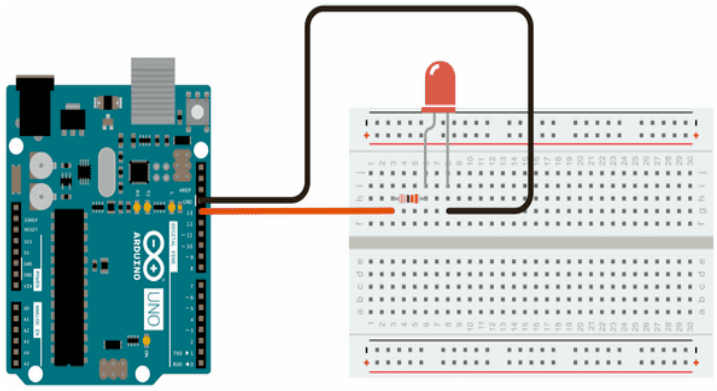
Ex no: 1 **LED ON/OFF Pattern**

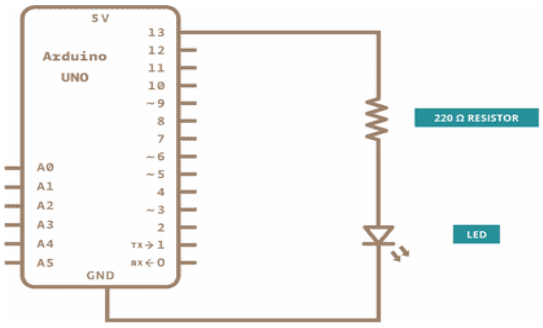
Aim: In this project with an Arduino to see physical output: it blinks the on-board LED.

Requirement:

- Arduino Board optional
- LED
- 220 ohm resistor

Circuit diagram:





Procedure:

This example uses the built-in LED that most Arduino boards have. This LED is connected to a digital pin and its number may vary from board type to board type. To make your life easier, we have a constant that is specified in every board descriptor file. This constant is *LED_BUILTIN* and allows you to control the built-in LED easily. Here is the correspondence between the constant and the digital pin.

• D13 - 101					
• D13 - Due					
• D1 - Gemma					
• D13 - Intel Edison					
• D13 - Intel Galileo Gen2					
• D13 - Leonardo and Micro					
• D13 - LilyPad					
• D13 - LilyPad USB					
• D13 - MEGA2560					
• D13 - Mini					
• D6 - MKR1000					
• D13 - Nano					
• D13 - Pro					
• D13 - Pro Mini					
• D13 - UNO					
• D13 - Yún					
• D13 - Zero					

If you want to lit an external LED with this sketch, you need to build this circuit, where you connect one end of the resistor to the digital pin correspondent to the *LED_BUILTIN* constant. Connect the long leg of the LED (the positive leg, called the anode) to the other end of the resistor. Connect the short leg of the LED (the negative leg, called the cathode) to the GND. In the diagram below we show an UNO board that has D13 as the LED_BUILTIN value.

The value of the resistor in series with the LED may be of a different value than 220 ohm; the LED will lit up also with values up to 1K ohm.

Program:

After you build the circuit plug your Arduino board into your computer, start the Arduino Software (IDE) and enter the code below. You may also load it from the menu File/Examples/01.Basics/Blink . The first thing you do is to initialize LED_BUILTIN pin as an output pin with the line

```
pinMode(LED_BUILTIN, OUTPUT);
```

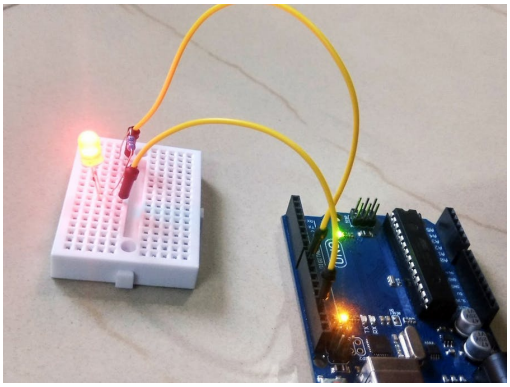
In the main loop, you turn the LED on with the line:

digitalWrite(LED_BUILTIN, HIGH);
This supplies 5 volts to the LED anode. That creates a voltage difference across the pins of the LED, and lights it up. Then you turn it off with the line:
digitalWrite(LED_BUILTIN, LOW);
That takes the LED_BUILTIN pin back to 0 volts, and turns the LED off. In between the on and the off, you want enough time for a person to see the change, so the delay() commands tell the board to do nothing for 1000 milliseconds, or one second. When you use the delay() command, nothing else happens for that amount of time. Once you've understood the basic examples, check out the BlinkWithoutDelay example to learn how to create a delay while doing other things.
Once you've understood this example, check out the DigitalReadSerial example to learn how read a switch connected to the board.

```
/*
  Blink
Turns an LED on for one second, then off for one second, repeatedly.
Most Arduinos have an on-board LED you can control. On the UNO, MEGA and ZERO
  it is attached to digital pin 13, on MKR1000 on pin 6. LED_BUILTIN is set to  the correct LED
pin independent of which board is used.
 If you want to know what pin the on-board LED is connected to on your Arduino
  model,
 This example code is in the public domain.
 https://www.arduino.cc/en/Tutorial/BuiltInExamples/Blink
*/
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}
// the loop function runs over and over again forever
void loop()
 {
digitalWrite(LED_BUILTIN, HIGH);  // turn the LED on (HIGH is the voltage level)
  delay(1000);                      // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);                      // wait for a second
```

}

Output:



Result:

Thus the circuit were created and executed successfully.

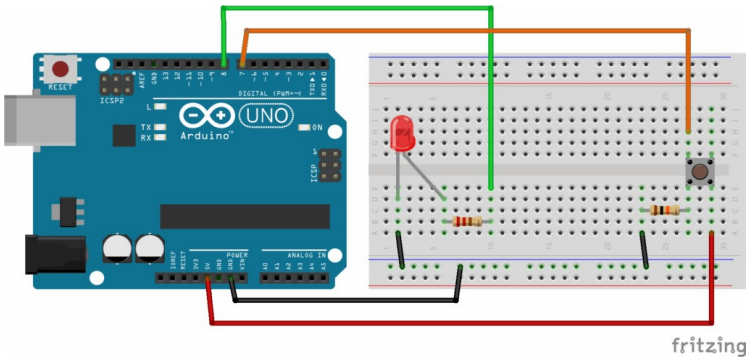
Ex no:2 **Arduino – Turn LED ON and OFF with Button**

Aim: In this Arduino project how to turn an LED on and off with a push button.

Requirements:

- Arduino board (any board, if you don't have Uno you can easily adapt by finding corresponding pins).
- Breadboard.
- LED – any color.
- Push button.
- 220 Ohm resistor for the LED. If you don't have this specific value, any resistor from 330 to 1k Ohm will do.
- 10k Ohm resistor for the push button. If you don't have, you can go until 20k-50k Ohm.
- A bunch of male to male wires (including if possible black, red, and other colors).

Circuit diagram:



Procedure:

- First, make sure to power off your Arduino – remove any USB cable.
- Plug a black wire between the blue line of the breadboard and a ground (GND) pin on the Arduino board.
- Plug the LED. You can notice that the LED has a leg shorter than the other. Plug this shorter leg to the ground (blue line here) of the circuit.
- Connect the longer leg of the LED to a digital pin (here pin no 8, you can change it). Add a 220 Ohm resistor in between to limit the current going through the LED.
- Add the push button to the breadboard, like in the picture.

- Connect one leg of the button to the ground, and put a 10k Ohm resistor in between. This resistor will act as a "pull down" resistor, which means that the default button's state will be LOW.
- Add a red wire between another leg of the button and VCC (5V).
- Finally, connect a leg of the button (same side as the pull down resistor) to a digital pin (here 7).

Program:

Turn on the LED when button is pressed, turn it off otherwise

```
#define LED_PIN 8
#define BUTTON_PIN 7
void setup()
{
pinMode(LED_PIN, OUTPUT);
pinMode(BUTTON_PIN, INPUT);
}
void loop() {
if (digitalRead(BUTTON_PIN) == HIGH) {
digitalWrite(LED_PIN, HIGH);
}
else {
digitalWrite(LED_PIN, LOW);
}
}
```

Toggle LED's state with the push button – first iteration

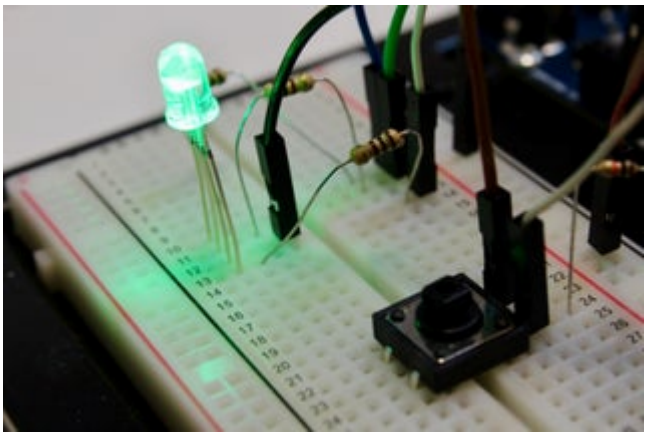
```
#define LED_PIN 8
#define BUTTON_PIN 7
byte lastButtonState = LOW;
byte ledState = LOW;
void setup() {
pinMode(LED_PIN, OUTPUT);
pinMode(BUTTON_PIN, INPUT);
}
void loop() {
byte buttonState = digitalRead(BUTTON_PIN);
if (buttonState != lastButtonState) {
```

```
lastButtonState = buttonState;
if (buttonState == LOW) {
ledState = (ledState == HIGH) ? LOW: HIGH;
digitalWrite(LED_PIN, ledState);
}
}
}
```

Turn LED on and off with button – using debounce

```
#define LED_PIN 8
#define BUTTON_PIN 7
byte lastButtonState = LOW;
byte ledState = LOW;
unsigned long debounceDuration = 50; // millis
unsigned long lastTimeButtonStateChanged = 0;
void setup() {
  pinMode(LED_PIN, OUTPUT);
  pinMode(BUTTON_PIN, INPUT);
}
void loop() {
  if (millis() - lastTimeButtonStateChanged > debounceDuration) {
    byte buttonState = digitalRead(BUTTON_PIN);
    if (buttonState != lastButtonState) {
      lastTimeButtonStateChanged = millis();
      lastButtonState = buttonState;
      if (buttonState == LOW) {
        ledState = (ledState == HIGH) ? LOW: HIGH;
        digitalWrite(LED_PIN, ledState)
     }
    }
  }
}
```

Output:



Result:

Thus the circuit were created and executed successfully.

Ex no:3 **Arduino – LED, Push Button, and Potentiometer**

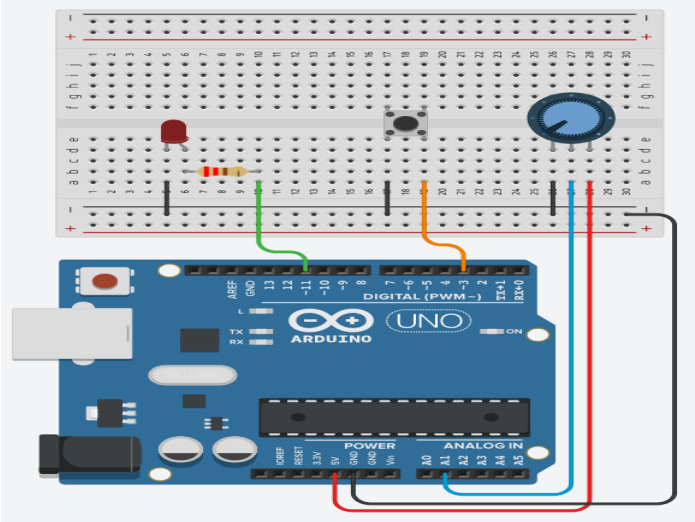
Aim:

you will discover different examples of what you can do with an LED, a push button, and a potentiometer

Requirement:

- Arduino board – I use Uno, but whatever is fine, as long as you have at least a PWM compatible digital pin (for the LED), and an analog pin (for the potentiometer).
- Breadboard.
- Some male-female wires.
- LED.
- 220 Ohm resistor.
- Push button.
- Potentiometer.

Circuit diagram:



Working procedure:

- Connect a GND pin from the Arduino to the line made for GND on the breadboard (the "minus" line).
- LED: Connect the shorter leg to the GND line, and the longer leg to a PWM compatible digital pin, with a 220 Ohm resistor in between. Note: PWM compatible pins are the ones with a "~" next to them.
- Push button: Plug the push button in the middle of the breadboard. Connect one side to the ground, and the other side to a digital pin.

- Potentiometer: Connect one of the extreme leg (right or left) to the ground, and the other extreme leg to 5V on the Arduino. Connect the middle leg to an analog pin.

Program:

```
#define LED_PIN 11
#define BUTTON_PIN 3
#define POTENTIOMETER_PIN A1
void setup()
{
  pinMode(LED_PIN, OUTPUT);
  pinMode(BUTTON_PIN, INPUT_PULLUP);
}
void loop()
{
  if (digitalRead(BUTTON_PIN) == LOW) {
    int potentiometerValue = analogRead(POTENTIOMETER_PIN);
    int brightness = map(potentiometerValue, 0, 1023, 0, 255);
analogWrite(LED_PIN, brightness);
  }
}
```

App 2 – Button powers on/off the LED, potentiometer sets LED's brightness

```
#define LED_PIN 11
#define BUTTON_PIN 3
#define POTENTIOMETER_PIN A1
unsigned long debounceDuration = 50; // millis
unsigned long lastTimeButtonStateChanged = 0;
byte lastButtonState = HIGH;
bool LEDOn = true;
void setup()
{
  pinMode(LED_PIN, OUTPUT);
  pinMode(BUTTON_PIN, INPUT_PULLUP);
}
void loop()
{
  unsigned long timeNow = millis();
  if (timeNow - lastTimeButtonStateChanged > debounceDuration) {
    byte buttonState = digitalRead(BUTTON_PIN);
```

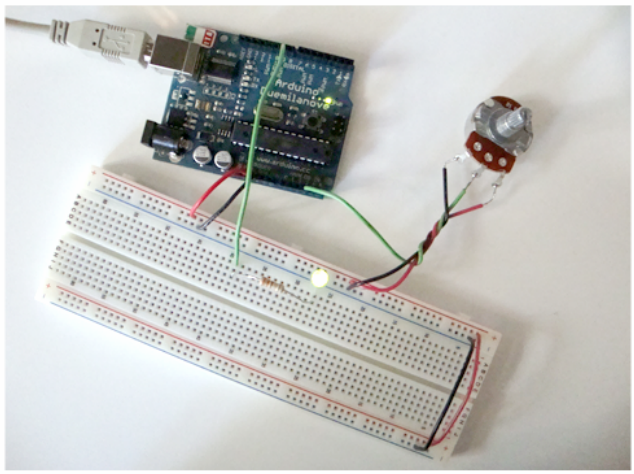
```
    if (buttonState != lastButtonState) {
      lastTimeButtonStateChanged = timeNow;
      lastButtonState = buttonState;
      if (buttonState == HIGH) { // button has been released
        LEDOn = ! LEDOn;
      }
    }
  }
    if (LEDOn) {
    int potentiometerValue = analogRead(POTENTIOMETER_PIN);
    int brightness = map(potentiometerValue, 0, 1023, 0, 255);
    analogWrite(LED_PIN, brightness);
  }
  else {
    digitalWrite(LED_PIN, LOW);
  }
}
```

App 3 – LED blinks on its own, button pauses blink, potentiometer sets blink delay

```
#define LED_PIN 11
#define BUTTON_PIN 3
#define POTENTIOMETER_PIN A1
unsigned long delayBetweenBlink = 500; unsigned
long lastTimeLEDBlinked = 0; unsigned long
debounceDuration = 50; // millis unsigned long
lastTimeButtonStateChanged = 0; byte
lastButtonState = HIGH;
bool blinkEnabled = true;
byte LEDState = LOW; void
setup()
{
  pinMode(LED_PIN, OUTPUT);
  pinMode(BUTTON_PIN, INPUT_PULLUP);
}
void loop()
{
 unsigned long timeNow = millis();
  if (timeNow - lastTimeButtonStateChanged > debounceDuration) { byte
    buttonState = digitalRead(BUTTON_PIN);
    if (buttonState != lastButtonState) { lastTimeButtonStateChanged =
      timeNow;
```

```
      lastButtonState = buttonState;
      if (buttonState == HIGH) { // button has been released blinkEnabled
        = !blinkEnabled;
      }
    }
  }
  int potentiometerValue = analogRead(POTENTIOMETER_PIN);
  delayBetweenBlink = map(potentiometerValue, 0, 1024, 0, 2000); if
  (blinkEnabled) {
    if (timeNow - lastTimeLEDBlinked > delayBetweenBlink) { lastTimeLEDBlinked
      = timeNow;
      LEDState = (LEDState == HIGH) ? LOW : HIGH;
      digitalWrite(LED_PIN, LEDState);
    }
  }
}
```

Output:



Result:

Thus the circuit were created and executed successfully.

Ex.no-4 **Temperature and Humidity Sensor Interface**

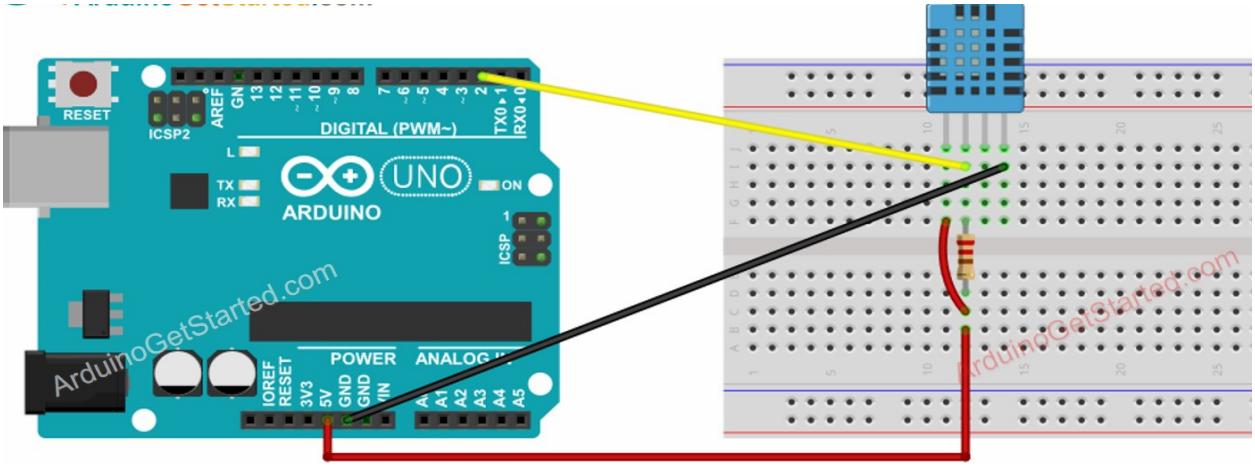
Aim:

In this project, we are going to measure ambient temperature and humidity and display it on a 16x2 LCD screen

Requirements:

- Arduino Nano
- Arduino Nano cable
- USB 2.0 cable
- Temperature and Humidity Sensor DHT11
- 10KΩ resistor
- Breadboard
- Jumper wires

Circuit Diagram:



Program:
Arduino Code – DHT11

```
#include "DHT.h"
#define DHTPIN 2
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);
void setup()
{
Serial.begin(9600);
dht.begin(); // initialize the sensor
}
void loop() {
// wait a few seconds between measurements.
delay(2000);
```

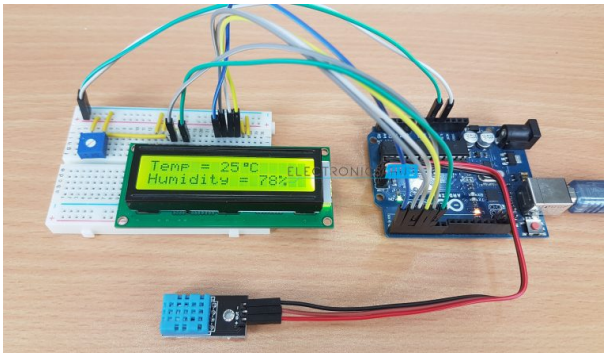
```
// read humidity
float humi  = dht.readHumidity();
// read temperature as Celsius
float tempC = dht.readTemperature();
// read temperature as Fahrenheit
float tempF = dht.readTemperature(true);

// check if any reads failed
if (isnan(humi) || isnan(tempC) || isnan(tempF)) {
Serial.println("Failed to read from DHT sensor!");
} else {
Serial.print("Humidity: ");
Serial.print(humi);
Serial.print("%");
Serial.print("  |  ");
Serial.print("Temperature: ");
Serial.print(tempC);
Serial.print("°C ~ ");
Serial.print(tempF);
Serial.println("°F");
}
}
```

// Including library for dht

```
#include<dht.h>
#include<LiquidCrystal.h>
LiquidCrystal lcd(2, 3, 4, 5, 6, 7);
#define dht_dpin 12
dht DHT;
byte degree[8] =
            {
              0b00011,
              0b00011,
              0b00000,
              0b00000,
              0b00000,
              0b00000,
              0b00000,
              0b00000
            };

void setup()
{
 lcd.begin(16, 2);
 lcd.createChar(1, degree);
 lcd.clear();
 lcd.print("   Humidity   ");
 lcd.setCursor(0,1);
 lcd.print("  Measurement ");
 delay(2000);
```

```
 lcd.clear();
 lcd.print("Circuit Digest ");
 delay(2000);
}
void loop()
{
  DHT.read11(dht_dpin);
  lcd.setCursor(0,0);
  lcd.print("Humidity: ");
  lcd.print(DHT.humidity);   // printing Humidity on LCD
  lcd.print(" %");
  lcd.setCursor(0,1);
  lcd.print("Temperature:");
  lcd.print(DHT.temperature);   // Printing temperature on LCD
  lcd.write(1);
  lcd.print("C");
  delay(500);
}
```

Output:



Result:

Thus the circuit were created and executed successfully.

Ex.no-5 Arduino Flame Sensor Interfacing to Build a Fire Alarm System

Aim:

In this project build a fire alarm system using Arduino flame sensor interface.

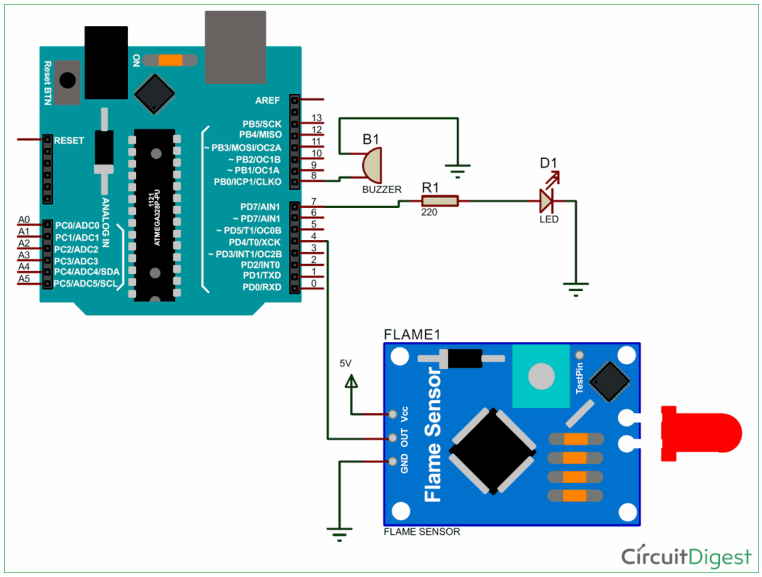
Requirements:

- Arduino Uno (any Arduino board can be used)
- Flame sensor module.
- LED.
- Buzzer.
- Resistor.
- Jumper wires.

Applications of flame sensors

- Hydrogen stations
- Combustion monitors for burners
- Oil and gas pipelines
- Automotive manufacturing facilities
- Nuclear facilities
- Aircraft hangars
- Turbine enclosures

Circuit Diagram:



Program:

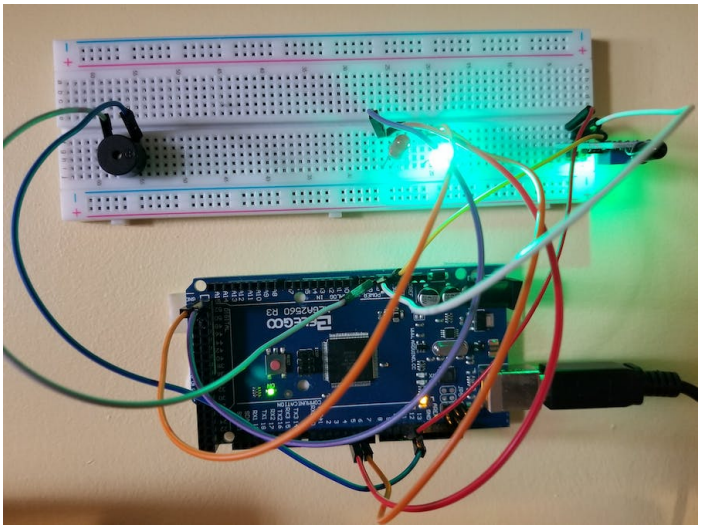
```
void setup()
{
  Serial.begin(9600) ;
  pinMode(buzzer, OUTPUT) ;
  pinMode(LED, OUTPUT) ;
  pinMode(flame_sensor, INPUT) ;
}
```

This line of code reads the digital output from flame sensor and stores it in the variable "*flame_detected*".

```
flame_detected = digitalRead(flame_sensor) ;
if (flame_detected == 1)
  {
    Serial.println("Flame detected...! take action immediately.");
    digitalWrite(buzzer, HIGH);
    digitalWrite(LED, HIGH);
    delay(200);
    digitalWrite(LED, LOW);
    delay(200);
  }
  else
  {
    Serial.println("No flame detected. stay cool");
    digitalWrite(buzzer, LOW);
    digitalWrite(LED, LOW);
  }
  delay(1000);

int buzzer = 8;
int LED = 7;
```

```
int flame_sensor = 4;
int flame_detected;
void setup()
{
  Serial.begin(9600);
  pinMode(buzzer, OUTPUT);
  pinMode(LED, OUTPUT);
  pinMode(flame_sensor, INPUT);
}
void loop()
{
  flame_detected = digitalRead(flame_sensor);
 if (flame_detected == 1)
  {
    Serial.println("Flame detected...! take action immediately.");
    digitalWrite(buzzer, HIGH);
    digitalWrite(LED, HIGH);
    delay(200);
    digitalWrite(LED, LOW);
    delay(200);
  }
  else
  {
    Serial.println("No flame detected. stay cool");
    digitalWrite(buzzer, LOW);
    digitalWrite(LED, LOW);
  }
  delay(1000);
}
```

Output:



Result:

Thus the circuit were created and executed successfully.

Ex.no-6

Remote controlled AC Fan Regulator

AC Fan Speed Control using Arduino and TRIAC

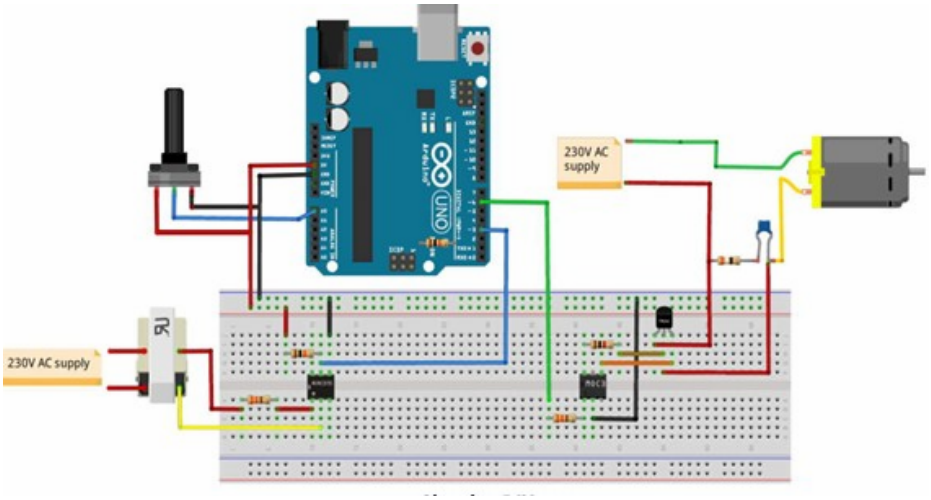
Aim :

In this tutorial, we are going to make a project of AC Fan speed control using Arduino and TRIAC. The circuit diagram discussed in this project is only for educational purposes. Be advised that working with 220V AC mains voltage requires extreme precaution and safety procedures should be followed.

Requirements:

S.no	Component	Value	Quantity
1	Stepdown Transformer	0-9V, 500mA	1
2	Zero crossing detector (optocoupler)	4N25	1
3	Arduino	Uno	1
4	Potentiometer	10KΩ	1
5	Opto-coupler	MOC3021	1
6	TRIAC	BT136	1
7	Axial AC fan	230 VAC	1
8	Resistor	330Ω, 10KΩ, 39Ω, 1KΩ	1, 1, 1, 1
12	Capacitor	10nF	1

Circuit Diagram:


230V AC supply
230V AC supply

Working Procedure :

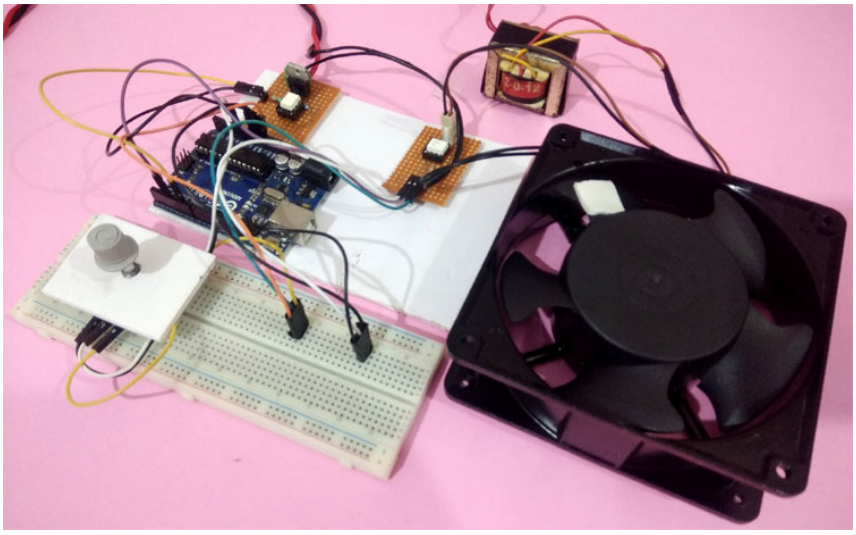
The first step in the working of this circuit is to detect the zero-crossing point, it is the point on the voltage curve where the voltage changes the direction. A 220V mains voltage is stepped down to 9V using a step-down transformer and then it is sent to a 4N25 Optocoupler. This optocoupler has a built-in LED and an output transistor that de-actives when the AC wave goes close to the zero-crossing point and they both are activated when the signal increases to the peak point.

Using this pulse the zero-crossing point is detected by the Arduino. Then we have to control the time period for which the power will on and off. It is done by a BT136 Triac. The speed of the fan is adjusted or varied through a 10K potentiometer. The Arduino will take the input from the potentiometer and generates a PWM signal which is fed into the TRIAC and optocoupler circuit that will result in operating the AC fan at the desired speed. The PWM signal will decide the amount of voltage output to the AC motor that controls the speed of it.

Arduino Code

```
: int TRIAC =
6;
int speed_val=0;
void setup()
{
 pinMode(TRIAC, OUTPUT);
 attachInterrupt(digitalPinToInterrupt(3), zero_crossing, CHANGE);
}
void zero_crossing()
{
 int chop_time = (200*speed_val);
 delayMicroseconds(chop_time);
 digitalWrite(TRIAC, HIGH);
 delayMicroseconds(10);
 digitalWrite(TRIAC, LOW);
}
void loop()
{
   int pot=analogRead(A0);
   int data1 = map(pot, 0, 1023,10,40);
   speed_val=data1;
}
```

Output:



Result:

Thus the circuit were created and executed successfully.

Ex.no : 7

Motion Detection
Interfacing Arduino with PIR Motion Sensor

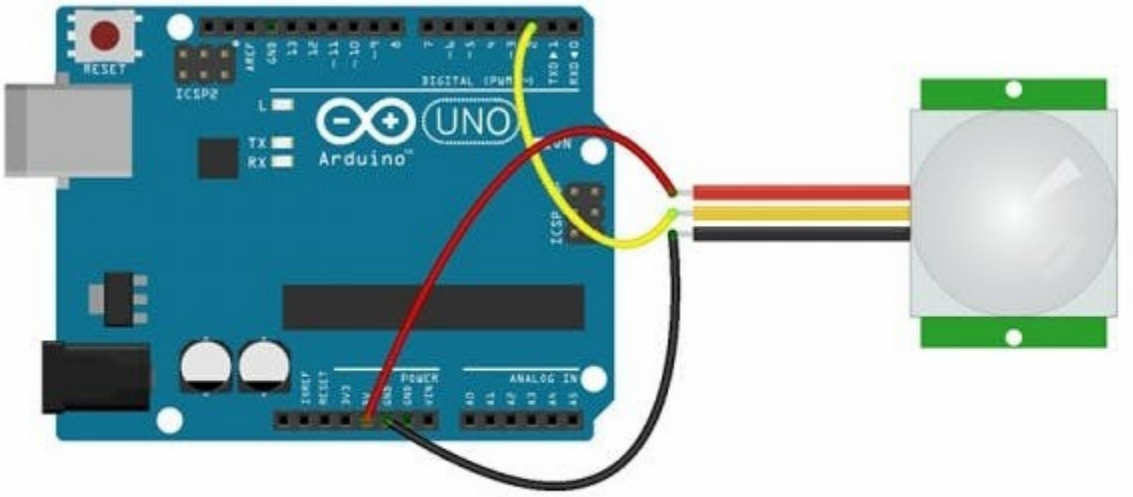
Aim:

To using the PIR Motion Sensors to detect human movement or occupancy in security systems. Initial setup and calibration of these sensors takes about 10 to 60 seconds.

Requirements:

- Arduino UNO
- HC - SR501 PIR Motion Sensor
- 5mm Led
- Resistor 10 (ohm)
- Jumper Wires
- Breadboard

Circuit Diagram :



Working Procedure:

The HC-SR501's infrared imaging sensor is an efficient, inexpensive and adjustable module for detecting motion in the environment. The small size and physical design of this module allow you to easily use it in your project. The output of PIR motion detection sensor can be connected directly to one of the Arduino (or any microcontroller) digital pins. If any motion is detected by the sensor, this pin value will be set to "1". The two potentiometers on the board allow you to adjust the sensitivity and delay time after detecting a movement.

The HC-SR501's infrared imaging sensor is an efficient, inexpensive and adjustable module for detecting motion in the environment. The small size and physical design of this module allow you to easily use it in your project.

The output of PIR motion detection sensor can be connected directly to one of the Arduino (or any microcontroller) digital pins. If any motion is detected by the sensor, this pin value will be set to "1". The two potentiometers on the board allow you to adjust the sensitivity and delay time after detecting a movement. There is a jumper behind this module. If you move the jumper to L position, the sensor will 'toggle' (change state) whenever motion is detected. This is unlikely to be of much use in a practical applications. This mode is called non-triggering or Single Triggering mode.

Moving the jumper to the H position will result in the more usual sensor logic. The sensor will turn on when motion is detected and turn off a while after the last motion is detected. This sensor will reset the timer (which would otherwise turn the output off) each time motion is detected; this would be applicable, for example, for room occupancy lighting control where you don't want the lights to blink off while the unit resets. This is called Retriggering mode. (or repeatable trigger mode).

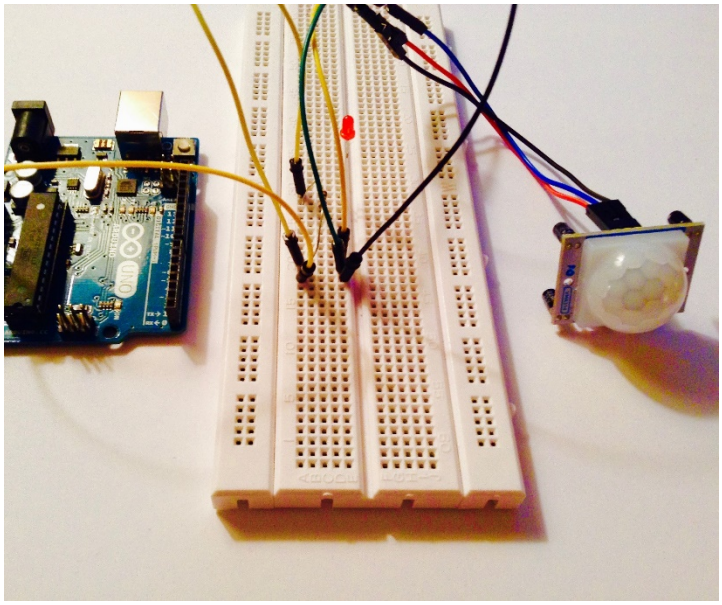
There are also two potentiometers behind this module. By changing the SENSITIVITY potentiometer, you can reduce or increase the sensitivity of the sensor (clockwise increase), and also by changing TIME potentiometer the output delay after movement detection will be changed.

Program:

```
int ledPin = 13;                // LED 4

int pirPin = 2;                 // PIR Out pin

int pirStat = 0;                  // PIR status

void setup() {

 pinMode(ledPin, OUTPUT);
 pinMode(pirPin, INPUT);
 Serial.begin(9600);
}

void loop(){

 pirStat = digitalRead(pirPin);
 if (pirStat == HIGH) {            // if motion detected
```

```
  digitalWrite(ledPin, HIGH);  // turn LED ON
  Serial.println("Hey I got you!!!");
 }

 else {
  digitalWrite(ledPin, LOW); // turn LED OFF if we have no motion

 }
}
```

Output:



Result:

Thus the circuit were created and executed successfully.

Ex.no : 8

Playing Music
Simple Arduino Audio Player with Amplifier with LM386

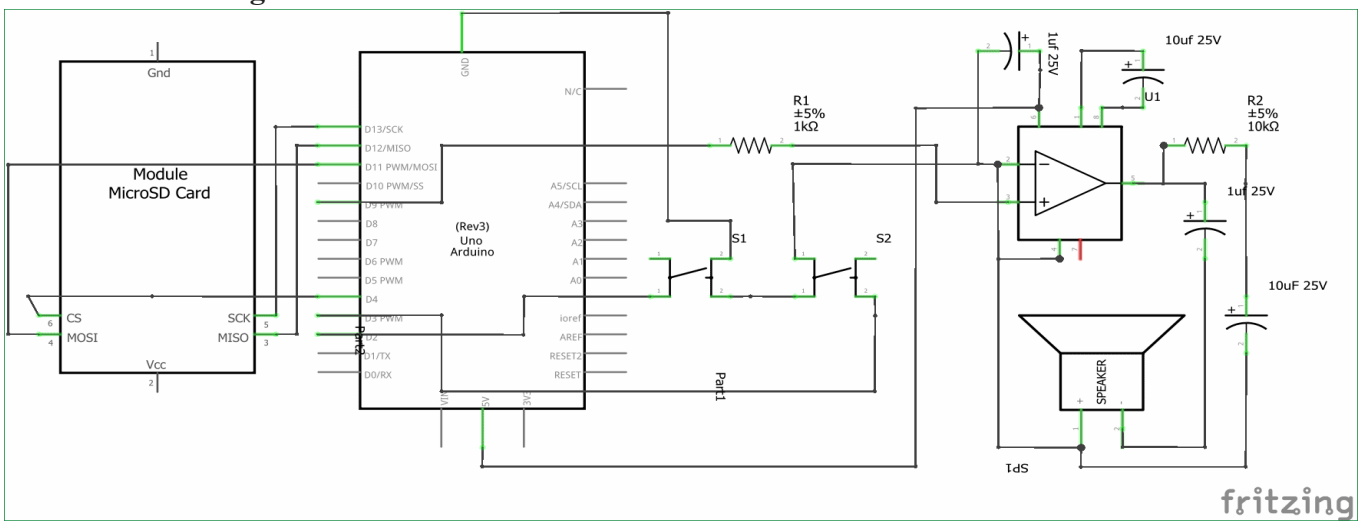
Aim:

In this project, we will play the .wav music files stores in an SD card. We will program the Arduino to read these .wav files and play the audio on a speak through an LM386 Audio amplifier.

Requirements:

- Arduino UNO
- SD Card Reader module
- SD card
- LM386 Audio Amplifier
- 10uf Capacitor (2 Nos)
- 100uf Capacitor (2 Nos)
- 1K,10K Resistor
- Push buttons (2 Nos)
- Breadboard
- Connecting Wires

Circuit Diagram :



Working Procedure:

Once we are ready with the Hardware and the SD card, we are just one step away playing those songs. Insert the card into your SD card module and follow the steps below.

Step 1: As said earlier we will be using a library to make this project work. The link for the library is given below. Click on it and select "Clone or download" and choose download as ZIP.

- TMRpcm library

Step 2: Add this Zip file into your Arduino IDE by selecting Sketch->Include Library -> Add .ZIP Library as shown below and select the ZIP file that we just downloaded.

Step 3: The complete program of the **arduino music player project** is given at the end of this article, simply copy it and paste it in the Arduino Program. Now, click on Upload and get ready to play your audio files. The program is self explanatory since they have the comment lines. But, I have also explained the ability of the TMRpcm library below.

Playing an audio file:

You can play any audio that is stored in Wav format inside the SD card module by using the line below.

```
music.play("3.wav");
//object name.play ("FileName.wav");
```

You can use this line at places where you want to trigger the Audio

Pause an audio File:

To pause an Audio file, you can simply call the line below.

```
music.pause();
//objectname.pause();
```

Forwarding/Rewinding an Audio:

There are not direct ways to forward or rewind an Audio file, but you can use the line below to play a song at a particular time. This can be used to forward/rewind with some additional programming.

```
music.play("2.wav",33); //Plays the song from 33rd second
//objectname.play("Filename.wav",time in second);
```

Setting the quality of the audio:

The library gives us two qualities to play the music, one is to play as normal mode the other to play with 2X oversampling.

```
music.quality(0); //Normal Mode
music.quality(1); //2X over sampling mode
```

Setting the Volume of the audio:

Yes, you can control the volume of the audio through software. You can simply set the volume by using the line below. Higher music volumes tend to affect the quality of the audio, hence use hardware control when possible.

```
music.setVolume(5);        //Plays the song at volume 5
//objectname.setVolume(Volume level);
```

After programming your Arduino simply press the button connected to pin 2 and your Arduino will play the first song (saved as 1.wav) for you. Now you can press the button again to

change your track to the next song that is to play 2.wav. Likewise you can navigate to all four songs.

You can also play/Pause the song by pressing the button connected to pin 3. Press it once to pause the song and press it again to play it from where it stopped. Watch the **video below** for complete working (or maybe to relax yourself with some songs).

Arduino Code :

Arduino Based Music Player

This example shows how to play three songs from SD card by pressing a push button

The circuit:

* Push Button on pin 2 and 3

* Audio Out - pin 9

* SD card attached to SPI bus as follows:

** MOSI - pin 11

** MISO - pin 12

** CLK - pin 13

** CS - pin 4

Program:

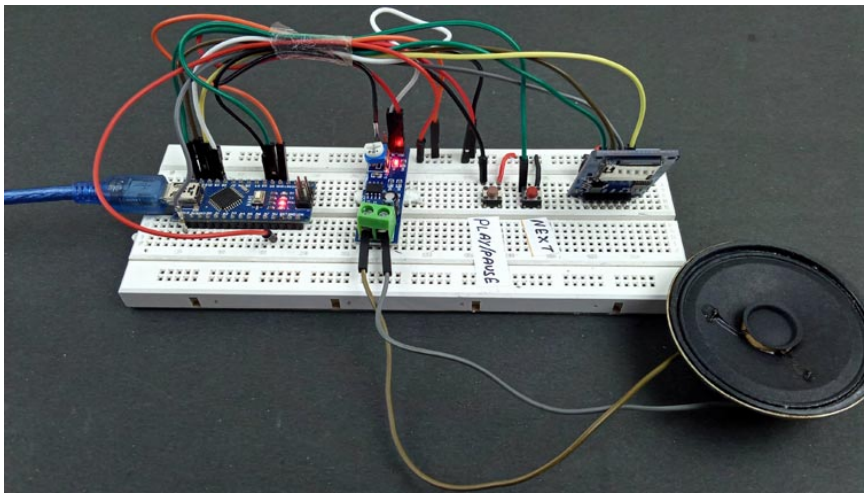
```
#include "SD.h" //Lib to read SD card

#include "TMRpcm.h" //Lib to play auido

#include "SPI.h" //SPI lib for SD card
```

```
#define SD_ChipSelectPin 4 //Chip select is pin number 4

TMRpcm music; //Lib object is named "music"

int song_number=0;

boolean debounce1=true;

boolean debounce2=true;

boolean play_pause;

void setup(){

music.speakerPin = 9; //Auido out on pin 9

Serial.begin(9600); //Serial Com for debugging

if (!SD.begin(SD_ChipSelectPin)) {

Serial.println("SD fail");

return;

}

pinMode(2, INPUT_PULLUP); //Button 1 with internal pull up to chage track

pinMode(3, INPUT_PULLUP); //Button 2 with internal pull up to play/pause

pinMode(3, INPUT_PULLUP); //Button 2 with internal pull up to fast forward

music.setVolume(5);    //  0 to 7. Set volume level

music.quality(1);        //  Set 1 for 2x oversampling Set 0 for normal

//music.volume(0);        //   1(up) or 0(down) to control volume

//music.play("filename",30); plays a file starting at 30 seconds into the track
```

```
}
void loop()
{
 if (digitalRead(2)==LOW  && debounce1 == true) //Button 1 Pressed
 {
 song_number++;
 if (song_number==5)
 {song_number=1;}
 debounce1=false;
 Serial.println("KEY PRESSED");
 Serial.print("song_number=");
 Serial.println(song_number);
 if (song_number ==1)
 {music.play("1.wav",10);} //Play song 1 from 10th second
 if (song_number ==2)
 {music.play("2.wav",33);} //Play song 2 from 33rd second
 if (song_number ==3)
 {music.play("3.wav");} //Play song 3 from start
 if (song_number ==4)
 {music.play("4.wav",25);} //Play song 4 from 25th second
```

```
if (digitalRead(3)==LOW  && debounce2 == true) //Button 2 Pressed

{

music.pause();  Serial.println("PLAY / PAUSE");

debounce2=false;

}

if (digitalRead(2)==HIGH) //Avoid debounce

debounce1=true;

if (digitalRead(3)==HIGH)//Avoid debounce

debounce2=true;

}

}
```

Output:



Result :

Thus the circuit were created and executed successfully.

Ex.no : 9 **Arduino Traffic Light Controller**

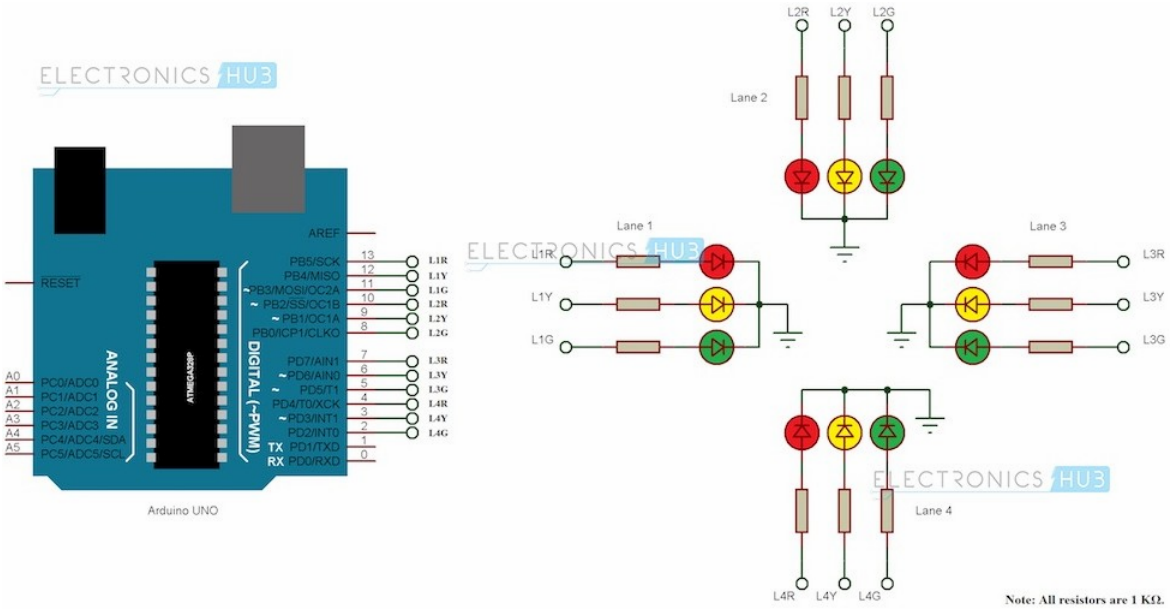
Aim:

In this project, an Arduino based Traffic Light Controller system is designed. It is a simple implementation of traffic lights system but can be extended to a real time system with programmable timings, pedestrian lighting etc.

Requirements:

- Arduino UNO
- 1KΩ Resistor X 12
- Red LEDs X 4
- Yellow LEDs X 4
- Green LEDs X 4
- Connecting wires
- Prototyping board
- Power adapter

Circuit Diagram:



Working Procedure:

The real time traffic light controller is a complex piece of equipment which consists of power cabinet, main controller or processor, relays, control panel with switches or keys, communication ports etc. In this project, a simple traffic light system for a 4 way intersection is implemented using Arduino UNO. Although it is not the ideal implementation for real life scenarios, it gives an idea of the process behind the traffic light control system

The aim of the project is to implement a simple traffic light controller using Arduino UNO, where the traffic is controlled in a pre-defined timing system. The working of the project is very simple and is explained below. Consider the following gif image showing a loop of traffic light operations. The project is also implemented in the same manner.

In that, first the Lane 1 gets its Green light turned. Hence, in all the other Lanes, their corresponding Red lights are turned on. After a time delay of predefined time say 5 seconds, the Green light in the Lane 3 must be turned on and the Green light in the Lane 1 must be turned off. As a warning indicator, the Yellow light in Lane 1 is tuned on indicating that the red light is about to light up. Similarly, the yellow light in the Lane 3 is also turned as an indication that the green light about to be turned on.

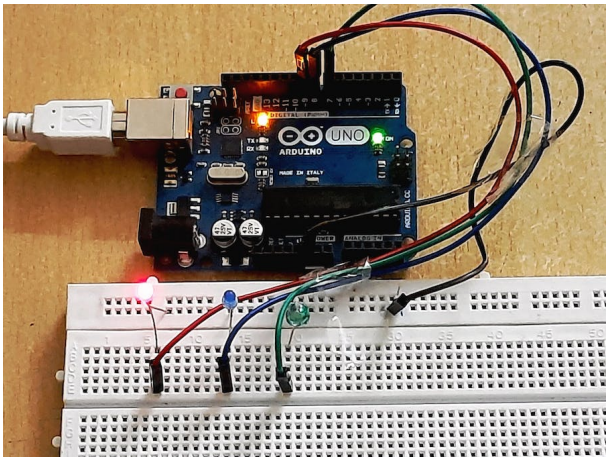
The yellow lights in Lanes 1 and 3 are turned for a small duration say 2 seconds after with the red light in the Lane 1 is turned on and green light in Lane 3 is also turned on. The green light in Lane 3 is also turned on for a predefined time and the process moves forward to Lane 4 and finally Lane 2. The system then loops back to Lane 1 where the process mentioned above will be repeated all over again.

Arduino Code:

```
int Lane1[] = {13,12,11}; // Lane 1 Red, Yellow and Green int
Lane2[] = {10,9,8};// Lane 2 Red, Yellow and Green  int Lane3[] =
{7,6,5};// Lane 3 Red, Yellow and Green
int Lane4[] = {4,3,2};// Lane 4 Red, Yellow and Green

void setup()
{



  for (int i = 0; i < 3; i++)
  {
    pinMode(Lane1[i],        OUTPUT);
    pinMode(Lane2[i],        OUTPUT);
    pinMode(Lane3[i],        OUTPUT);
    pinMode(Lane4[i], OUTPUT);
  }
  for (int i = 0; i < 3; i++)
  {
    digitalWrite(Lane1[i],        LOW);
    digitalWrite(Lane2[i],        LOW);
    digitalWrite(Lane3[i],        LOW);
    digitalWrite(Lane4[i], LOW);
  }

}
```

```
void loop()
 {
  digitalWrite(Lane1[2], HIGH);
  digitalWrite(Lane3[0], HIGH);
  digitalWrite(Lane4[0], HIGH);
  digitalWrite(Lane2[0], HIGH);
  delay(7000); digitalWrite(Lane1[2],
  LOW); digitalWrite(Lane3[0], LOW);
  digitalWrite(Lane1[1], HIGH);
  digitalWrite(Lane3[1], HIGH);
  delay(3000); digitalWrite(Lane1[1],
  LOW); digitalWrite(Lane3[1], LOW);
  digitalWrite(Lane1[0], HIGH);
  digitalWrite(Lane3[2], HIGH);
  delay(7000); digitalWrite(Lane3[2],
  LOW); digitalWrite(Lane4[0], LOW);
  digitalWrite(Lane3[1], HIGH);
  digitalWrite(Lane4[1], HIGH);
  delay(3000); digitalWrite(Lane3[1],
  LOW);
 digitalWrite(Lane4[1], LOW);
 digitalWrite(Lane3[0], HIGH);
 digitalWrite(Lane4[2], HIGH);
 delay(7000); digitalWrite(Lane4[2],
 LOW); digitalWrite(Lane2[0], LOW);
 digitalWrite(Lane4[1], HIGH);
 digitalWrite(Lane2[1], HIGH);
 delay(3000); digitalWrite(Lane4[1],
 LOW); digitalWrite(Lane2[1], LOW);
 digitalWrite(Lane4[0], HIGH);
 digitalWrite(Lane2[2], HIGH);
 delay(7000); digitalWrite(Lane1[0],
 LOW); digitalWrite(Lane2[2], LOW);
 digitalWrite(Lane1[1], HIGH);
 digitalWrite(Lane2[1], HIGH);
 delay(3000); digitalWrite(Lane2[1],
 LOW); digitalWrite(Lane1[1], LOW);
 }
```

Output:



Result :

Thus the circuit were created and executed successfully.

Ex.no : 10

Password based Door Lock System using Arduino

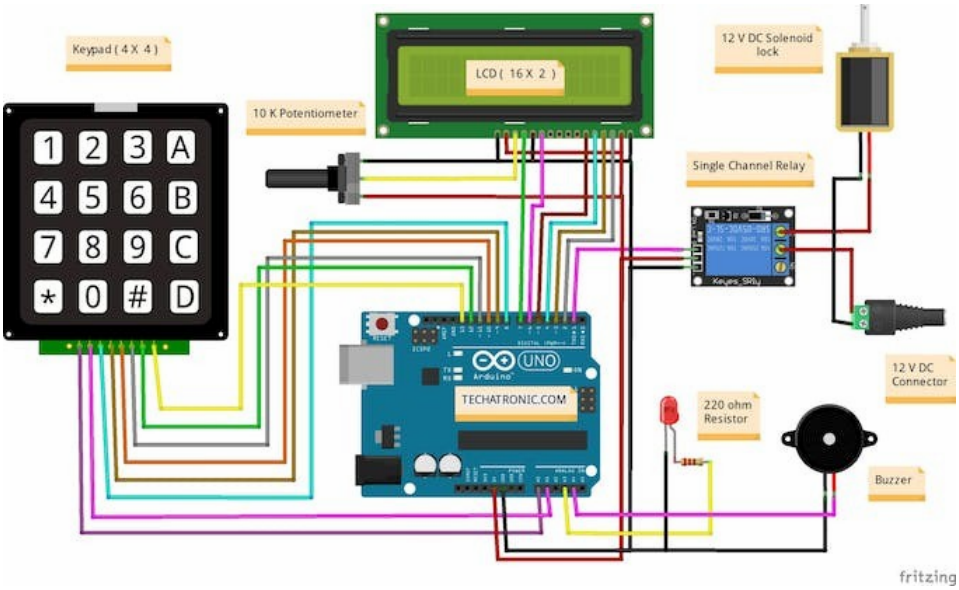
Aim:

In this project, we are using a keypad as we have to enter the passcode so that the system grants us access.

Requirements:

- Arduino Nano
- Arduino nano cable
- LED
- 220-ohm Resistor
- 5V Relay
- Solenoid lock
- Zero PCB
- Soldering Iron
- Soldering Wire
- Hookup wire.
- 16×2 LCD

Circuit Diagram:



Working Procedure:

Connect the 5 volts adapter to the circuit so that Arduino can start working. We are also sharing the code and circuit diagram for this project. There is a solenoidal lock connected with the relay module and controlled by the Arduino. If you want to open the door then enter the passcode and you can also see the values on a 16x2 LCD display that we use here. There is a pre- defined passcode in the system so when someone enters the key the system will start comparing that data with the input data and if both are the same then the door will open else the door will not open until you enter the correct key.

Arduino Code:

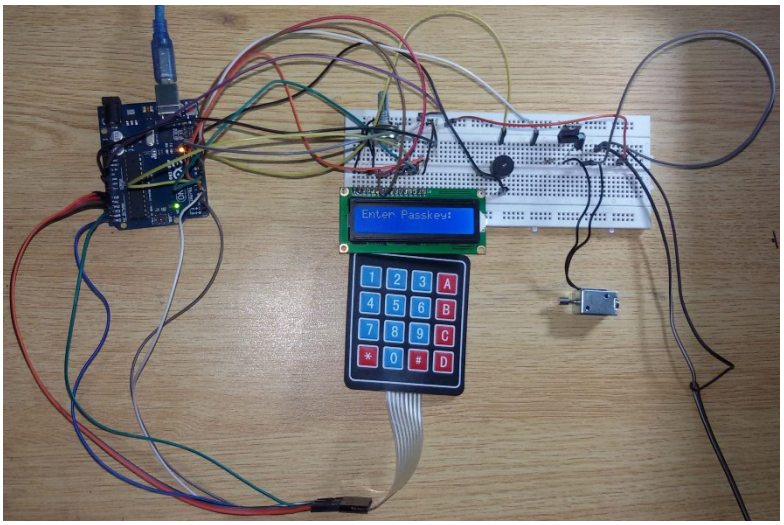
```
#include <Keypad.h>
#include<LiquidCrystal.h>
#include<EEPROM.h>
#define led A3
#define buzzer A4
LiquidCrystal lcd(7,6,5,4,3,2); char
password[4];
char pass[4],pass1[4];
int i=0;
char customKey=0;
const byte ROWS = 4; //four rows const byte
COLS = 4; //four columns char
hexaKeys[ROWS][COLS] = {
{'1','2','3','A'},
{'4','5','6','B'},
{'7','8','9','C'},
{'*','0','#','D'}
};

byte rowPins[ROWS] = {A0, A1, 8, 9};
//connect to the row pinouts of the keypad
byte colPins[COLS] = {10, 11, 12, 13};
//connect to the column pinouts of the keypad
//initialize an instance of class NewKeypad
Keypad customKeypad = Keypad( makeKeymap(hexaKeys), rowPins, colPins, ROWS,
COLS);
void setup()
{
lcd.begin(16,2);
pniMode(led, OUTPUT);
pinMode(buzzer,OUTUT);
```

```
      pinMode(1, OUTPUT);
      //  pinMode(m12, OUTPUT); lcd.print(" Electronic "); lcd.setCursor(0,1);
      lcd.print(" Keypad Lock "); delay(2000);


 lcd.clear();
 lcd.print("Enter Ur Passkey:");
 lcd.setCursor(0,1);
 for(int j=0;j<4;j++)
 EEPROM.write(j, j+49); for(int
 j=0;j<4;j++)
 pass[j]=EEPROM.read(j);
}
void loop()
{
 customKey = customKeypad.getKey();
 if(customKey=='#')
 change();
 if (customKey)
 {
    password[i++]=customKey;
    lcd.print(customKey); beep();
 }
 if(i==4)
 {
  delay(200);
  for(int j=0;j<4;j++) pass[j]=EEPROM.read(j);
  if(!(strncmp(password, pass,4)))
  {
     digitalWrite(1,  HIGH
     );
    digitalWrite(led, HIGH); beep();
    lcd.clear();  lcd.print("Passkey
    Accepted"); delay(2000);
    lcd.setCursor(0,1); lcd.print("#.Change
    Passkey"); delay(2000);
    lcd.clear();
    lcd.print("Enter Passkey:");
    lcd.setCursor(0,1);
    i=0;
    digitalWrite(led, LOW); digitalWrite(1,
```

```
      LOW);
    }
    else
    {
      digitalWrite(buzzer, HIGH); lcd.clear();
      lcd.print("Access Denied...");
      lcd.setCursor(0,1); lcd.print("#.Change
      Passkey"); delay(2000);
      lcd.clear();  lcd.print("Enter
      Passkey:"); lcd.setCursor(0,1);
      i=0;
      digitalWrite(buzzer,  LOW);
    }
  }
}
void change()
{
  int j=0; lcd.clear();
  lcd.print("UR Current Passk");
  lcd.setCursor(0,1);
  while(j<4)
  {
    char key=customKeypad.getKey(); if(key)
    {
      pass1[j++]=key;
      lcd.print(key);
      beep();
    }
    key=0;
  }
  delay(500);
  if((strncmp(pass1, pass, 4)))  { lcd.clear();
    lcd.print("Wrong Passkey...");
    lcd.setCursor(0,1); lcd.print("Better Luck
    Again"); delay(1000);
  }
  else
  {
    j=0;
  lcd.clear();
  lcd.print("Enter New Passk:"); lcd.setCursor(0,1);
```

```
  while(j<4)
  {
   char key=customKeypad.getKey(); if(key)
   {
    pass[j]=key; lcd.print(key);
    EEPROM.write(j,key); j++;
    beep();
   }
  }
  lcd.print("  Done                    ");
  delay(1000);
  }
  lcd.clear();
  lcd.print("Enter Ur Passk:");
  lcd.setCursor(0,1); customKey=0;
}
void beep()
{
  digitalWrite(buzzer, HIGH);
  delay(20); digitalWrite(buzzer,
  LOW);
}
```

Output:



Result:

Thus the circuit were created and executed successfully.