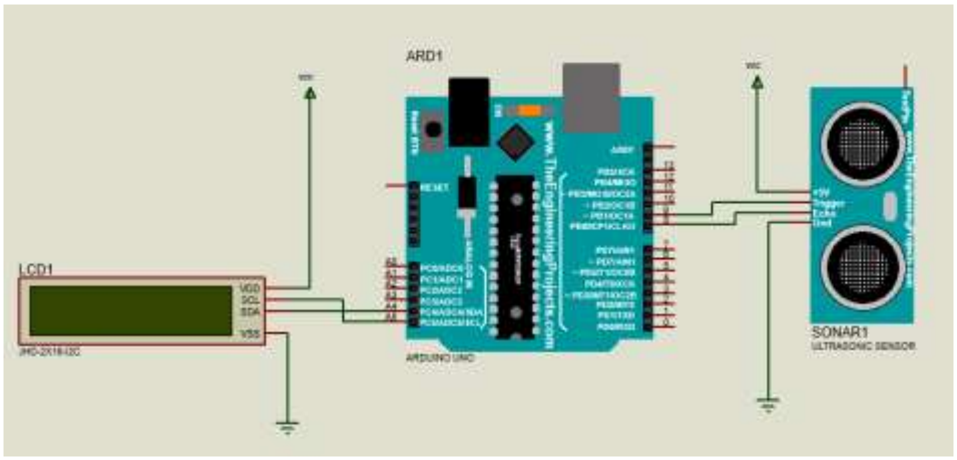


Figure: Distance measurement using an ultrasonic sensor and displays it in centimeters on an I2C LCD screen

```
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27, 16, 2); // I2C address 0x27, 16 column and 2 rows
int trigPin = 9;    // TRIG pin
int echoPin = 8;    // ECHO pin
float duration_us, distance_cm;
void setup() {
  lcd.init();               // initialize the lcd
  lcd.backlight();          // open the backlight
  pinMode(trigPin, OUTPUT); // config trigger pin to output mode
  pinMode(echoPin, INPUT);  // config echo pin to input mode
}
void loop() {
  // generate 10-microsecond pulse to TRIG pin
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  // measure duration of pulse from ECHO pin
  duration_us = pulseIn(echoPin, HIGH);
  // calculate the distance
  distance_cm = 0.017 * duration_us;
  lcd.clear();
  lcd.setCursor(0, 0); // start to print at the first row
  lcd.print("Distance: ");
  lcd.print(distance_cm);
  delay(500);
}
```

Therefore, this code measures distance using an ultrasonic sensor (HC-SR04) and displays the result on a 16x2 I2C LCD. The **TRIG pin** (connected to pin 9) is used to send a short 10-microsecond pulse that triggers the sensor to emit an ultrasonic sound wave. The **ECHO pin** (connected to pin 8) then listens for the echo of that sound wave bouncing back from an object. The time it takes for the echo to return is measured using pulseIn, and this duration is converted into a distance in centimeters using the formula distance = 0.017 * duration_us. The calculated distance is then displayed on the LCD every half a second.

9. RELAY SWITCH

Question: What are the common and difference between controlling led and light bulb?

Answer:

- Common: Just like controlling led, Arduino's output is used to turn them ON/OFF.
- Difference: For led (≤5V) from Arduino can be used directly to power it. Therefore, LED can be connected directly to Arduino's pin. For light bulb external power source with (high power and/or high current) should be used, which can burn Arduino. Therefore, bulb can not be connected directly with Arduino's pins. Therefore, a relay is used between Arduino and light bulb tu protect Arduino from high voltage and current.

About Relay:

Relay is a programmable electrical switch which can be controlled by Arduino or any microcontroller. It is used to programmatically control ON/OFF devices which use high voltage and/or high current.

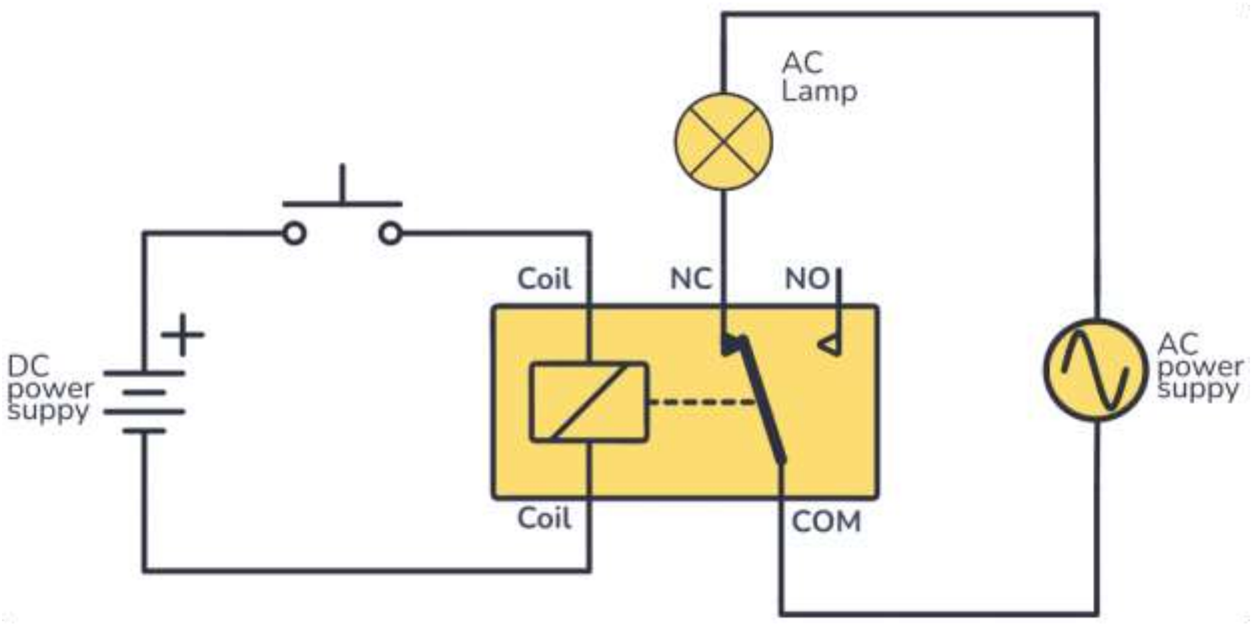


Figure: Relay Switch

Mains Voltage Connections

The high-voltage side has two connectors, each with three sockets: common (COM), normally closed (NC), and normally open (NO):

- **COM**: common pin
- **NC (Normally Closed):** the normally closed configuration is used when you want the relay to be closed by default, meaning the current is flowing unless you send a signal from the Arduino to the relay module to open the circuit and stop the current.
- **NO (Normally Open):** the normally open configuration works the other way around: the relay is always open, so the circuit is broken unless you send a signal from the Arduino to close the circuit.

Warning: For these projects which involve connection to the mains voltage, you should know what your doing otherwise you may shock yourself. If your not 100% sure what your doing don't touch anything for safety.

Project: Servo and Relay Control with Button Input

This project provides basic Arduino-based automation using a push-button to control both a servo motor and a relay. When the button is pressed, the system activates the relay, simulating the control of a high-power device and prints "pressed" to the serial monitor. Upon releasing the button, the relay is turned off, a "released" message is printed, and the servo performs a full sweep from 0 to 180 degrees and back to 0, demonstrating mechanical motion control. The project teaches essential concepts such as digital input/output, review on servo control using the Servo library, relay operation, and serial communication.

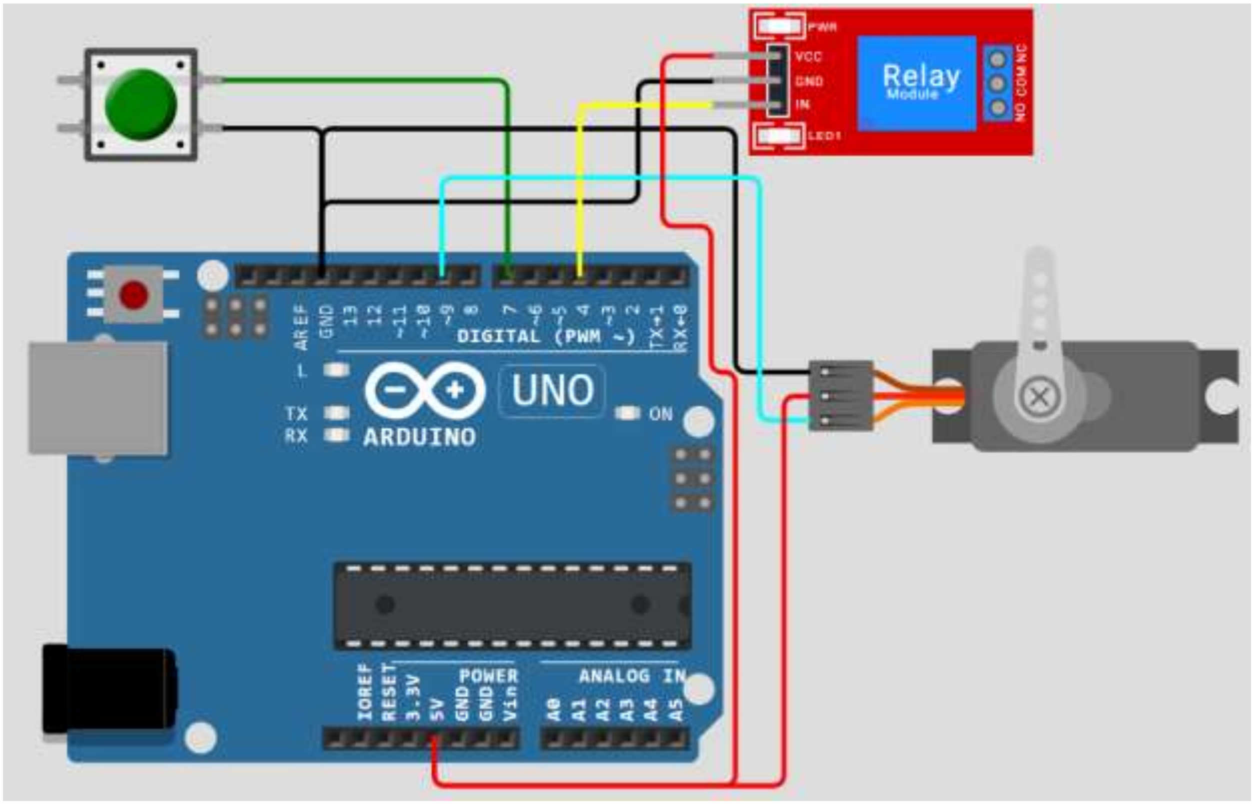


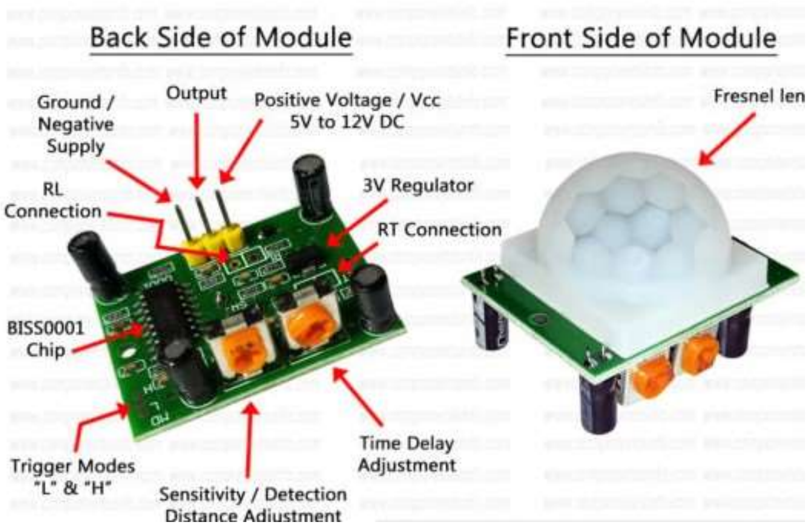
Figure: Circuit diagram

CODE:

```
#include <Servo.h>
int buttonpin= 7;
int relaypin = 4;
int last_state= HIGH;

Servo myservo;

long pos = 0;

void setup(){
  Serial.begin(115200);
  pinMode(buttonpin, INPUT_PULLUP);
  pinMode(relaypin, OUTPUT);
  myservo.attach(9);
}

void loop() {
  int value=digitalRead(buttonpin);
  if(last_state!=value){
    last_state=value;
    if(value==HIGH){
      digitalWrite(relaypin,LOW);
      Serial.println("released");
      for(pos=0;pos<=180;pos+=1){
        myservo.write(pos);
        delay(15);
      }
      for(pos=180;pos>=0;pos-=1){
        myservo.write(pos);
        delay(15);
      }
    }
    else{
        digitalWrite(relaypin,HIGH);
        Serial.println("pressed");
    }
  }
}
```

10. HS-SR501 (PIR) MOTION SENSOR

A PIR (Passive Infrared) motion sensor is an electronic sensor used in motion detectors such as automatic triggered lighting devices and protection systems that measure devices emitting infrared light in their field of view.



Working Mechanism

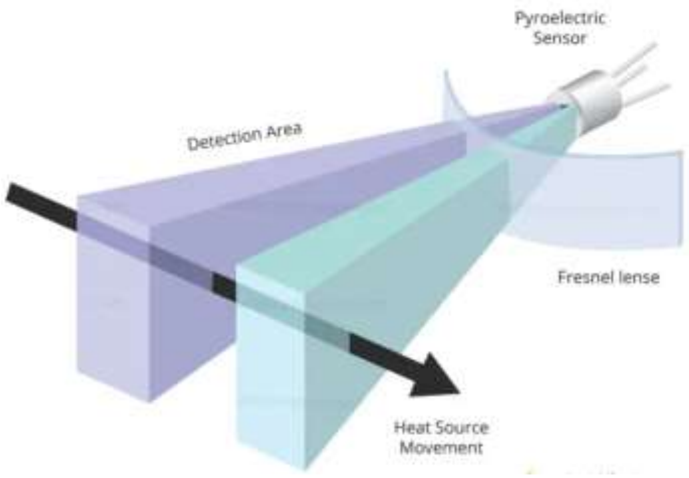
Everything around us including our bodies gives off heat energy as infrared radiation. This happens as long as objects are warmer than absolute zero (0 Kelvin or -273.15°C). The warmer an object is, the more infrared radiation it releases. We can't see this radiation with our eyes because it's in the infrared part of the light spectrum, beyond what humans can detect visually.

A basic infrared receiver can detect the presence of infrared radiation, but to detect movement, we need something that can notice changes in infrared levels. This is exactly what PIR (Passive Infrared) sensors do. They're specially designed to detect changes in infrared radiation caused when warm objects, like people, move within their detection range.

Parts of PIR sensor:

I. **Pyroelectric sensor**: This is the heart of the device, appearing as a round metal piece with a rectangular crystal in the center.

The pyroelectric sensor has a window with two rectangular slots made from a material (usually silicon with a special coating) that allows infrared radiation to pass through. Behind this window are two separate infrared-detecting electrodes, one produces a positive output, while the other produces a negative output.

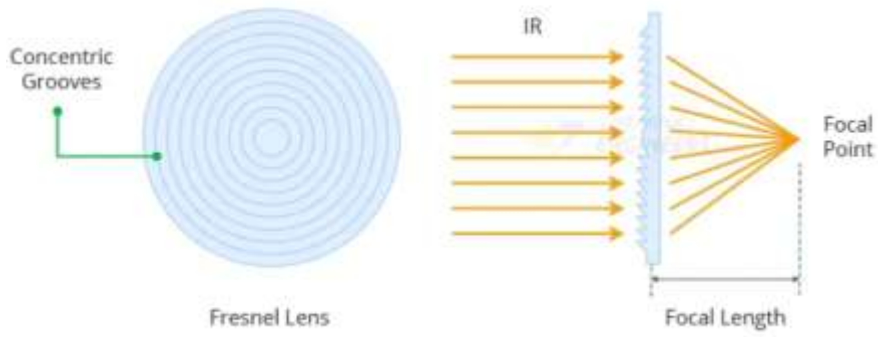


When nothing is moving near the sensor, both parts detect the same amount of infrared radiation, so they cancel each other out and no signal is produced.

However, **when a warm object like a person moves** within the sensor's detection area, one part detects a change in infrared radiation before the other. This creates a difference in signals between the two parts, which the sensor recognizes as movement

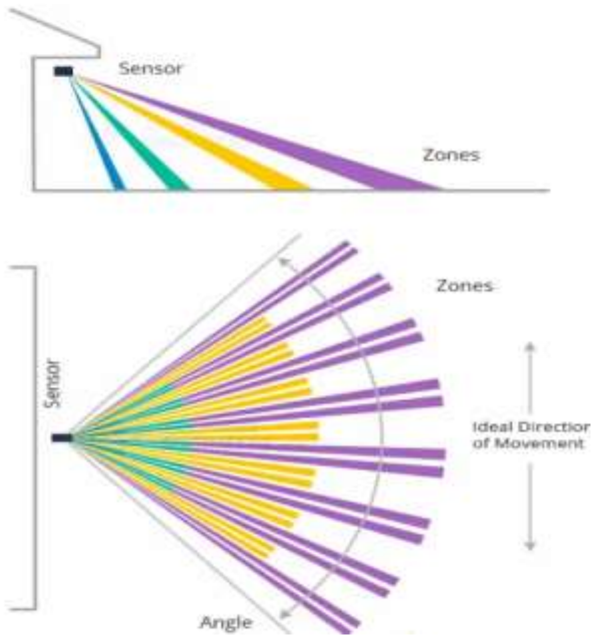
II. **Fresnel lens:** A special lens that collects and focuses infrared signals onto the pyroelectric sensor.

A Fresnel lens is made of plastic with a series of concentric circular grooves carved into it. Each groove acts as a refracting surface that focuses parallel light rays onto a focal point, similar to how a regular optical lens works, but in a much flatter design.



To increase the detection range and field of view, the lens is divided into multiple sections, with each section acting as its own individual Fresnel lens.

These different sections create various detection zones that overlap with one another. This is why the centers of the lenses appear to be pointing in different directions each one directs infrared radiation from a different area to the PIR sensing element, allowing the sensor to cover a much wider area.



That’s why PIR sensors can detect movement across a room even though the actual sensing element inside is quite small.

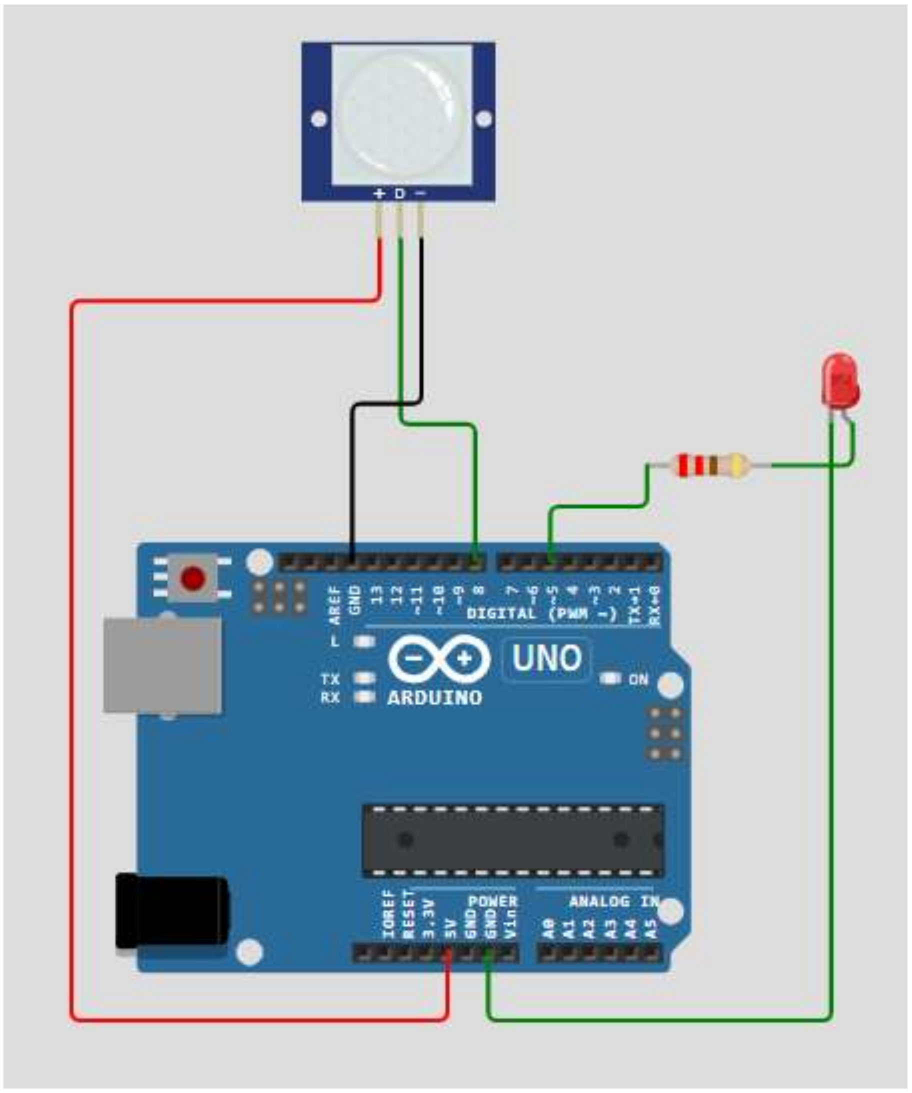


Figure: Led control with PIR Sensor

CODE:

```
int ledPin = 5;                // choose the pin for the LED
int inputPin = 8;               // choose the input pin (for PIR sensor)
int pirState = LOW;             // we start, assuming no motion detected
int val = 0;                    // variable for reading the pin status

void setup() {
  pinMode(ledPin, OUTPUT);      // declare LED as output
  pinMode(inputPin, INPUT);     // declare sensor as input
  Serial.begin(9600);
}

void loop(){
  val = digitalRead(inputPin);  // read input value

  if (val == HIGH)        // check if the input is HIGH
  {
    digitalWrite(ledPin, HIGH);  // turn LED ON

    if (pirState == LOW)
       {
      Serial.println("Motion detected!");        // print on output change
      pirState = HIGH;
    }
  }
  else
  {
    digitalWrite(ledPin, LOW); // turn LED OFF

    if (pirState == HIGH)
       {
      Serial.println("Motion ended!");        // print on output change
      pirState = LOW;
    }
  }
}
```

11. DS3231 Real-Time Clock (RTC)

The DS3231 can keep track of seconds, minutes, hours, days, dates, months, and years. It's smart enough to know which months have fewer than 31 days and adjusts automatically. It also handles leap years correctly, though only up to the year 2100.

This chip can show time in either 12-hour format (with AM/PM indicators) or 24-hour format. The chip also includes two programmable alarms that can go off at specific times during the day.

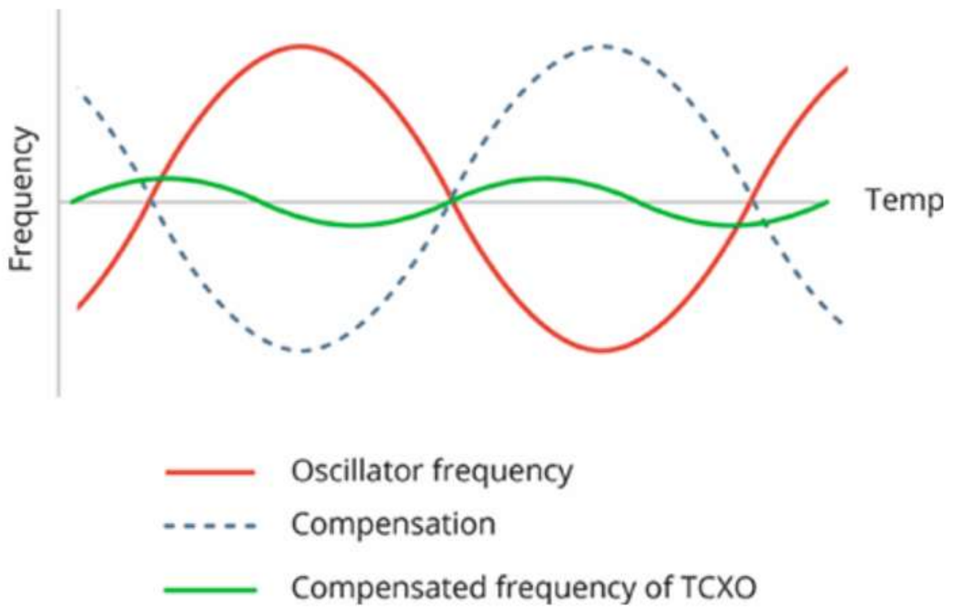
The chip has a special pin called INT/SQW that can do two different things. It can either send out an interrupt signal when an alarm goes off, or it can produce a steady square wave signal at one of four frequencies: 1Hz, 4kHz, 8kHz, or 32kHz.

There's another pin called the 32K pin that outputs an extremely stable and accurate timing signal. This signal stays precise even when the temperature changes, which makes it useful for applications that need exact timing or for keeping other electronic circuits synchronized.

Temperature Compensated Crystal Oscillator(TCXO)

Most other RTC modules, like the DS1307, need an external 32kHz crystal oscillator to keep time. The problem with these crystals is that their frequency can change slightly with temperature. This tiny change might not seem like much, but over days and weeks, the clock could end up being minutes off.

The DS3231 solves this problem by using a built-in 32kHz temperature-compensated crystal oscillator (TCXO), which is highly resistant to changes in temperature. This helps the clock stay accurate over time.



The TCXO consists of three key parts working together: a temperature sensor, a 32kHz crystal oscillator, and control logic. If the temperature causes the crystal's frequency to change, the chip automatically adjusts the clock by adding or removing ticks to keep it accurate.

Battery Backup

The DS3231 chip has a battery backup feature that keeps the clock running even when the main power is disconnected. On the back of the module, there is a battery holder designed for a 20mm 3V lithium coin cell battery. Inside the chip, there's a built-in smart power-sensing circuit that constantly checks the main power. If this circuit detects that the main power has been lost, it automatically switches to the backup battery.

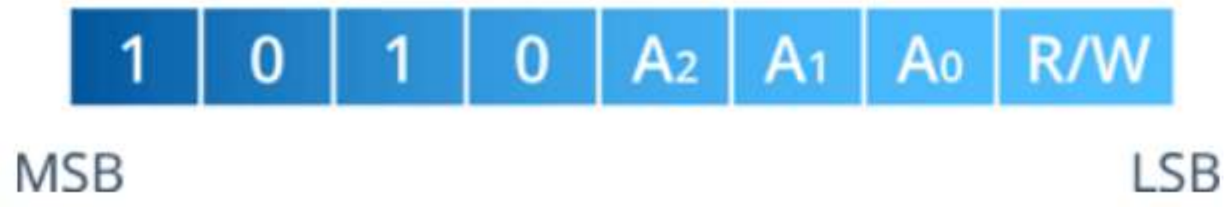
WARNING: These modules usually come with a 200Ω resistor soldered next to a 1N4148 diode (which you can see in the image). Together, these components form a simple charging circuit designed for rechargeable LIR2032 batteries. However, some DS3231 modules come with non-rechargeable CR2032 batteries instead. If the module has a non-rechargeable battery, you must remove the resistor! Attempting to charge a non-rechargeable battery is dangerous. It can damage the battery and may even cause it to leak or explode!

Onboard 24C32 EEPROM

In addition to the main DS3231 chip, the module also includes a 24C32 EEPROM chip. EEPROM stands for Electrically Erasable Programmable Read-Only Memory. While this chip isn’t used for timekeeping, it’s handy for storing data, like logs or settings, that can be kept even when the power is turned off.

The 24C32 EEPROM can store 32 kilobits of data, and it’s designed to handle up to 1,000,000 write cycles, meaning that data can be saved many times before it wears out.

In 24C32, there are three address bits located at the end of the 7-bit I2C address register (right before the Read/Write bit).



Since there are three address inputs that can be either HIGH or LOW, you can create eight different address combinations ($2^3 = 8$).

All three address inputs are pulled HIGH by default using onboard pullup resistors. This gives the EEPROM a default I2C address of 0x57.

When you short a solder jumper, you pull that address input LOW. If you were to short all three jumpers, the address would change to 0x50. So the range of possible addresses goes from 0x50 to 0x57.

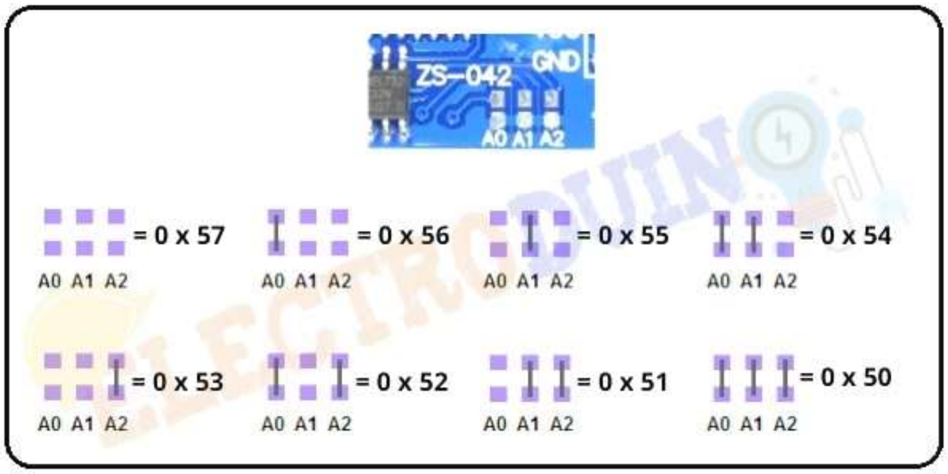


Figure: Different I2C addresses can be set according to the above table

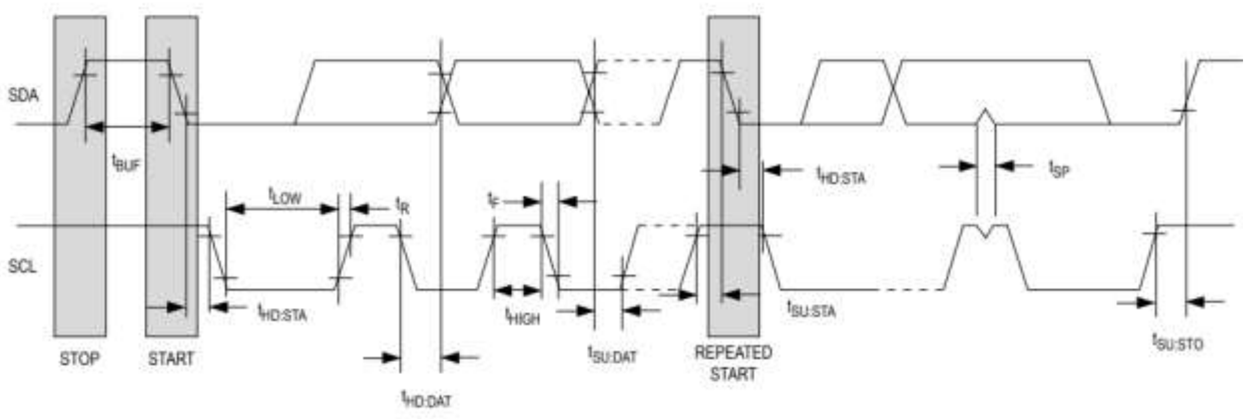


Figure: Data transfer on I2C Serial Bus

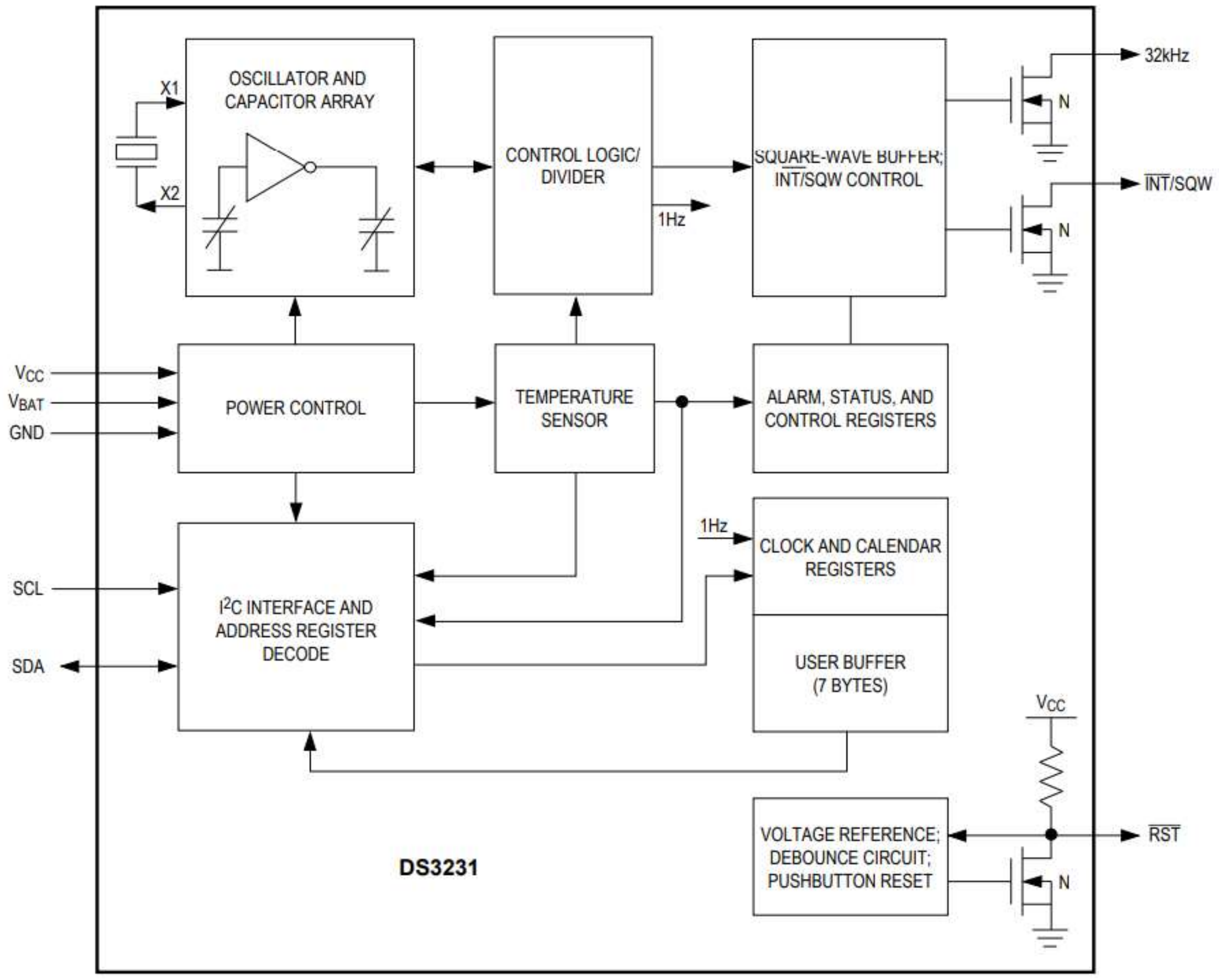

X1
X2
OSCILLATOR AND CAPACITOR ARRAY
CONTROL LOGIC/ DIVIDER
1Hz
SQUARE-WAVE BUFFER; INT/SQW CONTROL
32kHz
N
INT/SQW
N
VCC
VBAT
GND
POWER CONTROL
TEMPERATURE SENSOR
ALARM, STATUS, AND CONTROL REGISTERS
1Hz
CLOCK AND CALENDAR REGISTERS
USER BUFFER (7 BYTES)
SCL
SDA
I2C INTERFACE AND ADDRESS REGISTER DECODE
VCC
RST
VOLTAGE REFERENCE; DEBOUNCE CIRCUIT; PUSHBUTTON RESET
N
DS3231

Figure: Block Diagram

Operating voltage	2.3 to 5.5V (3.3 or 5V typical)
Current Consumption	< 300µA (typ.)
Accuracy	± 2ppm
Battery	CR2032 (3V Coin)

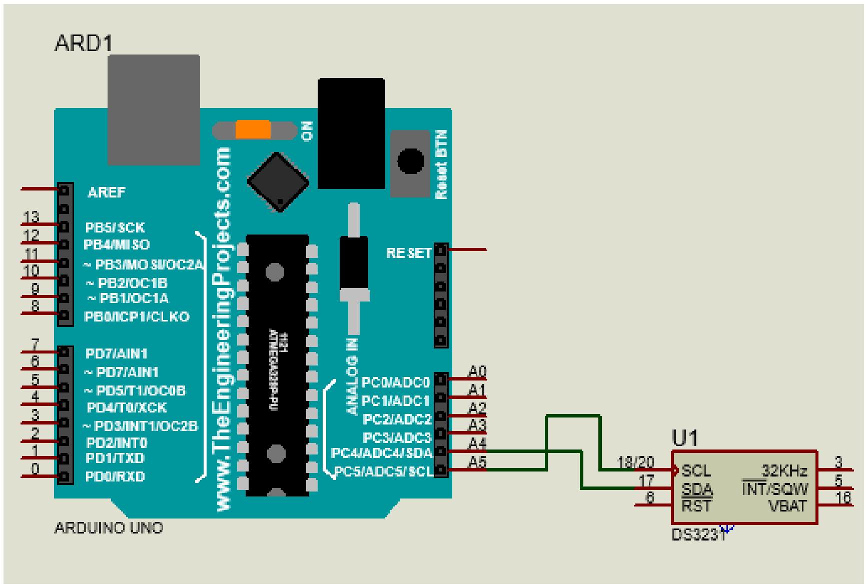


Figure: Schematic Diagram

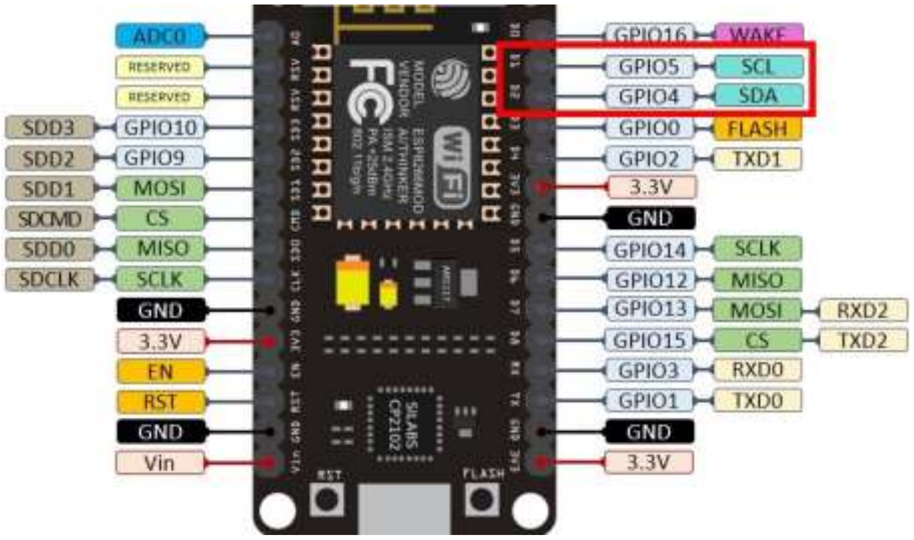
CODE (set and read the date, time, and temperature)

```
#include "Arduino.h"
#include "uRTCLib.h"
uRTCLib rtc(0x68);
char daysOfTheWeek[7][12] = { "Sunday", "Monday", "Tuesday", "Wednesday", "Thursday",
"Friday", "Saturday" };
void setup() {
  Serial.begin(9600);
  URTCLIB_WIRE.begin();
  rtc.set(0, 56, 12, 2, 14, 4, 25);
  // rtc.set(second, minute, hour, dayOfWeek, dayOfMonth, month, year)
  // set day of week (1=Sunday, 7=Saturday)
}
void loop() {
  rtc.refresh();
  Serial.print("Current Date & Time: ");
  Serial.print(rtc.year());
  Serial.print('/');
  Serial.print(rtc.month());
  Serial.print('/');
  Serial.print(rtc.day());
  Serial.print(" (");
  Serial.print(daysOfTheWeek[rtc.dayOfWeek() - 1]);
  Serial.print(") ");
  Serial.print(rtc.hour());
  Serial.print(':');
  Serial.print(rtc.minute());
  Serial.print(':');
  Serial.println(rtc.second());
  Serial.print("Temperature: ");
  Serial.print(rtc.temp() / 100);
  Serial.println("°C");
  Serial.println();
  delay(1000);
}
```

ESP8266 NodeMCU

12. Introduction to ESP8266 NodeMCU

The ESP8266 NodeMCU is a Wi-Fi System on a Chip (SoC). It can be used as a standalone device, or as a UART to Wi-Fi adaptor to allow other microcontrollers to connect to a Wi-Fi network. For example, you can connect an ESP8266 to an Arduino to add Wi-Fi capabilities to your Arduino board. The most practical application is using it as a standalone device.



ESP8266 PinOUT

INSTALLING ESP8266 CORE

I. In your Arduino IDE 2, go to File > Preferences

II. Copy and paste the following line to the Additional Boards Manager URLs field (https://arduino.esp8266.com/stable/package_esp8266com_index.json)

III. Open the Boards Manager. You can go to Tools > Board > Boards Manager**…** or you can simply click the Boards Manager icon in the left-side corner.

IV. Search for ESP8266 and press the install button for esp8266 by ESP8266 Community. After this, restart your Arduino IDE.
Then, go to **Tools** > **Board** and check that you have ESP8266 boards available.

Testing Installation and uploading code to ESP8266

To test the ESP8266 add-on installation, we'll upload a simple code that blinks the on-board LED (GPIO 2).

```
#include <Arduino.h>
#define LED 2
void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200);
  pinMode(LED, OUTPUT);
}
void loop() {
  // put your main code here, to run repeatedly:
  digitalWrite(LED, HIGH);
  Serial.println("LED is on");
  delay(1000);
  digitalWrite(LED, LOW);
  Serial.println("LED is off");
  delay(1000);
}
```

Uploading the sketch

I. Connect your ESP8266 development board to your computer using a USB cable

II. Select your board before uploading the code. On the top drop-down menu, click on "**Select other board and port…**"

III. Select the ESP8266 board model you're using, and the COM port. In our example, we're using the NodeMCU 1.0 board. Click **OK** when you're done.

NOTE: If you don't see the COM port or it it's grayed out, you probably need to install the USB-to-UART drivers (CP2101 or CH340 drivers) **on your computer from "CP210x Windows Drivers"**

- Now, you just need to click on the **Upload** button
- After a few seconds, the upload should be complete
- The ESP8266 on-board LED should be blinking every second

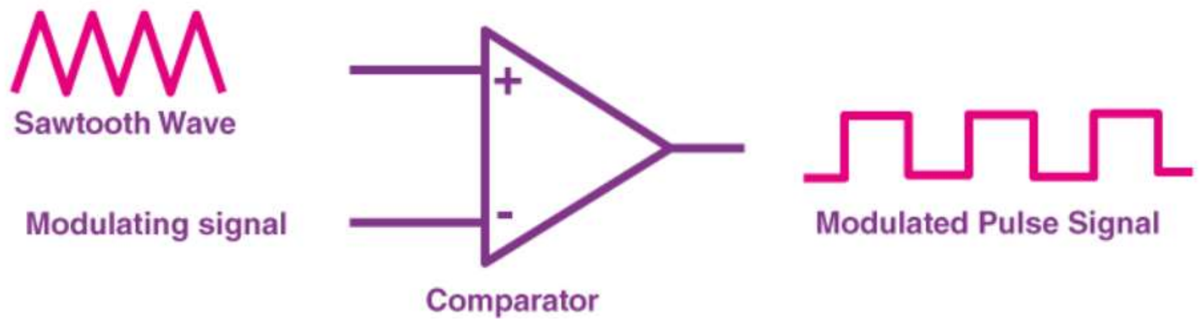
IV. You can click on the Serial Monitor icon to open the Serial Monitor tab, you've installed the ESP8266 Boards successfully in Arduino IDE and now you know how to upload code to the board.

13. PWM

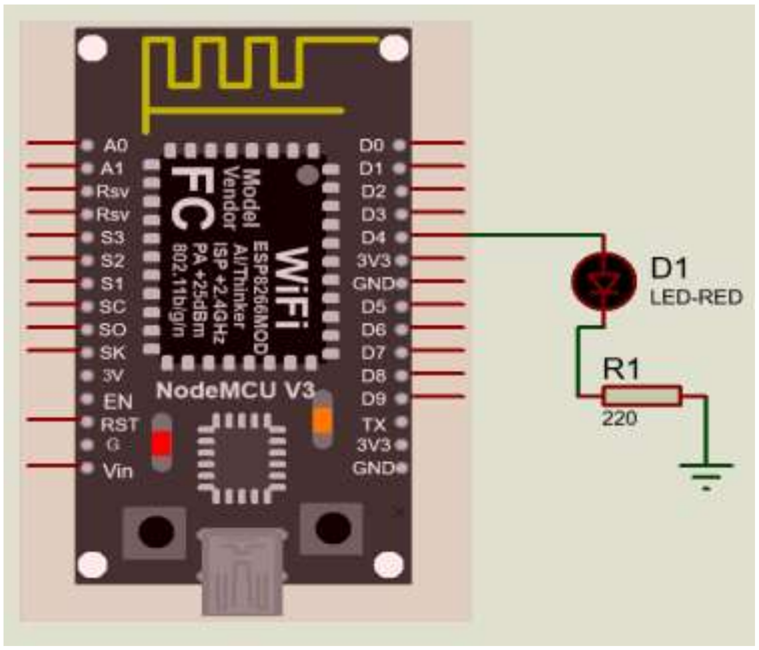
Pulse width modulation (PWM) is a commonly used control technique that generates analog signals from digital devices such as microcontrollers. The signal thus produced will have a train of pulses, and these pulses will be in the form of square waves. Thus, at any given time, the wave will either be high or low.

How PWM is generated

A pulse width modulating signal is generated using a comparator. The modulating signal forms one part of the input to the comparator, while the non-sinusoidal wave or sawtooth wave forms the other part of the input. The comparator compares two signals and generates a PWM signal as its output waveform.



If the sawtooth signal is more than the modulating signal, then the output signal is in a "High" state. The value of the magnitude determines the comparator output which defines the width of the pulse generated at the output.



Circuit Diagram

CODE:

```
// the number of the LED pin
const int ledPin = D4;
void setup() {
  // set the LED as an output
  pinMode(ledPin, OUTPUT);
}
void loop(){
  // increase the LED brightness
  for(int dutyCycle = 0; dutyCycle <= 255; dutyCycle++){
    // changing the LED brightness with PWM
    analogWrite(ledPin, dutyCycle);
    delay(15);
  }
  // decrease the LED brightness
  for(int dutyCycle = 255; dutyCycle >= 0; dutyCycle--){
    // changing the LED brightness with PWM
    analogWrite(ledPin, dutyCycle);
    delay(15);
  }
}
```

14. ESP8266 AS WIFI STATION

CODE:

```
#include<ESP8266WiFi.h>
void ConnectToWifi(void);
void setup() {
  Serial.begin(115200);
  ConnectToWifi();
}
void loop() {
}
void ConnectToWifi (){
WiFi.mode(WIFI_STA); //nodemcu as station
WiFi.begin("********", "********"); //(ssid) and password
Serial.print("connecting to wifi");
while(WiFi.status() !=WL_CONNECTED){
  Serial.print('.');
  delay(200);
}
Serial.print("IP Address:");
Serial.println(WiFi.localIP());
}
```

15. DHT11 WITH ESP8266

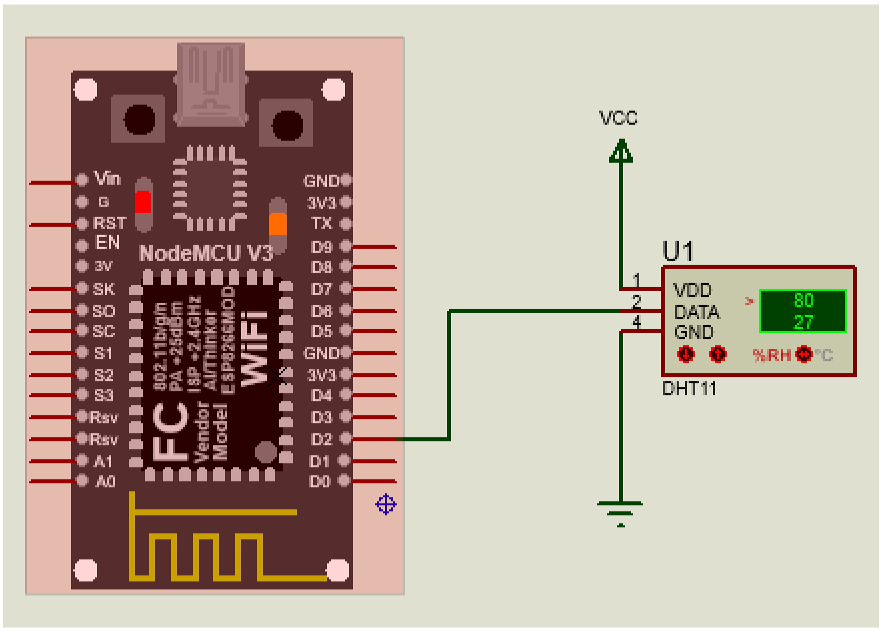


Figure: Schematic diagram

CODE:

```
#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>
#include <DHT.h>
const char* ssid = "xxxx";
const char* password = "xxxxx";
#define DHTPIN D2
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);
ESP8266WebServer server(80);
void setup(){
  Serial.begin(115200);
  Serial.println("Starting setup...");
  WiFi.begin(ssid, password);
  Serial.println("Connecting to WiFi...");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println();
  Serial.println("Connected to WiFi. IP address: " + WiFi.localIP().toString());
  server.on("/", HTTP_GET, handleRoot);
  server.begin();
  Serial.println("Server started. Ready to serve.");
}
void loop(){
  server.handleClient();
  delay(2000);
}
void handleRoot() {
  float temp = dht.readTemperature();
  float hum = dht.readHumidity();
  String html = "<!DOCTYPE HTML>\r\n";
  html += "<html>\r\n";
  html += "<head><meta name=\"viewport\" content=\"width=device-width, initial-
scale=1\"></head>\r\n";
  html += "<body>\r\n";
  html += "<h1>DHT11 Sensor Data</h1>\r\n";
  html += "<p>Temperature: " + String(temp) + " °C</p>\r\n";
  html += "<p>Humidity: " + String(hum) + " %</p>\r\n";
  html += "</body>\r\n";
  html += "</html>\r\n";
  server.send(200, "text/html", html);
}
```